Context Management and Self-Adaptivity for
Situation-Aware Smart Software Systems

by

Norha Milena Villegas Machado
B. Systems Engineering, Icesi University, 2002
EM. Management of Information Systems, Icesi University, 2005

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Context Management and Self-Adaptivity for
Situation-Aware Smart Software Systems

by

Norha Milena Villegas Machado
B. Systems Engineering, Icesi University, 2002
EM. Management of Information Systems, Icesi University, 2005

Supervisory Committee

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science)

Dr. Margaret-Anne Storey, Departmental Member
(Department of Computer Science)

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Dr. Pan Agathoklis, Outside Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science)

Dr. Margaret-Anne Storey, Departmental Member
(Department of Computer Science)

Dr. Ulrike Stege, Departmental Member
(Department of Computer Science)

Dr. Pan Agathoklis, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Our society is increasingly demanding situation-aware smarter software (SASS) systems, whose goals change over time and depend on context situations. A system with such properties must sense their dynamic environment and respond to changes quickly, accurately, and reliably, that is, to be context-aware and self-adaptive.

The problem addressed in this dissertation is the dynamic management of context information, with the goal of improving the relevance of SASS systems' context-aware capabilities with respect to changes in their requirements and execution environment. Therefore, this dissertation focuses on the investigation of dynamic context management and self-adaptivity to: (i) improve context-awareness and exploit context information to enhance quality of user experience in SASS systems, and (ii) improve the dynamic capabilities of self-adaptivity in SASS systems.

Context-awareness and *self-adaptivity* pose significant challenges for the engineering of SASS systems. Regarding context-awareness, the first challenge addressed in this dissertation is the impossibility of fully specifying environmental entities and the

corresponding monitoring requirements at design-time. The second challenge arises from the continuous evolution of monitoring requirements due to changes in the system caused by self-adaptation. As a result, context monitoring strategies must be modeled and managed in such a way that they support the addition and deletion of context types and monitoring conditions at runtime. For this, the user must be integrated into the dynamic context management process. Concerning self-adaptivity, the third challenge is to control the dynamicity of adaptation goals, adaptation mechanisms, and monitoring infrastructures, and the way they affect each other in the adaptation process. This is to preserve the effectiveness of context monitoring requirements and thus self-adaptation. The fourth challenge, related also to self-adaptivity, concerns the assessment of adaptation mechanisms at runtime to prevent undesirable system states as a result of self-adaptation.

Given these challenges, to improve context-awareness we made three contributions. First, we proposed the *personal context sphere* concept to empower users to control the life cycle of personal context information in user-centric SASS systems. Second, we proposed the SmarterContext ontology to model context information and its monitoring requirements supporting changes in these models at runtime. Third, we proposed an efficient context processing engine to discover implicit contextual facts from context information specified in changing context models.

To improve self-adaptivity we made three contributions. First, we proposed a framework for the identification of adaptation properties and goals, which is useful to evaluate self-adaptivity and to derive monitoring requirements mapped to adaptation goals. Second, we proposed a reference model for designing highly dynamic self-adaptive systems, for which the continuous pertinence between monitoring mechanisms and both changing system goals and context situations is a major concern. Third, we proposed a model with explicit validation and verification (V&V) tasks for self-adaptive software, where dynamic context monitoring plays a major role.

The seventh contribution of this dissertation, the implementation of SmarterContext infrastructure, addresses both context-awareness and self-adaptivity.

To evaluate our contributions, qualitatively and quantitatively, we conducted several comprehensive literature reviews, a case study on user-centric situation-aware online shopping, and a case study on dynamic governance of service-oriented applications.

Contents

Supervisory Committee ii

Abstract iii

Table of Contents v

List of Tables xi

List of Figures xiii

Acknowledgements xvi

Dedication xviii

1 Introduction 1

1.1 Motivation . . . 1

1.2 Problem Statement, Challenges and Research Questions . . . 4

1.2.1 Problem Statement . . . 4

1.2.2 Research Challenges . . . 5

1.2.3 Research Questions . . . 6

1.3 Methodological Aspects . . . 6

1.3.1 Research Methodology . . . 6

1.3.2 Research Approach . . . 8

1.3.3 Dissertation Goals . . . 13

1.4 Research Contributions . . . 14

1.4.1 Context Management . . . 14

1.4.2 Self-Adaptivity . . . 16

1.4.3 Publications Derived from this Dissertation . . . 18

1.5 Dissertation Outline . . . 20

1.6 Chapter Summary . . . 23

2 Research Background **25**
2.1 The Smart Internet . . . 25
2.1.1 The Smart Internet Vision . . . 26
2.1.2 Smart Interactions and Smart Services . . . 28
2.1.3 Context-Awareness Requirements in the Smart Internet . . . 29
2.1.4 The Personal Web . . . 31
2.1.5 Smarter Commerce . . . 33
2.1.6 Future Internet Perspectives . . . 35
2.2 Context Awareness . . . 37
2.2.1 Situation Awareness . . . 38
2.2.2 Characterization of Context Information . . . 39
2.2.3 Context Modeling . . . 42
2.2.4 Context Life Cycle Management . . . 45
2.3 Semantic Web . . . 49
2.3.1 Linked Data and Resource Description Framework . . . 51
2.3.2 Vocabularies and Ontologies . . . 53
2.4 Feedback Control for Self-Adaptivity . . . 56
2.4.1 Feedback Loops . . . 56
2.4.2 Adaptive Control . . . 59
2.5 Self-Adaptive Software Systems . . . 61
2.5.1 A Survey of Self-Adaptive Approaches . . . 62
2.6 Discussion . . . 64
2.6.1 The Smart Internet and the Personal Web . . . 64
2.6.2 Context-Awareness . . . 65
2.6.3 Self-Adaptive Software Systems . . . 66
2.7 Chapter Summary . . . 67

3 The Personal Context Sphere **68**
3.1 Improving User Quality of Experience with Personal Context . . . 70
3.1.1 Situation-Aware Smarter Shopping . . . 71
3.1.2 User-driven Context Management . . . 72
3.2 Realizing the Visions of the Smart Internet and the Personal Web . . . 74
3.2.1 Personal Context Spheres and the Smart Internet . . . 75
3.2.2 Personal Context Spheres and the Personal Web . . . 76

3.3 User-controlled Data Privacy and Security . . . 77
3.4 Chapter Summary . . . 79

4 The SmarterContext Ontology 81
4.1 Design Methodology and Requirements . . . 81
4.1.1 Methodological Aspects . . . 82
4.1.2 Requirements Analysis . . . 83
4.1.3 Extensibility and Modularity . . . 84
4.2 The Foundational Module: General Context . . . 86
4.3 Application to the Personal Web . . . 91
4.3.1 The Personal Web Context (PWC) Module . . . 91
4.3.2 Application to Situation-Aware Smarter Shopping . . . 95
4.3.3 Context Representation and Personal Context Spheres . . . 99
4.4 Chapter Summary . . . 99

5 The SmarterContext Reasoning Engine 101
5.1 Runtime Graph-based Models for Dynamic Context Reasoning . . . 103
5.1.1 Labeled Digraphs . . . 104
5.1.2 Resource Description Framework (RDF) Graphs . . . 104
5.1.3 Contextual RDF Graphs . . . 105
5.2 Context Reasoning with RDFS and OWL-Lite . . . 107
5.2.1 RDFS and OWL-Lite Deduction Rules . . . 107
5.2.2 SmarterContext Deduction Rules . . . 112
5.3 Context Reasoning with Structural Context Patterns . . . 114
5.3.1 Structural Context Patterns in SmarterContext . . . 115
5.4 Chapter Summary . . . 140

6 A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems 141
6.1 A Characterization Model for Self-Adaptive Software . . . 143
6.2 From Adaptation Properties and Goals to Monitoring Requirements . . . 146
6.2.1 Quality Attributes as Adaptation Goals . . . 147
6.2.2 Adaptation Properties . . . 148
6.2.3 Mapping Adaptation Properties to Quality Attributes . . . 151
6.2.4 Adaptation Metrics . . . 154
6.3 Chapter Summary . . . 157

7 Dynamico: A Reference Model for Improving Self-Adaptivity 158
7.1 Improving the Dynamic Capabilities of Software Systems 160
7.1.1 Dynamic SOA Governance 160
7.1.2 The Need for Dynamic Context Monitoring 161
7.2 Design Drivers in the Engineering of Self-Adaptive Software 163
7.2.1 The Three Levels of Dynamics 163
7.3 Our Proposed Reference Model 164
7.3.1 Addressing Separation of Concerns 165
7.3.2 The Control Objectives Feedback Loop 168
7.3.3 The Adaptation Feedback Loop 170
7.3.4 The Context Monitoring Feedback Loop 171
7.3.5 Feedback Loop Interactions 173
7.4 Governing and Controlling Feedback Loop Interactions 175
7.5 Possible Dynamico Variations 176
7.6 Chapter Summary 177

8 A Model for Explicit Runtime V&V Tasks in Self-Adaptive Systems 179
8.1 The Viability Zone 181
8.1.1 V&V under Viability Zone Dynamics 182
8.2 The What, Where and When Questions of Runtime V&V 183
8.2.1 What: Requirements and Adaptation Properties 183
8.2.2 Where: Separation of Concerns 183
8.2.3 When: V&V in the Adaptation Process 185
8.3 Making V&V Explicit in the Self-Adaptation Loop 186
8.3.1 The Runtime Validator and Verifier 188
8.3.2 The V&V Monitors 189
8.3.3 Assurance of Runtime V&V Tasks 189
8.4 Runtime V&V Enablers 190
8.4.1 Requirements and Adaptation Properties at Runtime 191
8.4.2 Models at Runtime 191
8.4.3 Dynamic Context Monitoring 193
8.5 Chapter Summary 195

9 Implementation of the SmarterContext Infrastructure 197

9.1 Realizing Dynamic Context Monitoring . . . 198
9.1.1 The SmarterContext Generic Architecture . . . 198
9.1.2 Services in the SmarterContext Generic Architecture . . . 201
9.1.3 The SmarterContext Monitoring Feedback Loop Architecture 203
9.1.4 Services in the Monitoring Feedback Loop Composite . . . 205
9.2 Implementation of the Context Reasoning Engine (CoRE) . . . 207
9.2.1 Implementation of Contextual RDF Graphs . . . 207
9.2.2 The SmarterContext CoRE Artifacts . . . 208
9.2.3 Optimization of the SmarterContext Reasoning Engine . . . 209
9.3 Realizing User-Controlled Privacy and Security . . . 215
9.3.1 Modified Features of PRE in Surprise . . . 217
9.3.2 Redefinition of PRE Policies in Surprise . . . 217
9.3.3 New Features . . . 219
9.4 Chapter Summary . . . 220

10 Evaluation **222**
10.1 E-commerce Value Creation with Personal Context Spheres . . . 223
10.1.1 Enabling new E-Commerce Business Models with SmarterContext . . . 224
10.1.2 Exploiting the Value of Context through SmarterContext 225
10.2 Improving User Quality of Experience . . . 230
10.2.1 Applying SmarterContext to User-Centric SASS E-Commerce Applications . . . 230
10.2.2 Realizing Self-Adaptation . . . 232
10.2.3 Improving the Accuracy of Deal Recommendations with SmarterContext . . . 234
10.3 Improving Self-Adaptivity . . . 242
10.3.1 Applying SmarterContext to Dynamic SOA Governance . 243
10.3.2 Realizing Dynamic Monitoring Strategies . . . 245
10.3.3 Experimental Evaluation . . . 250
10.3.4 Evaluation Criteria . . . 252
10.3.5 Performance . . . 252
10.3.6 Engineering Effort . . . 254
10.3.7 Maintaining System Quality Attributes . . . 255
10.4 Efficiency of the SmarterContext Reasoning Engine . . . 256

10.4.1 Worst Case Running Time Analysis . . . 257
10.4.2 Improving Efficiency with Structural Context Patterns and Indices . . . 263
10.5 Chapter Summary . . . 265

11 Summary and Conclusions **266**
11.1 Dissertation Summary . . . 266
11.1.1 Addressed Challenges and Goals . . . 267
11.1.2 Contributions . . . 268
11.2 Limitations . . . 272
11.3 Future Work . . . 276

Glossary **280**

Acronyms **285**

References **287**

Apendices **311**

A Context Information Survey: Methodology and Demography **311**

B A Personal Context Sphere Example **316**

C Characterizing Self-Adaptive Software Systems **318**

D Modified Features of PRE in Surprise **321**

E Collaborative Filtering **323**
E.0.1 Calculating Similarities . . . 323
E.0.2 Rating Prediction . . . 324
E.0.3 Baseline Approaches . . . 324

List of Tables

Table 2.1 RDF Schema classes used in the SmarterContext ontology. . . 53
Table 2.2 RDF Schema properties used in the SmarterContext ontology. 54
Table 2.3 OWL-Lite properties used in the SmarterContext ontology. . . 55
Table 2.4 Characterization summary of the self-adaptation approaches analyzed in our survey. 63

Table 4.1 Linked data principles applied to our SmarterContext ontology 82
Table 4.2 RDF and OWL schemas, and ontologies used in the smarter commerce case study. 86
Table 4.3 Context entities defined in the SmarterContext ontology' GC module 88
Table 4.4 Object properties defined in the SmarterContext ontology's GC module 89
Table 4.5 Data properties defined in the SmarterContext ontology's GC module 90
Table 4.6 Context entities defined in the SmarterContext ontology's PWC module 92
Table 4.7 Object properties defined in the SmarterContext ontology's PWC module 94
Table 4.8 Data properties defined in the SmarterContext ontology's PWC module 95
Table 4.9 Context entities defined in the SmarterContext ontology's Shopping module 96
Table 4.10 Object properties defined in the SmarterContext ontology's Shopping module 97
Table 4.11 Data properties defined in the SmarterContext ontology's Shopping module 98

Table 5.1 SmarterContext properties for supporting context reasoning based on structural patterns. 116

Table 6.1 Classification of adaptation properties 151
Table 6.2 Mapping properties to quality attributes 152

Table 7.1 Adding a new throughput SLO to the existing SLA 162

Table 9.1 Self-Adaptive Capabilities of SmarterContext 206

Table 10.1 Validation results for SmarterDeals. 241
Table 10.2 Mapping between control objectives and monitoring strategies 251
Table 10.3 Average settling time and overhead (ms) of the monitoring infrastructure adaptation for SLA-v1 and SLA-v2 253
Table 10.4 Processing time of transitive closure for gc:locatedIn 264

Table B.1 RDF triples that define the PCS for user Norha 317

Table C.1 Characterization of the self-adaptation approaches analyzed in our survey 319
Table C.2 Characterization of the self-adaptation approaches analyzed in our survey (cont.) 320

Table D.1 Modified features of PRE in Surprise 322

List of Figures

Figure 1.1 Research methodology 7
Figure 1.2 Context management across the context life cycle 10
Figure 1.3 Classical block diagram of a feedback control system 12
Figure 1.4 Dissertation summary 24

Figure 2.1 Classification of context information 40
Figure 2.2 Notation used in context feature diagrams 43
Figure 2.3 Features of context modeling 43
Figure 2.4 Context modeling approaches 43
Figure 2.5 Features for representing context entities and situations . . . 44
Figure 2.6 Features of (a) timeliness and (b) quality modeling 45
Figure 2.7 Top level features of context management approaches 46
Figure 2.8 Features of context acquisition 47
Figure 2.9 Features of context handling. 47
Figure 2.10 Features of context exploitation 48
Figure 2.11 Features for the maintenance and evolution of context managers . 49
Figure 2.12 A simple RDF statement 53
Figure 2.13 The MAPE-K loop 58
Figure 2.14 The Autonomic Computing Reference Architecture (ACRA) . 59
Figure 2.15 Model Reference Adaptive Control (MRAC) 60
Figure 2.16 Model Identification Adaptive Control (MIAC) 61

Figure 3.1 Realizing user-driven context management with personal context spheres 74

Figure 5.1 Motivating the Transitivity pattern 118
Figure 5.2 Transitivity Pattern's graph structure 119
Figure 5.3 A context model with three instances of the Transitivity pattern . 121
Figure 5.4 Symmetry pattern (a) motivation, (b) graph structure 122

Figure 5.5 A context model with two instances of the Symmetry pattern . . 124
Figure 5.6 Inverse pattern (a) motivation, (b) graph structure 124
Figure 5.7 A context model with two instances of the Symmetry pattern . . 127
Figure 5.8 Motivating the Join pattern 127
Figure 5.9 Join pattern's graph structure 128
Figure 5.10 A context model with two instances of the Join pattern 130
Figure 5.11 Motivating the Generalization pattern 132
Figure 5.12 Generalization pattern's graph structure 132
Figure 5.13 A context model with two instances of the Generalization pattern 134
Figure 5.14 Motivating the Delegation pattern 136
Figure 5.15 Delegation pattern's graph structure 137
Figure 5.16 A context model with two instances of the Delegation pattern . . 139

Figure 7.1 General components of DYNAMICO 166
Figure 7.2 The three levels of dynamics in context-driven self-adaptive software 167
Figure 7.3 DYNAMICO with a detailed view of the controllers for the three levels of dynamics. 169
Figure 7.4 DYNAMICO abstracted as a feedback loop for governing changes in control objectives. 176
Figure 7.5 Reference model variation for supporting adaptive control . . 177

Figure 8.1 Runtime V&V tasks made explicit in the self-adaptation loop . . 187
Figure 8.2 Realizing model evolution with MRAC and MIAC in runtime V&V 193

Figure 9.1 SCA artifacts legend 199
Figure 9.2 SMARTERCONTEXT general architecture 200
Figure 9.3 Context monitoring feedback loop architecture 204
Figure 9.4 An abstraction of the SMARTERCONTEXT CORE implementation 209
Figure 9.5 Data structures for context pattern partitioning and indexing 211
Figure 9.6 A partial graph-based view of user Norha's PCS and `geo` vocabulary 213
Figure 9.7 Indexing Norha's PCS for patterns *Transitivity* and *Join* . . . 214
Figure 9.8 Indices of the `geo` vocabulary 215
Figure 9.9 Context inference based on the *Transitivity* pattern 216

Figure 9.10 A partial view of Norha's PCS with sensitive contextual data 217
Figure 9.11 Encryption semantics in (a) PRE and (b) SURPRISE 218
Figure 9.12 The PREPolicySC specification 219

Figure 10.1 An abstract view of the SMARTERCONTEXT solution for e-commerce 231
Figure 10.2 The SMARTERCONTEXT architecture for the smarter shopping case study 232
Figure 10.3 SmarterDeals: our context-aware recommender system 237
Figure 10.4 Our recommendation algorithm for daily-deals 239
Figure 10.5 Improvement of ACF and SMARTERDEALS with respect to classic CF. 242
Figure 10.6 Relative performance of SMARTERDEALS (S) with respect to ACF (A) 242
Figure 10.7 The SMARTERCONTEXT architecture for the dynamic SOA governance case study 244
Figure 10.8 The control objectives ontology 246
Figure 10.9 A control objectives specification example 248
Figure 10.10 Synthesizing monitoring strategies dynamically 250
Figure 10.11 A partial graph-based view of user Norha's PCS and `google` vocabulary 259
Figure 10.12 Indices of generalizable predicates for a user's PCS 259
Figure 10.13 Partial view of index `rdfs:subClassOf` for the vocabulary of products 260
Figure 10.14 Processing Time of SMARTERCONTEXT versus Pellet 263

Figure A.1 Distribution of contributions by year 312
Figure A.2 Overall distribution of contributions by (a) research community, and (b) approach type 314
Figure A.3 Distribution of contributions by (a) context modeling features, and (b) selected features of context management 315

Figure E.1 Traditional user-based collaborative filtering approach 325
Figure E.2 Adjusted Collaborative Filtering Approach 327

ACKNOWLEDGEMENTS

I am infinitely grateful to my supervisor Prof. Hausi A. Müller for his invaluable support during my doctorate program. I want to thank him for being an excellent professor and mentor, for all his patience while I was learning how to conduct research, and for all the time he spent reviewing my writing with authentic commitment to guide my learning process. I want to acknowledge my supervisor for being the kind of professor who really cares about students not only as learners, but also as human beings. I still have a long way to learn, and in this journey I sincerely hope to continue receiving his inestimable advice.

I would like to thank Profs. Margaret-Anne Storey, Ulrike Stege, and Pan Agathoklis for serving on my PhD supervisory committee, as well as for their valuable feedback on my work. I also thank Prof. Patrick Martin for acting as the external examiner of my thesis, and Prof. Colin Bradley for serving as the chair of my final oral examination.

I am indebted to my friend and former professor Grace Lewis for her encouragement and support during my PhD program, and for recommending me to Hausi as a student.

I want to show my appreciation to all current and former members of the *Rigi* research group for their feedback, support, and all the moments we shared at our lab. In particular, Sahar Ebrahimi and her co-supervisor, Prof. Alex Thomo, for their interest in applying my research to Sahar's Master thesis. I also would like to thank my colleagues and friends from *Pita*, *Segal*, and *Chisel* research groups for the conversations at ECS-668 room, which I very much enjoyed.

I want to thank my colleagues and friends from Icesi University in Colombia. Especially, Gabriel Tamura for the fruitful discussions and collaboration on our common research topics; Juan C. Muñoz for his contributions to the privacy module of the SmarterContext framework; Sandra Céspedes, Lorena Castañeda, Álvaro Pachón, Gonzalo Ulloa, and Francisco Piedrahita for their continuous encouragement.

One of the most rewarding aspects of my doctoral program was the opportunity to collaborate with top quality research communities and centres. I own my gratitude to Profs. Laurence Duchien and Lionel Seinturier for inviting and hosting me as a visiting scientist at INRIA Lille Nord Europe, and to all members of *ADAM* team for their interest in my research. Their school of thought visibly shaped the vision of research of the ADAM team members (including Gabriel's) and influenced my own.

I also thank all researchers from *SEAMS*, *CASCON*, *MESOCA*, and *CSER* communities for the opportunity of presenting my work at their venues and their valuable feedback. In particular, I want to extend my acknowledgements to the co-authors of my publications and all researchers who participated in Dagstuhl Seminars 10431 on Software Engineering for Self-Adaptive Systems and 11481 on Models@run.time. I am especially thankful to the *Center for Advance Studies (CAS)* of IBM Canada for the opportunity of being a CAS student. My three-year experience at CAS allowed me to exchange valuable knowledge with practitioners, and inspired me with ideas for the application of my research to industry. I want to thank CAS Director Joanna Ng, as well as Alex Lau, Diana Lau, and Jimmy Lo for their support in all the aspects of my CAS project. I also thank *The Writing Center* tutors from University of Victoria for helping me develop my writing abilities.

I would like to acknowledge the institutions that sponsored my research. This dissertation was possible thanks to the funding by University of Victoria, the National Sciences and Engineering Research Council (NSERC) of Canada, Icesi University and Colfuturo (Colombia), and IBM Corporation. I also thank the administrative staff members of the Department of Computer Science of University of Victoria, especially Wendy Beggs and Nancy Chan, for their friendly assistance at all the stages of my PhD.

Last but not least, I am indebted to my close friends and family for their support. Especially, to the most important person in my life, my husband Gabriel, who unconditionally supported me with his endless love and patience on every single day of my life as a PhD student.

DEDICATION

To my beloved husband, Gabriel.

Chapter 1

Introduction

1.1 Motivation

Currently, systems are evolving from software intensive systems to socio-technical ecosystems, where dynamic groups of users, stakeholders, organizations, and software and hardware infrastructures have to interact in complex and changing environments [NFG+06]. These interactions generate an enormous amount of meaningful data that offer multiple possibilities to generate value. The web is a good representative of such systems. Generally, its stakeholders have uncertain, conflicting, and changing requirements; furthermore, its control is highly complex and decentralized, and its ubiquitous services and users increasingly demand more dynamic, situation-aware and personalized services. By the end of 2012 the global internet population expected to have about 2.39 billion users in total (75.6% of the world population), 113.9 million mobile users with 106.7 million using smartphones, and 154.6 million online buyers.[1] In 2011, every minute of the day buyers spent about $272,070 USD on web shopping, Google received over 2 million search queries, and brands and organizations received 34,722 "likes" on Facebook.[2] Under this complexity, software systems must understand their environment to support users in their matters of concern effectively. Moreover, software systems must accommodate themselves to address changes in user requirements and business goals, or to adjust their computational capacity according to particular situations. Thus, *context-awareness* and *self-adaptivity* become fundamental concerns in the engineering, operation, maintenance and evolution of software systems.

[1] http://www.webpronews.com/internet-usage-2012-2012-02

[2] http://www.domo.com/blog/2012/06/how-much-data-is-created-every-minute

This dissertation focuses on *situation-aware smarter software (SASS) systems*. We define these systems as software systems whose requirements may change continuously, and are highly affected by dynamic context information. That is, the satisfaction of their requirements depends on context situations that change at runtime, and generally cannot be anticipated at design time. In cases where context situations are highly dynamic and uncertain, these systems must modify themselves at runtime in the form of fully or semi-automatic self-adaptation to satisfy their context-dependent requirements. The research challenges addressed in this dissertation concern two main features of SASS systems: *situation-awareness* (also called context-awareness) [VM10b] and *self-adaptivity* [dLGM+13].

The first main motivation of this dissertation was to investigate the application of dynamic context management to improve situation-awareness and exploit the value of context information in SASS systems, thus improving their user quality of experience (QoE). Situation-awareness refers to the capability of a system to gather and process information from its environment to understand the situation of external and internal entities that can affect the system in the accomplishment of its goals. We call these entities *relevant context entities*. Thus, a *context situation* refers to the information that describes the states of a set of relevant context entities at a particular point in time. Most importantly, for a system to understand its context situation, it must be instrumented with suitable mechanisms to represent and manage the life cycle of its relevant context. That is, the system must be able to model, acquire, process, provide, and dispose context information. We are particularly interested in situation-aware software where end-users and system operators play active roles in the context management process, and where the set of relevant context entities may change while the system is in execution thus changing the context management requirements at runtime. Situation-awareness driven by users is a crucial feature in user-centric web applications such as online shopping for smarter commerce [IBM11b]. In these applications relevant context entities correspond, among others, to personal agendas, personal shopping lists, and products and services offered through online catalogs. Meaningful context observations can be gathered from these entities and from the interactions of the user with them. For example, location context can be obtained from events registered in personal agendas, products the user is willing to buy from personal shopping lists, and product or service preferences from the interactions of the user with online catalogs.

The second main motivation of this dissertation was to investigate software en-

gineering models to improve the dynamic capabilities of self-adaptivity through dynamic context monitoring. Self-adaptivity concerns the capability of a system to modify itself, at runtime, according to environmental changes that can affect the system's expected behavior. Self-adaptivity can be classified as structural and behavioral, depending on whether self-adaptation tasks modify the system's structure or business logic. In traditional software engineering, software systems are expected to support rigid and stable business structures and user needs, have low maintenance and minimum intervention of users in the behavior of the system, and ensure high degrees of user acceptance [TBK99]. In contrast, the engineering of self-adaptive software systems demands new software engineering models based on continuous analysis, dynamic requirements negotiation, and incomplete requirements specification. System requirements satisfaction in these systems is regulated and controlled by continuously adjusting or enhancing system behaviour [BSG+09]. Self-adaptive systems adapt in response to changes in their environments, either to ensure the continuous satisfaction of their functional and non-functional requirements, or to provide ubiquitous and situation-aware functionalities. This adaptive behavior is commonly supported by control loop-based mechanisms composed of monitors, analyzers, planners and executors. Monitors keep track of context information that can affect the system's behavior; analyzers use monitored facts to infer symptoms that may trigger adaptation events; planners define the steps that must be performed to tailor the system's behavior; and executors apply these plans at runtime to adapt the system's behavior accordingly. After executing plans, the monitoring phase feeds the loop back and the adaptation process continues over time [IBM06, MPS08].

An important part of this dissertation concerns the key role that dynamic monitoring mechanisms play in the effectiveness and assurance of self-adaptation. For example, consider a service-oriented application enabled with self-adaptivity to adjust its behavior with the goal of maintaining its response time within the contracted levels defined in a Service-level Agreement (SLA). For this, monitors keep track of the system's response time, analyzers compare these measures against the contracted conditions to identify symptoms and decide about the need for adaptation, planners define an architecture re-configuration plan to target the desired response time, and executors perform the re-configuration. With static monitoring, changes in the contracted conditions such as the addition of a new SLA to guarantee availability will require the manual modification of the monitoring mechanism. In the meantime, the effectiveness of the adaptation process will be compromised since one monitoring

condition—availability—is missing. That is, the monitoring mechanism is no longer relevant with respect to the adaptation's goals.

1.2 Problem Statement, Challenges and Research Questions

As explained in the motivation, to advance the state of the art of SASS systems we identified two main fields of research namely situation-awareness and self-adaptivity. In light of this, our research has been driven by the following two main motivations:

M1. The need for improving situation-awareness and the exploitation of the value of context information to enhance QoE in SASS systems.

M2. The need for improving the dynamic capabilities of self-adaptivity in SASS systems by controlling the relevance of context monitoring with respect to current context situations and system goals.

1.2.1 Problem Statement

The research problem addressed in this dissertation is stated as follows:

> *In software systems whose goals change over time and depend on dynamic context-situations, the relevance of the monitoring mechanisms with respect to system goals and current context situations is continuously challenged. Thus, for these systems to become more situation-aware and smarter (i) context information and its monitoring requirements must be modeled in such a way that they exhibit an explicit relationship with the system's goals; (ii) the resulting models must support variations in the set of relevant context entities at runtime; (iii) the monitoring strategies, including the sensing infrastructure, must adapt dynamically at runtime to address changes in the environment and the system's goals; and (iv) the user must be integrated into the context management loop to drive changes in context models.*

1.2.2 Research Challenges

The following are the research challenges addressed in this dissertation. We classified these challenges according to the two main features of SASS systems (i.e., context-awareness and self-adaptivity). Challenges *CH1* and *CH2* concern the research related to dynamic context management, whereas challenges *CH3* and *CH4* concern the improvement of the dynamic capabilities of self-adaptivity.

Dynamic Context Management (context-awareness)

CH1. Given the changing nature of SASS systems' requirements and their context situations, it is impossible to fully specify relevant context entities and the corresponding monitoring requirements at design time. Therefore, we require context models that are manipulable and adaptable at runtime to support changes in the initial specifications of context entities and their monitoring requirements.

CH2. Given the changing nature of monitoring requirements in SASS systems, to guarantee situation awareness monitoring strategies must self-adapt at runtime in such a way that they support the addition of new, and deletion of existing context types and corresponding monitoring conditions at runtime. Moreover, the user must be integrated into the context management loop to control this dynamic behavior.

Self-Adaptivity

CH3. To improve the effectiveness of adaptation mechanisms, and thus of self-adaptation, we require reference models to guide the design of self-adaptive software with explicit control of the dynamicity of adaptation goals, adaptation mechanisms, and monitoring infrastructures, and the way they affect each other in the adaptation process.

CH4. Given the dynamic behavior of SASS systems, they require the assessment of adaptation mechanisms at runtime to determine whether an undesirable system state could be reached as a result of a faulty adaptation process.

1.2.3 Research Questions

From the four research challenges stated above we defined a general research question stated as follows:

> *How to leverage context information and dynamic context management to improve context-awareness and the dynamics of self-adaptivity in SASS systems?*

From this general research question, we derived the following four concrete research questions:

Q1. What infrastructure is needed to make context first-class in SASS systems? (a) What models are suitable for the specification of dynamic context and its monitoring requirements? (b) What software infrastructure is needed for dynamic context management?

Q2. How to integrate users into the dynamic context management process to improve user quality of experience (QoE) in SASS systems?

Q3. How can we design SASS systems to maintain the relevance of context-awareness with respect to changing system goals throughout the adaptation process?

Q4. How do we apply dynamic context monitoring to the assurance of self-adaptation?

1.3 Methodological Aspects

1.3.1 Research Methodology

To develop this dissertation we used a concurrent mixed-methods approach [Cre09]. This approach pragmatically combines systematic literature reviews[3] [KC07] and case studies [ESSD08, Yin03] (cf. Figure 1.1). The benefit of a mixed methods methodology is that it allows researchers to choose the most appropriate method—qualitative or quantitative—to answer each of the research questions and validate the hypotheses. Moreover, concurrent mixed-methods is a suitable approach for supporting an iterative loop from theory to design to experiment and back to theory again.

[3]The concepts *systematic literature review* and *survey* are used interchangeably in this dissertation.

Figure 1.1 summarizes our research methodology, which was primarily based on qualitative data obtained from systematic literature reviews and two case studies. Qualitative and quantitative data were obtained and analyzed concurrently and iteratively. We also performed comprehensive studies of the related work that fed the analysis of findings.

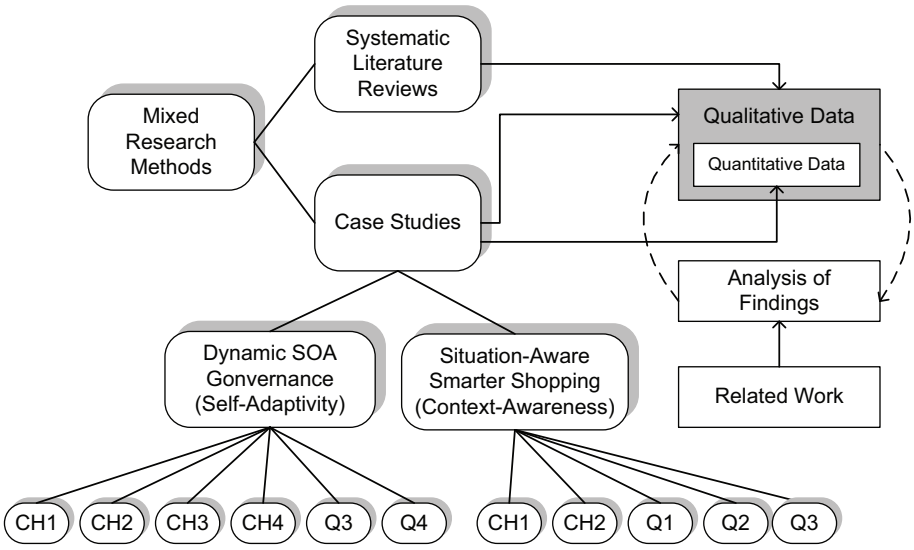


Figure 1.1: Research methodology

The case study on dynamic service-oriented applications (SOA) governance consisted in demonstrating how by implementing dynamic monitoring to keep track of the changing environment that affects a service-oriented system, it is possible to better guarantee the contracted qualities specified in SLAs. For example, facing changes in contracted conditions having no dynamic monitoring will compromise the effectiveness of the governance process until the monitoring infrastructure is modified manually. This is due to a lack of relevance between the implemented monitoring infrastructure and the monitoring requirements that change upon the re-negotiation of the SLA. This case study drove us in the observation of the self-adaptivity phenomenon throughout the state of the art, and in the analysis to propose our models for the engineering of self-adaptive systems. This case study was mainly related to research questions *Q3* and *Q4* and addressed the four research challenges. The case study on situation-aware smarter shopping drove us in the investigation of methods and techniques to leverage context-awareness in user-centric SASS systems. Particularly, to improve user quality of experience (QoE) in user-centric situation-aware online shopping applications. This case study, conducted with the IBM Centre for

Advanced Studies (CAS) at the IBM Toronto Software Laboratory, supported the implementation of several proofs of concept to evaluate the contributions related to research questions *Q1–Q3* and research challenges *CH1* and *CH2*.[4]

In our research, systematic literature reviews supported the collection and analysis of qualitative data, whereas case studies supported the collection and analysis of both qualitative and quantitative data. We hypothesized that for software systems to become more situation-aware, an important factor is to integrate the user into the context management loop. To validate this hypothesis, we surveyed and analyzed approaches for managing the life cycle of context information [VM10b], and conducted a case study on situation-aware smarter shopping where user-driven dynamic context management improves the accuracy of product and service recommendations thus improving the QoE [EVMT12]. Based on the qualitative data obtained from our survey, we proposed a solution to empower users with full control of their personal context information and its management process [VMMn+11, MTVM12, VM13]. Then, through the case study where we simulated context information with real data obtained from the Yelp academic data set [Yel04], we proved, with qualitative and quantitative data, that the accuracy of product recommendations and QoE improve when assisting users in managing personal context information dynamically. Another of our hypotheses was that to improve self-adaptivity, monitoring strategies, including the sensing infrastructure, must adapt dynamically to address changes in the environment and the system's goals. To validate this hypothesis, we conducted a systematic literature review on the design and evaluation of self-adaptive software systems [VMT+11c]. The qualitative data obtained from this study not only supported our hypothesis, but also provided us with qualitative data valuable for our contributions [VTM+13, TVM+13]. Finally, the case study on dynamic SOA governance provided us with quantitative evidence that allowed us to prove the importance of preserving the context relevance of monitoring strategies to improve self-adaptivity and thus situation-awareness in adaptive software systems [VM10a, VMT11b].

1.3.2 Research Approach

The first step in our research was to conduct an exploratory study that allowed us to start answering the research questions and define the goals of this research. For

[4]The research project derived from this case study received "Project of the Year 2011 Award" by IBM CAS Canada Research.

this, we conducted two comprehensive literature reviews, one for each of the main components of this dissertation: context-awareness and self-adaptivity. The findings of these exploratory studies constitute the backbone of this research as follows.

Characterizing Context Information

The first literature review we conducted focused on context-awareness [VM10b], particularly in the characterization of context information and its life cycle. We performed this study with the goal of identifying context modeling and management requirements for supporting smart interactions and services in SASS systems related to the *smart internet* [CCNY10]. Our survey was driven by four main research questions related to the context-awareness challenges we identified for the smart internet: (i) How to identify relevant context and corresponding context management objectives for a particular set of interactions involved in a particular situation? (ii) How to acquire, compose, and distribute context information to multiple execution endpoints in an efficient manner? (iii) How to manage the dynamic nature of context information and the uncertainty of the context-aware system's requirements? (iv) How to control and govern context management by ensuring its provisioning according to the user's situation and/or system's requirements?

To conduct this exploratory study, we proposed an operational definition of context derived from three important existing definitions: the classic definition proposed by Dey *et al.* [ADB+99], the operational extension of this definition given by Zimmermann *et al.* [ZLO07], and the notion of *context life cycle* applied by Hynes [Hyn09]. An operational definition has the advantage of being more concrete and directly implementable than just abstract/conceptual definitions. Our operational definition is as follows:

> *Context is any information useful to characterize the state of individual entities and the relationships among them. An entity is any subject which can affect the behavior of the system and/or its interaction with the user. This context information must be modeled in such a way that it can be pre-processed after its acquisition from the environment, classified according to the corresponding domain, handled to be provisioned based on the system's requirements, and maintained to support its dynamic evolution.*

In summary, three important aspects constitute the foundation of our operational definition of context. The information which characterizes the situation of subjects

relevant to the interactions between users and systems, the models required to represent this information, and the management of this information along its life cycle. Note that our operational definition does not distinguish between environmental and system entities, therefore relevant context information can arise from entities located either outside or inside the boundaries of the system.

To survey the 53 papers that resulted from the filtering process, we applied the method of Feature-Oriented Domain Analysis (FODA) proposed by researchers of Carnegie Mellon Software Engineering Institute (CMU SEI) [KCH+90]. The main focus of this method is the identification of prominent or distinctive features of software systems in a domain. Using this method, we identified relevant features of context modeling and context management approaches, as well as, context modeling and management requirements for supporting context-awareness in the smart internet. In each case, a set of feature diagrams was presented in the form of a hierarchy of common and variable features characterizing the selected context modeling and context management approaches. Figure 1.2 presents the central aspects of context management across the context information life cycle identified in our study. The feature models defined in our survey were based on the aspects presented in this figure. Appendix A provides methodological and demographic details of our survey.

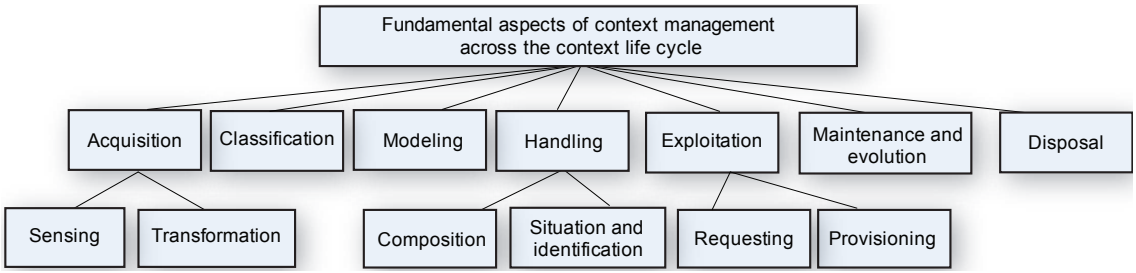


Figure 1.2: Context management across the context life cycle

Findings. The findings of this survey can be summarized as follows:

- The two main components of context-awareness are *context representation* and *context management*. Context representation refers to the modeling of context information, whereas context management refers to the administration of its life cycle.

- Only 4 of the 53 surveyed approaches considered the need for adapting monitoring infrastructures at runtime [CCRR02, SACD08, HIR08, Euz08]. How-

ever, in contrast to the contributions related to this dissertation, these approaches provide no implementation with corresponding validation and/or focus only on self-healing capabilities rather than dynamic changes in monitoring requirements.

- We found no approach that empowers the end-user to decide about the set of relevant context entities, and/or to control the life cycle of context information.

Control Theory as the Foundational Theory

For the second literature review [VMT+11c], we selected the *feedback loop* model, the cornerstone of control theory, as the foundational theory that would drive the exploration of the state-of-the-art self-adaptive software systems and the investigation of the research questions related to self-adaptivity. We selected the feedback loop as our foundational theory because it has been widely accepted by the *software engineering for adaptive and self-managing systems (SEAMS)* research community[5] as a suitable conceptual model for self-adaptive software [Sha95, MPS08, BSG+09, HGB10]. We used the feedback loop as a theoretical lens to define and answer more specific research questions.

In this theory, the feedback loop or closed loop, as depicted in Figure 1.3, is the *model* used to automate the control of dynamic systems. These control mechanisms are realized by comparing the *measured outputs* (A) of the target system behavior to the control objectives given as *reference inputs* (B), yielding the *control error* (C), and then adjusting the *controlling inputs* (D) accordingly for the target system to behave as defined by the reference input [HDPT04]. The control input relies on the mathematical model of the controller, known as the transfer function, to modify the behavior of the target system. The measured output can also be affected by external *disturbances* (E), or even by the *noise* (F) caused by the system adaptation itself. *Transducers* (G) translate the signals coming from sensors, as required by the comparison element (H). An important challenge in control theory is controller design. That is, the definition of transfer functions that relate the system's outputs and inputs to obtain a suitable control input that will modify the system to achieve the desired outputs. Similarly, in software engineering of adaptive systems the challenge is to design adaptation mechanisms (i.e., controllers) to modify the system according to context observations and the expected behavior. The goal of using self-adaptivity

[5] http://en.wikipedia.org/wiki/Software_Engineering_for_Adaptive_and_Self-Managing_Systems

is to satisfy functional requirements and regulate the satisfaction of non-functional ones, under changing conditions of the environment and the target system (i.e., the software system to be adapted).

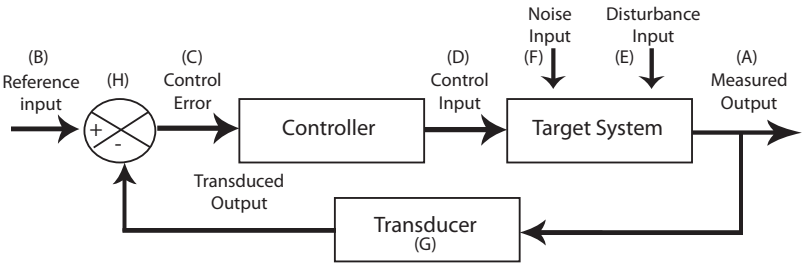


Figure 1.3: Classical block diagram of a feedback control system [HDPT04]

Findings. The findings of this literature review can be summarized as follows:

- Most research contributions using the feedback loop as a reference model follow the MAPE-K loop abstraction (also called autonomic element) from autonomic computing [KC03, IBM06, MKS09]. That is, the equivalent to the controller in Figure 1.3 is designed as a set of autonomic elements composed of monitors, analyzers, planners, executors and knowledge bases.

- Despite the acknowledgement of the feedback loop as an important reference model for self-adaptive software, its visibility, the visibility of its elements and the relationships among them remain often hidden. As a result, there is a lack of separation of concerns among adaptation controllers, monitoring mechanisms and target systems. This compromises the dynamicity of the adaptation mechanism, and in many cases the operation of the target system.

- In contrast to control theory, in the engineering of software systems adaptation properties are generally neither identified nor evaluated explicitly [VMT+11c]. Therefore, there is a lack of mechanisms to assess self-adaptive software systems. Moreover, the lack of adaptation properties leads to a lack of metrics mapped to system goals which compromises not only the effectiveness of monitoring mechanisms but also self-adaptivity.

- Given the abstract nature of software properties and the discrete nature of software systems, controller design, as realized in control theory, is still an unsolved problem in the engineering of adaptive software systems. Moreover,

controller design approaches such as linear system theory, successfully applied to some areas of computing systems [HDPT04], are not applicable to many of the adaptation concerns in adaptive software [VMT+11c].

1.3.3 Dissertation Goals

From the findings of our exploratory study, and taking into account our research questions, we stated the goals of this dissertation as follows:

G1. To propose a context representation mechanism to model dynamic context information and corresponding monitoring requirements in any application domain.

G2. To propose an adaptive context management solution that, assisted by the user, supports context gathering, reasoning, provisioning and disposal under changing context monitoring requirements.

G3. To characterize adaptation properties and system goals for guiding the specification of context monitoring requirements in quality-driven self-adaptive software systems.

G4. To propose a reference model for engineering adaptive software that helps improve self-adaptivity, by controlling the dynamicity of adaptation goals, adaptation mechanisms and monitoring mechanisms, and the way they affect each other in the adaptation process.

G5. To investigate the application of dynamic context monitoring to the runtime V&V of self-adaptive software.

G6. To implement a proof-of-concept of the adaptive context management solution for both case studies: situation-aware smarter shopping, and dynamic SOA governance.

These goals determine the scope of this dissertation. Goals *G1* and *G2* are related to context-awareness and thus address challenges *CH1* and *CH2* (cf. Section 1.2.2). These two goals relate to research questions *Q1* and *Q2* (cf. Section 1.2.3). Goals *G3-G5* concern self-adaptivity, address challenges *CH3* and *CH4* (cf. Section 1.2.2), and concern questions *Q3* and *Q4* (cf. Section 1.2.3). Goal *G6* concerns the implementation of our monitoring infrastructure, thus is related to all challenges and research questions.

1.4 Research Contributions

This section summarizes the contributions of this dissertation in relation to the addressed challenges as follows.

1.4.1 Context Management

Concerning context-awareness, we classify our contributions into two groups: *context representation* and *context management*. In this dissertation we refer to them as the SmarterContext framework or the SmarterContext infrastructure. The research findings concerning our context management research contributions were published in [VM10a, VM10b, VMT11b, VMMn+11, MTVM12, EVMT12, VM13].

C1: The Personal Context Sphere

According to goal *G2*, our solution to dynamic context management must integrate users as main controllers of the context management process (i.e., acquisition, modeling, handling, exploitation, and maintenance and evolution). Such a solution to context management must integrate the user to decide about (i) the relevance of context entities, (ii) shareable context information, (iii) trustable context providers and consumers, and (iii) the context life cycle. To integrate the user as a controller of the context management process we proposed the *personal context sphere (PCS)*. A PCS is a distributed repository supplied by a cloud infrastructure provider that stores the context information gathered about the user. Each PCS belongs to a user. Moreover, access to the information stored in a PCS is fully controlled by its owner. For this, users integrate into their PCSs the identification of third parties with which they want to share their personal data. Moreover, users can define different levels of granularity for granting access to different parties. Personal context spheres are the cornerstone of our context management approach for improving context-awareness in user-centric SASS systems. Through our case study on situation-aware smarter shopping we evaluated the feasibility of implementing this concept to improve the value of context information in several scenarios of electronic commerce. In particular, we analyzed how PCSs improve the accuracy of product and service recommendations, and enable new opportunities of value creation for buyers, and revenue generation for sellers and cloud infrastructure providers. This contribution addresses goals *G1* and *G2*.

C2: The SmarterContext Ontology

To address goals *G1* and *G2* we proposed the SmarterContext ontology. This ontology constitutes our approach to context representation and its design was driven by the characterization of context of our systematic literature review (cf. Section 1.3.2). It supports the specification of (i) context entities with corresponding attributes, (ii) context relationships, (iii) context monitoring requirements, and (iv) context privacy and security policies. The SmarterContext ontology is a vocabulary based on Resource Description Framework (RDF) [MM04] that characterizes context types and the relationships among them. We designed our ontology with modular and extensible capabilities to make it applicable to any context-related application domain. It supports two types of extensibility. Vertical extensibility is supported by extending its core module to define more particular context types and relationships according to the domain. For example *location context*, a general type defined in the core module, can be extended to define the *geographical location* type. Horizontal extensibility is supported by integrating concrete vocabularies useful to characterize a concrete context type. For example, instead of defining exhaustively every possible geographical location instance, our ontology supports the integration of specific vocabularies for geographical regions such as countries or cities. In this way, the semantics provided by the ontology is highly flexible to address the requirements of any specific location-aware application. We evaluated the generality of our ontology using our case studies on dynamic SOA governance and situation-aware smarter shopping.

C3: The SmarterContext Reasoning Engine

To address goal *G2* we proposed a context reasoning engine based on the SmarterContext ontology. Our approach to context reasoning supports the inference of implicit contextual facts from explicit contextual data. These explicit data is specified in context models compliant with the SmarterContext ontology. The SmarterContext engine supports the addition and deletion of context types and context reasoning rules dynamically. In the first version of our engine, reasoning rules are based on deduction rules supported by the RDFS specification [W3C04c], a subset of the axioms defined in OWL-Lite [W3C04d], and the set of particular rules defined in the SmarterContext ontology for each application domain. Furthermore, to improve performance when processing high volumes of RDF data, we proposed a set of contextual patterns which are RDF sub-graphs that represent types of contextual

facts commonly used in a particular domain. This optimized version relies on structural indices and an efficient use of data structures to improve context processing time. To validate our engine we evaluated its performance, and the improvement of the accuracy of a recommendation algorithm that relies on SmarterContext to suggest shopping deals to users [EVMT12].

1.4.2 Self-Adaptivity

The following three contributions correspond to our approaches to improve the dynamic capabilities of self-adaptivity in SASS systems. All of them have been widely disseminated in several research communities related to self-adaptivity, mainly through book chapters, Dagstuhl seminars, symposiums and workshops [VM10a, VMT11a, VMT11b, VMT+11c, FVP+11, dLGM+13, VTM+13, TVM+13].

C4: A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems

This contribution addresses goals *G3* and *G5*. This framework, intended for evaluating quality-driven self-adaptive software systems, provides a catalog of adaptation properties mapped to software quality attributes. This catalog is useful to assist software engineers in the identification of adaptation goals, desired adaptation properties, monitoring requirements and suitable metrics to assess adaptation mechanisms. This contribution is derived from the second survey of self-adaptation approaches conducted in our exploratory study (cf. Section 1.3.2) [VMT+11c]. We have presented and discussed our framework at different venues of related research communities including the *Dagstuhl Seminar 10431 on Software Engineering for Self-Adaptive Systems*,[6] the 6^{th} *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011)*,[7] and the *Dagstuhl Seminar 11481 on Models@run.time*.[8] Moreover, it has been well received among researchers as a useful tool for engineering self-adaptive software.

[6] http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=10431
[7] http://www.hpi.uni-potsdam.de/giese/gforge/events/2011/seams2011
[8] http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=11481

C5: DYNAMICO—A Reference Model for Designing Self-Adaptive Systems

To address goal *G4*, we proposed DYNAMICO, a reference model for engineering adaptive software. This reference model helps guaranteeing the coherence of (i) adaptation mechanisms with respect to changes in adaptation goals; and (ii) monitoring mechanisms with respect to changes in both adaptation goals and adaptation mechanisms. DYNAMICO improves the engineering of self-adaptive systems by addressing (i) the management of adaptation properties and goals as control objectives; (ii) the separation of concerns among feedback loops required to address changing control objectives over time; and (iii) the management of dynamic context as an independent control function to preserve context-awareness in the adaptation mechanism. We proposed DYNAMICO to provide a guide for researchers and practitioners to design adaptive software systems where separation of concerns, dynamic monitoring, and runtime requirements variability are critical drivers. Our reference model was published in the book that resulted from the Dagstuhl Seminar 10431 on Software Engineering for Self-Adaptive Systems that held in November 2010 [VTM+13].

C6: A Model for Explicit Runtime V&V Tasks in Self-Adaptive Systems

This contribution concerns goal *G5*. Software validation and verification (V&V) ensures that software products satisfy user requirements and meet their expected quality attributes throughout their life cycle. While high levels of adaptation and autonomy provide new ways for software systems to operate in highly dynamic environments, developing certifiable V&V methods for guaranteeing the achievement of self-adaptive software goals is one of the major challenges facing the entire research field. This contribution is divided in two concrete contributions. First, we analyzed fundamental challenges and concerns for the development of V&V methods and techniques that provide certifiable trust in self-adaptivity. Second, we proposed a model for including V&V operations explicitly in feedback loops for ensuring the achievement of software self-adaptation goals. Both of these contributions provide valuable starting points for V&V researchers to help advance this field. This model was published also in the book that resulted from Dagstuhl Seminar 10431 [TVM+13].

C7: Implementation of the SmarterContext Infrastructure

The implementation of the SmarterContext infrastructure constitutes contribution C7 and addresses both aspects of our research: context-awareness and self-adaptivity. We implemented a proof-of-concept of the SmarterContext infrastructure for the case study on situation-aware smarter shopping. The main components of this implementation are (i) the SmarterContext engine, and (ii) the personal context sphere (PCS). The vehicle for context gathering are the interactions of users with web applications. Context reasoning, gathering and provisioning rely on RDF context models based on the SmarterContext ontology. RDF models are used to represent PCSs, the context requirements of context consumers, and the policies for securing personal data. We also implemented a proof concept of our monitoring infrastructure for our case study on dynamic SOA governance. The main components of this version of the implementation are (i) the dynamic context manager that realizes a feedback loop in charge of controlling the self-reconfiguration of the monitoring infrastructure according to changes in monitoring requirements, and (ii) the monitoring infrastructure composed of monitors deployed at runtime.

1.4.3 Publications Derived from this Dissertation

Book Chapters

- N. M. Villegas and H. A. Müller. Managing Dynamic Context to Optimize Smart Interactions and Services, pp. 289-318. Vol. 6400 of LNCS, Springer, 1 Ed., 2010 [VM10b].

- R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cikic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, and D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, J. Wuttke. *Software Engineering for Self-Adaptive Systems: A second Research Roadmap*, pp. 1-26. Vol. 7475, Springer, 2013 [dLGM+13].

- G. Tamura, N. M. Villegas, H. A. Müller, J. P. Sousa, B. Becker, M. Pezzè, G.

Karsai, S. Mankovskii, W. Schäfer, L. Tahvildari, and K.Wong. *Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems*, pp. 108-132. Vol. 7475 of LNCS, Springer, 2013 [TVM+13].

- N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas. *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems*, pp. 265-293. Vol. 7475 of LNCS, Springer, 2013 [VTM+13].

- N. M. Villegas and H. A. Müller. *The SmarterContext Ontology and its Application to the Smart Internet: A Smarter Commerce Case Study*, p. . Vol. of LNCS, Springer, Ed., 2013. In press [VM13].

International Conferences and Symposiums

- M. E. Frîncu, N. M. Villegas, D. Petcu, H. A. Müller, and R. Rouvoy. *Self-Healing Distributed Scheduling Platform.* In: 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 225-234, IEEE Computer Society, Washington, DC, USA, 2011 [FVP+11].

- N. M. Villegas, H. A. Müller, J. C. Muñoz, A. Lau, J. Ng, and C. Brealey. *A Dynamic Context Management Infrastructure for Supporting User-driven Web Integration in the Personal Web.* In: 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2011), pp. 200-214, IBM Corp., Markham, ON, Canada, 2011 [VMMn+11].

- N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas. *A Framework for Evaluating Quality-driven Self-Adaptive Software Systems.* In: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), pp. 80-89, ACM, New York, NY, USA, 2011 [VMT+11c].

- J. C. Muñoz, G. Tamura, N. M. Villegas, and H. A. Müller. SURPRISE*: User-controlled Granular Privacy and Security for Personal Data in* SMARTERCONTEXT. In: 2012 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2012), pp. 131–145, Riverton, NJ, USA, 2012. IBM Corp. [MTVM12].

- S. Ebrahimi, N. M. Villegas, H. A. Müller, and A. Thomo. SmarterDeals*: A Context-aware Deal Recommendation System based on the* SmarterContext *Engine*. In: 2012 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2012), pp. 116–130, Riverton, NJ, USA, 2012. IBM Corp. [EVMT12].

International Workshops

- N. M. Villegas and H. A. Müller. *Context-driven adaptive monitoring for supporting SOA governance*. In: Proceedings 4th International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2010), Carnegie Mellon University Software Engineering Institute, 2010 [VM10a].

- N. M. Villegas, H. A. Müller, and G. Tamura. *Optimizing Run-Time SOA Governance through Context-Driven SLAs and Dynamic Monitoring*. In: 2011 IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2011), pp. 1-10, IEEE, 2011 [VMT11b].

Electronic Refereed Magazines

- N. M. Villegas, H. A. Müller, and G. Tamura. *On Designing Self-Adaptive Software Systems*. Revista S&T, Vol. 9, No. 18, pp. 29-51, 2011 [VMT11a].

1.5 Dissertation Outline

The remaining chapters of this dissertation are organized as follows:

Chapter 2: Research Background. This chapter presents the background topics of this dissertation. The first topic is the *smart internet*, a vision of the next generation of the Internet that relies on context-aware computing and self-adaptivity to deliver smart services and support smart interactions. The smart internet provides compelling applications for the contributions of this thesis. The second topic is *context-awareness*. This section presents the characterization of context information and the set of requirements, in the form of features, for context modeling and management. The third topic is *Semantic Web (SW)*. This section explains the technologies, languages and tools that we gleaned from SW to propose our approach to

context representation and context processing. The fourth topic is the *feedback control for self-adaptivity* model. This section presents the feedback loop and adaptive control as foundational models for engineering dynamic behavior in software systems. It also discusses the importance of making the feedback loop's components and the relationships among them explicit in self-adaptive software systems. The last topic is *self-adaptive software*. This section summarizes the state of the art of self-adaptive systems and discusses the self-adaptivity challenges we addressed in this dissertation. This chapter ends with a discussion of the most important aspects of each background topic with respect to this dissertation.

Chapter 3: The Personal Context Sphere. This is the first contribution chapter of this dissertation. It presents our solution to empower end-users as main controllers of their personal context information life cycle. We provide a detailed explanation on how to improve user quality of experience (QoE) by combining context-management, self-adaptivity and the user's personal context sphere (PCS).

Chapter 4: The SmarterContext Ontology. This chapter presents our approach to context representation, the methology we followed for its design, its requirements analysis, and its extensible and modular features.

Chapter 5: The SmarterContext Reasoning Engine. This chapter presents our approach to context reasoning based on the SmarterContext ontology. It explains the two reasoning approaches supported by SmarterContext. The first one is based on both RDF and OWL and relies on existing semantic web reasoning engines. The second one, which provides a better performance, is based on RDF only and the identification of *context patterns* in PCSs. Context patterns constitute our proposal to graph-based representations of simple contextual facts, and provide the semantics required to infer implicit contextual facts from explicit contextual data in PCSs, without the complexity added by OWL.

Chapter 6: A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems. This chapter presents our characterization model for self-adaptive software that resulted from the exploratory study we performed on self-adaptivity. It also presents our catalog of adaptation properties and goals and their mapping to quality attributes. This chapter explains how this catalog can be used for the identification of adaptation goals, adaptation properties, and assessment metrics, which are crucial for deriving monitoring requirements in self-adaptive software.

Chapter 7: Dynamico: A Reference Model for Designing Self-Adaptive Systems. This chapter starts with a discussion about selected key aspects to improve

the dynamic capabilities of self-adaptive software systems. These aspects constitute the design drivers that guide the engineering of SAS systems based on DYNAMICO, our proposed reference model. This chapter explains in great detail the main components of the reference model represented by three different feedback loops namely *control objectives, adaptation* and *context monitoring*. It also discusses the interactions among these feedback loops and their governance.

Chapter 8: A Model for Explicit Runtime V&V Tasks in Self-Adaptive Systems. This chapter starts with an introduction to the *viability zone* of self-adaptivity, an important concept for ensuring the dynamic behavior of SAS systems. Then, it discusses assessment of adaptive software in light of three main questions: the *what*, *where*, and *when* of runtime V&V. The main contribution of this chapter is our proposal to make explicit the V&V tasks, supported by runtime monitoring, throughout the adaptation process.

Chapter 9: The SmarterContext Infrastructure. This is the last contribution chapter of this dissertation. It presents the generic software architecture of our context management solution and how it is used to realize dynamic context monitoring. It also provides implementation details of the two versions of the SMARTERCONTEXT engine, and the user-controlled privacy and security mechanisms that are supported by SMARTERCONTEXT.

Chapter 10: Evaluation. This chapter presents the validation results of this dissertation. The validation of our contributions was performed using qualitative and quantitative evaluation. Qualitative evaluation is performed through the analysis of how with our contributions we advance the state of the art of self-adaptivity and user QoE in SASS systems, using the case studies on dynamic runtime SOA governance and situation-aware smarter shopping, respectively. Quantitative evaluation focuses on the following aspects. With respect to user-centric SASS systems it focuses on accuracy and performance. To evaluate the first one, we analyzed the improvement of accuracy in a recommendation algorithm when combined with our SMARTERCONTEXT engine. To evaluate the second one, we measured the improvement of efficiency in the version of our recommendation engine that relies on RDF only using our concept of context patterns. With respect to self-adaptivity, we conducted several experiments that allowed us to demonstrate the suitability of our solution with respect to three important properties: settling time, overhead, and engineering effort.

Chapter 11: Summary. This chapter summarizes the research work and contributions of this dissertation. It concludes with an analysis of limitations and future

work.

1.6 Chapter Summary

This chapter presented the motivation, research challenges, questions, goals and methodology, as well as an overview of the contributions and publications related to this dissertation. We introduced the two main topics of our research, *context-awareness* and *self-adaptivity*, and explained how from our initial exploratory study, focused on these two topics, we arrived to our approaches and made seven contributions to improve context-awareness and self-adaptivity in situation-aware smart software (SASS) systems. Figure 1.4 summarizes this dissertation including the relationships among research challenges, questions, goals, contributions, publications, and evaluation methods.

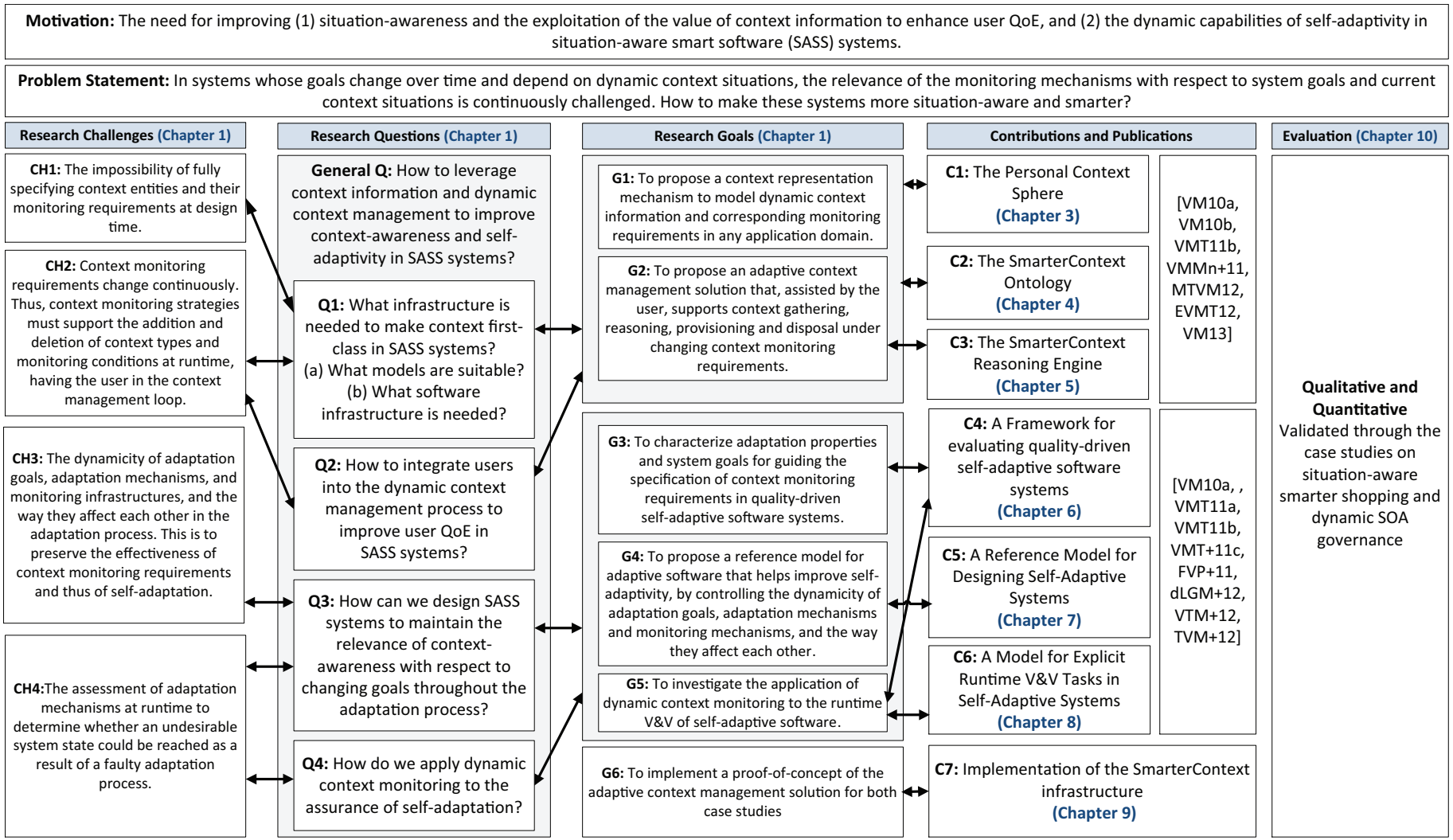


Figure 1.4: Dissertation summary. Chapters 2 and 11, not included in the summary, correspond to chapters *Research Background* and *Summary and Conclusions*, respectively.

Chapter 2

Research Background

2.1 The Smart Internet

The concepts *smart systems*, *smart internet*, and *smarter commerce* are borrowed from a set of IBM initiatives proposed recently with the vision of building a *smarter planet*.[1] By "smarter" they refer to an *instrumented*, *interconnected*, and *intelligent* world, where people's and businesses' daily life activities rely on hardware, software, and information and communications technologies. The term *instrumented* refers to the embedding of physical sensors in common objects with which people interact. Instances of these objects are vehicles, products offered in a retailing store, health care devices, clothing, appliances, electronic devices and smartphones. The term *interconnected* refers to people, systems, and objects communicating and interacting with each other in entirely new ways. The term *intelligent* refers to systems responding to changes quickly, accurately and reliably. In this dissertation the term instrumented refers not only to embedded physical sensors, but also to the use of other instruments for observation, measurement and control of different phenomena in situation-aware smart software (SASS) systems and their execution environments, which include users. This section presents background and research challenges of the smart internet and its applications that are relevant to the contributions of our research.

[1] http://www.ibm.com/smarterplanet/us/en/overview/ideas

2.1.1 The Smart Internet Vision

The smart internet is conceived as a new generation of the internet where web entities, represented by online services and content, are discovered, aggregated and delivered dynamically, automatically, and interactively according to users' needs and situations [CCNY10].[2] In the smart internet, interactions must be situation-aware and smart, that is they must be realized with awareness of, and adaptation to users' individual and collective context situations. The smart internet vision has been motivated by the problems of the server-centric approach in the current internet [NCCY10a]. Ng *et al.* stated these shortcomings as the (i) lack of integration of web content and services from the user's perspective; (ii) lack of individualization to deliver content and services based on the users' current needs and situations; (iii) absence of server-initiated connections on behalf of users and with awareness of their context situations; (iv) lack of a notion of service level collaboration where multiple users can work together on a service instance; and (v) lack of control by users over web entities. Our investigation on dynamic context management addresses directly research challenges related to shortcomings (ii) and (v). Since the lack of user context-awareness is a common factor in all these aspects, our research contributes towards solving all of them.

The Smart Internet Principles

The smart internet defines three principles that distinguish it from the internet [NCCY10d]. All of them pose unsolved problems that we address in our research on context-awareness and self-adaptivity. These principles and corresponding challenges are stated as follows:

1. *A user-centric model for instinctive interaction:* It refers to a user-model mainly based on users and their situations, rather than the server and its interactions with the user. That is, it is a situation-aware model from the perspective of the user. This user-centric model must guide the design of both systems and interactions. The delivery of instinctive interactions has three critical implications. First, interactions must be driven by the user's current needs and situations and realized as natural as possible, from the user perspective, by miming real world interactions of the corresponding application domain. Second, content delivered to the user as a result of these

[2] In this dissertation the term *user* refers to both people and organizations.

instinctive interactions must be aggregated according to the user's needs and situations. Moreover, interactions must be tailored according to the user's situation. Third, the control over services and content must be transferred from the server to the user. This is to empower users to drive the discovery of services and content, and the aggregation and delivery of information according to personal context situations. To contribute to the realization of user-centric models for instinctive interactions we address two main challenges: (i) the need for a model that specifies user situations and relevant information to support context-aware interactions (i.e, a model of the user's context relevant to current tasks and situations), and (ii) the empowerment of users, through instinctive mechanisms, to control the life cycle of and access to the information stored in these models explicitly.

2. *Session for users and their matters of concern (moc):* The internet today implements sessions as a persistence mechanism to keep track of the user's interactions from the perspective of the server. However, sessions end when users interrupt their interactions with the server. Moreover, information stored in sessions is available within the boundaries of a particular server only, even when this information could be used to support tasks of the user with applications deployed in other servers. As a result, users have to remember and provide this information when needed across multiple servers. In the smart internet the concept of session goes beyond the server to focus on the user. That is, interactions can be asynchronous and processed across different web applications deployed on different servers, and sessions store mocs persistently to be used along the user's entire web experience. To address the need for this new concept of session, we proposed the concept of *personal context sphere (PCS)*. A PCS can be seen as a user-centric "super-session" where the user controls fully the integration of new information into the super-session, as well as the access to and the sharing of the information stored in this session. We call it "super-session" because its boundaries go beyond the interaction of the user with a particular application. That is, the information stored in PCSs is obtained from and shared with any application with which the user interacts, of course observing the user's privacy preferences. Hence, the boundaries of the information accessible by a particular system are given by the privacy policies configured by users, and managed by the computing infrastructure of

PCSs.

3. *Collaborative and collective web interactions:* This principle refers to the collaboration among users to accomplish collective goals or resolve common mocs. It has two main implications. *Dynamic social binding*, which allows users to select other users dynamically to interact at different levels depending on the moc. *Collective intelligence*, which refers to the collective building of knowledge enabled by historical information of users' behavior gathered from web interactions.

2.1.2 Smart Interactions and Smart Services

The smart internet includes two specific fields of research: (i) *smart interactions*, which concerns the *user model* for the smart internet, and *smart services*, which concerns the *web model* for the smart internet [CCNY10]. The application of our research on context-awareness to user-centric SASS systems addresses mainly the user model for the smart internet and thus concerns smart interactions.

Smart Interactions

According to the smart internet vision, smart interactions research should address at least five categories of challenges [NCCY10b]. The contributions of this dissertation address all of them. The first one concerns *metaphors* to elicit instinctive responses to goals and concerns. Examples of metaphors widely used in the current web are the bookmark and the shopping cart. A general metaphor proposed for the smart internet is the "matter of concern" (moc), a way of connecting user needs and situations (i.e., personal context) to web content and services. Therefore, a user's moc characterizes the context information relevant to the user's web interactions. The second category of challenges is *task simplification*. This concerns the development of innovative ways for engineering simpler web tasks for users, without compromising efficiency. An important research question is how to empower the user to act as a supervisory controller who specifies goals to be achieved and monitors the results without worrying about technical details. The third type of challenges refers to *cognitive support*. It concerns the engineering of web artifacts to assist users in thinking about and solving problems related to personal needs. The goal is to enable web systems to complete tasks in the way users want, but requiring from users minimum knowledge about

the process. For this, web systems must keep track of the states of web entities related to the user's task, taking into account the current moc. The fourth set of challenges is *adaptive aggregation*, which refers to the capability of interactions to adapt according to the user's context situation. The objective is to deliver relevant content and services from multiple sources according to user mocs. The last category of challenges is *dynamic collaboration* that refers to the smart internet capability to assist the user, at runtime, in integrating other users into a collective moc.

Smart Services

Part of the vision of the smart internet is to transfer service control from web programmers to users [NCCY10c]. In light of this, smart interactions impose new requirements for web services based on high degrees of personalization and dynamic adaptation to context situations. Therefore, a new web infrastructure is required to handle mocs' states and sessions, and map services delivered by several providers to the dynamic context of the user. Some key requirements to advance towards this direction, and that are within the scope of our research, are: infrastructure to support *dynamic adaptation* of services according to the user's situation; *inference capabilities* to process dynamic user context; practical *ontology support* to represent and reason about information; *self-discovery* of web entities relevant to user mocs. This dissertation addresses all of these requirements, as well as *privacy and security centered and controlled by the user*.

2.1.3 Context-Awareness Requirements in the Smart Internet

This subsection introduces key requirements on context modeling and context management that we identified for the smart internet. These requirements resulted from the *Smart Internet Technologies Working Conference (SITCON)*[3] co-located with CASCON 2009 [CCNY10], and from our exploratory study on dynamic context for supporting smart interactions and services [VM10b].

[3] http://research.cs.queensu.ca/~cordy/SITCON/

Dynamic Context Modeling

For supporting smart interactions, smart service infrastructures require the definition of models adaptable at runtime for the specification of personal context and web entities involved in user mocs. These models involve properties, relationships among users and web entities, and contextual facts about these entities. Thus, given a specific moc, context models for supporting smart interactions and services are required to (i) represent the variety of context information that characterizes the state of entities that affect the user interactions in the moc; (ii) represent the context monitoring requirements to keep track of changes in the situations of these entities that can affect the user's tasks in the moc; (iii) adapt according to dynamic changes in either the user's moc or in the state of its relevant entities, to ensure a consistent representation of context information.

Dynamic Context Management

Smart services require models and infrastructure to support the three dimensions of the smart internet: instinctive user model, sessions for users and their matter of concerns, and collective and collaborative web interactions [NCCY10d]. For instinctive interactions to be user-centric, the management of context information must be relevant to the user's mocs. Moreover, collaborative and collective web interactions require high levels of context-awareness to infer, for instance, collaboration opportunities among users with similar interests. As a result, for a particular moc and its corresponding context model, dynamic context management infrastructures are required to (i) define context management strategies dynamically based on the context monitoring requirements specified in the model. Most importantly, a context management infrastructure should be able to not only detect changes in the moc, but also changes in the state of the involved entities to adapt the context model dynamically; (ii) gather relevant context according to context monitoring requirements; (iii) handle contextual facts and reason about context situations represented by the context model; (iv) provide context information to multiple and distributed execution endpoints; (v) be self-adaptive, and ideally self-managing, to address issues such as the dynamic deployment of new components for context acquisition—for instance, to gather context information from web applications integrated into the user's moc at runtime. Self-adaptation is crucial for supporting dynamic changes in context monitoring requirements. For example, changes in a moc can impact context

requirements, thus demanding the reconfiguration of the infrastructure to guarantee context monitoring pertinence with respect to the new situation. Finally, a context management infrastructure is required to monitor itself for regulating the satisfaction of its requirements.

2.1.4 The Personal Web

The *personal web (PW)* is a concrete realization of the smart internet that focuses on the user as the center of web integration [Ng10]. The PW was proposed in 2009 as part of the first research roadmap on the smart internet [CCNY10]. The objective of the PW is to empower the user to assemble and aggregate integrable content and services that are useful for the user's tasks in a particular moc. Thus, smart interactions are required to support, from a people-centric perspective, the discovery, aggregation and delivery of web entities from the internet. Moreover, smart services must provide the infrastructure required by these interactions to assist users in web integration [CCNY10]. The vision of the PW is to enable regular web users (i.e., people with non-specific technical skills) to control the integration of web entities according to their personal mocs. The semantics that define relevant types of web entities, the relationships among them, and the way the user interacts with these entities is defined, conceptually, as the *personal web sphere*.

The seminal paper on the PW posits a set of principles that define core characteristics of the PW [Ng10]. Ng classified these principles into two categories: conceptual and technical principles. We summarize the conceptual principles as follows:

1. *The user as the center of web integration:* The integration of web content and services must be user-oriented. The PW envisions the integration of the user's data, resulting from web interactions, into a common repository that belongs to the user, and that remains available to be exploited in future web interactions.

2. *The user's context shapes the scope and semantics of web integration:* Web integration in the PW must be performed around users and the semantics of their personal context. The semantics of personal context at a specific point in time, which defines a moc, must drive the discovery of relevant content and services.

3. *The user controls web integration:* The user model of the PW must support web integration dynamically, completely operated and controlled by general internet users, and without requiring any programming skills.

4. *The web works on behalf of the user:* This refers to the goal of the smart internet of reducing the cognitive load of users. For this, automation and self-adaptation with the user acting as a supervisory controller play a key role.

5. *The PW is social:* The user is involved in different social networks, therefore the infrastructure of the PW must support different interactions among personal web spheres of several users.

We summarize the technical principles as follows:

1. *Abstracting programming complexity to pass control from programmers to users:* The goal is to produce new programming models and APIs that support users in web composition and personalization.

2. *Less is more in meta-model for semantics and user operations for web composition:* This principle applies to different components of the PW architecture. For example, expressive languages such as the Web Ontology Language (OWL) [W3C04d] are not suitable for end-users to specify relationships with web entities easily. The PW requires simpler mechanisms to empower users in the form of instinctive interactions.

3. *Designing for open integration:* This principle refers to system interoperability, which makes Resource Description Framework (RDF) [MM04, BHBL09], a preeminent enabling technology for the PW.

4. *Modeling the relationships of users with things:* Data models generally focus on representing relationships among things. An important modeling requirement for user models in the PW is the specification of how people relate to entities that are relevant to user mocs. Linked data (LD) [BL06] is proposed as a suitable approach for user models in the PW. These models, called *personal linked data graphs, can be gathered by the user from different web domains and combined into one semantic context. In this way, new knowledge can be inferred from multiple graphs to better understand the user's intent. According*

to the PW's vision, simple user-driven web integration can be realized through the integration of personal linked data graphs into a personal semantic context.

2.1.5 Smarter Commerce

Smarter commerce,[4] which provides several compelling applications for our research on context management and self-adaptivity, is a recent IBM strategic initiative that places the customer in the center of the commerce business processes. Its most important capability, from a customer's perspective, is an effective and user-centric shopping experience. According to a recent report by International Data Corporation, smarter commerce exploits emerging technologies and social media to enable the profound change in the way consumers research, shop for, and purchase goods and services [IBM11a]. Particularly in online shopping, businesses are continuously looking for innovative ways to exploit customer information gathered from Web 2.0 applications such as social networks, wikis, blogs and many others related to the smart internet. However, current online shopping solutions still require innovative mechanisms to offer smart recommendations on products. Moreover, for businesses to deliver effective and pleasant shopping experiences, it is necessary to improve their knowledge about buyers' changing situations and preferences, and to increase the flexibility of their commerce processes and infrastructures to address changes in these situations and preferences expeditiously.

Smarter commerce seeks for innovative approaches to renovate the value chain, given that the customer, empowered by technology, transparency, and abundant information, is now changing the dynamics between buyers and sellers [IBM12]. Customers today are more connected and demanding, and expect to be involved with sellers whenever and wherever they want, using physical, digital, and mobile interactions. They evaluate and compare products and services with barely one click. Due to all these technological facilities, customers demand more personalized products and services and more effective and pleasant shopping experiences.

Current online shopping platforms lack knowledge about users and their matters of concern. We have identified three main reasons for this lack of awareness about buyers' situations in e-commerce. First, users have no control over the personal information they provide to e-commerce applications. Therefore, they avoid sharing information that could be useful to understand their shopping needs. Second, the

[4] http://www.ibm.com/smarterplanet/ca/en/smarter_commerce/overview/index.html

information gathered about users by a particular retailer remains within the boundaries of its e-commerce application. Third, current e-commerce implementations are ineffective in identifying the context entities that are relevant to the buyer, keep track of the interactions of the buyer with these entities to gather meaningful context information, and manage the life cycle of these contextual data.

Situation awareness is recognized as a critical, and often elusive, foundation for good decision making in dynamic systems. Context information about buyers' situations is a major source for value creation and revenue generation in smarter commerce. Today there is an abundance of information available to online shoppers. However, this does not mean that online shopping experiences are more situation-aware. What truly matters is not how much data sellers can provide on the many channels, but how relevant this information is for users to make purchase decisions effectively. Suppliers greatly benefit from continuous interaction with customers by anticipating customer behaviors and keeping them loyal. Customers provide information about their situations in the form of space and time volumes as well as personal context spheres to enable suppliers to make inferences.

One of the case studies related to this dissertation presents how through SmarterContext, our dynamic context management solution, we improve situation-awareness in the interactions between online customers and suppliers [VMMn+11]. For this, our SmarterContext engine gathers meaningful information from the interactions of users with web entities to understand users' intents and situations, thus leveraging the value of personal context information in online shopping. In other words, SmarterContext enables situation-awareness. First, it tracks past and present web interactions to provide sellers with relevant context information about users. Thus, in contrast to existing solutions, the information gathered from the interactions with a seller's platform is exploited not only by this seller, but also by any other seller authorized by the user. Second, access to personal information is controlled by the user [MTVM12]. SmarterContext provides shoppers with privacy and security guarantees for sharing context information that sellers can use to improve the relevance of product and service offers [EVMT12]. Therefore, in contrast to existing e-commerce applications, the user is the one who decides about the information to be shared and the third parties authorized to access this information.

Business analytics (BA) helps businesses identify and analyze trends and patterns, and anticipate events. This information is crucial for leveraging decision making in

the process of optimizing business goals. In smarter commerce, and particularly in online shopping, BA provides useful information to improve the relevance of product and service offers with respect to customers' preferences. The goal is to satisfy customers' expectations while improving the business revenue. Despite a range of products available for BA in e-commerce, existing approaches take only advantage of information gathered from the interaction between the shopper and the particular retailer's shopping infrastructure. User-centric BA take advantage of relevant context about the customer and his/her behavior gathered throughout the shopping experience including other relevant shopping sites. Instead of analyzing customer preferences using traditional business intelligence dimensions, the integration of personal context information as a new set of dimensions is a promising approach to understand customers' expectations in situation-aware commerce scenarios.

On the one hand, dynamic self-adaptive software systems are able to reason about their own behavior to adapt themselves in response to changes in their execution environments, either to ensure the continuous satisfaction of their functional and non-functional requirements, or to provide ubiquitous and context-dependent smart services [CLG+09]. On the other hand, permanent analysis, dynamic requirements negotiation and incomplete requirements specification are inherent in the engineering of e-commerce solutions as envisioned by smarter commerce. Therefore, self-adaptive software is required to ensure the continuous satisfaction of functional requirements for situation-aware commerce while preserving the agreed conditions on quality of service levels [VTM+13]. The need for self-adaptation in online commerce platforms arises because of the dynamic nature of shopper preferences and situations, and the dependency of e-commerce infrastructures on context situations such as shopping seasons and popularity of online product and service offers.

2.1.6 Future Internet Perspectives

The development of the *future internet* or *next generation internet* is a research topic that has gained the attention of several research communities worldwide. Therefore, several initiatives have been contributed to address new internet challenges that propose new internet models and architectures based on existing and new web technologies. The smart internet is one of such initiatives, and the personal web is an application of it that puts the user in the center of the web.

Another future internet initiative is the *user-centric internet* [Int11]. This initia-

tive, led by the Internet Society (ISOC),[5] focuses on openness, transparency, edge-based intelligence and, most importantly, users. It is important to point out that although this initiative recognizes "user-centricity" and the ability of users to create content and applications as well as to use web applications in personalized ways, it seeks to ensure that the primacy of the user is not forgotten in the new architectures, commercial offerings and policies that are being proposed as part of future internet initiatives. In contrast to the PW, in ISOC's user-centric internet, end-users are not necessarily the ultimate concern of the internet. In particular, business-oriented quality of service (QoS) is also an important concern.

The *Internet of Things (IoT)* is another initiative related to the future internet. It focusses mainly on the exploitation of smart objects to improve services provided to users rather than user needs themselves [MSPC12]. The IoT research community defines smart objects as physical objects able to interconnect with each other and/or with humans to offer services. In light of this, the IoT's main challenge is the development of technologies and solutions to support smart objects able to be identify, communicate and interact, either among themselves, networks of interconnected objects, or end-users.

Many other related research initiatives can be grouped into the *ubiquitous web* [HS10]. These initiatives arise from the proliferation of social and mobile web applications, where the main challenge concerns the integration of data and the extraction of knowledge that could be useful to support user activities. In ubiquitous web applications, the effective combination of Web 2.0 functionalities with semantic web is a major field of research.

Despite differences among the several visions of the future internet what all these initiatives have in common is context-awareness and self-adaptivity. The increased computation capacity of mobile devices connected to the internet empowers users to use and even create applications to be used in any situation and anywhere [NB09]. On the one hand, these applications become rich sources of context information that must be managed to provide services and content to users. On the other hand, self-adaptivity is a foundational element of context-awareness since the primary goal of context-aware applications is to sense and understand their environment to respond accordingly. As in the PW, in ISOC's user-centric internet it is necessary to understand the situations of users to provide them with personalized services. In the IoT smart objects are producers and consumers of context information. In the Ubiquitous

[5] http://www.internetsociety.org

Web, applications rely on data gathered from sensors, social networks and mobile devices. These data must be integrated, aggregated and correlated to feed ubiquitous applications with meaningful knowledge about situations.

2.2 Context Awareness

The dynamic capabilities of SASS systems and self-adaptation are highly affected by entities in the execution environment, including system requirements and the system itself. The observable characteristics of these entities are known as context information. According to Whittle *et al.*, the uncertainty inherent in self-adaptation is generated by two main sources [WSB+10]. The first one, *environmental uncertainty*, is the uncertainty due to changing environmental conditions. The second one, *behavioral uncertainty*, originates from changes in software requirements or in the behavior of the system. Therefore, context monitoring in this dissertation concerns not only entities external to the system, but also entities within the boundaries of the system and system goals.

So far, most monitoring mechanisms for supporting context-aware and self-adaptive systems have been based on the classical definition of context [VMT+11c]. This definition characterizes context as any information that describes the situation of entities that can affect the system's behavior [ADB+99]. It is important to point out that this definition does not consider changes in the states of these entities while the system that is intended to be context-aware is in execution. On the contrary, in SASS systems, as defined in this dissertation (cf. Section 1.1), context is not simply the state of a predefined environment with fixed entities. It is part of a process of interacting with a continuously changing and uncertain environment [CCD05, Hyn09]. Therefore, to conduct our research on context awareness, we proposed a more operational definition in which context and their monitoring requirements are modeled as first-class entities, in such a way that it can be acquired from the environment, manipulated along its life cycle explicitly by taking into account its dynamic nature, and provisioned based on changing system requirements (cf. Section 1.3.2) [VM10b].

Dynamic context information differs from static context in aspects related to its modeling and management. Concerning context modeling, static context specifies, at design-time, relevant context entities and the interactions among them, which remain immutable at runtime. The birthday and gender of a user are instances of static context. Therefore, monitoring mechanisms based on static context keep track of

entities specified at design-time. Once the system is in execution, the addition of new entities is not supported by the static context specification. On the contrary, dynamic context requires modeling techniques that support changes in the specification of context entities and corresponding monitoring requirements at runtime. For example, location, and product and service preferences are instances of highly dynamic context. Nevertheless, in many existing "user-centric" solutions these types of context are considered static, which hampers the user's web experience. Such is the case with online deal applications such as Groupon,[6] where the user specifies preferred location and coupon preferences during the sign-up process only. Thus, the offers may be ineffective due to a lack of user situation awareness.

With respect to context management, monitoring strategies to keep track of static context are well known at design-time and remain fixed at runtime, whereas monitoring strategies to manage dynamic context are required to change over time. Dynamic context management is key to leverage the dynamic capabilities of SASS systems and manage the uncertainty that can affect their behavior.

This background section characterizes context information, as well as the features on context modelling and context management that we surveyed in our exploratory study on dynamic context [VM10b].

2.2.1 Situation Awareness

Situation awareness has long been recognized as a key to success in military command and control, in emergency situations, air traffic control, and medicine. While there are many definitions of situation awareness, Endsley's definition is widely accepted [End95]: The perception of elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future. The first step in situation awareness, the perception of elements in the environment, concerns the sensing of their status, attributes and dynamics. For example, a pilot would perceive attributes of elements such as aircraft, climate, and warning signals; an automobile driver must sense obstacles, transit signals, and the dynamic behavior of other vehicles. Similarly, a situation-aware smart online shopping application must observe aspects such as the location of customers to offer products within their neighborhood, the time of the year to deliver offers according to the shopping season, and the number of online users interested in last minute of-

[6] http://www.groupon.com

fers to accommodate the e-commerce infrastructure capacity accordingly. The second step, the comprehension of the situation, concerns the understanding of environmental elements' properties in light of the operator or system's goals. For example, an increase in the number of online users, beyond normal thresholds, could mean that the e-commerce infrastructure needs to self-adapt to process a higher number of purchase orders per time unit. Similarly, a change in a shopper's preferred location could imply changes in the offers delivered to this person. The third step, the projection of the status of environmental entities in the future, refers to the prediction of the state of these elements in the future to support decision making. For example, by projecting the number of purchase orders to be processed in the following minutes, under a peak system load, it should be possible to decide the number of new software components to be deployed to expand the system capacity.

Two common concerns in situation awareness are time and space. Time is important because situation awareness is built up over time rather than instantaneously. Therefore, situation awareness—context awareness—mechanisms, must observe, understand and project the status of environmental entities over time. Space is fundamental to define the scope of the environment that must be observed. For example, shopping sites can apply user context, such as "likes" and wish lists, to limit the space of products that are interesting to the user and based on this personalize product offers. With the involvement of smart personal devices in the shopping experience customers provide time and space points of referential information. Businesses can expand a user's data point to a volume of time and space of information to understand and anticipate a customer's situation and relevant buying options. The user's current GPS location can be expanded to a close proximity circle and used to provide, for example, relevant offers in the neighborhood. The moment in time can be expanded to a period in time to provide, for example, seasonal offers. By combining location and time we compute situation volumes of time and space to understand the customer's motivation in time and space in our situation-aware smarter shopping case study.

2.2.2 Characterization of Context Information

Context information describes situations [Sch02]. Therefore, situation awareness relies on the management of the context information life cycle. To manage context information in SASS systems, it is useful to characterize context information types and management categories. Context information can be organized along several axes

from static to dynamic, from non-volatile to volatile, from non-transient to transient, and many others. Moreover, the dimensions for such classification can vary from one domain to another. According to the findings of our survey, context information can be organized along five main categories, however, many other categories can be derived by combining the categories presented in Figure 2.1.

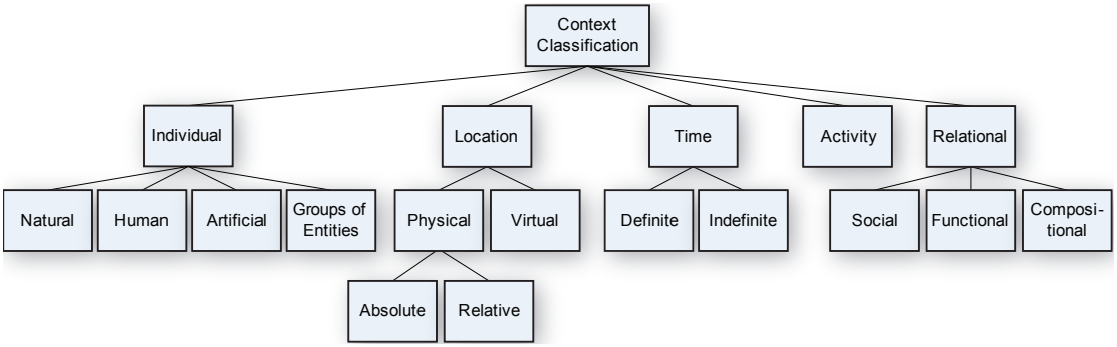


Figure 2.1: Classification of context information. *Individual context* is related to information observed from independent entities or groups of them that are not interacting with others. *Time, location* and *activity* refer to the *when, where* and *what* of a context situation. *Relational context* emerges from the relationship among the other types.

The first category, *individual context*, includes anything that can be observed about an isolated subject (i.e., the state of the subject). Moreover, one subject can expose more than one individual context depending on whether or not it plays multiple roles. Moreover, individual context can be classified depending on the subject type into *natural, human, artificial,* or *groups of entities* [ZLO07]. The first subcategory, *natural context,* represents living and non-living entities that are not the direct result of any human activity (e.g., weather conditions). The second one, *human context,* refers to any information about the behaviour of users and their preferences such as security profiles, language settings, and their way of interacting with the system (e.g., a buyer's security profile that protects sensitive information such as payment methods). The third one, *artificial context,* describes the state of entities resulting from human actions or technical processes. Some examples of artificial context are information related to buildings, and hardware and software configurations (e.g., the software architecture configuration of an e-commerce platform). Individual context can emerge from *groups of subjects* that share common characteristics but not necessarily interact with each other. Most importantly, group memberships may emerge dynamically at runtime (e.g., users with similar shopping preferences).

Many aspects of the interactions between users and systems are related to the

second category. *Location*, physical or virtual, represents the place of settlement or activity of an object. Examples of physical location are the absolute coordinates of the user's location (e.g., a shopping mall's address), the position of an object with respect to another (e.g., the directions to reach a store from the user's current location), or simply the name of a city. An example of virtual location is the notion of Uniform Resource Identifier (URI), for example, to identify the location of a web service.

The third main category is *time context*. Most interactions between users and systems are influenced by this dimension. Time information not only provides context about a specific date and time, but also categorical information such as holidays, working days, and meeting schedules. Dey and Abowd proposed an initial categorization of context based on primary and secondary information. In this former classification, secondary context can be inferred from the attributes of entities with primary context [ADB+99]. We call primary and secondary context as *explicit* and *implicit* context, respectively. Thus, context reasoning concerns the identification of implicit contextual facts from explicit ones, and is important to enrich situational knowledge. As an example, an online shopping application can correlate user preferences with shopping seasons sending custom offers to users during certain times of the year. In the same way, a SASS system can suggest an activity to the user, based on past experiences of the same type of activity (e.g., a recommender system). Time context can be either *definite* or *indefinite*. The former represents time frames with specific begin and end points (i.e., a definite duration). The latter expresses a recurrent event that occurs while another situation takes place. In other words, it is not possible to know its duration in advance. The fourth category, *activity context*, answers questions regarding future, current, and past goals, as well as actions and tasks of an object (e.g., the purchase order process performed by an e-commerce system).

According to Zimmerman *et al.*, all these context categories can be related to each other based on the three relational subcategories depicted in Figure 2.1: *social, functional,* and *compositional* [ZLO07]. A sound representation of context must provide mechanisms to express semantic dependencies between two or more instances of context information. *Social context* emerges from the interrelation among individual human and group context. Examples of this context type are affiliations, colleagues, friends and customers. Social context constitutes not only the foundations for applica-

tions provided by companies such as Facebook[7] and LinkedIn,[8] but also an important source of value creation and revenue generation for their businesses. *Functional context* refers to the information related to the usage that an object can make of another. For example, the personalized functionalities exposed by a service to a user, and the use that a user makes of the email service provided by Google[9] (e.g., when the user marks messages to rank their importance).

2.2.3 Context Modeling

Context modeling is an important component of the context information life cycle. Context models represent the relevant aspects of entities that affect the interactions between users and systems, as well as the situations that trigger dynamic changes in such interactions. Control loops play a crucial role monitoring context in supporting smart interactions and services. We use the term *context control objectives* to refer to the context monitoring requirements for supporting self-adaptation in a SASS system. These aspects include not only information regarding individual, time, location, and activity context, but also the relations among them.

Several approaches for context representation have been proposed in recent years. In particular, many of them have been analyzed by Bettini *et al.* [BBH+09], Moore [Moo07], and Strang and Linnhoff-Popien [SLP04]. Context feature diagrams presented in this subsection follow the notation depicted in Figure 2.2. Figure 2.3 presents selected features from existing context modeling approaches and context management infrastructures. The *approach-type* feature group represents any of the possible ways of representing context such as logic-based and ontology-based models. The second group of features, *context entities and situations representation*, concerns the modeling of important aspects such as granularity, properties, constraints, and relationships. The third one, *timeliness modeling*, refers to the representation of past and future states. The fourth set of features, *quality modeling*, corresponds to metadata for representing quality attributes of context information. Finally, the *software engineering* set of features includes tools to support context requirements analysis, context model design, code generation, and runtime processing. In this background subsection we elaborate on the first four groups of features. Further information about the last modeling feature is available in our context survey [VM10b].

[7] http://www.facebook.com
[8] http://www.linkedin.com
[9] https://mail.google.com

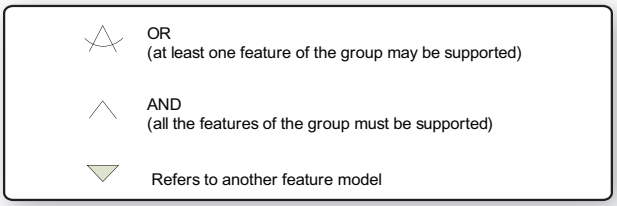


Figure 2.2: Notation used in context modeling and context management feature diagrams

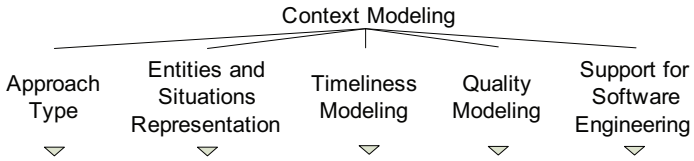


Figure 2.3: Features of context modeling. Appropriate context models must support features of each of these six categories.

Context Modeling Techniques

Figure 2.4 presents the most common context modeling approaches. The two first features are known as the early approaches. *Key-value pairs* are used to represent context information as a list of attributes with their corresponding values. *Markup models* generally are in the form of XML models. *Object-oriented models* use the object-oriented paradigm to represent dynamic characteristics of context [CRS07, SHK07]. *Meta-model-based models* emerged recently in model driven engineering (MDE) domains to support the dynamic generation of new model instances based on meta-models defined at design-time [TKAZC09, Rei08, Bun04, SRP+05]. *Logic-based* and *ontology-based models* are used to define formal specifications of context entities and the relations among them in a particular domain [SLPF03, KS07]. Finally, *fact-based models*, such as the object-role approach applied by Henricksen *et al.*, emerged from the need for providing formal models that are able to support query processing and reasoning [HIM05].

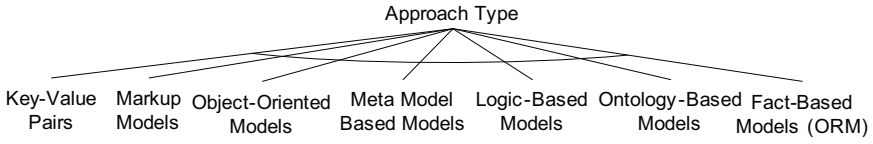


Figure 2.4: Context modeling approaches

Representation of Context Entities and Situations

The features depicted in Figure 2.5 are important attributes for the representation of context entities and situations. Appropriate context models must support the detection of conditions for moving between situations and contexts. Moreover, they must allow the representation of context information types, as well as different levels of granularity, constraints, relations among context entities, and spatial and scope information about the specific context.

The first feature within this group is *granularity,* which concerns the level of detail of contextual data and the scope of this information. *Context type* and *context property* features refer to the capability of models to represent raw context after its preprocessing during the acquisition process. Thus, reasoning about dynamic context depends on the suitability of such models to express data gathered from the environment. Moreover, a context model must represent different types of context information and the context management system must handle this information depending on its type [BBH+09]. The capability of expressing *constraints* is also a key factor for modeling context information. Context modeling techniques are required to support both, consistency verification of the model and context reasoning techniques. Moreover, they must assist the context management infrastructure in dealing with incomplete or conflicting context information. Thus, constraints are necessary for consistency verification. In the same way, the *representation of relationships* among context entities is required to derive new contextual facts from existing context observations. Finally, spatial and scope representation are useful for quality-based reasoning, management, and provisioning. In particular, *spatial representation* is useful to limit the reasoning space to avoid performance degradation. Similarly, *scope representation* is useful to enforce policies such as privacy of context information [SLP04].

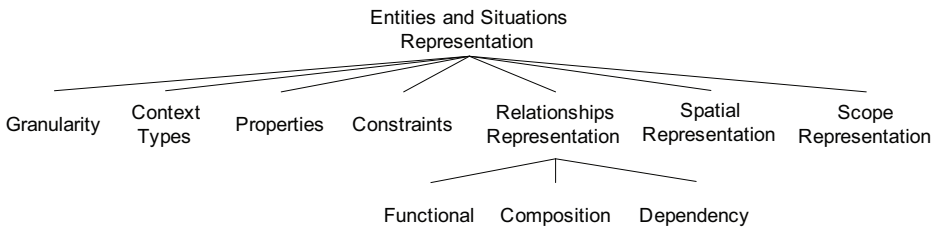


Figure 2.5: Features for representing context entities and situations. Reasoning about dynamic context depends on the suitability of context models to express these features.

Timeliness and Quality Modeling

The representation of past and future states, and quality attributes of context information are also important features of context modeling techniques (cf. Figure 2.6). On the one hand, context-aware applications need information regarding past and future states of their relevant context entities. Therefore, the *timeliness* feature, characterized by Bettini *et al.*, must be captured by context models and managed by context management systems [BBH+09]. Nevertheless, the management of historical information is challenging if the context is highly dynamic. It may not be feasible to store every variation along the time dimension. Consequently, summarization techniques are required. On the other hand, data gathered from the environment can have different levels of quality or can even be incorrect. Krause and Hochstatter characterized quality problems of context information: (i) Contextual data could be out-dated and no longer applicable to a particular situation; (ii) default profile information could not apply to the current situation or physical constraints; and (iii) temporary effects could limit the precision of sensors [KH05]. Therefore, *quality modeling* must express both information about the quality of the data that they are modeling (i.e., quality attributes), and information about quality policies and metrics (e.g., confidence, freshness, or resolution).

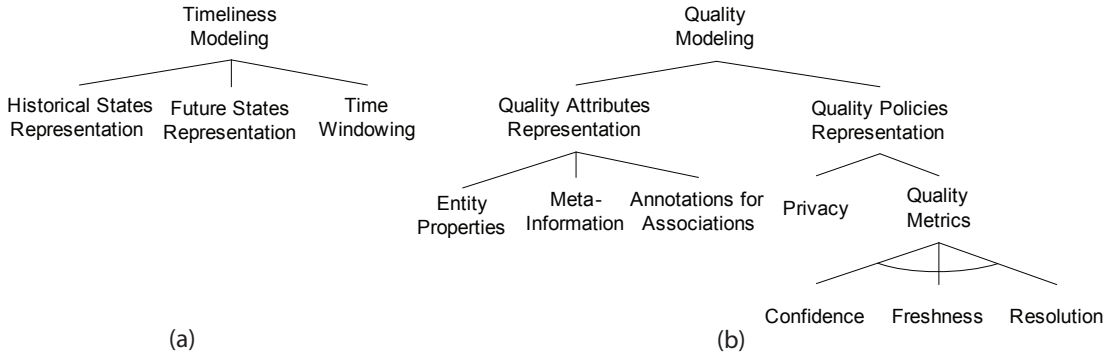


Figure 2.6: Features of (a) timeliness and (b) quality modeling

2.2.4 Context Life Cycle Management

From a SASS system, we expect that it is able to detect its current state or context and determine how to tailor its behavior based on relevant context. For this, SASS systems must be instrumented with context management capabilities. Features of context

management infrastructures belong to any of the six groups depicted in Figure 2.7. The second feature, *modeling*, was addressed in the previous subsection.

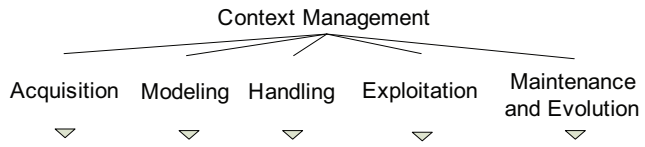


Figure 2.7: Top level features of context management approaches

Context Acquisition

Context acquisition refers to the gathering of explicit context from the environment. Context management infrastructures must implement mechanisms to obtain observable data from physical sensors, and determine comparable measures by performing basic transformations.

Figure 2.8 depicts context acquisition feature groups. The first group corresponds to *distributed context sources*. According to this set of features, context management infrastructures can obtain context information from one, or more than one of these subtypes. Thus, due to this distributed character of context information, such infrastructures must implement the second feature group, *context source discovery*. As an illustration of the third group of features, *gathering*, the service-based infrastructure proposed by Hynes *et al.* implements both pulling and pushing as mechanisms to receive contextual data. Context management infrastructures should be able to gather context information by implementing these mechanisms. That means, retrieving data and receiving unsolicited information from the source. Similarly, the implementation of *levels of indirection* is a highly desirable feature for supporting dynamic context gathering. *Pre-processing* features such as *filtering* and *classification* are related to basic transformations of context information to provide symbolic observables at the appropriate level of abstraction [CA06, Cro03]. Finally, *reuse* refers to the need for maintained existing sensors and gathered context to be used in the future.

Context Handling

SASS applications use environmental information to both decide whether or not to adapt their behavior, and provide context-aware functionality. Thus, context management infrastructures must provide cognitive support to infer context information

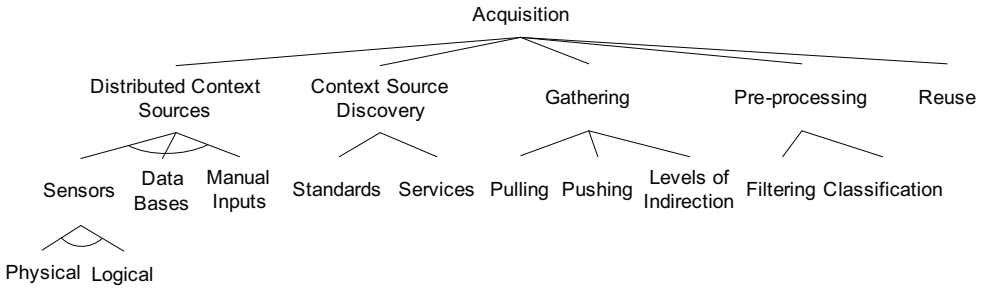


Figure 2.8: Features of context acquisition. Context is gathered from heterogeneous and distributed sources using different mechanisms such as pulling and pushing. Raw context must be pre-processed and classified before being manipulated at higher levels of abstraction.

from sensed context, and reason about high level context abstractions (i.e., derivation of implicit contextual facts).

Features of context handling are depicted in Figure 2.9. *Derivation of contextual facts* is the first feature of this group. Contextual facts can be derived from explicit context observations represented in context models. The second group of features describes *techniques for supporting context handling*. Depending on the specific application domain, context management infrastructures studied in our survey support at least one of these techniques. Context handling techniques are used to reason about the information represented in a context model. Moreover, a context management infrastructure must implement *consistency verification* of the context model for guaranteeing the accuracy of the reasoning process and ensuring the dynamic adaptation of such models [BBH+09].

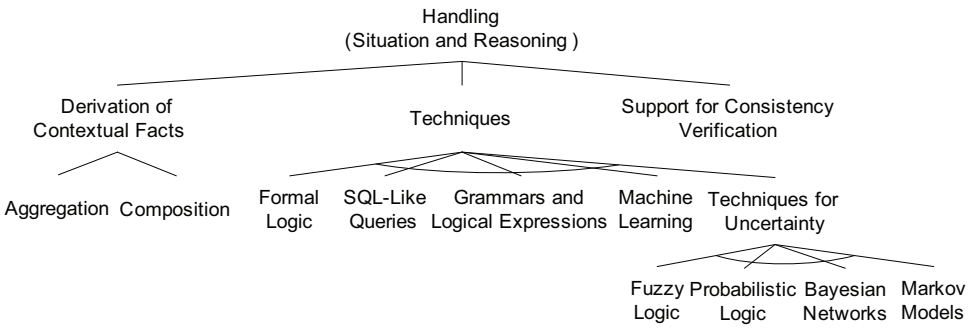


Figure 2.9: Features of context handling. Handling concerns mechanisms to infer context information from sensed data, and reason about high level context abstractions.

Context Exploitation

Context exploitation concerns the request and provisioning of environmental information. Context management infrastructures must provide services to support not only the gathering of context information, but also its provisioning to multiple endpoints.

User and business goals are continuously evolving according to their changing environment. Consequently, smart interactions and smart services have the challenge of supporting the user's tasks and stakeholders' needs, even when their requirements are highly dynamic and uncertain. Figure 2.10 illustrates the selected features of context exploitation. According to the first branch, context information must be disseminated by implementing *policy-based mechanisms*, as it was proposed by Salomie *et al.* [SACD08], Strassner *et al.* [SdSJNR+09], and Samaan *et al.* [SHK07]. The second feature, *provisioning endpoints*, concerns the communication mechanisms between context consumers and context management infrastructures. *Dynamic context sharing* is the third feature of context exploitation. Even when the context can be related to a specific domain, fundamental context categories such as *individual, time,* and *location* can be shared by a group of different applications. Moreover, this group can vary over time. This feature optimizes the use of resources and the quality of the information provided to context-aware systems and supports the dynamic collaboration among services as demanded by the smart internet. Finally, context exploitation requires efficient *persistence mechanisms* as environmental information is managed along the time dimension.

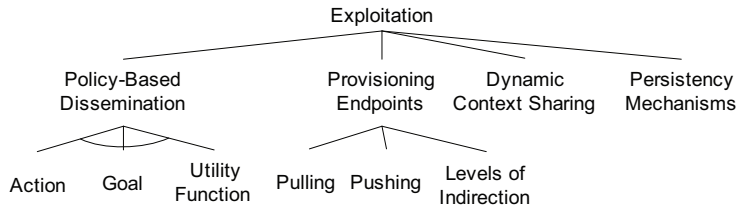


Figure 2.10: Features of context exploitation. These features concern the provisioning of contextual facts to context-aware applications.

Maintenance and Evolution of Context Management Infrastructures

Coutaz *et al.* have discussed the issue of dynamic context evolution [CCD05]. This set of features, depicted in Figure 2.11, concerns the capability of context management infrastructures to evolve, as required by the dynamic nature of context, for instance,

by reconfiguring their architecture to support new context requirements, providers or consumers at runtime.

Context disposal refers to the fully or semi-automatic removal of context information when it is no longer useful. Policies and enforcement mechanisms can be defined to manage the disposal of context information according to certain conditions such as freshness, location or context sources [Hyn09]. Contextual data must also be managed to ensure the persistence of valuable information for context reasoning. Furthermore, the second feature, *management of contextual data*, must be based on practical considerations such as performance, computational cost, and distributed storage.

The *scalability* feature is a direct result of the distributed nature of context information. From this perspective, a context management infrastructure must monitor its performance to manage its degradation as the number of context sources and context consumers increases [PCP07]. Hu *et al.* have discussed the importance of implementing features such as *self-configuring* and *self-healing* in context management infrastructures. Dynamic context management systems should implement (i) dynamic reconfiguration to support new types, sources, and consumers of context information, or self-recover from failures at runtime, and (ii) self-monitoring mechanisms to support their dynamic behavior.

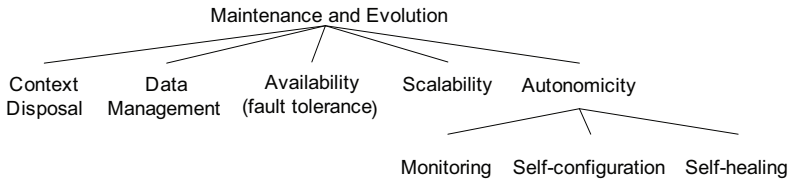


Figure 2.11: Features for the maintenance and evolution of context managers

2.3 Semantic Web

The semantic web can be defined as an extension of the world wide web (WWW), or web for short, that enables systems to smartly search, combine, and process web data based on the meaning that these data have to humans [HKR09]. This extension exploits the potential of the web since it enables data sharing effectively across the internet [BLHH+06].

Semantic web applications can be defined as software solutions that use any of

the well standardized ontology languages such as Resource Description Framework (RDF) [MM04] or Ontology Web Language (OWL) [W3C04d] to support data exchange and integration, as well as knowledge representation and reasoning [HKR09]. Instances of application domains for the SW include web data exchange and syndication, semantic wikis, semantic portals, semantic metadata in data formats, semantic web in life sciences, and ontologies for standardizations. Hitzler *et al.* provide several examples of semantic web applications within these domains [HKR09]. In particular, SW applications in life sciences have demonstrated the benefits of using vocabularies and ontologies for knowledge representation and reasoning. These applications include the *Gene Ontology*,[10] which includes many of the world's most important genome repositories; and *SNOMED CT*,[11] the ontology that defines clinical terminology and that is widely used by health software applications.

This dissertation applies SW to the management of dynamic context to support context-awareness in SASS systems. Particularly, in user-centric situation-aware smarter shopping. We apply SW ontologies to the characterization and representation of context information in any application domain. Different approaches to context-awareness have used ontologies to represent and reason about context situations [SLPF03, KS07, PGR07, Moo07, Rei08, SW09, HRH09]. Since ontologies are designed for open data integration, they support the addition of new types, relationships and reasoning rules into existing specifications. Moreover, SW-based reasoning mechanisms will understand new vocabularies even if they were integrated while the context-aware system was already in execution. Therefore, ontologies and SW reasoning engines are natural mechanisms to represent context information and infer new contextual facts in dynamic and uncertain environments. Indeed, the SW community has developed several vocabularies that are useful to characterize specific context types such as time,[12] location,[13] social relationships,[14] mobile platforms and preference profiles,[15] and e-commerce offers.[16] Nevertheles, to the best of our knowledge, there is no approach, besides our SmarterContext solution, that provides a foundational context taxonomy that allows the integration of specific vocabularies that characterize concrete types of context information for particular application domains.

[10] http://www.geneontology.org
[11] http://www.ihtsdo.org/snomed-ct
[12] http://www.w3.org/TR/owl-time
[13] http://www.w3.org/2003/01/geo
[14] http://www.foaf-project.org
[15] http://www.w3.org/Mobile/CCPP
[16] http://www.heppnetz.de/projects/goodrelations

SW technologies provide the means to build models that allow the description of anything in the web, reason about the knowledge encoded by these models, and transmit this knowledge among web entities [HKR09]. Our SmarterContext solution exploits these foundations to manage context information as required by SASS systems and the smart internet. First, RDF [MM04] provides the framework to represent context entities and describe relevant information about them. SmarterContext uses this framework to define an RDF-based ontology that characterizes context information across different application domains. Second, RDF, RDF Schema (RDFS) [W3C04c], and OWL [W3C04d] provide the semantic mechanisms to reason about context entities. SmarterContext exploits these reasoning capabilities by specifying context reasoning rules for particular application domains. Moreover, rules can be defined hierarchically such that more general rules are useful across the corresponding sub-domains. Finally, XML, RDF and OWL provide the interoperability mechanisms to exchange context information among different sources. These interoperability mechanisms support the gathering of context from providers, and the provisioning of context to consumers.

The following two subsections present RDF and OWL, two important layers of the SW that provide part of the foundations for dynamic context management in our SmarterContext solution.

2.3.1 Linked Data and Resource Description Framework

The vision of the SW relies on linked data, a common framework based on RDF for sharing data and integrating a variety of applications [BLHH+06, HKR09]. LD has been recognized as an important enabling technology for the smart internet and the personal web because it allows the creation of typed links between data from different sources (cf. Section 2.1.4). This implies that data is published on the web such that it is machine-processable, has an explicit meaning, and can be linked to other data sets. SmarterContext uses LD and realizes context gathering and provisioning by making context information discoverable and machine processable. Furthermore, SmarterContext realizes context reasoning as its ontology provides explicit semantics that allows inferring implicit contextual facts.

LD uses RDF to describe things in any application domain using typed statements also known as labeled links. Building on top of RDF, the SmarterContext ontology provides the entity types (context things), object properties (labeled links

between entity types), and data properties (labeled links between entity attributes and their corresponding values) to describe relevant context entities (entities that can affect the situation of users and/or systems). Berners-Lee proposed a list of LD principles for publishing and connecting data using the infrastructure of the web, while being compliant with its architecture and standards [BL06]:

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- Provide useful information about things identified by URIs in a standardized way (RDF, SPARQL).
- Include links to other URIs, so that they can discover more things.

LD is based on two fundamental web technologies, Uniform Resource Identifiers (URIs) [BLFM05], and the HyperText Transfer Protocol (HTTP) [FGM+99]. A URI is a compact sequence of characters that identifies an abstract or physical resource. The HTTP protocol provides a mechanism for retrieving information about entities identified by URIs. RDF is based on the principle that things can be described by making statements about their properties and corresponding values. For this, RDF encodes data in the form of statements defined as *subject*, *predicate*, *object* triples [MM04]. The subject is the entity the statement is about, the predicate is the property being described about this entity, and the object corresponds to the value of the described property.

Figure 2.12 exemplifies the graph representation of a simple RDF statement with corresponding subject, predicate and object. This statement provides context information about a user's preferred location: "Norha" (the subject) has "preferred location" (the predicate) "Victoria" (the object). Subject, predicate and object are identified by a URI. For convenience, RDF specifications use a shorthand for referring to URI references (QName). In this way, the full URI is defined by appending the local identifier to the abbreviation (QName prefix). For example, the statement presented in Figure 2.12 involves two QName prefixes. `pwc:` to abbreviate the namespace of one of the modules of the SmarterContext ontology: `http://smartercontext.org/vocabularies/pwc/v6.0/pwc.owl#`, and `geo:` the namespace of a vocabulary used for geographical locations: `http://smartercontext.org/vocabularies/rdf /geo.rdf#`.

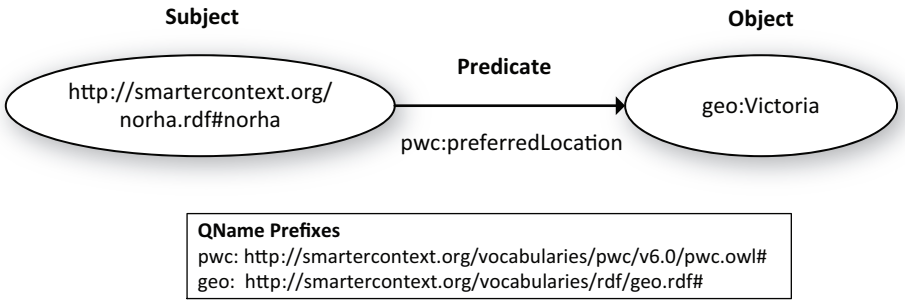


Figure 2.12: A simple RDF statement

2.3.2 Vocabularies and Ontologies

Vocabularies describe entities in the world and the relationships among them [BL06]. In the SW, vocabularies are collections of classes and properties expressed using RDFS [W3C04c] and OWL [W3C04d].

RDF Vocabulary Definition Language (RDF Schema)

RDF Schema, RDFS for short, is a semantic extension of RDF. It defines classes and properties that can be used to describe classes, properties, and other RDF resources. Tables 2.1 and 2.2 summarize the classes and properties from the RDFS specification [W3C04c] used in our SmarterContext ontology. `rdfs:` is the QName prefix used for `http://www.w3.org/2000/01/rdf-schema#`, the namespace of RDS.

Table 2.1: RDF Schema classes used in the SmarterContext ontology.

Class	Description
rdfs:Resource	Any entity described by RDF — e.g., user Norha from Figure 2.12.
rdfs:Class	The class of resources that are RDF classes — e.g., a class User that defines the entity Norha.
rdfs:Literal	The class of resources that are values such as strings or integers. Literals may be typed or untyped — e.g., the values for the age and the address of user Norha.
rdfs:Datatype	Any datatype defined in the XML Schema [W3C01] — e.g., `xsd:string`.

Table 2.2: RDF Schema properties used in the SmarterContext ontology.

Property	Description
rdfs:range	Defines the universe of possible values of a property — e.g., the possible values of the property `pwc:preferredLocation` correspond to entities of type *Location* context.
rdfs:domain	States that any resource with a given property is an instance of one or more classes — e.g., any resource that has the *marital status* property is an instance of class *User*.
rdf:type	States that a resource is an instance of a class — e.g. Norha is an instance of type *User* (cf. Figure 2.12).
rdfs:subClassOf	States that all the instances of a class are instances of another one — e.g., every instance of class User is an instance of class *Human Entity*.
rdfs:subPropertyOf	States that all the resources related by a property are also related by another one — e.g., if Norha is related to Peter by the property *child of*, and *child of* is a subproperty of *relative of*, then Norha is related to Peter by the property *relative of*.

Ontologies and the Web Ontology Language (OWL)

RDFS is suitable for modeling simple ontologies due to its limited expressiveness and knowledge inference support [HKR09]. To model more complex knowledge, the SW provides OWL, an expressive representation language based on formal logic. OWL supports the description of more complex relationships between classes and properties than the ones supported by RDFS.

OWL is used to model ontologies. An OWL ontology is a set of classes, properties, and individuals useful to describe entities and the relationships among them in a particular application domain. *Classes* are instances of `owl:Class`, which is a subclass of `rdfs:Class`. Therefore, as described in Table 2.1, OWL classes are RDF resources of type class. Classes are also known as *entity types*. *Individuals* correspond to instances of classes. OWL defines two types of properties, *abstract properties* and *concrete properties*. Abstract properties, also called *object properties*, relate individuals with individuals, whereas concrete properties link individuals with data values. Both are subtypes of `rdf:Property`. `owl:` is the QName prefix used for the `http://www.w3.org/2002/07/owl#` namespace.

The SmarterContext ontology is based on RDF and a subset of the OWL-

Lite [W3C04b] specification. Taking into account that pure RDFS is not sufficient for context reasoning as envisioned in SmarterContext, we decided to use the simpler version of OWL called OWL-Lite, which provides enough support for context representation and reasoning in the SmarterContext framework. Table 2.3 describes the OWL-Lite properties used in our SmarterContext ontology.

Table 2.3: OWL-Lite properties used in the SmarterContext ontology.

Feature	Description
owl:inverseOf	If property P_1 is stated to be the inverse of property P_2, then if X is related to Y by P_2, then Y is related to X by P_1 — e.g., properties *hosts* and *hosted by* are inverse. Thus, if CASCON 2012 is *hosted by* Hilton Markham, then Hilton Markham *hosts* CASCON 2012.
owl:TransitiveProperty	If a property P is transitive, then if X is related to Y by P, and Y is related to Z by P, then X is related to Z by P — e.g., property *located in* is a transitive property. If Norha is *located in* Victoria, and Victoria is *located in* British Columbia, then Norha is *located in* British Columbia.
owl:FunctionalProperty	A property that has at most one value — e.g., the birth year of a human entity.
owl:SymmetricProperty	If a property P is symmetric and X is related to Y by P, then Y is related to X by P — e.g., property *near to* is symmetric. If Victoria is *near to* Vancouver, then Vancouver is *near to* Victoria.

2.4 Feedback Control for Self-Adaptivity

In Section 1.3.2 we presented the *feedback loop* from control theory as the model we selected to drive our research on self-adaptivity. This section discusses key concepts, reference architectures, and identified benefits and challenges in the application of feedback loops to the engineering of self-adaptive software systems.

2.4.1 Feedback Loops

To keep objectives controlled in a target system, several strategies have been proposed. The three most common strategies are (i) the *regulatory* control, which ensures that the measured output is as close as possible to the reference input; (ii) the *disturbance rejection*, to control the effects of disturbances on the measured output; and (iii) the *optimization* control, which continuously seeks to obtain the best value of the measured output, as effectively as possible [HDPT04]. These strategies imply variations on the controller element but are realizable with the general structure of the block diagram. To compute the controlling signals, there are several possible mechanisms. In control theory, the representative mechanism is the *system transfer function,* a mathematical model built upon the physical properties and characteristics of the target system. Depending on these characteristics, the transfer function can be built, for instance, with proportional, derivative and integral (PID) terms. The parameters in a PID controller have special significance given that there exist precise and sophisticated methods for tuning their associated parameters.

Even though the application of control theory to industrial processes is well understood, its application to the control of software systems has at least two significant challenges: first, control theory is based on continuous mathematics, and second, it relies on measurements taken from, and actions performed into, physical, self-contained and self-performing artifacts (e.g., sensors, gauges and valves/actuators for temperature, pressure and other variables). As their associated variables are in the continuous-time domain, the use of continuous mathematics in this theory fits perfectly. In contrast, software systems are composed of intangible artifacts with discrete-time behavior and not always well characterized properties. Thus, direct sensing must be performed by CPU time-consuming software artifacts, and the adaptation mechanisms must reason on the target system's discrete-time output. Moreover, to exploit the possibilities of software adaptation fully, the output of the adaptation mechanism must be more structured than controlling signals to be transduced

by electro-mechanical devices. This output may take the form, for example, of a plan of ordered actions to be instrumented by the software actuators on the target software components. Fortunately, there exists also the theory of linear discrete-time systems, which closely resembles the theory of linear continuous-time systems. However, linear discrete-time mechanisms are not suitable for every kind of adaptive software system.

The benefits of integrating feedback loop-based models into the engineering of self-adaptive software systems have been pointed out by several researchers in the field. [OMT08, MPS08, BSG+09, CLG+09]. Oreizy *et al.* define runtime adaptation in the form of two processes that exploit feedback loops to manage adaptation and system evolution, respectively. The evolution management process feedback loop is in charge of monitoring the consistency between architectural models and the actual system implementation. Whenever this consistency is no longer satisfied, the evolution management process feeds monitored information back to the adaptation management process feedback loop, which is in charge of reconfiguring the system's architecture [OMT08]. Müller *et al.* outline the benefits of specifying the feedback loops and their major components explicitly and independently. Furthermore, they articulate the usefulness of defining the interactions among the elements of a feedback loop explicitly, from analysis and design to implementation [MPS08]. Giese *et al.* also argue for the decoupling of feedback loops in control-based reference architectures to address the satisfaction of quality attributes (control objectives), the management of the context complexity, and the interactions among multiple feedback loops and their elements [BSG+09]. Cheng *et al.* also emphasize the importance of making explicit not only the feedback loops, but also their elements and properties [CLG+09]. In fact, Müller *et al.* [MPS08], as well as Kramer and Magee [KM07], attest that even though feedback loops have been recognized as fundamental design elements for self-adaptation, the related design documents and research publications usually hide the visibility of both the adaptation controller and the feedback loops. As a result, there currently exists no explicit methods for analysis, validation and verification useful to measure the effectiveness of adaptation mechanisms in software systems [VMT+11c]. Based on these remarks, an important challenge addressed in this dissertation is to increase the visibility of feedback loops, including their components and relationships among them, by making them explicit entities of software architecture design and, thus, directly analyzable, assessable and comparable.

Valuable papers have been published making significant advances in the field. For instance, the feedback control architecture for adaptive systems proposed by Shaw

decouples the elements of a feedback loop (i.e., comparison, plan correction, and effect correction), and identifies the importance of context relevance for the adaptation process [Sha95, MPS08]. In the same way, the autonomic manager (MAPE-K loop) presented in Figure 2.13, and the autonomic computing reference architecture (ACRA) depicted in Figure 2.14 are important contributions of IBM that also make the feedback loops in autonomic systems explicit [IBM06].

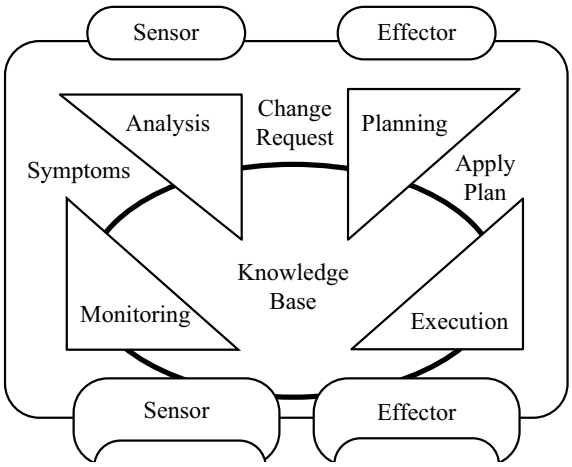


Figure 2.13: The MAPE-K loop [KC03]

The autonomic manager is an implementation of the controller element in the generic control feedback loop depicted in Figure 1.3. At the same time, the autonomic manager controls the managed element by implementing an intelligent control loop composed of the monitor, the analyzer, the planner, the executor, and the knowledge base elements. This knowledge base is an important element to share information along the loop. Moreover, it provides persistence for historical information and policies required to correlate complex situations. ACRA provides a reference architecture as a guide to organize and orchestrate an autonomic system. Autonomic systems based on ACRA are defined as a set of hierarchically structured building blocks composed of orchestrating managers, resource managers and managed resources. ACRA organizes resource management policies into layers where system administrators' (manual manager's) policies control lower level policies. System operators have access to all ACRA levels.

Despite ACRA and the MAPE-K loop that have helped considerably improve the visibility of feedback loops, the internal components of each control loop, and the control loop itself, still remain hidden inside the autonomic manager. Certainly, the

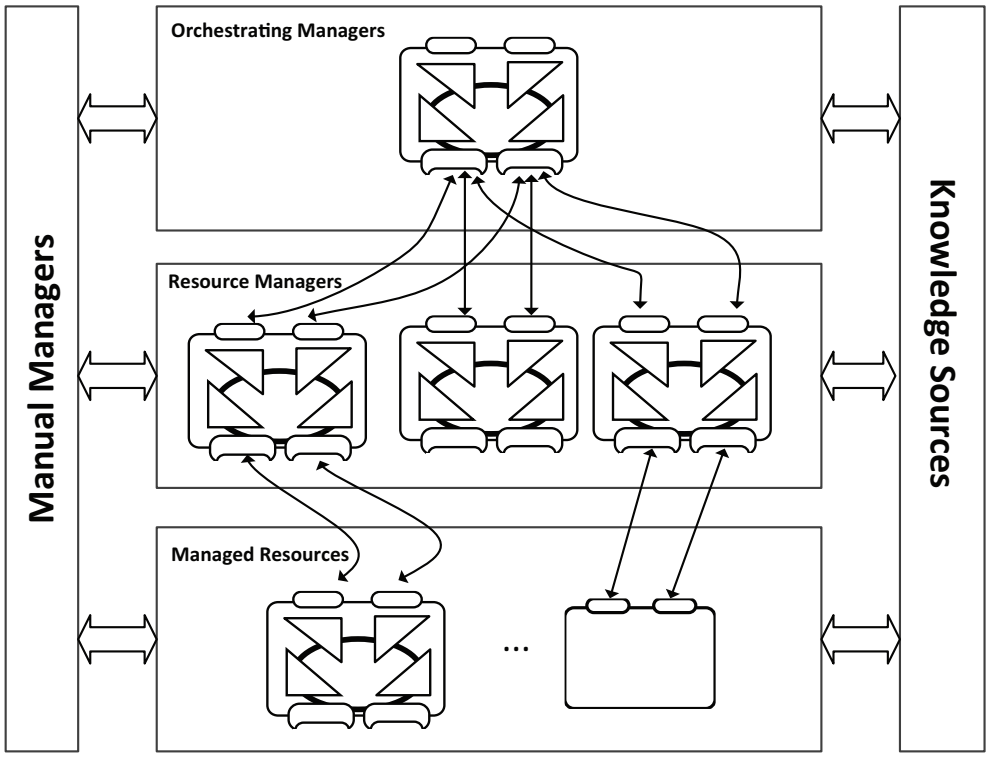


Figure 2.14: The Autonomic Computing Reference Architecture (ACRA) [IBM06]

specification of the autonomic manager, provided in the IBM architectural blueprint for autonomic computing, characterizes the manager as *a component that implements an intelligent control loop* [IBM06]. Moreover, even when the ACRA architecture drivers are clearly the feedback loops in the form of autonomic managers, their internal elements (i.e., the elements of the MAPE-K loop) are highly coupled. Therefore, even though the multiple feedback loops defined in an ACRA-based model can be distributed —for instance to improve the system scalability— this distribution is limited by the autonomic manager boundaries. Each autonomic manager implements the entire cycle to collect and aggregate information from the environment (monitor), to correlate the collected information and identify symptoms for supporting the adaptation decision making (analyzer), to plan the adaptation process (planner), and to perform the adaptation plan (executor).

2.4.2 Adaptive Control

From the perspective of control theory, adaptive control concerns the automatic adjustment of controllers. Adaptive control researchers investigate parameter adjust-

ment algorithms that allow the adaptation of the control mechanisms while guaranteeing global stability and convergence [DH02a]. Control theory offers several reference models for realizing adaptive control. We focus our attention on two of them, *model-reference adaptive control* (MRAC) and *model identification adaptive control* (MIAC).

Model Reference Adaptive Control (MRAC). MRAC, also known as *model reference adaptive system* (MRAS), is used to implement controllers that support the modification of parameters online to adjust the way the target system is affected (cf. Figure 2.15). A reference model, specified in advance, defines the way the controller's parameters affect the target system to obtain the desired output. Parameters are adjusted by the adaptation algorithm based on the control error, which is the difference of the measured output and the expected result according to the model.

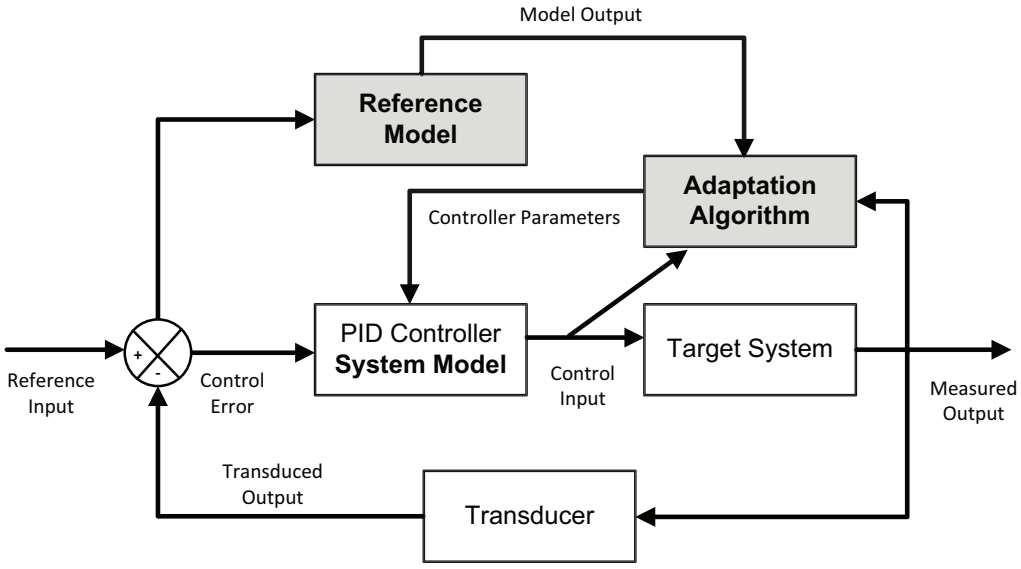


Figure 2.15: Model Reference Adaptive Control (MRAC)

Model Identification Adaptive Control (MIAC). In MIAC, the reference model that allows parameter estimation is identified or inferred at runtime using system identification methods. As depicted in Figure 2.16, the control input and measured output are used to identify the reference model (system identification). Then, the new model parameters are calculated and sent to the adjustment mechanism which calculate the parameters that will modify the controller.

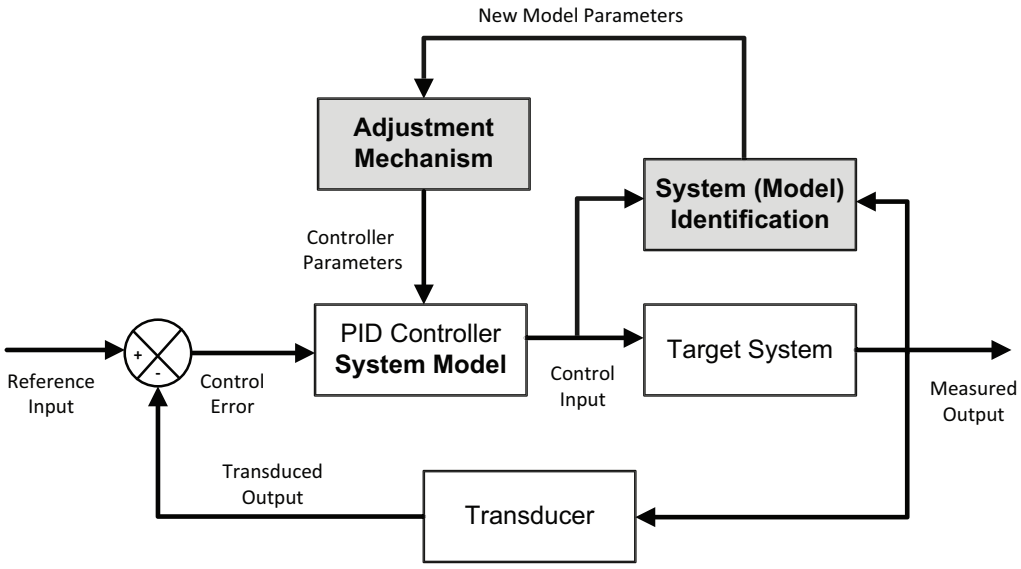


Figure 2.16: Model Identification Adaptive Control (MIAC)

2.5 Self-Adaptive Software Systems

The necessity of a change of perspective in the engineering of software systems has been widely discussed during the last decade by several researchers and practitioners in different software application domains [NFG+06, Dah10, CLG+09]. In particular, Truex *et al.* posited that software engineering has been based in part on an incorrect set of goals, from the assumption that software systems should support rigid and stable business structures and requirements, have low maintenance, and fully fulfill their requirements from the initial system delivery [TBK99]. In contrast to this static and "stable" vision, they proposed a new set of goals based on permanent analysis, dynamic requirements negotiation and incomplete requirements specification. Their proposal is aligned with the vision of self-adaptive systems, where dynamic adaptation is necessary to ensure the continuous satisfaction of their functional requirements while preserving the agreed conditions on Quality of Service (QoS) levels. These QoS levels are usually represented in the form of Service Level Agreements (SLAs), and their enforcement mechanisms are based on contracts and policies, among others [TT09, TCCD12]. To achieve the continuous satisfaction of changing requirements, the development of this kind of systems requires adaptation mechanisms able to perform short-term adaptations on them, and manage their long-term evolution [OMT08]. As part of this adaptation and evolution, system analysis must be performed at runtime, and its requirements satisfaction must be monitored and

regulated by continuously adjusting or enhancing its behavior [BSG+09, CLG+09].

2.5.1 A Survey of Self-Adaptive Approaches

To conduct our research on self-adaptivity, we analyzed 16 published approaches dealing with self-adaptive software systems [VMT+11c]. This survey was one of the basis for two of the contributions of this dissertation: our framework for evaluating quality-driven self-adaptive software systems, and our reference model for designing self-adaptive software systems (cf. contributions C4 and C5 in Figure 1.4).

We started this analysis with 34 research papers with different proposals for self-adaptive software systems published over the past decade. From this set, we filtered-out 18 mainly because either they presented very generic proposals (i.e., with non-measurable self-adaptive properties) or they did not provide enough information for our characterization purposes. From the analysis of the 16 remaining (and even considering the 18 papers that we filtered out), it is worth noting that the lack of metrics to evaluate self-adaptivity was a common observation. This lack of metrics hampers the realization of dynamic monitoring because it is impossible to identify changes in monitoring requirements, at runtime, having no explicit mapping between adaptation goals and metrics.

To characterize the selected approaches, we proposed a classification spectrum where self-adaptation mechanisms range from pure control theory to pure software engineering with many hybrid approaches in-between. Table 2.4 summarizes the characterization and classification of the approaches we considered in our survey.

In control engineering-based approaches, control actions (the means to affect the target system to obtain the desired effect) are continuous signals that affect behavioral parameters of the managed system. The structure of the managed system in these approaches is generally non-modifiable while its behavior is modeled mathematically [PGH+02]. In contrast, software engineering-based approaches are characterized by implementing discrete control actions that affect the managed system's software architecture (i.e., the system structure). In these approaches the adaptation is supported by a model of the managed system's structure and reflection capabilities that allow the modification of the system structure [AFF+01, DC04, FHS+06, GCH+04, KCC+07, LLC10, MG05, SBDP08, TCCD12]. In hybrid adaptive systems, control actions are generally discrete operations that affect either the computing infrastructure executing the managed system or the set of processes com-

Table 2.4: Characterization summary of the self-adaptation approaches analyzed in our survey.

Characteristic	Count [List of Approaches]
	Spectrum Classification
Control Engineering	1 [PGH+02]
Hybrid	5 [BG11, CCF04, CCG+09, EER+10, Whi05]
Hybrid-Software	1 [SLBL10]
Software Engineering	9 [AFF+01, DC04, FHS+06, GCH+04, KCC+07, LLC10, MG05, SBDP08, TCCD12]
	Monitoring Mechanisms
Monitor internal context	15
Monitor external context	2 [KCC+07, MG05]
Non specified	1 [SBDP08]
	Controller's Structure
Feedback control	2 [PGH+02, SBDP08]
Adaptive control	9 [AFF+01, BG11, CCF04, EER+10, FHS+06, GCH+04, KCC+07, SLBL10, Whi05]
Reconfigurable Control	4 [CCG+09, DC04, LLC10, MG05, TCCD12]
	Managed System's Structure
Non-modifiable	4 [BG11, EER+10, PGH+02, Whi05]
Modifiable with reflection	12 [AFF+01, CCF04, CCG+09, DC04, FHS+06, GCH+04, KCC+07, LLC10, MG05, SBDP08, SLBL10, TCCD12]
	Adaptation Properties
Settling time	4 [AFF+01, CCF04, KCC+07, Whi05]
Small overshoot	4 [AFF+01, CCF04, KCC+07, PGH+02]
Scalability	3 [AFF+01, DC04, FHS+06]
Stability	2 [AFF+01, PGH+02]
Accuracy	2 [CCG+09, SLBL10]
Termination	2 [EER+10, TCCD12]
Consistency	3 [LLC10, MG05, TCCD12]
Robustness	1 [DC04]
Security	0

prising the managed system. Usually, the structure of the managed system is non-modifiable [BG11, CCF04, CCG+09, EER+10, Whi05]. For example, we classified the approach proposed by Solomon *et al.* [SLBL10] as a hybrid approach because their

control actions affect the architecture of the managed system and the analysis process is based on a behavioral model of the managed system to decide when to adapt.

According to our classification spectrum, most approaches were identified as software engineering-based and hybrid adaptive systems. With respect to the adaptation goal (the justification for the system to be self-adaptive), we did not identify a relationship with our proposed spectrum. This means that it is possible to find any of the adaptation goals along the entire spectrum. Concerning reference inputs (specifications of the system's desired state), most approaches use contracts as the way to specify reference values for the adaptation goal and the corresponding measured outputs. All approaches that address monitoring explicitly, do so by monitoring logical properties of computational elements (internal context), while two of them take the external context into account [KCC+07, MG05]. Regarding the controller structure (feedback, adaptive or reconfigurable control), all approaches, except [PGH+02] and [SBDP08] that implement a simple feedback loop, implement either adaptive or reconfigurable control. Finally, the most common evaluation mechanism is the implementation of running examples based on simulated environments.

2.6 Discussion

This section discusses challenges addressed in this dissertation in light of related work.

2.6.1 The Smart Internet and the Personal Web

The smart internet has been defined as an extension of the internet today that builds on existing foundational technologies such as URIs, HTML, and HTTP but centers on people with the goal of realizing user-centric web integration. Although several research initiatives have been proposed to help develop the next generation internet, excepting the smart internet and the PW, neither IoT, the ubiquitous web, nor the user-centric internet have as their ultimate goal to place the user as the center of both web interactions and the integration of services and content. Therefore, because of this strong user-centric vision, we adopted the smart internet, and concretely e-shopping based on the PW, as the application domain for the validation of the contributions related to this dissertation.[17]

[17]The smart internet and PW are strategic research and business initiatives of IBM, one of our main research partners. This was another important motivation for us to adopt them as part of the application domains of our research.

The three major characteristics of the smart internet are its instinctive user model, a session that focuses on user needs and goes beyond the interactions of the user with a particular server, and collective web interactions. Context-awareness and self-adaptivity are key concerns for all three characteristics, as well as for next generation internet initiatives such as the one mentioned above. The PW is a concrete realization of the smart internet that focuses on the user as the center of web integration. Thus, smart interactions are required to support, from a people-centric perspective, the discovery, aggregation and delivery of resources from the internet. Moreover, smart services must provide the infrastructure required to support changes in these interactions according to changes in the user's situation. We grouped the challenges of the smart internet and the PW addressed in this dissertation as follows: (i) the need for a user-centric model of personal context information to improve situation-awareness in interactions and services; (ii) the empowerment of users to manage context and web integration through instinctive interactions supported by these models; and (iii) the realization of a new concept of session, based on the moc and personal web sphere metaphors, that focuses on users and is available across different servers.

2.6.2 Context-Awareness

Situation-awareness has been defined in terms of three elements: the perception of elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future. Moreover, situations are described in terms of dynamic context information. As a result of our survey, we proposed a general classification of context information, and a set of feature-based models for context representation and dynamic context management. Regarding context representation (cf. Section 2.2.3), the main challenge that we addressed in this dissertation is the design of context models adaptable at runtime that allow the specification of dynamic context entities, the relationships among them, and corresponding monitoring requirements. Concerning the management of the context information life cycle (cf. Section 2.2.4), the challenge addressed in this dissertation is the design and implementation of self-adaptive context management infrastructures that support context gathering, handling, provisioning, and disposal while addressing dynamic changes in context models. That is, changes in context monitoring requirements at runtime. From the perspective of the smart internet and user-centric SASS systems, the research objects of this dissertation, another important challenge is to put the

user in charge of the context management loop. None of the surveyed contributions in our comprehensive literature review provides a context modeling and management solution that (i) is driven by users, (ii) supports changes in context specifications and context management strategies at runtime, and (iii) is suitable for any application domain. Further analysis of related work on context awareness is provided in Appendix A, our survey book chapter [VM10b], and [VM10a, VMT11b, VMMn+11].

2.6.3 Self-Adaptive Software Systems

Over the past decade, self-adaptation has increasingly become a fundamental concern in the engineering of software systems to reduce the high costs of software maintenance and evolution and to regulate the satisfaction of functional and extra-functional requirements under changing conditions of system execution. Even though adaptation mechanisms have been widely investigated in the engineering of dynamic software systems, their application to real problems is still limited due to a lack of both methods to validate and verify complex, adaptive, nonlinear applications [Dah10], and reference models to drive the engineering of adaptation mechanisms with explicit feedback loops.

From our survey on self-adaptive systems we inferred two main conclusions. The first conclusion concerns adaptation properties and evaluation metrics. Adaptation properties and the corresponding metrics are rarely identified or explicitly addressed in papers dealing with the engineering of dynamic software systems. Consequently, without explicit adaptation properties it is impossible to monitor, assess and certify adaptive system behavior.

The second conclusion concerns the visibility of feedback loops and the need for adaptive monitoring in the engineering of adaptation mechanisms. Although the feedback loop model of control theory has been used as a reference in many self-adaptive systems in different application domains, the visibility of the feedback loop as the crucial architectural element to govern software adaptation remains often hidden. In many cases, the managed application is intertwined with the adaptation mechanism, rendering it as hard to analyze, reuse, and manipulate [HDPT04, BSG+09, MKS09]. In other cases, such as those following the multi-layer architectures (e.g., ACRA [IBM06], FORMS [WMA10] and Kramer and Magee's [KM07]), their designs assume completely closed and controlled context situations where monitoring requirements are not subject to change, even though several feedback loops can be evidenced in

them. However, for many systems it is not affordable to discard unexpected context changes and dynamic changes in adaptation goals and user requirements, such as SLA re-negotiation at runtime. In these cases, statically deployed context monitoring elements are not enough to cope with these levels of dynamics, which are implied by context unpredictability.

2.7 Chapter Summary

This chapter presented the research background topics that are relevant to this dissertation.

Section 2.1, *the smart internet,* focused on principles and research challenges of the smart internet and the personal web (PW). Section 2.2, *context awareness*, summarized the state of the art on context representation and context management, presented our proposed general classification of context information, and the feature-based models we proposed for context representation and context management. Our classification of context is useful to characterize context entities in any application domain, whereas our feature-based models are useful to guide the evaluation and design of context modeling and context management solutions. Section 2.3, *semantic web (SW)*, introduced SW applications and presented SW technologies that provide the foundations for context representation and reasoning in our dynamic context management solution. Linked data, recognized as an enabling technology for the smart internet, provides the framework for representing context in the form of RDF statements, and for exchanging context models in the form of RDF graphs with context providers and consumers. RDFS and OWL-Lite provide the expressiveness required for reasoning on contextual data. Section 2.4 presented important challenges in the application of the feedback loop from control theory as the foundational model for the engineering of self-adaptive software systems. Section 2.5 summarized our survey on self-adaptive software systems. Finally, Section 2.6 closed the chapter by pointing out the most important aspects, from the perspective of this dissertation, of the presented background topics.

Chapter 3

The Personal Context Sphere

In Chapter 2 we presented the smart internet and the personal web (PW). The smart internet is conceived as a new generation of the internet whose vision focuses on the discovery, aggregation and delivery of web entities dynamically, interactively, and according to user needs and situations (cf. Section 2.1.1). The PW is one of the possible realizations of the smart internet whose vision focuses on the user as the center and controller of web integration. That is, the discovery, aggregation, and delivery of web entities is driven mainly by the user (cf. Section 2.1.4). The visions of the smart internet and the PW are part of the research motivation of this dissertation. In particular, they relate to our second research question: *How to integrate users into the dynamic context management process to improve user quality of experience (QoE) in situation-aware smart software (SASS) systems?*

To answer this research question, we addressed the three general principles of the smart internet, the five categories of challenges defined for smart interactions, and the five conceptual principles of the PW. For this, we focused on the two main metaphors of the smart internet and PW: the *matter of concern (moc)* and the *personal web sphere*. The moc metaphor is proposed as a way of connecting user needs and situations (i.e., personal context) to web entities. The personal web sphere is proposed as the semantics that define relevant web entities, the relationships among them, and how the user interacts with these entities. For these two metaphors self-adaptivity and context-awareness play major roles. Regarding self-adaptivity, since the state of web entities, web entities themselves, and user situations change continuously, and web entities are related to user needs, the interactions of the user with these web entities must change over time. Therefore, the smart internet requires services with self-adaptive capabilities to tailor web interactions at runtime. Concerning context-

awareness, mocs provide the means to identify a user's context situation and the web entities that are relevant to the user's tasks in that particular situation. Moreover, the personal web sphere, as defined in the PW, provides the meta-model that characterizes web entities and their relationships with the user. Thus, a user moc can be seen as an *instance* of a *contextual model* derived from the personal web sphere meta-model of the PW. For a particular application domain (e.g., online shopping), an instance of the contextual model characterizes the user's context situation in terms of its activities and corresponding relevant web entities. A snapshot of this instance at a particular point in time provides the user's context information relevant to the user's current moc. For supporting situation-awareness, this context information must be acquired from the environment, integrated into the contextual model, processed to infer new contextual facts related to that situation, provided to be used by web applications in the accomplishment of the user's tasks, and disposed when it is no longer valid.

To contribute user-centric and user-controlled contextual models that are useful to improve context awareness, and to realize a new concept of a session centered on users (cf. the challenges of the smart internet and the PW summarized in Section 2.7), we proposed the *Personal Context Sphere (PCS)*. In abstract terms, a PCS is a personal context model that contains context information about web entities relevant to the user's tasks in a particular application domain. In concrete terms, a PCS is a distributed repository provided by a third party (e.g., a cloud infrastructure provider) that stores both the context information of a user gathered along the user's entire web experience, and the context consumers and providers with which the user shares this information. Most importantly, the access to a PCS is controlled fully by its owner, the user. For this, users are empowered to integrate into their PCSs web entities that act as context consumers and context providers of the user's context stored in the PCS, and to define the privacy and security policies for this information. In this way, context information can only be gathered from, and provisioned to, web entities integrated by the user into her/his PCS. Web entities enabled to act as context consumers and/or context providers correspond to mobile applications, web applications, and web services. We call these entities *personal web-enabled (PWE) entities*.

PCSs are the cornerstone of SmarterContext, our solution to user-driven dynamic context management. This chapter presents the PCS concept and how it is used in SmarterContext to empower end-users as main controllers of their per-

sonal context information, and how, with this solution, we address the principles of the smart internet and the PW in our case study on situation-aware smarter shopping.

This chapter is organized as follows. Section 3.1 introduces our situation-aware smarter shopping case study and explains how PCSs can be exploited to improve user quality of experience in the PW. Section 3.2 revisits the principles of the smart internet and the PW presented in Chapter 2 to explain how we address several of their challenges with our PCS approach. Section 3.3 analyzes the user-centric data and privacy security features of our solution. Finally, Section 3.4 summarizes the chapter.

3.1 Improving User Quality of Experience with Personal Context

User interactions with web entities are valuable sources of meaningful information to understand user intents and needs. Moreover, the information gathered from the interactions of the user with a particular web application can be exploited to improve the user quality of experience (QoE) provided not only by the same application, but also by others. Context-awareness and self-adaptivity are key to provide pleasant user experiences in the smart internet. Context-awareness allows smart internet applications to understand user needs and thus deliver content and services that users are willing to receive. Moreover, since users are the ones who certainly know best what their needs and interests are, they must be integrated into the context management loop to decide about the relevance of context information and the way this information must be used. Self-adaptivity enables the smart internet infrastructure with dynamic capabilities to adjust the delivery of relevant content and services to users, according to changes in their context situations.

Our approach to user-driven context management based on PCSs can improve QoE in any application domain. This is mainly because context information stored into the user's PCS can be exploited by any application authorized by the user as context consumer and/or context provider, rather than by the third party who initially obtains the information only. For example in health care, the PCS represents the patient's electronic medical record (EMR), and her preferred health care service providers represent context consumers and providers (PWE entities). At any time the patient wants to share her EMR with a new practitioner, the only step she has

to perform is to add this health care provider into her PCS as a new PWE entity. Similarly, she can eliminate health providers from her PCS whenever she wants to cancel context sharing with those entities. Suppose she integrated her family doctor into her context sphere. Because of this integration, a communication mechanism will be established between the patient's PCS and the doctor's health care software application. Thus, every time she consults her doctor, her PCS will be updated with new context observations. Patients can also integrate personal health monitors that keep track of their treatment progress. In this way, specialist software applications integrated into patient PCSs will receive context information about the patient's conditions opportunely. Moreover, by correlating PCSs of family members it could be possible to infer symptoms for early detection of diseases. We can imagine many opportunities to realize the vision of the PW and improve patient QoE in the health care domain with SmarterContext and PCSs. In this section we explain in detail how we realize the vision of the PW and improve QoE in online shopping applications by empowering users to manage their personal context information.

3.1.1 Situation-Aware Smarter Shopping

Imagine Norha, a frequent online buyer who has registered with our SmarterContext solution to create her shopping PCS. In this registration process Norha provided personal context information such as her gender, age, marital status, and preferred locations. She also integrated her husband's PCS application, her mobile shopping list application, and several web and mobile shopping applications. SmarterContext gathers relevant context about Norha's preferences and situations from different sources such as her mobile devices, and her web interactions. Suppose Norha is logged into her PCS which includes mobile shopping applications of Target, Sears, and Walmart. As a result, SmarterContext is now able to exchange Norha's relevant context information with these retailers. Suppose she has an electric grill in her shopping list, and integrated Victoria (BC) as her preferred location.

From the very first time Norha browses any of the integrated shopping applications, the corresponding retailers can take advantage of Norha's personal context to improve her shopping QoE. Suppose Norha is browsing Target's product catalog. Since this application receives Norha's data from her PCS, it knows Norha's situation and preferences, and can suggest relevant products accordingly. Norha can interact

with the products in the online catalog through instinctive web interactions such as likes, tags, wish lists, and rankings. For example, she added a pair of earrings from Sears's online catalog to her wish list. Product categories, rather than individual products, involved in these interactions constitute relevant context that is integrated by our user into her PCS, assisted by the SmarterContext's gathering service.

Suppose now that Norha is visiting Edmonton and has arrived at West Edmonton Mall. As soon as she gets into the mall, the smarter commerce application in her mobile device suggests deals available at the stores located in the mall, according to her preferences and shopping list. Finally, social relationships are also sources of relevant context information. For example, since Norha integrated her husband's PCS application into her PCS, products in her shopping list or wish list could be suggested as relevant products to him. When her husband integrates Norha's PCS application into his PCS, products in his lists will be suggested to Norha.

3.1.2 User-driven Context Management

The management of context information involves four main phases: context gathering, reasoning, provisioning, and disposal. Thus, in user-driven context management the user must act as the controller of these activities and context models represented by PCSs. The PCS is our key contribution to realize user-driven context management and web integration with our SmarterContext solution.

Context gathering. It refers to the acquisition of information about users and web entities with which users interact in a particular moc. In user-centric applications such as the shopping scenario described above, instinctive user interactions constitute the main vehicle for context gathering. For example, when Norha adds a product into her wish list in Sears's shopping site, this wish interaction marks the category of this product as a meaningful piece of context information. Once the interaction of the user with this retailer's site finishes, this context observation is sent to Norha's PCS and included into her personal context model thus realizing user-driven web integration. It is important to point out that Sears's shopping application was added by Norha as a context consumer and provider (PWE entity) into her PCS. Context gathering also is realized when the user integrates context PWE entities such as Sears's application or her mobile shopping list, and configures her personal profile with information such as her age and marital status. Automatic context gathering, instinctively driven by the user through common web interactions, improves her QoE since she does not have to

collect meaningful context manually to be used every time she shops different retailer web sites. Through instinctive web interactions users integrate relevant context and web entities into their PCSs to make them available throughout their web experiences.

Context processing. It concerns the identification of implicit contextual facts from explicit contextual data stored in PCSs. Context processing is crucial to optimize user QoE since implicit contextual facts improve the knowledge about the user. As a result, retailers can improve the relevance of their product and service offers. For example, products in Norha's wish list are not explicitly included in her husband shopping list. However, a retailer can take advantage of Norha's wish list and the "married to" relationship status (explicit contextual facts) to suggest Norha's desired products as products her husband could buy.

Context provisioning. It refers to the provisioning of personal context information stored in PCSs to context consumers authorized by the user. Context provisioning based on PCSs improves QoE because web applications can take advantage of the knowledge about the user accumulated along the user's entire web experience, rather than the knowledge gathered only from the interactions of the user with a particular web application. For example, when Norha arrives at West Edmonton Mall, the retailers that Norha integrated into her PCS can suggest relevant deals available at the mall, by taking advantage of her preferences gathered from her interactions with different shopping sites.

Context disposal. It refers to the deletion, temporal or permanent depending on the context type, of personal context from PCSs. PCSs enable users to dispose of context information by two main mechanisms: directly through their PCS's user interface, or instinctively through web interactions. An example of instinctively disposing is the deletion of a product category from the user's Sears wish list. Given the perishable nature of context information, context disposal is a critical requirement for improving user QoE through personal context. For example, the user's shopping experience will be negatively affected if she receives irrelevant product offers due to stale context information.

Figure 3.1 illustrates the role of the user's PCS in user-driven context management. Label 1 represents the user in a shopping moc. Label 2 represents PWE entities that have been integrated by the user into her PCS, retailer applications in this case. Label 3 represents the user interface that allows the user to interact directly with the personal context in her PCS. Label 4 represents the user's PCS. The smaller circles in Label 4 illustrate context PWE entities integrated by the user such as her

mobile shopping list or favorite retailers. PCSs allow the user to manage her context information through two mechanisms, *direct* context management and *instinctive* context management. Direct context management is realized through the PCS's user interface (cf. Label 3) and allows the user to provide, modify, and dispose of personal context interacting directly with her PCS. Instinctive context management is realized through common web interactions of the user with web applications (cf. Label 2). For example, when the user adds products into her wish list, these product categories are fed into her PCS as new contextual facts that describe her shopping desires.

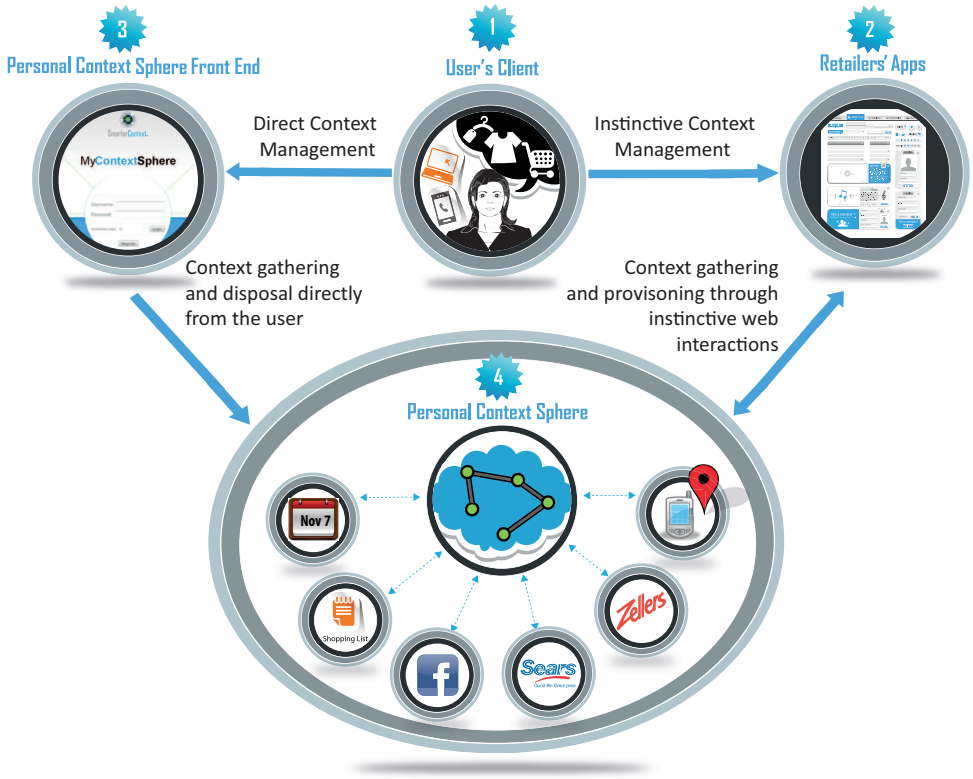


Figure 3.1: Realizing user-driven context management with personal context spheres

3.2 Realizing the Visions of the Smart Internet and the Personal Web

The personal context sphere (PCS) is the foundational concept that allows us to face many of the challenges stated by the smart internet and the personal web (PW). This section explains how we address these challenges in both cases.

3.2.1 Personal Context Spheres and the Smart Internet

Addressing the Smart Internet's Principles

The first principle of the smart internet refers to the need for *a user-centric model for instinctive interaction* (cf. Section 2.1.1). This is precisely what a PCS is, *a user-centric model for context management through instinctive interactions*. A PCS is a personal context model that specifies the context information that is relevant to the user, and assists users in the management of personal context through instinctive interactions along their entire web experience. Our PCS concept addresses the need for a user-centric model in the smart internet because (i) the user is the central context entity. Moreover, the relationships among relevant context entities and the user are specified explicitly; (ii) the user has full control over the context information stored in her/his PCS; and (iii) the delivery of content based on the user's situation by web applications is enhanced with the personal context provided from the user's PCS. PCSs are based on instinctive interactions because context information is mainly gathered from the common interactions between the user and context providers (e.g., web applications). That is, the user is not obligated to do extra work for the specification of personal context. Furthermore, through common web interactions the user is completely empowered to decide on the relevance of context information.

The second principle of the smart internet concerns the need for a *session concept that focuses on users and their mocs*. A PCS can be seen as a user "super-session" that goes beyond a particular set of interactions of a user with a specific application. In contrast to current web models where personal information is only exploited by the application that obtains it, PCSs implement a context exploitation mechanism that traverses different applications along the user's entire web experience. That is, context gathering and provisioning are asynchronous and involve all the applications authorized by the user rather than only one. Moreover, PCSs are also persistence mechanisms that keep valid context available to be used in future interactions.

The last principle of the smart internet refers to the need for improving *collaboration and collectivity* focusing on two main implications, *dynamic social binding* and *collective intelligence*. PCSs support both of these implications through the integration of multiple context spheres. For example, in the scenario presented in Section 3.1.1 the user integrated her husband's PCS application into her context sphere. As a result, her PCS is socially bound to her husband's PCS. Because of this social binding, the context information in both PCSs can be correlated to im-

prove the knowledge about the context situations of these two users, individually and collectively. For example, family shopping could be more efficient when analyzing shopping lists collectively.

Realizing Smart Interactions

In the research background of the smart internet we presented the term *smart interaction*, how it concerns the user model for the smart internet, and its five categories of research challenges namely *metaphors, task simplification, cognitive support, adaptive aggregation,* and *dynamic collaboration*. Our PCS approach contributes to the realization of all these challenges. With respect to the main metaphor of the smart internet, *matter of concern (moc)*, the PCS is a concrete realization of this metaphor since it relates user personal context to the activities the user performs with smart internet applications. Regarding task simplification, PCSs not only simplify user-driven context management, but also the tasks the user performs along her/his web experience by providing the context information that applications require to improve their situation-awareness. Concerning cognitive support, the personal context stored in users PCSs is processed to improve the knowledge about the user. In this way, our SmarterContext solution simplifies user web experience by releasing the user from thinking about the relevance of web content and services. Adaptive aggregation is addressed by providing meaningful personal context information from the users' PCSs to smart internet applications. With this, interactions can be tailored according the user's needs. Finally, dynamic collaboration is improved when smart internet applications take advantage of context information correlated from multiple PCSs.

3.2.2 Personal Context Spheres and the Personal Web

In Section 2.1.4 we presented the conceptual and technical principles of the PW. In this subsection we explain how our PCS approach addresses PW's conceptual principles, as the conceptual model that characterizes the situation of the user.

The first conceptual principle is *the user as the center of web integration*. PCSs are the "center of gravity" of user-driven web integration based on context management. Indeed, PCSs support the vision of the PW by allowing the integration of personal contextual data resulting from user web interactions into a repository that belongs to the user, is fully controlled by its owner, and is available to be exploited in future interactions. The second principle in this group is *the user's context shapes the scope*

and semantics of web integration. The semantics of personal context stored in context spheres allows us to understand user mocs. Therefore, content and services delivered to users by consumers of this context information should be focused on the user's situation. The third principle, *the user controls web integration*, also was a key driver in our PCS approach. The integration of context information into the user's context sphere is completely driven by the user through either direct interactions with the PCS, or instinctive interactions offered by smart internet applications (cf. Figure 3.1). The fourth principle is that *the web works on behalf of the user*. This is one of the main goals of context management using PCSs: to relax the cognitive load of users. For this, we empower the user to control the context management process while our SmarterContext solution does the computational work required to manage the life cycle of context information. The last conceptual principle is that *the PW is social*. Our approach exploits social relationships to improve the knowledge about users. For example, by recommending products to a user based on the shopping or wish list of his/her spouse, or based on the preferences of his/her friends and taking into account special dates and events.

3.3 User-controlled Data Privacy and Security

The smart internet relies on the exploitation of context information obtained from interactions of users with web applications. Since context-awareness leads to privacy problems, a critical aspect for the success of the smart internet is the adoption of mechanisms that guarantee the protection of user's sensitive information, while still leveraging its value to provide users with more situation-aware web experiences. This section analyzes key features of our solution that empower users with privacy and data security control for the access to their information stored in PCSs.

Our solution to context management guarantees privacy and security for personal information stored in PCSs using user policies. The level of protection depends on the context types from which the information is derived. From the perspective of privacy and security, we classify personal context information as *sensitive*, context information that must be encrypted along its life cycle, and *non-sensitive* context. For example, in our e-commerce case study sensitive context corresponds to credit cards, user identities, personal agendas and current and preferred locations. Instances of non-sensitive contextual data could be user preferences about product and service categories. Since users are the main controllers of PCSs, our solution supports them

in specifying the information they want to share and the corresponding third parties with which this information can be exchanged.

With our solution based on PCSs we realize the vision of exploiting information about users, gathered from their past and present interactions with web applications, to support smart internet applications in delivering more pleasant and effective user experiences. Most importantly, for a particular person, SmarterContext only shares context information with the applications and services integrated into that user's PCS. For example, when Norha created her PCS, she provided her preferred shipping and billing addresses using the PCS's user interface.

In our solution users are able to configure privacy and security levels selectively. For example, Norha can approve the disclosure of her credit card information with one of the shopping web sites in her context sphere only. Thus, our solution guarantees that the user's credit card information will not be shared with any other context consumer. Moreover, given the sensitive nature of this information, Norha's credit card data are protected against unauthorized access by third parties both during their provision to the authorized consumers, and while stored in her PCS.

PCSs have the potential to trigger radical changes in existing e-commerce models. For example, information about users stored in PCSs can also be exploited to improve the accuracy of results in business analytics. Of course, without compromising the privacy and security of sensitive personal data. To exploit the value of non-sensitive data, our solution implements partial encryption and non-sensitive context sharing in an anonymous way.

The key features implemented in our SmarterContext solution to guarantee privacy and security for personal context stored in PCSs are the following:

1. *Dynamic selectivity*: SmarterContext exchanges selected personal context information only with PWE entities integrated into PCSs. Moreover, it supports dynamic changes in the list of PWE entities authorized by the user. To exchange non-sensitive information with third parties not included in the user's PCS, the SmarterContext engine disassociates the user identity from this information to guarantee its anonymity.

2. *Dynamic granularity:* Since the user may decide to share her context information partially, SmarterContext allows the user to configure different levels of granularity (e.g., in the form of policies) for the access to her personal context information. Moreover, variations in granularity levels is also

supported.

3. *Partial encryption:* Sensitive personal information is encrypted when stored, gathered, and provisioned. However, for SmarterContext to exploit us much as possible context information stored in PCSs, only sensitive data is encrypted according to the specified policies. This is because the inference of context information relies on the semantic analysis of context data represented by URIs stored in RDF graphs. Therefore, encrypted URIs provide no semantics that can be interpreted by SmarterContext.

Concerning the first feature (dynamic selectivity), when the user integrates PWE entities into her PCS, she specifies whether SmarterContext is allowed to either gather and send, only send, or only gather context information from and to the corresponding third party. In this sense, data security in SmarterContext guarantees that sensitive personal data (i.e., any datum associated with the identity of the user) is available only to PWE entities or other PCSs that have been authorized by the user. Regarding the second feature (dynamic granularity), in SmarterContext the user may decide to share some types of information only. Moreover, these types can vary from one third party to another. Finally, concerning the third feature (partial encryption), since the SmarterContext infrastructure (including PCS repositories) is maintained by cloud providers (also third parties), data protection is guaranteed even in cases where this information is accessed by third parties not explicitly authorized. In such cases, the information will remain inaccessible because it is encrypted. Finally, PCSs are encrypted partially for SmarterContext to reason about non-sensitive context information and exploit it, even with third parties that have not been integrated into the user's PCS.

3.4 Chapter Summary

The *personal context sphere (PCS)* constitutes the first contribution of this dissertation. It allows us to advance towards the accomplishment of our context-awareness research goals *G1-G3*, and to answer research question *Q2* (cf. Figure 1.4). PCSs provide the conceptual model, centered on and controlled by the user, required to represent and manage context information about web entities relevant to the user's tasks in a particular application domain in the smart internet and the personal web (PW). This chapter presented the PCS and how it (i) constitutes a user-centric model of

personal context information to improve situation-awareness in interactions and services; (ii) empowers users to manage context and web integration through instinctive interactions supported by these models; and (iii) provides a new concept of session, based on the matter of concern (moc) and personal web sphere metaphors, that focuses on users and is available across different servers. These three aspects constitute the summary of the research challenges posed by the smart internet and the PW. Our PCS approach provides a sound solution to these challenges (cf. Section 2.7). For this, we conceived SmarterContext with two main features. First, it keeps track of past and present user web interactions, with any web application, to provide sellers with relevant context information about users. Thus, in contrast to existing solutions, the information gathered from the interactions with a seller's platform is exploited not only by this seller, but also by any other seller authorized by the user. Second, SmarterContext supports users in controlling the access to their personal information, and provides them with privacy and security guarantees for sharing context information that sellers can use to improve the relevance of product and service offers. Therefore, in contrast to existing e-commerce applications, the user is the one who decides about the information to be shared and the third parties authorized to access this information.

The next chapter presents the SmarterContext ontology, our approach to represent context information, in particular, personal context data stored in PCSs.

Chapter 4

The SmarterContext Ontology

This chapter presents the second contribution of this dissertation, the SmarterContext ontology. This ontology constitutes our solution to context representation. The advantage of using ontologies for knowledge representation, context in this research, is that they provide explicit descriptions of the concepts and the relationships among them that characterize a domain. Moreover, having being developed for a general domain, they always can be particularized to be applied in more concrete domains. Therefore, our ontology not only characterizes general context types, but also can be extended to be applied in any domain of situation-aware smart software (SASS) systems, including the smart internet.

To explain our ontology, this chapter presents in Section 4.1 methodological aspects of the ontology, as well as the requirements that drove us in its design. Section 4.2 explains the entities and properties defined in our ontology's foundational module, the *general context* taxonomy. Section 4.3 explains how the ontology can be applied to particular problem domains and illustrates its application to the personal web, in particular to our situation-aware smarter shopping case study. Finally, Section 4.4 summarizes the chapter.

4.1 Design Methodology and Requirements

We designed the SmarterContext ontology to leverage interoperability and to model relationships among context entities which include users and systems. Our ontology exploits linked data (LD) [BHBL09] and OWL-Lite [W3C04d] to represent, share, and reason about context information. Table 4.1 summarizes the application

of the LD's principles proposed by Berners-Lee (cf. Section 2.3.1) in our SmarterContext ontology.

Table 4.1: Linked data principles applied to our SmarterContext ontology

Linked Data Principles [BL06]	Application in SmarterContext
Use URIs as names for things.	URIs identify context types, context entities, context relationships (context things).
Use HTTP URIs so that people can look up things' names.	Context types and context entities are discoverable in the web by people and systems using URIs.
Provide useful information about things identified by URIs in a standardized way (RDF, SPARQL).	Context repositories, known as personal context spheres (or repositories of context things), can be queried using RDF tools and SPARQL endpoints.
Include links to other URIs, so that they can discover more things.	Smart interactions allow the discovery of new context entities thus feeding context spheres with new contextual facts.

With our SmarterContext ontology and our personal context sphere (PCS) contribution presented in the previous chapter we leverage the semantic web (SW) potential by defining the semantics required for exploiting user-centric context information in SASS systems' application domains. Besides taking advantage of the data sharing and interoperability capabilities provided by the SW, our contributions enable SASS systems with the context management required to support situation-aware user-centric functionalities. Our ontology can be extended by either creating further layers in its hierarchical structure, or integrating existing domain-specific vocabularies (ontologies) into its layers. According to our systematic literature review on context awareness, the SmarterContext ontology is the only existing context representation mechanism that defines a common framework to specify context models applicable in any domain. Moreover, SmarterContext was designed to represent dynamic context rather than static context only.

4.1.1 Methodological Aspects

The genesis of our dynamic context management solution, and in particular of the SmarterContext ontology, was our comprehensive survey on context modeling

and context management approaches in different problem domains of context-aware computing (cf. Section 2.2 and Appendix A). From this study, we proposed a general classification of context information. This general classification, depicted in Figure 2.1, allowed us to define our *General Context (GC)* taxonomy, which constitutes the fundamental module of the SmarterContext ontology.

4.1.2 Requirements Analysis

We designed the SmarterContext ontology with the goal of providing a context representation mechanism based on self-adaptive context models that support, at runtime, changes in the specification of dynamic context entities, the relationships among them, and corresponding monitoring requirements. In light of this and based on the key features for context modeling obtained from our systematic literature review, we identified a list of requirements for context representation in SASS systems. Our context representation solution addresses these requirements, which are summarized as follows:

R-1. Context representations must be interoperable to support the exchange of context information between context management infrastructures and both context consumers and providers.

R-2. Context models must support the representation of context entities, the relationships among them, the properties that characterize their situation, and the relationships between these entities and the object of context awareness (i.e., users and SASS systems).

R-3. Context models must support timeliness. That is, the representation of past, present and future situations.

R-4. Context representations must be adaptable at runtime according to changes in the situation of users and systems. That is, context entities and monitoring requirements may appear, disappear or be modified dynamically without affecting the effectiveness of the context management mechanism.

R-5. Context models must be flexible to support context representation according to the semantics of a particular domain.

Figure 2.5 in our background section on context awareness presents identified features for representing context entities and their situations. Based on these features, we analyzed different approaches to context representation and reasoning (i.e.,

key-value pairs, markup models, object-oriented models, meta-model-based models, logic-based models, ontology-based models, and fact-based models—cf. Section 2.2.3). Based on this analysis and the requirements listed above, we concluded that ontologies and logic-based models, as used in the semantic web, are suitable approaches to context modeling. However, we found no ontological approach that supports context representation as specified by this set of requirements, thus we developed the SmarterContext ontology.

Regarding requirement R-1, the knowledge represented in a semantic format, such as the one provided by the SmarterContext ontology, is better suited for interoperability between context providers and consumers. Concerning requirement R-2, RDFS and OWL-Lite provide sufficient expressiveness to characterize context types with corresponding properties, and to represent context relationships, constraints and granularity levels. Concerning requirements R-3 and R-4, context models based on RDF graphs support the representation of contextual data over time, and can be modified at runtime to add or delete contextual facts according to changes in context situations [VMMn+11, VMT11b]. Regarding requirement R-5, the extensibility capabilities of the SmarterContext ontology for the specification of both, entities and reasoning rules, provide a sound mechanism to tailor context specifications according to the semantics and context management concerns of concrete application domains.

4.1.3 Extensibility and Modularity

Modularization, as in many other domains, is a best practice in ontology design [HKR09]. The increasing size and complexity of context models require collaborative design. Moreover, the design of loosely coupled ontologies facilitates their processing, maintenance and evolution. Modular ontologies also benefit privacy and security requirements since it is easier to control the level of exposure of sensible data [MTVM12].

We designed SmarterContext as a modular and extensible ontology. Its foundational module, *general context (GC)*, enables context representation and reasoning for any problem domain related to SASS systems. Because of its modular structure, the SmarterContext ontology supports vertical and horizontal extensibility. *Vertical extensibility* makes the SmarterContext ontology applicable to different problem domains. It is realized by defining more specialized modules that inherit from the GC module or other modules derived from GC. The application of the

SmarterContext ontology to a particular domain may imply the definition of several hierarchical levels. For example, to support context-awareness in the personal web (PW) we derived from GC the *personal web context (PWC)* module. The PWC module supports context representation and reasoning in any problem domain of the PW. To apply SmarterContext to a particular application of the PW, we recommend to extend the PWC module further by defining more particular context types and context reasoning rules according to the specific domain. For example, we derived from PWC the *shopping* module to support context representation and reasoning in our smarter commerce case study [VMMn+11]. *Horizontal extensibility* is realized by importing existing ontologies or vocabularies into any of the modules defined in the SmarterContext ontology.

Several ontologies are available for representing things in the SW. Examples of these ontologies are FOAF[1] (friend-of-a-friend), the ontology to connect people on the SW [GCB07]; GoodRelations,[2] the ontology for describing product and service offers on the web [Hep08]; and GeoNames, the ontology that adds geospatial semantic information to the web.[3] In contrast to existing ontologies, SmarterContext provides the common framework required by the smart internet to augment the semantics of existing ontologies to make them suitable for context representation. Our SmarterContext ontology provides the knowledge representation mechanism required to elevate the visibility of context information as demanded by SASS systems.

Without SmarterContext, existing SW ontologies are useful to characterize and reason about entities in concrete application domains with no awareness of users' and systems' situations. For example, FOAF supports the representation of social relationships, GoodRelations the representation of online product and service offers, the Google Product taxonomy the representation of products, and GeoNames provides information about geographical places. The integration of these vocabularies into SmarterContext, which can be realized through either vertical or horizontal extensibility, instantaneously augments their semantics by converting their concepts into context entity types that relate to each other to describe information about the user's situation. Therefore, these ontologies would represent not only web resources independent of the user, but relevant context about the user's situation. For example, FOAF would represent social context relationships that could be exploited to discover

[1] http://www.foaf-project.org/
[2] http://www.heppnetz.de/projects/goodrelations/
[3] http://www.geonames.org/ontology/documentation.html

shopping preferences from the user's social network. GoodRelations and the Google Product taxonomy would describe not only products and services, but also the interactions between shoppers and online offers, thus enabling innovative approaches to leverage web interactions in business intelligence (BI) applications. Finally, the GeoNames ontology would represent not only places in the world, but geographical locations meaningful to improve the user's web experience.

Table 4.2 provides the list of namespaces and corresponding prefixes (abbreviations of URIs) that identify the schemas and ontologies used in our situation-aware smarter shopping case study. When applicable, we indicate whether the ontology or vocabulary extends the SmarterContext ontology horizontally or vertically (cf. column labeled as V/H). Protégé [KFNM04], the tool we use to create and edit ontologies in our research, can easily be used to visualize the ontologies described in this chapter.

Table 4.2: RDF and OWL schemas, and ontologies used in the smarter commerce case study.

Prefix	Namespace	V/H
gc:	smartercontext.org/vocabularies/gc/v6.0/gc.owl#	
pwc:	smartercontext.org/vocabularies/pwc/v6.0/pwc.owl#	V
shopping:	smartercontext.org/vocabularies/shopping/v6.0/shopping.owl#	V
geo:	smartercontext.org/vocabularies/rdfv2/geo.rdf#	H
google:	smartercontext.org/vocabularies/rdfv2/googleproducts.rdf#	H
deals:	smartercontext.org/vocabularies/rdfv2/dealcategories.owl#	H
xsd:	www.w3.org/2001/XMLSchema#	-
rdf:	www.w3.org/1999/02/22-rdf-syntax-ns#	-
rdfs:	www.w3.org/2000/01/rdf-schema#	-
owl:	www.w3.org/2002/07/owl#	-

4.2 The Foundational Module: General Context

The *general context (GC)* module defines context types (classes), abstract properties (relationships between classes), and concrete properties (links between attributes of individuals and their corresponding values) applicable to any problem domain in SASS

systems, for instance the smart internet. For actual schemas and ontologies used in our smarter commerce case study please refer to Table 4.2.

Context Entities in the GC Module

Table 4.3 below details the entities defined in the GC module. For each class, the table presents the corresponding context entity type (column *Entity*), the class from which the entity is derived (column *Superclass*), and a description of the role that the entity plays in the SmarterContext ontology (column *Description*). This column schema is used to describe the class types of the ontologies presented in the subsequent sections of this chapter.

Object Properties in the GC Module

The representation of context relationships is a crucial feature for context reasoning. Table 4.4 presents the object properties (abstract properties) defined in the GC module. GC's object properties constitute context relationships between context entities defined in the GC module or in any of its extensions. For each object property, Table 4.4 presents in column *Domain* the values of the domain, in column *Range* the values of the range, in column *Features* whether the property is transitive (T), functional (F), or symmetric (S), and in column *Inverse Of* the inverse properties. This column schema is used to describe object properties in the subsequent subsections of this chapter.

The `associationRelationship` object property represents aggregation and association context relationships (different than functional and social relations). Both its domain and range (the values for the `rdfs:domain` and `rdfs:range` properties) correspond to entities of the type `contextEntity`. `locationRelationship` includes any object property with range equal to the `LocationContext` type. GC defines no domain for the `LocationContext` property as it may depend on the application domain's semantics. That is, the domain of this property must be defined by the ontology that extends GC in the corresponding application (e.g., in certain domains it makes sense to talk about the location of an artificial entity, however it makes no sense in many other domains). `hostedBy` and `locatedIn` are sub-properties of `locationRelationship`. The value of `hostedBy` in a triple represents a `LocationContext` entity that hosts the `ActivityContext` entity represented by the subject. The value of `locatedIn` represents the location where

Table 4.3: Context entities defined in the SmarterContext ontology' GC module

Entity (Class)	Superclass	Description
ContextEntity	owl:Thing	The superclass of any context type.
ActivityContext	ContextEntity	Actions and tasks performed by an object - e.g., attending a meeting.
IndividualContext	ContextEntity	Anything that can be observed about an isolated object - e.g., the object location.
ArtificialEntity	IndividualContext	Entities resulting from human actions or technical processes - e.g., buildings, hardware and software configurations.
GroupEntity	IndividualContext	Groups of subjects that share common characteristics but not necessarily interact with each other - e.g., the buyers that like the same special offer.
HumanEntity	IndividualContext	Any information about a person's behavior, preferences, characteristics and way of interacting with a system - e.g., an online shopper.
NaturalEntity	IndividualContext	Living and non-living entities which are not the direct result of any human activity - e.g., weather conditions.
LocationContext	ContextEntity	The place of settlement or activity of an object.
PhysicalLocation	LocationContext	A physical place of settlement or activity of an object - e.g., University of Victoria.
GeoLocation	PhysicalLocation	The latitude and altitude that describe a physical location.
VirtualLocation	LocationContext	Location describable by a URI - e.g., the processing node to which a service is deployed.
Endpoint	VirtualLocation	A URI that identifies the location of a computational resource - e.g., the SOAP address of a context sensor exposed as a service.
TimeContext	ContextEntity	Provides context about a specific date and time, but also categorical information such as holidays, working days, and meeting schedules - e.g., Boxing Day.
DefiniteTime	TimeContext	Represents time frames with specific begin and end points - e.g., the duration of a conference.
IndefiniteTime	TimeContext	Expresses a recurrent event which is happening while another situation is taking place. It is not possible to know its duration in advance - e.g., the uptime of a cloud service.

Table 4.4: Object properties defined in the SmarterContext ontology's GC module

Property	Domain	Range	Features	Inverse Of
association Relationship	ContextEntity	ContextEntity	-	-
location Relationship	-	LocationContext	-	-
hostedBy	ActivityContext	LocationContext	-	hosts
locatedIn	IndividualContext LocationContext	LocationContext	T	-
functional Relationship	ContextEntity	ContextEntity	-	-
hosts	LocationContext	ActivityContext	-	hostedBy
social Relationship	GroupEntity HumanEntity	GroupEntity HumanEntity	-	-

the subject (an `IndividualContext` or `LocationContext` entity) is located. The `functionalRelationship` object property refers to information about the usage that an object can make of another. As indicated by its domain and range, functional relationships can exist between any pair of context entities. The value of the `hosts` property, which inherits from `functionalRelationship`, corresponds to a scheduled event that has place in the `LocationContext` entity represented by the subject. Finally, the `socialRelationship` object property emerges from the interrelation between individuals of type `HumanEntity` and `GroupEntity`. Samples of this relational context are affiliations, colleagues, and customers.

Data Properties in the GC Module

Table 4.5 details the data properties (concrete properties) defined in the GC module. Data properties allow the description of context attributes (i.e., characteristics of context entities). All these properties correspond to functional properties, that is, properties that have at most one value. The domain corresponds to the context entity type for which the data property is defined. The range details the valid data types for the values of the properties. In many cases, the range of a data property is restricted to a set of specific values. Such is the case of the `geoLocationClassification` data property (cf. Table 4.5).

Table 4.5: Data properties defined in the SmarterContext ontology's GC module

Property	Domain	Range	Value Description
address	GeoLocation	xsd:string	Corresponds to a String literal that denotes the exact location of a GeoLocation context entity.
endDateTime	DefiniteTime	xsd:dateTime	A dateTime value that denotes the end time of a DefiniteTime context entity (the last value of the time interval).
geoLocation Classification	GeoLocation	"City", "Country", "Neighborhood", "Place", "Region"	Classifies GeoLocation context types. It could be extended or changed according to the domain.
latitude	GeoLocation	xsd:string	The angular distance north or south of the Equator, in degrees, minutes, and seconds of a GeoLocation context entity.
longitude	GeoLocation	xsd:string	The angular distance, in degrees, minutes, and seconds, of GeoLocation context entity east or west of the Prime (Greenwich) Meridian.
startDateTime	DefiniteTime	xsd:dateTime	A dateTime value that denotes the beginning of a DefiniteTime context entity (the initial value of the time interval).
zipCode	GeoLocation	xsd:string	A string value that corresponds to the postal code of the GeoLocation entity represented by the subject.

Horizontal Extension in the GC Module

The elements defined in the GC module can be extended horizontally by importing concrete vocabularies. The goal of the SmarterContext ontology is to provide a general framework for integrating context information rather than defining exhaustively sub-ontologies for every possible context type in any application domain. In our situation-aware smarter shopping case study we extended the GC module by defining a vocabulary to characterize `GeoLocation` entities and the relationships among them

(cf. prefix `geo:` in Table 4.2).

4.3 Application to the Personal Web

This section explains how to extend the SmarterContext ontology, vertically and horizontally, to represent context information in any problem domain of the PW and in particular, in situation-aware smarter shopping.

4.3.1 The Personal Web Context (PWC) Module

The PWC module extends the GC module vertically to define context types, object properties, and data properties required to represent and reason about context information in context-aware applications within the personal web domain. For the prefix and namespace of the PWC module please refer to Table 4.2.

Context Entities in the PWC Module

Table 4.6 details the entities defined in the PWC module. `PWConcern` allows smart interactions and services to understand the nature of user matters of concern (mocs) at a specific time (e.g., whether the user is browsing the web for shopping or social activities). The `PWConcern` entity defines seven categories of personal web concerns: *Academic, Business, Entertainment, Healthcare, Shopping, Social,* and *Travel.* Of course, these categories can be extended further as required.

Object Properties in the PWC Module

Table 4.7 details the object properties (abstract properties) defined in the PWC module. Column *Features* indicates whether the property is transitive (T), functional (F), or symmetric (S). PWC object properties, which extend the object properties defined in the GC module, allow the definition of context relationships between context entities defined in the PWC module or in any of its extensions.

The `concerns` property associates context entities of type `ScheduledEvent, gc:NaturalEntity, gc:ArtificialEntity, gc:LocationContext,` and `gc:GroupEntity` with relevant categories defined as entities of type `PWConcern` (e.g., a personal calendar event associated with a shopping concern). `hasIntegrated` allows the integration of any instance of type `IndividualContext` to the user's

Table 4.6: Context entities defined in the SmarterContext ontology's PWC module

Entity (Class)	Superclass	Description
PWConcern	gc:ActivityContext	Classifies web resources and activities a user performs in the web - e.g., shopping, academic, healthcare.
ScheduledEvent	gc:ActivityContext	A calendar event defined in a personal agenda - e.g., a business trip.
PhysicalEntity	gc:ArtificialEntity	A context entity that is unavailable as a web entity. E.g., the user's preferred currency.
WebResource	gc:ArtificialEntity	Web elements with which the user interacts such as web sites, and web services - e.g., Walmart's shopping site.
PWESite	WebResource	Represents a web site compliant with the SmarterContext framework (i.e., a web site able to interoperate with the SmarterContext infrastructure).
WebEntity	WebResource	Any entity available on the web different than PWE sites and web services - e.g., products or services offered online, friends of the user.
WebService	WebResource	Any web service relevant to the user - e.g., a service for payments with credit cards.
User	gc:HumanEntity	Refers to any information about the user's behavior and preferences - e.g., security profiles, language preferences, and personal information.

context sphere (e.g., a personal agenda application integrated through a web service). `isNearTo` associates two entities of type `gc:GeoLocation` as close to each other (within a short distance). Since property `isNearTo` is symmetric, it applies to both entities. Property `preferredLocation` defines a `gc:GeoLocation` entity as the user's favorite place of settlement. As it is a functional property, each user can have one preferred location at most. `concerns,` and `hasIntegrated` are sub-properties of `gc:associationRelationship`. `isNearTo`, and `preferredLocation` are sub-properties of `gc:locationRelationship`, which is sub-property of `gc:associationRelationship`.

Property `identifiedBy` is useful for identifying context entities of type `WebResource`. The value of this property is an entity of type `gc:Endpoint`.

For example a shopping web site identified by its URL `http://www.amazon.ca/`. `scheduledFor` is used to define the schedule of calendar events. Both `identifiedBy` and `scheduledFor` are functional properties and inherit from `gc:functionalRelationship`. Another PWC object property that extends from `gc:functionalRelationship` is `userInteraction`. This property is absolutely crucial for context integration in the PW since the user is the one who certainly knows about web entities and their relationship with her own situation. User interactions provide the means to identify context entities relevant to the user's situation throughout her web experience. For example, through a simple interaction such as *liking* a product, the user provides the SmarterContext infrastructure with relevant information about her preferences. This product category thus becomes a relevant context entity [VMMn+11]. The current version of the SmarterContext ontology subdivides user interactions defined in the PWC module in the following object properties: `dislikes, isInterestedIn, likes, ranked`, and `tagged`. These interactions are useful to integrate any entity of type `IndividualContext` into the user's context sphere, and are applicable to any application domain within the PW.

Social context relationships (`gc:socialRelationship`) emerge from the interrelation between entities of type `GroupEntity` and `HumanEntity`. The PWC module defines the following object properties to represent social relationships in any application domain of the PW: `affiliatedWith, associates, colleagueOf, engagedTo, friendOf`, and `relativeOf`, which is subdivided in `childOf, marriedTo, parentOf`, and `siblingOf`. `affiliatedWith` and `associates` are inverse.`colleagueOf, engagedTo`, and `friendOf` are symmetric properties and apply between two entities of type `gc:HumanEntity`. `childOf` and `parentOf` are inverse properties.

Data Properties in the PWC Module

Table 4.8 details the data properties (concrete properties) defined in the PWC module. The data properties `birthYear, emailAccount, givenName, hasGender, lastName`, and `preferredLanguage` define attributes of context entities of type `gc:HumanEntity`. `rankingValue` rates any entity of type `gc:IndividualContext`, and is used together with the `ranked` object property. `scheduledEventDescription` and `scheduledEventTittle` describe attributes of calendar events (i.e., instances of type `ScheduledEvent`).

Table 4.7: Object properties defined in the SmarterContext ontology's PWC module

Property	Domain	Range	Features	Inverse Of
concerns	ScheduledEvent gc:NaturalEntity gc:ArtificialEntity gc:LocationContext gc:GroupEntity	PWConcern	-	-
hasIntegrated	User	gc:IndividualContext	-	-
isNearTo	gc:GeoLocation	gc:GeoLocation	S	-
preferredLocation	User	gc:GeoLocation	F	-
identifiedBy	WebResource	Endpoint	F	-
scheduledFor	ScheduledEvent	gc:DefiniteTime	F	-
userInteraction	gc:HumanEntity	gc:IndividualContext	-	-
dislikes	User	gc:IndividualContext	-	-
isInterestedIn	User	gc:IndividualContext	-	-
likes	User	gc:IndividualContext	-	-
ranked	User	gc:IndividualContext	-	-
tagged	User	gc:IndividualContext	-	-
affiliatedWith	gc:HumanEntity	gc:GroupEntity	-	associates
associates	gc:GroupEntity	gc:HumanEntity	-	affiliatedWith
colleagueOf	gc:HumanEntity	gc:HumanEntity	S	-
engagedTo	gc:HumanEntity	gc:HumanEntity	S	-
friendOf	gc:HumanEntity	gc:HumanEntity	S	-
relativeOf	gc:HumanEntity	gc:HumanEntity	S	-
childOf	gc:HumanEntity	gc:HumanEntity	-	parentOf
marriedTo	gc:HumanEntity	gc:HumanEntity	S	-
parentOf	gc:HumanEntity	gc:HumanEntity	-	childOf
siblingOf	gc:HumanEntity	gc:HumanEntity	S	

Table 4.8: Data properties defined in the SmarterContext ontology's PWC module

Property	Domain	Range	Value Description
birthYear	gc:Human Entity	xsd:int	Functional. The year a human entity was born.
emailAccount	gc:Human Entity	xsd:string	An email account associated with the human entity represented by the subject.
givenName	gc:Human Entity	xsd:string	Functional. The given name of the human entity represented by the subject.
hasGender	gc:Human Entity	"Female", "Male", "NotSpecified"	Functional. The gender of the human entity represented by the subject.
lastName	gc:Human Entity	xsd:string	Functional. The last name of the human entity represented by the subject.
preferred Language	gc:Human Entity	xsd:string	The preferred language of the human entity represented by the subject.
rankingValue	gc:Individual Context	xsd:int	Functional. The ranking value assigned by the user to the entity represented by the subject.
scheduledEvent Description	Scheduled Event	xsd:string	Functional. The description of the scheduled event represented by the subject.
scheduledEvent Title	Scheduled Event	xsd:string	Functional. The title of the scheduled event represented by the subject.

4.3.2 Application to Situation-Aware Smarter Shopping

This subsection explains the application of the SmarterContext ontology to our case study on situation-aware smarter shopping, a particular application of the PW. That is, how we derived the *shopping module* from the PWC module. For the prefix and namespace of the Shopping module please refer to Table 4.2. The shopping scenario of our case study corresponds to the one described in Section 3.1.1.

Context Entities in the Shopping Module

Table 4.9 details the classes defined as context entity types in the shopping module.

Table 4.9: Context entities defined in the SmarterContext ontology's Shopping module

Entity (Class)	Superclass	Description
Currency	pwc:Physical Entity	Represents one of the user's preferred currencies - e.g., CAD, USD.
Delivery Method	pwc:Physical Entity	Represents one of the user's preferred delivery methods - e.g., Fedex.
Payment Method	pwc:Physical Entity	Represents one of the user's preferred payment methods - e.g., credit card, PayPal.
ProductService Category	pwc:WebEntity	Denotes a product or a service category offered online - e.g., Clothing, Electronics.

Object Properties in the Shopping Module

Table 4.10 details the types required to represent context relationships in the Shopping module. The `relatedProductOrService` object property, which extends from `gc:associationRelationship`, denotes that two product or service categories are related to each other (e.g., complementary products such as necklaces and earrings). `preferredCurrency, preferredDeliveryMethod`, and `preferredPaymentMethod` inherit from `gc:functionalRelationship` and associate currencies, delivery methods and payment methods to the user. Four new types of user interactions extend `pwc:userInteraction` in the Shopping module. The first one, `doesNotWish`, indicates that the product or service category represented by the object cannot be part of the user's wish list. The second one, `purchased`, allows the SmarterContext infrastructure to identify products the user purchased during her interactions with a particular shopping site. The third one, `toBuy`, indicates that the product or service category represented by the object is in the user's shopping list. Finally, the `wishes` object property represents that the corresponding product or service category was added by the user into her wish list.

Table 4.10: Object properties defined in the SmarterContext ontology's Shopping module

Property	Domain	Range	Features	Inverse Of
relatedProduct orService	ProductService Category	ProductService Category	S	-
preferred Currency	pwc:User	Currency	-	-
preferred DeliveryMethod	pwc:User	DeliveryMethod	-	-
preferred PaymentMethod	pwc:User	PaymentMethod	-	-
doesNotWish	pwc:User	ProductService Category	-	-
purchased	pwc:User	ProductService Category	-	-
toBuy	pwc:User	ProductService Category	-	-
wishes	pwc:User	ProductService Category	-	-

Data Properties in the Shopping Module

Table 4.11 details the data properties (concrete properties) that allow the definition of context attributes for context entities in the Shopping module.

Horizontal Extension in the Shopping Module

The Shopping module of the SmarterContext ontology is extended horizontally by importing two RDF vocabularies that characterize products and services. Both vocabularies extend the `ProductServiceCategory` context type. The first vocabulary corresponds to the *Google Product Taxonomy* [Goo12]. This taxonomy categorizes products in Google's search results. Google provides this taxonomy in two formats, plain text and comma-separated values (CSV). We converted this hierarchical set of product categories into an RDF vocabulary compliant with the semantic web. The prefix and namespace we defined for this vocabulary correspond to `google:` and `http://smartercontext.org/vocabularies/rdf/googleproducts.rdf#`, re-

Table 4.11: Data properties defined in the SmarterContext ontology's Shopping module

Property	Domain	Range	Value Description
billingAddress	Payment Method	xsd:string	Functional. A string that represents the billing address of a PaymentMethod context entity.
cardNumber	Payment Method	xsd:string	Functional. A string that represents the card number of a PaymentMethod context entity.
expirationMonth	Payment Method	xsd:int	Functional. An int that represents the expiration month of a PaymentMethod context entity.
expirationYear	Payment Method	xsd:int	Functional. An int that represents the year in which PaymentMethod context entity expires.
nameOnCard	Payment Method	xsd:string	Functional. A string that represents the card holder's name of a PaymentMethod context entity.
payment MethodType	Payment Method	xsd:string	Functional. A string that represents the type of a payment method - e.g., Visa, Mastercard, PayPal.
targetedFor Gender	Product Service Category	Female, Male, None	Functional. A string that indicates whether the product or service category is intended for a particular gender.
verification Number	Payment Method	xsd:string	Functional. A string that represents the security number of a PaymentMethod context entity.

spectively. The second vocabulary corresponds to the *Groupon Deal Categories* [Gro12]. This taxonomy, available in JSON, XML and CSV formats, provides the complete set of categories used by Groupon to characterize deals offered to users via email. To integrate this taxonomy into the Shopping module of the SmarterContext ontology, we generated it as an RDF vocabulary whose prefix corresponds to `deals:` (cf. Table 4.2).

4.3.3 Context Representation and Personal Context Spheres

Our proposed PCSs are implemented as RDF graphs where resources and predicates (nodes and arcs) are compliant with the types defined in our SmarterContext ontology. In general, these types correspond to classes, object properties and data properties derived from the GC module.

In our situation-aware smarter commerce case study these types are defined in the PWC and Shopping modules, including their horizontal extensions. A partial version of a context sphere that represents the relevant context of user Norha in the shopping scenario described in Sect 3.1.1 is detailed in Appendix B.

4.4 Chapter Summary

This chapter presented the second main contribution of this dissertation: the SmarterContext ontology. This ontology, derived from the findings of our literature review on context awareness, is our solution to context modeling, which exploits linked data and is applicable to any problem domain of SASS systems due to its extensibility features. To the best of our knowledge, our SmarterContext ontology is the only context modeling approach that addresses the five requirements identified as crucial for the specification of context information in SASS systems. These requirements are (i) interoperability, (ii) entities and situations representation, (iii) timeliness, (iv) runtime self-adaptivity, and (v) flexibility. Moreover, we conclude from our survey that our context modeling approach is the only one that satisfies the whole set of features for the representation of context entities and situations discussed in Section 2.2.3. Granularity is supported through the extensible capabilities of our ontology and its data properties. The representation of context types and properties through the definition of class entities, and object and data properties, respectively. Constraint expressiveness is realized through the specification of the domain and range for every object and data property. With this, we verify the consistency of context models. Finally, spatial representation is realized through PCSs

This chapter presented the context types defined in the primary module of the SmarterContext ontology. To illustrate the application of our ontology, the chapter also presented the PWC module, which extends GC and was proposed to support context representation in any domain of the personal web; and the shopping module, which extends PWC and supports context representation in our situation-aware

smarter commerce case study.

The ontologies presented provide the semantics required to reason about context information with our SmarterContext solution. The next chapter presents our SmarterContext reasoning engine and the way it exploits these ontologies to infer implicit contextual facts from explicit data modeled in the form of contextual RDF graphs.

Chapter 5

The SmarterContext Reasoning Engine

This chapter presents our third main contribution, our SmarterContext *reasoning engine*. According to our literature review on context awareness, *context handling* is a fundamental aspect in the management of the context information life cycle (cf. Section 1.2). Furthermore, for software systems to become more situation-aware and smarter, they require context management support not only to gather context from the environment, but also to infer implicit contextual facts from sensed context observations, and reason about this knowledge to deliver content and functionalities according to user situations and system context-driven requirements.

We designed our context reasoning engine, CoRE for short, based on the features for context handling that we identified in our systematic literature review (cf. Figure 2.5), and taking into account the dynamic nature of context in situation-aware smart software (SASS) systems. Our SmarterContext CoRE supports the inference of implicit contextual facts from explicit contextual data compliant with the SmarterContext ontology. Contextual data compliant with SmarterContext correspond to RDF graphs whose nodes and arcs correspond to either types derived from the SmarterContext ontology or values defined in vocabularies compatible with it. We call these RDF graphs *contextual RDF graphs*. A personal context sphere (PCS), the repository of personal context presented in Chapter 3, constitutes a contextual RDF graph.

We developed two versions of our SmarterContext CoRE. The first version exploits RDFS [W3C04c] and OWL-Lite [W3C04d], as well as domain-dependent se-

mantic web (SW) rules to infer contextual facts based on existing descriptive logic reasoning engines such as Pellet [SPG+07a] and the ones supported by the Jena framework [CDD+04]. This first version, although powerful in terms of expressiveness, is challenged when reasoning on large-scale RDF context repositories due to the high computational overhead added by OWL-based inferences. One of the main reasons for this overhead is that type separation is not enforced in OWL [HKR09]. That is, when reasoning with OWL, classes, individuals, and properties are indistinguishable. Therefore, it is possible to use the same identifier as an individual in one statement and as a property in another statement. For example, `owl:DatatypeProperty` is not only equivalent to `rdf:Property`, but also is a subclass of `owl:ObjectProperty`. As a result, reasoning engines based on OWL can infer a considerable number of irrelevant statements, in the sense that they provide no meaningful knowledge, when reasoning about context situations within a particular application domain. For example, having the `pwc:givenName` data type property, an OWL-based engine will infer that `pwc:givenName` is a subclass of `owl:ObjectProperty`. This inference makes no sense in SmarterContext because in our solution object properties refer to context relationships, whereas data type properties refer to attributes of context entities. `pwc:givenName` is not a relationship between context entities, but an attribute of context entities of type `gc:HumanEntity`.

To avoid this unnecessary overhead, we developed a second version of our SmarterContext CoRE. This version relies on RDFS inferences and the identification of structural context patterns (i.e., graph patterns that represent common contextual facts) in contextual RDF graphs. We proposed structural context patterns to express context reasoning rules that replace the OWL-Lite inference functions required in SmarterContext. Moreover, our structural context patterns provide useful templates to assist users in the specification of domain-specific context reasoning rules compliant with the SmarterContext CoRE. We improved the processing time of our CoRE using indices associated with pattern-based predicates to group triples that are meaningful for the reasoning engine. In this way, to infer implicit context triples associated with a particular context type (e.g., `gc:LocationContext`), we have to traverse only the indices that correspond to predicates whose domain and/or range are/is related to location context. This mechanism was expressive enough for supporting context reasoning in the application domains that were the object of research in this dissertation (i.e., situation-aware smarter shopping and dynamic SOA governance).

To exploit contextual data using SmarterContext, our CoRE supports the following general operations:

1. Simple queries using either the API provided by Jena[1] or SPARQL endpoints [SPA08].

2. Inference of implicit contextual facts based on domain-specific context rules supported by RDFS and OWL-Lite. This refers to the first version of our CoRE.

3. Inference of implicit contextual facts based on domain-specific contextual rules supported by RDFS and the SmarterContext structural context patterns. This refers to the second version of our CoRE.

Under changing situations and system goals, the SmarterContext CoRE supports at runtime the addition of new context types and context reasoning rules into the SmarterContext ontology without requiring the manual deployment of context reasoning components. Context reasoning rules can be based on either pure semantic web (RDFS and OWL-Lite), or semantic web (RDFS) combined with our structural context patterns. Furthermore, since we represent monitoring strategies as RDF graphs, changes in monitoring requirements are fed into the context model, and thus processed by our CoRE dynamically, by either adding RDF graphs that represent new monitoring strategies or modifying the existing ones.

This chapter is organized as follows. Section 5.1 formalizes context models in our SmarterContext solution as *contextual RDF graphs* and provides the foundational definitions related to our CoRE. Section 5.2 presents the first version of the SmarterContext CoRE, which is based on RDFS and OWL-Lite inferences. Section 5.3 presents the optimized version of our CoRE, that is, our catalog of context patterns and how the CoRE uses them to reason without requiring OWL-based assertions. Finally, Section 5.4 summarizes and concludes the chapter.

5.1 Runtime Graph-based Models for Dynamic Context Reasoning

Context modeling is a foundational component of the context information life cycle (cf. Figure 1.2). In SASS applications, context models represent relevant data

[1] http://jena.apache.org/documentation/javadoc/jena

about entities, the relationships among them, the properties that characterize their situation, and the relationships between these entities and the user.

Context models in our SmarterContext solution are labeled digraphs [CL05], where vertices represent either web entities that provide relevant context information or the values of this information, and arcs represent the context properties that characterize the situation of these web entities. Labels of arcs are Uniform Resource Identifiers (URIs) [BLFM05] that correspond to context property types defined in our SmarterContext ontology. Labels of vertices are URIs that identify context types defined in our ontology or web entities classified into these types, literals that represent XML values or untyped sequences of characters, or empty labels [VMMn+11, VM13].

Context models in our SmarterContext solution are known as *contextual RDF graphs*, a particular application of RDF graphs, which are a concrete case of labeled directed graphs or digraphs [CL05].

5.1.1 Labeled Digraphs

Definition 1 (Labeled Digraph)**.** *A* digraph or *directed graph G is a pair (V, A), where V is a finite non-empty set of objects called* vertices, and A is a (possibly empty) set of ordered pairs of vertices called *arcs. The vertex set of G is denoted by $V(G)$, and the arc set is denoted by $A(G)$. The digraph G is* labeled if there exists a function $l : (V \cup A) \to L$ for some set L, where $\forall x, x \in (V \cup A)$, the element $l(x) \in L$ is known as the *label of x [TR02].*

5.1.2 Resource Description Framework (RDF) Graphs

RDF is a formal language for describing structured information in a machine-readable way [MM04]. An RDF file describes a digraph, where arcs and, in most cases, vertices are labeled with URIs to distinguish them [W3C04a, HKR09]. Vertices can be labeled with XML values, untyped sequences of characters, or empty labels. Vertices labeled with URIs are known as *RDF resources*, vertices labeled with XML values or untyped strings of characters are known as *literals*, and vertices labeled with empty labels are known as *blank nodes*. Labels of arcs correspond to *RDF properties* also called *RDF predicates*.

Graphs compliant with the RDF specification [MM04, W3C04c] must be composed of at least one arc. Moreover, an RDF graph can be completely described by its set of arcs. The string representation of an arc is known as an *RDF triple*. An RDF

triple is a string representation of an arc and defined as a set of three sequences of characters: the label of the source vertex known as the *subject*, the label of the arc known as the *predicate*, and the label of the target vertex known as the *object*.

Definition 2 (RDF Triple)**.** *Given are an RDF graph G with vertex set $V(G)$ and arc set $A(G)$; an arc $a = (v_s, v_t) \in A(G)$ with $v_s, v_t \in V(G)$; a set of labels $L = \{U \cup B \cup X \cup N\}$ in the form of sequences of characters with U as the subset representing URIs, B as the subset representing blank labels, X as the subset representing XML values, and N as the subset representing sequences of characters. The RDF triple that describes arc a is the 3-tuple $(l(v_s)\ l(a)\ l(v_t))$, where $l(v_s) \in (L - X - N)$ is the label of source vertex v_s, $l(a) \in U$ is the label of arc a, and $l(v_t) \in L$ is the label of target vertex v_t. In an RDF triple, $l(v_s)$ is called the* subject, *$l(a)$* is called the *predicate, and $l(v_t)$ is called the* object.

Definition 3 (RDF Graph)**.** *An* RDF graph is a digraph $G = (V, A)$ where V is a finite non-empty set of vertices called *RDF resources, A is a non-empty set of ordered pairs of vertices called* arcs, and each arc is described by an RDF triple.

5.1.3 Contextual RDF Graphs

Contextual RDF Graphs are models used in SmarterContext to represent, at runtime, information about relevant context entities, the relationships among them, and/or their monitoring strategies. A contextual RDF graph is a digraph, where the set of valid URIs used to describe its vertices and arcs is constrained to the set of URIs that identify the context types defined in the SmarterContext ontology, and the set of URIs that identify any web entity classified into these context types. Every contextual RDF graph can be completely characterized by its RDF triples called *contextual RDF triples*. A contextual RDF triple represents a context fact or context observation. In contextual RDF triples, non-empty labels of source vertices identify either context entities (i.e., web entities that provide relevant context, for example a web application the user interacts with), or context types defined in SmarterContext (e.g., gc:LocationContext). Labels of arcs correspond to predicates defined as context properties (i.e., object or data type properties) in the SmarterContext ontology (e.g., gc:locatedIn). Finally, non-empty labels of target vertices represent values of context properties that correspond to context types defined in the SmarterContext ontology, web entities classified into these context

types (e.g., city Victoria as the location of the user), XML values (e.g., the age of the user), or untyped sequences of characters (e.g., a tag that classifies a product). Blank labels represent blank nodes used to group objects that correspond to the values of multivalued properties (e.g., the preferred location of the user may be described in terms of the city, address, and postal code. A blank node is required to group these attributes as properties that describe the user's preferred location).

Definition 4 (Contextual RDF Triple)**.** *A* contextual RDF triple, also called a *context fact, is an RDF triple where the subset of URI labels $U \in L$ is constrained by the URIs defined in the* SmarterContext *ontology to identify context entity types and context properties, or by the URIs that identify entities classified into any of the types defined in* SmarterContext*, including compatible vocabularies.*

Definition 5 (Contextual RDF Graph)**.** *A* contextual RDF graph, also called a *context model, is an RDF graph where each arc of its arc set is described by a contextual RDF triple. That is, the universe of labels that describe its resources and predicates is constrained by the types of context entities and context properties defined in the* SmarterContext *ontology.*

Definition 6 (Context Entity)**.** *In a contextual RDF graph with V as its vertex set, a* context entity corresponds to the label of any element $v \in V$ where this label is any of the URIs that identifies context types defined in the SmarterContext ontology, including the URIs defined in any other vocabulary compliant with SmarterContext. In terms of contextual RDF triples, a context entity may correspond to either the label of the subject, or the label of the object for those cases where the object is neither an XML value nor an untyped sequence of characters.

The statement "compliant with SmarterContext" refers to types, including their instances, defined in other ontologies that have been integrated into the SmarterContext ontology by inheriting from any of its elements.

Definition 7 (Context Predicate)**.** *In a contextual RDF graph with A as its arc set, a* context predicate, also called a *context property, corresponds to the label of any element $p \in A$ and is any URI that represents either an object property or a data property defined in, or compliant with, the* SmarterContext *ontology. In terms of contextual RDF triples, context predicates correspond to the labels of their arcs.*

5.2 Context Reasoning with RDFS and OWL-Lite

Context reasoning in the first version of our SmarterContext CoRE relied on deduction rules supported by the RDFS specification (cf. Tables 2.1 and 2.2) and a subset of the axioms defined in OWL-Lite (cf. Table 2.3). Besides standard RDFS and OWL-Lite rules, SmarterContext allows the definition of particular reasoning rules according to the problem domain. The definition of domain-dependent reasoning rules is part of the vertical extensibility capabilities of SmarterContext discussed in Section 4.1.3.

Sections 5.2.1 and 5.2.2 below present selected rules that we defined for our situation-aware smarter shopping case study. Using these rules, our SmarterContext CoRE infers contextual facts about the preferences and situations of user Norha in the shopping scenario described in Section 3.1.1. To illustrate the application of these rules we use the partial view of Norha's personal context sphere[2] (PCS) detailed in Table B.1.

5.2.1 RDFS and OWL-Lite Deduction Rules

This subsection illustrates the application of standard RDFS and OWL-Lite axioms to context reasoning with the SmarterContext ontology. Jena[3] is the semantic web platform that supports context reasoning in the first version of our SmarterContext CoRE. Context reasoning rules in Jena are defined as a set of premises, a list of conclusions, and an optional name and optional direction. Each term of a Jena rule corresponds to either a triple pattern, an extended triple pattern, or a call to a built-in function [CDD+04]. In the following rules, question marks denote variables, uppercase letters represent types of the SmarterContext ontology, and lowercase letters represent instances of these types.

Reasoning using RDFS Subclasses

The following deduction rules exploit the semantic characteristics of the `rdfs:subClassOf` object property.

Rule 1. *(?A rdfs:subClassOf ?B), (?v rdf:type ?A) → (?v rdf:type ?B)*

[2] http://smartercontext.org/examples/thesis/norha.rdf
[3] http://jena.sourceforge.net/inference

Example:

(google:Earrings rdfs:subClassOf google:Jewelry),
(sears.rdf#18KGoldEarrings rdf:type google:Earrings) →
(sears.rdf#18KGoldEarrings rdfs:subClassOf google:Jewelry)

Rule 1 enables the inheritance of a resource membership in a class A to the superclasses of A. In the example, since the product category `google:Earrings` is a subclass of the product category `google:Jewelry`, and the concrete product `sears.rdf#18KGoldEarrings` is an instance of `google:Earrings`, then this product also is an instance of `google:Jewelry`. An application of this rule to smarter commerce is the inference of product and service preferences from the interactions of a user with particular products. For example, knowing that user Norha has earrings in her wish list (cf. Triple 19 in Table B.1), it is possible to infer that she would be interested in other jewelry categories.

Rule 2. *(?A rdfs:subClassOf ?B), (?B rdfs:subClassOf ?C) →*
(?A rdfs:subClassOf ?C)

Example:

(google:Earrings rdfs:subClassOf google:Jewelry),
(google:Jewelry rdfs:subClassOf google:Apparel_&_Accessories) →
(google:Earrings rdfs:subClassOf google:Apparel_&_Accessories))

Rule 2 implements the transitivity of the `rdfs:subClassOf` object property. The example illustrates that since `google:Earrings` is a subclass of `google:Jewelry`, and `google:Jewelry` is a subclass of `google:Apparel_&_Accessories`, then `google:Earrings` also is a subclass of `google:Apparel_&_Accessories`. In our shopping scenario, rules 1 and 2 can be combined to infer that the user may be interested in products of the category apparel & accessories, given that the earrings product category is in her wish list.

Reasoning using RDFS Subproperties

The following deduction rules exploit the semantic characteristics of the `rdfs:subPropertyOf` object property.

Rule 3. *(?A rdfs:subPropertyOf ?B), (?v ?A ?y) → (?v ?B ?y)*

Example:

(pwc:marriedTo rdfs:subPropertyOf pwc:relativeOf),
(norha.rdf#norha pwc:marriedTo gabriel.rdf#gabriel) →
(norha.rdf#norha pwc:relativeOf gabriel.rdf#gabriel)

Rule 3 states that any triple with a predicate defined by a property A is also valid for the predicates defined by the superproperties of A. The example illustrates this rule using properties that correspond to family relationships between human entities. Shopping preferences are undeniable affected by the preferences and needs of the shopper's close family. The SmarterContext ontology defines `pwc:marriedTo` as a subproperty of `pwc:relativeOf` (cf. Table 4.7). In the example, since user Norha is married to Gabriel (cf. Triple 15 in Table B.1), the SmarterContext CoRE will infer that Norha is a relative of Gabriel.

Rule 4. *(?A rdfs:subPropertyOf ?B), (?B rdfs:subPropertyOf ?C) → (?A rdfs:subPropertyOf ?C)*

Example:

(pwc:marriedTo rdfs:subPropertyOf pwc:relativeOf),
(pwc:relativeOf rdfs:subPropertyOf gc:socialRelationship) →
(pwc:marriedTo rdfs:subPropertyOf gc:socialRelationship)

Rule 4 implements the transitivity of the `rdfs:subPropertyOf` object property. The example illustrates that since `pwc:marriedTo` is a subproperty of `pwc:relativeOf`, and `pwc:relativeOf` is a subproperty of `gc:socialRelationship`, it is possible to infer that any pair of human entities that are married to each other, are not only relatives of each other, but also are socially related to each other. By combining rules 3 and 4, it is possible to infer from Norha's PCS that a social relationship holds between her and the human entity identified by URI `gabriel.rdf#gabriel`.

Reasoning using RDFS Property Restrictions

The following deduction rules exploit the semantic characteristics of `rdfs:domain` and `rdfs:range`.

Rule 5. *(?A rdfs:domain ?B), (?u ?A ?y) → (?u rdf:type ?A)*

Example:

(shopping:toBuy rdfs:domain pwc:User),
(norha.rdf#norha shopping:toBuy google:Electric_Grills) →
(norha.rdf#norha rdf:type pwc:User)

As presented in Table 2.2, the domain of a property in SmarterContext defines the valid context types for the subjects of the triples in which this property acts as the predicate. Therefore, Rule 5 is useful to infer from a triple, the type of the subject context entity by looking at the domain of the predicate. For example, the SmarterContext ontology defines the context type `pwc:User` as the domain of the property `shopping:toBuy`. Therefore, from the context fact `(norha.rdf#norha shopping:toBuy google:Electric_Grills)` (cf. Triple 13 in Table B.1), the SmarterContext CoRE infers that `norha.rdf#norha` is an entity of type `pwc:User`.

Rule 6. *(?A rdfs:range ?B), (?u ?A ?y) → (?y rdf:type ?B)*

Example:

(shopping:toBuy rdfs:range shopping:ProductServiceCategory),
(norha.rdf#norha shopping:toBuy google:Electric_Grills) →
(google:Electric_Grills rdf:type shopping:ProductServiceCategory)

Rule 6 allows the inference of class memberships for objects of triples. The range of a property in a particular triple defines the valid types for the objects of the triple (cf. Table 2.2). Based on this example it is possible to infer that `google:Electric_Grills` is an entity of type `shopping:ProductServiceCategory`, given that the latter is the range of the `shopping:toBuy` property in the SmarterContext ontology (cf. Table 4.10). For example, rules 5 and 6 are useful in smarter commerce to recommend product or service categories by inferring the types of particular products with which the user has interacted, and applying complementary rules such as rules 1 and 2.

Reasoning using OWL Transitive Properties

The following deduction rules exploit the semantic characteristics of transitive properties in OWL-Lite (cf. `owl:TransitiveProperty` in Table 2.3).

Rule 7. *(?A rdf:Type owl:TransitiveProperty), (?u ?A ?v), (?v ?A ?x) → (?u ?A ?x)*

Example:

(gc:locatedIn rdf:type owl:TransitiveProperty),
(norha.rdf#norha gc:locatedIn http://www.wem.ca),
(http://www.wem.ca gc:locatedIn geo:Edmonton) →
(norha.rdf#norha gc:locatedIn geo:Edmonton)

Rule 7 enables transitivity for any property that is defined as an OWL transitive property. In the example, since `gc:locatedIn` is a transitive property (cf. Table 4.4), Norha is located in West Edmonton Mall (`http://www.wem.ca`), and this mall is located in Edmonton, the SmarterContext CoRE infers from Norha's PCS that she is located in Edmonton. We have demonstrated how location-based contextual facts are crucial to suggest user-centric deals, products, and services in situation-aware smarter commerce applications effectively [EVMT12].

Reasoning using OWL Symmetric Properties

The following deduction rules exploit the semantic characteristics of symmetric properties in OWL-Lite (cf. `owl:SymmetricProperty` in Table 2.3).

Rule 8. *(?A rdf:Type owl:SymmetricProperty), (?u ?A ?v) → (?v ?A ?u)*

Example:

(pwc:colleagueOf rdf:type owl:SymmetricProperty),
(norha.rdf#norha pwc:colleagueOf tatiana.rdf#tatiana) →
(tatiana.rdf#tatiana pwc:colleagueOf norha.rdf#norha)

Symmetric properties state that if the context relationship represented by the property is valid for subject u and object v, it also is valid for v acting as the subject and u as the object of the same relationship. `pwc:colleagueOf` is a symmetric property in the SmarterContext ontology. Therefore, given that in Norha's PCS the human entity `tatiana.rdf#tatiana` is a colleague of Norha (cf. Triple 23 in Table B.1), our SmarterContext CoRE infers that Norha is a colleague of Tatiana.

Reasoning using OWL Inverse Properties

The following deduction rules exploit the semantic characteristics of the `owl:inverseOf` object property (cf. Table 2.3).

Rule 9. *(?A owl:inverseOf ?B), (?u ?A ?v) → (?v ?B ?u)*

Example:

(pwc:parentOf owl:inverseOf pwc:childOf),
(gabriel.rdf#gabriel pwc:parentOf jg.rdf#jg) →
(jg.rdf#jg pwc:childOf gabriel.rdf#gabriel)

An interesting application of inverse properties in smarter commerce, and in SASS systems in general, is the inference of social relationships between context entities. `pwc:parentOf` and `pwc:childOf` are examples of inverse properties defined in the SmarterContext ontology. For example, given the context fact `(gabriel.rdf#gabriel pwc:parentOf jg.rdf#jg)`, it is possible to infer the fact `(jg.rdf#jg pwc:childOf gabriel.rdf#gabriel)`. Particularly in shopping scenarios, the shopping list of a parent could be affected by the shopping list of his kid and vice versa (although the second case is generally less probable than the first one).

5.2.2 SmarterContext Deduction Rules

This subsection presents selected rules that have been defined in our SmarterContext CoRE to extend the standard reasoning capabilities provided by RDFS and OWL-Lite (cf. Sect. 5.2.1). The rules presented in this subsection exemplify the definition of domain-specific context reasoning rules.

Rule 10. ***[pwc:NearTo]***
(?a gc:locationRelationship ?b), (?b pwc:isNearTo ?c) → (?a pwc:isNearTo ?c)

Example:

(norha.rdf#norha pwc:preferredLocation geo:Victoria),
(geo:Victoria pwc:isNearTo geo:Vancouver) →
(norha.rdf#norha pwc:isNearTo geo:Vancouver)

We defined Rule 10 to infer location-based contextual facts for the identification of relevant products, services, and retailers in our situation-aware smarter commerce case study. This rule states that if an entity `a` is related to a location entity `b` by a `gc:locationRelationship`, and entity `b` is near to another location entity `c`, a valid conclusion is that entity `a` is near to `c`. In the example, given that Norha has as her preferred location Victoria (cf. Triple 18 in Table B.1), `pwc:preferredLocation` is a subproperty of `gc:locationRelationship`, Victoria is near to Vancouver,[4] it is possible to infer that Norha is near to Vancouver. As a result, Norha may be interested in products, services, and deals not only available in Victoria but also in Vancouver.

Rule 11. ***[shopping:FamilyShoppingList]***
(?a pwc:relativeOf ?b), (?a shopping:toBuy ?c) → (?b shopping:toBuy ?c)

Example:

(norha.rdf#norha pwc:marriedTo gabriel.rdf#gabriel),
(norha.rdf#norha shopping:toBuy google:Tennis_Shoes) →
(gabriel.rdf#gabriel shopping:toBuy google:Tennis_Shoes)

In our situation-aware smarter shopping case study, Rule 11 is useful to infer products that could be included in the user's shopping list because they are in the shopping lists of the user's relatives and vice versa. According to Norha's PCS, tennis shoes and electric grills are product categories in her shopping list (cf. triples 13 and 22 in Table B.1). Therefore, given that Norha is a relative of Gabriel, these two product categories could be suggested to Gabriel as products that could be included in his shopping list.

Rule 12. ***[shopping:SocialBasedShoppingPreferences]***
(?a gc:socialRelationship ?b), (?a pwc:likes ?c) → (?b pwc:likes ?c)

Example:

(norha.rdf#norha pwc:colleagueOf tatiana.rdf#tatiana),
(norha.rdf#norha pwc:likes google:XBox_360_Consoles) →
(tatiana.rdf#tatiana pwc:likes google:XBox_360_Consoles)

[4] http://smartercontext.org/vocabularies/rdfv2/geo.rdf#

Rule 12 is comparable to Rule 11 but applies to more general social relationships (besides `pwc:relativeOf`) and to user interactions different than `shopping:toBuy` (i.e., `pwc:likes`, `pwc:dislikes`, and `pwc:isInterestedIn`). An example of the application of this rule involves the facts `(norha.rdf#norha pwc:likes google:XBox_360_Consoles)` and `(norha.rdf#norha pwc:colleagueOf tatiana.rdf#tatiana)`. Since Norha and Tatiana are colleagues and possibly share interests and shopping preferences, it would be relevant to offer XBox 360 consoles or similar products to Tatiana.

Rule 13. ***[shopping:RelatedProductsPreferences]***
(?a shopping:relatedProductOrService ?b), (?c pwc:isInterestedIn ?a) $\rightarrow$ *(?c pwc:isInterestedIn ?b)*

Example:

(deals:Gyms_&_Fitness_Centers shopping:relatedProductOrService deals:Health_Clubs),
(norha.rdf#norha pwc:isInterestedIn deals:Gyms_&_Fitness_Centers) $\rightarrow$
(norha.rdf#norha pwc:isInterestedIn deals:Health_Clubs)

Relevant products, services and deals can be recommended by taking into account related products and services. Suitable ontologies for characterizing products, services and deals must support knowledge representation about related products (e.g., complementary products such as earrings and necklaces, or gyms and health clubs). In the example, Norha is interested in deals related to gyms and fitness centers, therefore and taking into account that these deals are related to health clubs, she could be interested in deals related to the `deals:Health_Clubs` category.

5.3 Context Reasoning with Structural Context Patterns

Section 5.2 presented the first version of our SmarterContext CoRE, which relies on standard RDFS and OWL rules, and domain-specific rules based on the entity and property types defined in the SmarterContext ontology.

OWL-based engines may suffer performance degradation when processing large reasoning spaces. This section presents an optimized version of our SmarterContext CoRE, which relies on RDFS and a set of structural context patterns that

substitute OWL-based inferences and provide general templates for the specification of domain-specific context reasoning rules. We defined structural patterns in the form of contextual RDF subgraphs that allow us to replace the OWL-Lite properties used in the first version of our SmarterContext CoRE: `owl:TransitiveProperty`, `owl:SymmetricProperty`, and `owl:inverseOf`. That is, the optimized version of our CoRE uses the standard RDFS reasoning rules of the first version (i.e., rules 1–6 as defined in Section 5.2.1), but replaces rules 7–9 with the context patterns presented in this section. Moreover, our structural context patterns provide general templates for the specification and processing of domain-dependent rules such as the one defined for our situation-aware smarter shopping case study (cf. rules 10–13 in Section 5.2.2). With this, we optimize the performance of our engine and advance towards a usable tool to assist users in the specification of personalized context reasoning rules in SmarterContext.

To realize the optimized version of our SmarterContext CoRE, we defined in our SmarterContext ontology several object property types, which are subtypes of `rdf:Property`. These property types are required for our CoRE to reason about context situations without using OWL-based axioms, and to process domain-specific rules defined by users. Table 5.1 summarizes these types indicating for each case the context pattern that uses the corresponding property and its equivalence with the first version of the SmarterContext CoRE. That is, whether the property type replaces an OWL object property or provides a template for the specification of domain-specific context reasoning rules.

5.3.1 Structural Context Patterns in SmarterContext

Context patterns define graph structures that together with the semantics provided by our SmarterContext ontology allow our SmarterContext CoRE to reason about context situations without using OWL. Each pattern's structure is composed of a set of RDF triples where the semantics of their context predicate types allow the inference of implicit contextual facts from explicit contextual data specified in contextual RDF graphs.

To describe each pattern we proposed a template composed of nine essential elements. Some of these elements are based on the template proposed by Gamma *et al.* to describe design patterns in object-oriented software [GHJV97]. We define the elements of this template as follows:

Table 5.1: SmarterContext properties for supporting context reasoning based on structural patterns.

SmarterContext Property	Context Pattern	Equivalence with CoRE V1
gc:transitivePredicate	Transitivity	owl:TransitiveProperty
gc:symmetricPredicate	Symmetry	owl:SymmetricProperty
gc:isInverseOf	Inverse	owl:inverseOf
gc:joinerPredicate gc:joinedPredicate gc:isJoinedBy	Join	Template (e.g. Rule 10)
gc:generalizablePredicate gc:isGeneralizableFor	Generalization	Template (e.g. Rule 13)
gc:delegablePredicate gc:delegationPredicate gc:isDelegableFor	Delegation	Template (e.g. Rules 11 and 12)

1. *Name:* The identifier of the context pattern.

2. *Intent:* The purpose of the context pattern. That is, how the pattern is used to derive implicit contextual facts from explicit contextual data specified in contextual RDF graphs.

3. *Motivation:* A scenario that illustrates the usefulness of the context pattern. This motivating section is used throughout the pattern description to explain the abstract concepts of the context pattern.

4. *Preconditions:* Prerequisites for the application of the pattern. These preconditions are associated with context predicates (i.e., arc types) defined in the SmarterContext ontology.[5]

5. *Graph Structure:* The graph-based representation of the context pattern.

6. *Definition:* The formal statement of the pattern.

[5]Preconditions can be verified online at http://smartercontext.org/vocabularies/gc/v6.0/gc.owl, http://smartercontext.org/vocabularies/pwc/v6.0/pwc.owl, and http://smartercontext.org/vocabularies/shopping/v6.0/shopping.owl, depending on the SmarterContext ontology module.

7. *Applicability:* An abstract description of the kinds of contextual facts that can be derived from the pattern, and the types of questions about context situations that the pattern helps answer.

8. *Pattern-based Reasoning Functions:* The reasoning methods defined in the SmarterContext CoRE to derive contextual facts based on the corresponding context pattern.

9. *Example:* An illustration of the context pattern application.

In the following pattern descriptions the symbol $\models$ denotes semantic consequence. $G[T]$ denotes that a contextual RDF triple T is an element in the set of triples that defines the contextual graph G. Graphs with white nodes correspond to pattern structures, whereas graphs with gray nodes represent pattern instances. Furthermore, according to definitions 6 and 7 in Section 5.1.3, context entities and context predicates correspond to the labels of vertices and arcs, respectively. Nevertheless, for clarity, pattern descriptions in this section may treat context entities as vertices, and context predicates as arcs.

Transitivity Pattern

Intent: An instance of the Transitivity pattern represents an implicit context fact (i.e., implicit contextual RDF triple) derivable from two explicit contextual RDF triples connected through a common context entity acting as "transitive entity," and that share a common context predicate known as the "transitive predicate" (cf. Figure 5.1). As the transitive predicate applies between the subject of the first triple and the transitive entity, as well as between the transitive entity and the object of the second triple, hence, the transitive predicate applies between the subject of the first triple and the object of the second one. The subject, predicate, and object of the derived triple correspond to the subject of the first explicit triple, the transitive predicate, and the object of the second explicit triple, respectively.

Motivation: Figure 5.1 presents a contextual RDF graph with two explicit contextual facts, "Norha is located in Victoria" and "Victoria is located in Vancouver Island." The context predicate `gc:locatedIn` is defined as a transitive predicate in our SmarterContext ontology. Since `gc:locatedIn` applies between `norha.rdf#norha` and `geo:Victoria`, and between `geo:Victoria` and

`geo:VancouverIsland`, hence, this predicate applies between `norha.rdf#norha` and `geo:VancouverIsland`. This means that the context fact "Norha is located in Vancouver Island" is a semantic consequence from this instance of the Transitivity pattern (cf. the triple associated with the dashed arc in Figure 5.1).

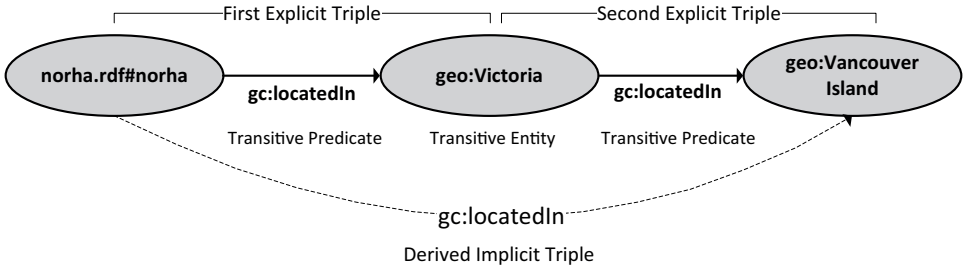


Figure 5.1: Motivating the Transitivity pattern

Preconditions: The common predicate must be a *transitive predicate* in the SmarterContext ontology.

The context model presented in Figure 5.1 is an instance of the Transitivity pattern, given that `gc:locatedIn` is defined as a transitive predicate in the general context (GC) module of the SmarterContext ontology. This is, the triple *(gc:locatedIn rdfs:subPropertyOf gc:transitivePredicate)* does exist in the SmarterContext ontology, concretely in the GC module.

Definition 8 (Transitive Predicate)**.** *Let l(p) be a context predicate defined in, or compliant with the* SmarterContext *ontology. l(p) is a "transitive predicate" iff it is an object subproperty, as defined in RDFS [W3C04c], of the gc:transitivePredicate* SmarterContext *object property type. That is, the triple (l(p) rdfs:subPropertyOf gc:transitivePredicate) exists in the ontology that defines l(p).*

The common predicate *gc:locatedIn* in the context model presented in Figure 5.1 is a transitive predicate. Thus, the contextual RDF triple *(gc:locatedIn rdfs:subPropertyOf gc:transitivePredicate)* exists in the general context (GC) taxonomy of the SmarterContext ontology.

Graph Structure: Figure 5.2 presents the Transitivity pattern's graph structure.

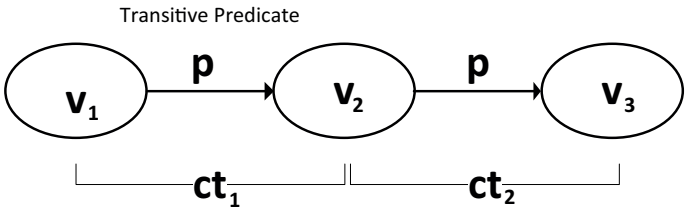


Figure 5.2: Transitivity Pattern's graph structure

Pattern Definition:

Definition 9 (Transitivity Pattern)**.** *Let p be an arc labeled with a transitive predicate $l(p)$; G a contextual RDF graph; and v_1, v_2, v_3 labeled vertices in G, where $l(v_1) \neq l(v_2) \neq l(v_3)$ correspond to URIs of either context entities or context types. The* Transitivity pattern is defined as the subset of contextual RDF triples $\{ct_1, ct_2\} \in G$, such that $ct_1 = (l(v_1)\ l(p)\ l(v_2))$ and $ct_2 = (l(v_2), l(p), l(v_3))$ (cf. Figure 5.2).

The context model presented in Figure 5.1 corresponds to an instance of the Transitive pattern with *l(p)=gc:locatedIn, ct_1=(norha.rdf#norha gc:locatedIn geo:Victoria) and ct_2=(geo:Victoria gc:locatedIn geo:VancouverIsland).*

Applicability: The Transitivity pattern is useful to derive implicit contextual facts from a transitive relationship between two explicit contextual RDF triples. That is, the object of the first triple is the subject of the second one, and both arcs correspond to the same context predicate (a transitive predicate). The subject, predicate, and object of the derived context fact correspond to the subject of the first explicit triple, the transitive predicate, and the object of the second explicit triple (cf. Figure 5.2). Formally, $G[(l(v_1)\ \ l(p)\ \ l(v_2)); (l(v_2)\ \ l(p)\ \ l(v_3))] \models (l(v_1)\ \ l(p)\ \ l(v_3))$, iff triple $(l(p)\ \ rdfs:subPropertyOf\ \ gc:transitivePredicate)$ is defined in the SmarterContext ontology.

Applying the Transitivity pattern to the model presented in Figure 5.1, the context fact "Norha is located in Vancouver Island" is a semantic consequence derived from the contextual RDF triples "Norha is located in Victoria" and "Victoria is located in Vancouver Island."

Pattern-based Reasoning Functions:

$ExistsTransitivity(l(v_1), l(p), l(v_2), G)$**.** Given the labels of two vertices, $l(v_1)$ and $l(v_2)$, a transitive predicate $l(p)$, and a context model G, this function answers

whether there is a path in G from v_1 to v_2, such that every arc's label of the path corresponds to $l(p)$. That is, the *ExistsTransitivity* function checks whether the implicit contextual RDF triple $(l(v_1)\ \ l(p)\ \ l(v_2))$ is derivable from G.

For example, the application of the $ExistsTransitivity(l(v_1), l(p), l(v_2), G)$ function with G as the graph presented in Figure 5.1, $l(v_1)$=*norha.rdf#norha,* *l(p)=gc:locatedIn,* and $l(v_2)$=*geo:VancouverIsland* returns `true`, since `gc:locatedIn` is a transitive predicate and the *(norha.rdf#norha gc:locatedIn geo:VancouverIsland)* triple is derivable from the two explicit triples stated in the model (cf. the triple associated with the dashed arrow in Figure 5.1).

$FindTransitivityClosure(l(v), l(p), G)$**.** Given a vertex's label $l(v)$, a transitive predicate $l(p)$, and a context model G, this function finds the longest simple path in G, such that v is the origin vertex and $l(p)$ is the label of every arc, and returns the set of vertices' labels in the path.

The application of the $FindTransitivityClosure(l(v), l(p), G)$ function with $l(v) = norha.rdf\#norha$, $l(p) = gc{:}locatedIn$, and G as the context model presented in Figure 5.3 returns the vertices' labels `geo:Victoria`, `geo:VancouverIsland`, `geo:BritishColumbia`, and `geo:Canada`. That is, user Norha is located in Victoria, and thus, she is located in Vancouver Island, as well as British Columbia and Canada.

Example: Figure 5.3 presents a context model for user Norha with three instances of the Transitivity pattern. Each instance of the pattern is defined by each of the following pairs of explicit contextual RDF triples:

1. *(norha.rdf#norha gc:locatedIn geo:Victoria)*
   (geo:Victoria gc:locatedIn geo:VancouverIsland),
2. *(geo:Victoria gc:locatedIn geo:VancouverIsland)*
   (geo:VancouverIsland gc:locatedIn geo:BritishColumbia), and
3. *(geo:VancouverIsland gc:locatedIn geo:BritishColumbia)*
   (geo:BritishColumbia gc:locatedIn geo:Canada).

To reason about the context model presented in Figure 5.3, the SmarterContext CoRE implements the functions *ExistsTransitivity*, and *FindTransitivityClosure*. In this example, the transitive predicate is `gc:locatedIn`, therefore, these

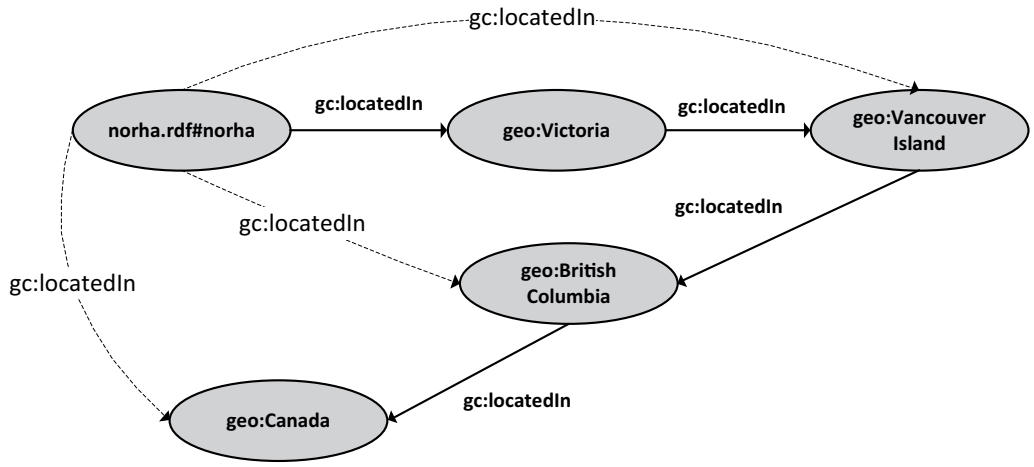


Figure 5.3: A context model with three instances of the Transitivity pattern

functions are useful to answer questions regarding the user's current location. The first function answers whether the user is located in/at a particular location. For instance, *ExistsTransitivity(norha.rdf#norha, gc:locatedIn, geo:Canada, G)* returns `true` indicating that the user is located in Canada. The second function returns the set of all the location context entities the user is currently located. This does not mean the user is located in/at different unrelated places, but that there exists a hierarchical relationship among these location entities. A call to the function *FindTransitivityClosure(norha.rdf#norha, gc:locatedIn, G)* returns Victoria, Vancouver Island, British Columbia and Canada as current locations of the user (cf. Figure 5.3).

Symmetry Pattern

Intent: An instance of the Symmetry pattern represents an implicit context fact (i.e., implicit contextual RDF triple) derivable from one explicit contextual RDF triple whose subject and object are linked through a "symmetric context predicate." As the predicate is symmetric, the context relationship expressed by the predicate also applies in the opposite direction (cf. Figure 5.1(a)).

Motivation: Figure 5.4(a) presents a contextual RDF graph with one explicit context fact, "Tatiana is colleague of Norha." The context predicate `pwc:colleagueOf` is defined as a symmetric predicate in the SmarterContext ontology. Hence, this predicate also applies between subject `norha.rdf#norha` and object `tatiana.rdf#tatiana`. This means that the context fact "Norha is colleague

of Tatiana" is a semantic consequence from this instance of the Symmetry pattern (cf. the triple associated with the dashed arc in Figure 5.4(a)).

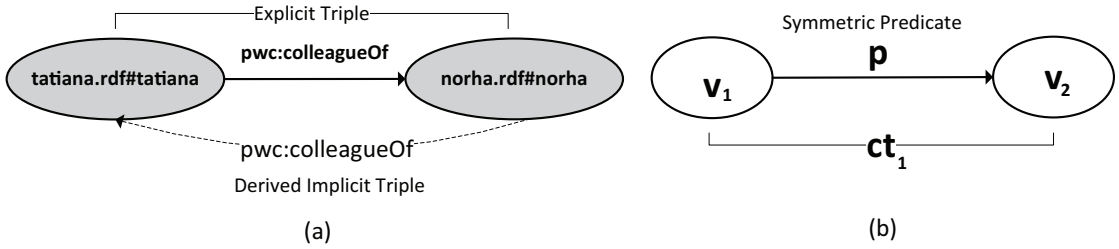


Figure 5.4: Symmetry pattern (a) motivation, (b) graph structure

Preconditions: The context predicate must be a *symmetric predicate* in the SmarterContext ontology.

The context model presented in Figure 5.4(a) is an instance of the Symmetry pattern, given that `pwc:colleagueOf` is defined as a symmetric predicate in the personal web context (PWC) module of the SmarterContext ontology. This is, the *(pwc:colleagueOf rdfs:subPropertyOf gc:symmetricPredicate)* triple does exist in the SmarterContext ontology, concretely in the PWC module.

Definition 10 (Symmetric Predicate)**.** *Let $l(p)$ be a context predicate defined in, or compliant with the* SmarterContext *ontology. $l(p)$ is a "symmetric predicate" iff it is an object subproperty, as defined in RDFS [W3C04c], of the gc:symmetricPredicate* SmarterContext *object property type. That is, the (l(p) rdfs:subPropertyOf gc:symmetricPredicate) triple exists in the ontology that defines l(p).*

Graph Structure: Figure 5.4(b) presents the Symmetry pattern's graph structure.

Pattern Definition:

Definition 11 (Symmetry Pattern)**.** *Let p be an arc labeled with a symmetric predicate $l(p)$; G a contextual RDF graph; and v_1, v_2, labeled vertices in G, where $l(v_1) \neq l(v_2)$ correspond to URIs of either context entities or context types. The* Symmetry pattern *is defined as the unit subset composed of the contextual RDF triple $\{ct\} \in G$, such that $ct = (l(v_1)\ \ l(p)\ \ l(v_2))$ (cf. Figure 5.4(b)).*

The context model presented in Figure 5.4(a) corresponds to an instance of the Symmetry pattern with *l(p)=pwc:colleagueOf*, and *ct=(tatiana.rdf#tatiana pwc:colleagueOf norha.rdf#norha)*.

Applicability: The Symmetry pattern is useful to derive a implicit context fact from an explicit symmetric relationship between two context entities. The subject, predicate, and object of the derived context fact correspond to the object, predicate, and subject of the explicit triple (cf. Figure 5.4(b)). Formally, $G[(l(v_1)\ \ l(p)\ \ l(v_2)]) \models (l(v_2)\ \ l(p)\ \ l(v_1))$, iff triple *(l(p) rdfs:subPropertyOf gc:symmetricPredicate)* is defined in the SmarterContext ontology.

Applying the Symmetry pattern to the model presented in Figure 5.4(a), the context fact "Norha is a colleague of Tatiana" is a semantic consequence derived from the contextual RDF triple "Tatiana is a colleague of Norha."

Pattern-based Reasoning Functions:

$ExistsSymmetry(l(v_1), l(p), l(v_2), G)$**.** Given the labels of two vertices, $l(v_1)$ and $l(v_2)$, a symmetric predicate $l(p)$, and a context model G, this function answers whether the symmetric relationship holds between $l(v_1)$ and $l(v_2)$. That is, the *ExistsSymmetry* function checks whether either $(l(v_1)\ \ l(p)\ \ l(v_2))$ or $(l(v_2)\ \ l(p)\ \ l(v_1))$ exists in G.

For example, the application of the $ExistsSymmetry(l(v_1), l(p), l(v_2), G)$ function with G as the graph presented in Figure 5.4(a), $l(v_1)$ = *norha.rdf#norha*, *l(p)=pwc:colleagueOf*, and $l(v_2)$ = *tatiana.rdf#tatiana* returns `true`, since `pwc:colleagueOf` is a symmetric predicate and the *(tatiana.rdf#tatiana pwc:colleagueOf norha.rdf#norha)* triple is stated in the model explicitly.

Example: Figure 5.5 presents a context model with two instances of the Symmetry pattern. Each instance of the pattern is defined by each of the following explicit contextual RDF triples:

1. *(norha.rdf#norha pwc:isMarriedTo gabriel.rdf#gabriel)*, and
2. *(tatiana.rdf#tatiana pwc:colleagueOf norha.rdf#norha))*.

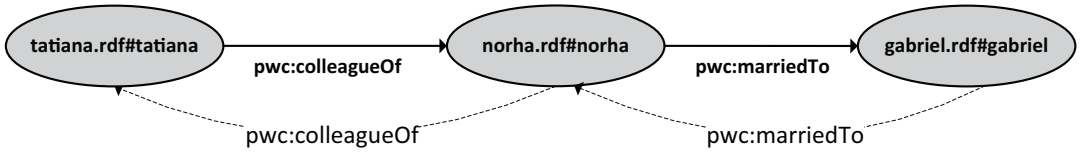


Figure 5.5: A context model with two instances of the Symmetry pattern

To reason about the context model presented in Figure 5.5, the SmarterContext CoRE implements the *ExistsSymmetry* function. In this example, the symmetric predicates are `pwc:isColleagueOf` and `pwc:isMarriedTo`, therefore, this function is useful to answer questions regarding the user's social relationships. For instance, *ExistsSymmetry(gabriel.rdf#gabriel, pwc:isMarriedTo, norha.rdf#norha, G)* returns `true`.

Inverse Pattern

Intent: An instance of the Inverse pattern represents an implicit context fact (i.e., implicit contextual RDF triple) derivable from an explicit contextual RDF triple whose subject and object are linked through a context predicate that is the "inverse of" the predicate of the implicit triple (cf. Figure 5.6(a)).

Motivation: Figure 5.6(a) presents a contextual RDF graph with one explicit context fact, "Gabriel is parent of JG." The context predicate `pwc:parentOf` is defined as the inverse of predicate `pwc:childOf` in the PWC module of the SmarterContext ontology. Hence, predicate `pwc:childOf` applies between subject `jg.rdf#jg` and object `gabriel.rdf#gabriel`. This means that the context fact "JG is child of Gabriel" is a semantic consequence from this instance of the Inverse pattern (cf. the triple associated with the dashed arc in Figure 5.6(a)).

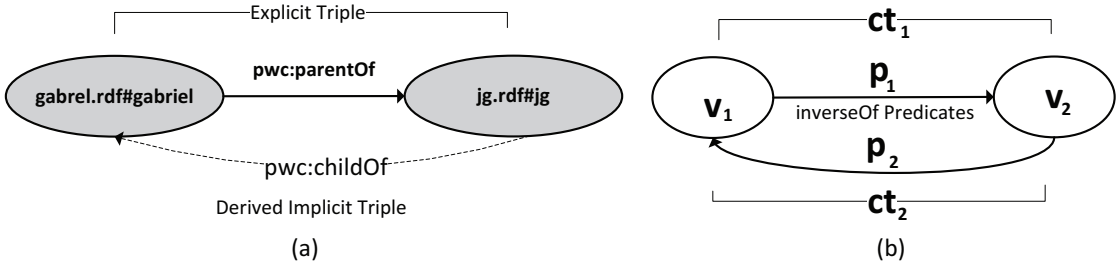


Figure 5.6: Inverse pattern (a) motivation, (b) graph structure

Preconditions: The context predicate must have an *inverse predicate* in the SmarterContext ontology.

The context model presented in Figure 5.6(a) is an instance of the Inverse pattern, given that `pwc:parentOf` is defined as inverse of predicate `pwc:childOf` in the personal web context (PWC) module of the SmarterContext ontology. This is, the *(pwc:parentOf gc:isInverseOf pwc:childOf)* and *(pwc:childOf gc:isInverseOf pwc:parentOf)* triples exist in the SmarterContext ontology, concretely in the PWC module.

Definition 12 (Inverse Predicate)**.** *Let $l(p_1)$ and $l(p_2)$ be context predicates defined in, or compliant with the* SmarterContext *ontology. $l(p_1)$ is a "predicate inverse of" $l(p_2)$ iff the triples $(l(p_1)\ \ gc:isInverseOf\ \ l(p_2))$ and $(l(p_2)\ \ gc:isInverseOf\ \ l(p_1))$ exist in the ontologies that define $l(p_1)$ and $l(p_2)$, respectively.*

Graph Structure: Figure 5.6(b) presents the Inverse pattern's graph structure.

Pattern Definition:

Definition 13 (Inverse Pattern)**.** *Let p_1 be an arc labeled with an inverse predicate $l(p_1)$ of a context predicate $l(p_2)$; G a contextual RDF graph; and v_1, v_2, labeled vertices in G, where $l(v_1) \neq l(v_2)$ correspond to URIs of either context entities or context types. The* Inverse pattern is defined as the subset composed of the contextual RDF triplea $\{ct_1, ct_2\} \in G$, such that $ct_1 = (l(v_1)\ \ l(p_1)\ \ l(v_2))$ and $ct_2 = (l(v_2)\ \ l(p_2)\ \ l(v_1))$ (cf. Figure 5.6(b)).

The context model presented in Figure 5.6(a) corresponds to an instance of the Inverse pattern with *ct=(gabriel.rdf#gabriel pwc:parentOf jg.rdf#jg)*.

Applicability: The Inverse pattern is useful to derive a implicit context fact from an explicit relationship between two context entities linked by a predicate $l(p_1)$ that is inverse of another predicate $l(p_2)$. The subject, predicate, and object of the derived context fact correspond to the object, the inverse predicate, and subject of the explicit triple (cf. Figure 5.6(b)). Formally, $G[(l(v_1)\ \ l(p_1)\ \ l(v_2)]) \models (l(v_2)\ \ l(p_2)\ \ l(v_1))$, iff triples $[(l(p_1)\ \ gc:isInverseOf\ \ l(p_2)),\ (l(p_2)\ \ gc:isInverseOf\ \ l(p_1))]$ are defined in the SmarterContext ontology.

Applying the inverse pattern to the model presented in Figure 5.6(a), the context fact "JG is child of Gabriel" is a semantic consequence derived from the contextual RDF triple "Gabriel is parent of JG."

Pattern-based Reasoning Functions:

$ExistsInverse(l(v_1), l(p_i), l(v_2), G)$**.** Given the labels of two vertices, $l(v_1)$ and $l(v_2)$, an inverse predicate $l(p_i)$, and a context model G, this function answers whether the $(l(v_1)\ \ l(p_i)\ \ l(v_2))$ triple can be inferred from G.

For example, the application of the $ExistsInverse(l(v_1), l(p_1), l(v_2), G)$ function with G as the graph presented in Figure 5.6(a), $l(v_1) = jg.rdf\#jg$, *l(p)=pwc:childOf*, and $l(v_2) = gabriel.rdf\#gabriel$ returns `true`, since `pwc:childOf` is an inverse predicate of `pwc:parentOf`.

$GetInverse(l(v_1), l(p), l(v_2), G)$**.** Given the labels of two vertices, $l(v_1)$ and $l(v_2)$, an inverse predicate $l(p)$, and a context model G, this function returns an implicit context triple in G defined as $(l(v_2)\ \ l(p_i)\ \ l(v_1))$, where $l(p_i)$ is an inverse predicate of $l(p)$.

For example, the application of the $GetInverse(l(v_1), l(p), l(v_2), G)$ function with G as the graph presented in Figure 5.6(a), $l(v_1)$=*gabriel.rdf#gabriel*, *l(p)=pwc:parentOf*, and $l(v_2)$=*jg.rdf#jg* returns the *(jg.rdf#jg pwc:childOf gabriel.rdf#gabriel)* triple, since `pwc:childOf` is an inverse predicate of `pwc:parentOf`.

Example: Figure 5.7 presents a context model with two instances of the Inverse pattern. Each instance of the pattern is defined by each of the following explicit contextual RDF triples:

1. *(gabriel.rdf#gabriel pwc:parentOf jg.rdf#jg)*, and
2. *(gabriel.rdf#gabriel pwc:affiliatedWith www.icesi.edu.co))*.

To reason about the context model presented in Figure 5.7, the SmarterContext CoRE implements the *ExistsInverse* and *GetInverse* functions. In this example, *ExistsInverse* is useful to answer whether JG is child of Gabriel

Figure 5.7: A context model with two instances of the Symmetry pattern

and whether Icesi University associates Gabriel. *GetInverse* is useful to obtain the two implicit contextual facts that can be inferred from the inverse predicates: *(www.icesi.edu.co pwc:associates gabriel.rdf#gabriel)* and *(jg.rdf#jg pwc:childOf gabriel.rdf#gabriel)*.

Join Pattern

Intent: The Join pattern represents an implicit context fact (i.e., implicit contextual RDF triple) derivable from two explicit contextual RDF triples that have different context predicates, and are connected through a context entity acting as a "join." This join entity corresponds to the object of the first triple and the subject of the second triple. The subject, predicate and object of the derived triple correspond to the the subject of the first triple, the predicate of the second triple, and the object of the second triple, respectively (cf. Figure 5.8).

Motivation: Figure 5.8 presents a simple context model that states two explicit contextual facts, "Norha is located in Victoria" (cf. "First Explicit Triple" in Figure 5.8) and "Victoria is near to Vancouver" (cf. "Second Explicit Triple" in Figure 5.8). Since this contextual RDF graph is an instance of the Join pattern, the implicit context fact "Norha is near to Vancouver" is derivable from this model (cf. the triple associated with the dashed arc in Figure 5.8).

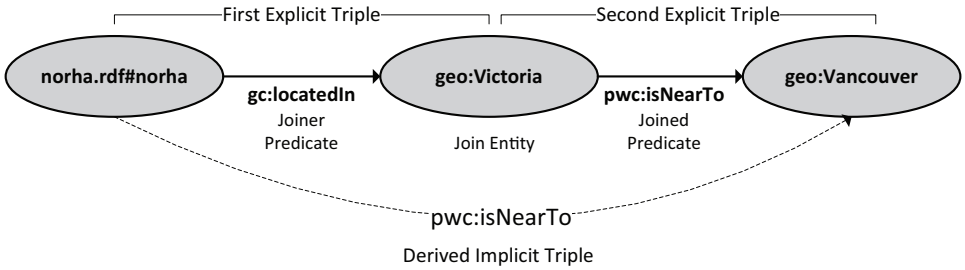


Figure 5.8: Motivating the Join pattern

Preconditions: The predicates of the first and second explicit triples must be defined as a "joiner predicate" and "joined predicate," respectively. Moreover, the predicate of the second explicit triple must be defined as a predicate "joined by" the predicate of the first contextual RDF triple. Since the SmarterContext ontology defines predicate `gc:locatedIn` as a joiner predicate, and predicate `pwc:isNearTo` as joined predicate joined by `gc:locatedIn`, the context model presented in Figure 5.8 corresponds to an instance of the Join pattern.

Definition 14 (Joiner Predicate)**.** *Let $l(p)$ be a context predicate defined in, or compliant with the* SmarterContext *ontology. $l(p)$ is a "joiner predicate" iff it is an object subproperty, as defined in RDFS, of the* `gc:joinerPredicate` SmarterContext *object property type. That is, the ($l(p)$ rdfs:subPropertyOf gc:joinerPredicate) triple exists in the ontology that defines $l(p)$.*

Definition 15 (Joined Predicate)**.** *Let $l(p_1)$ be a joiner predicate and $l(p_2)$ be a context predicate both defined in, or compliant with the* SmarterContext *ontology. $l(p_2)$ is a "joined predicate" iff it is an object subproperty, as defined in RDFS, of the gc:joinedPredicate* SmarterContext *object property type. That is, the triple ($l(p_2)$ rdfs:subPropertyOf gc:joinedPredicate) exists in the ontology that defines $l(p_2)$. Moreover, $l(p_2)$ is "joined by" $l(p_1)$ iff the contextual RDF triple ($l(p_2)$ $gc:isJoinedBy$ $l(p_1)$) exists in the ontology that defines $l(p_2)$.*

In the instance of the pattern presented in Figure 5.8, the arcs labeled as `gc:locatedIn` and `pwc:isNearTo` correspond to context predicates p_1 and p_2, respectively.

Graph Structure: Figure 5.9 illustrates the Join pattern's graph structure.

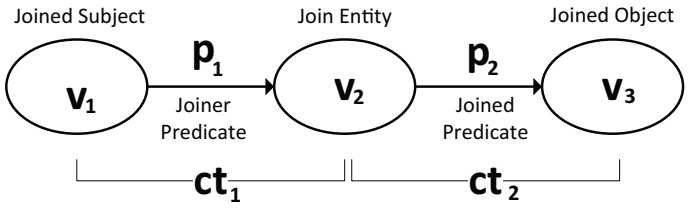


Figure 5.9: Join pattern's graph structure

Pattern Definition:

Definition 16 (Join Pattern)**.** *Let $l(p_2)$ be a context predicate joined by a joiner context predicate $l(p_1)$; G a contextual RDF graph; and v_1, v_2, v_3 labeled vertices in G, where $l(v_1) \neq l(v_2) \neq l(v_3)$ correspond to URIs of either context entities or context types. v_1 is named the "joined subject," v_2 is named the "joined object," and v_3 is named the "join entity." The* Join pattern is defined as the subset of contextual RDF triples $\{ct_1, ct_2\} \in G$, such that $ct_1 = (l(v_1)\ \ l(p_1)\ \ l(v_3))$ and $ct_2 = (l(v_3)\ \ l(p_2)\ \ l(v_2))$ (cf. Figure 5.9).

Since `pwc:isNearTo` is joined by `gc:locatedIn`, which is a joiner predicate, and *ct_1=(norha.rdf#norha gc:locatedIn geo:Victoria)* and *ct_2=(geo:Victoria pwc:isNearTo geo:Vancouver)*, the context model presented in Figure 5.8 is an instance of the Join pattern.

Applicability: The Join pattern allows the identification of implicit contextual facts derivable from a pair of contextual RDF triples joined by a common vertex acting as the object of the first triple and the subject of the second triple, where the predicate of the second triple is defined as joined by the predicate of the first triple, which in turn is a joiner predicate in the SmarterContext ontology. A context fact, defined by the joined subject, the joined predicate, and the joined object is derivable from an occurrence of the Join pattern (cf. Figure 5.9). Formally, $G[(l(v_1)\ \ l(p_1)\ \ l(v_3)), (l(v_3)\ \ l(p_2)\ \ l(v_2))] \models (l(v_1)\ \ l(p_2)\ \ l(v_2))$, iff triples [*$(l(p_1)$ rdfs:subPropertyOf gc:joinerPredicate), $(l(p_2)$ rdfs:subPropertyOf gc:joinedPredicate), $(l(p_2)$ gc:isJoinedBy $l(p_1))$*] are defined in the SmarterContext ontology.

The context fact "Norha is near to Vancouver" is a semantic consequence from the instance of the pattern depicted in Figure 5.8.

Pattern-based Reasoning Functions:

$ExistsJoin(l(v_1), l(v_2), l(p_1), l(p_2), G)$ Given a pair of vertex labels $l(v_1)$ and $l(v_2)$ with $l(v_1) \neq l(v_2)$, a joiner predicate $l(p_1)$, a predicate $l(p_2)$ joined by $l(p_1)$, and a context model G, this function evaluates whether the implicit contextual RDF triple $(l(v_1)\ \ l(p_2)\ \ l(v_2))$ is derivable from G.

Having G as the contextual RDF graph presented in Figure 5.8, the function $ExistsJoin(norha.rdf\#norha, geo : Vancouver, gc : locatedIn, pwc : isNearTo, G)$ returns `true`.

$FindJoinedObjects(l(p_1), l(p_2), l(v), G)$ Finds all the occurrences of the Join pattern for a joiner predicate $l(p_1)$, a context predicate $l(p_2)$ joined by p_1 and a joined subject $l(v)$ in a contextual RDF graph G, and returns the list of joined objects. That is, the list of context entities such that the Join pattern applies between the joined subject $l(v)$ and every entity in the list. Having G as the contextual RDF graph presented in Figure 5.10, the function $ExistsJoin(gc{:}located, pwc{:}isNearTo, norha.rdf\#norha, G)$ returns Vancouver and Seattle as the list of joined objects.

Example: Consider the following partial view of an RDF context model for user Norha.

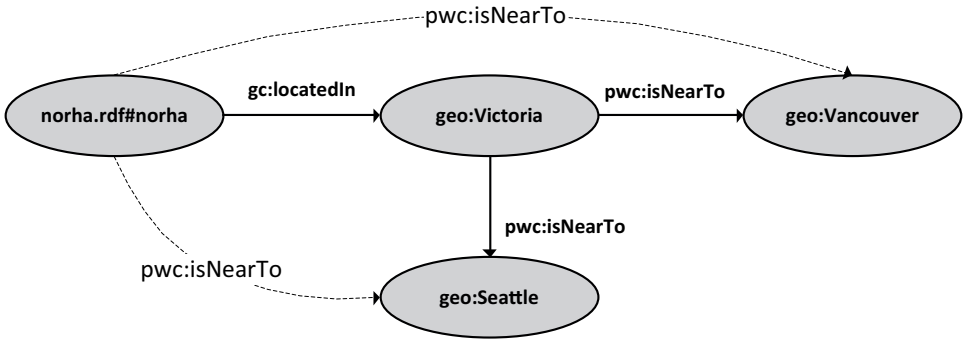


Figure 5.10: A context model with two instances of the Join pattern

The contextual RDF graph presented in Figure 5.10 is composed of the following explicit contextual RDF triples:

1. *(norha.rdf#norha gc:locatedIn geo:Victoria)*,
2. *(geo:Victoria pwc:isNearTo geo:Vancouver)*, and
3. *(geo:Victoria pwc:isNearTo geo:Seattle)*.

Since the SmarterContext ontology defines predicate `gc:locatedIn` as a joiner predicate and predicate `pwc:isNearTo` as joined by the predicate `gc:locatedIn`, this graph contains two instances of the Join pattern. SmarterContext exploits the

Join pattern through the functions *ExistsJoin* and *FindJoinedObjects* to answer questions regarding the location of the user. In a first case, given user `norha.rdf#norha` as the joined subject, location `geo:Vancouver` as the Joined object, `gc:locatedIn` as the joiner predicate, and `pwc:isNearTo` as the joined predicate, *ExistsJoin* returns `true`. That is, according to Norha's current location, it is true that she is located near to Vancouver. A similar case applies to the location context entity `geo:Seattle`. In another case related to *FindJoinedObjects*, given `gc:locatedIn` as the joiner predicate, `pwc:isNearTo` as the joined predicate, and user `norha.rdf#norha` as the joined subject, the answer is a list with Vancouver and Seattle as the locations the user is near to.

Generalization Pattern

Intent: An instance of the Generalization pattern represents an implicit context fact derivable from two contextual RDF triples, where the predicate of the first triple is defined as a "generalizable predicate" for the predicate of the second triple. That is, the first explicit triple is a "particular context fact" generalizable for the object of the second triple. The subject, predicate and object of the derived triple correspond to the subject of the first explicit triple, the generalizable predicate, and the object of the second explicit triple (cf. Figure 5.11).

Motivation: The context model presented in Figure 5.11 presents an instance of the Generalizable pattern. Since the SmarterContext ontology specifies that predicate `pwc:isInterestedIn` is a generalizable predicate for predicate `rdfs:subClassOf`, the context fact "Norha is interested in pants" is derivable from the triple "Norha is interested in jeans." That is, as the model states that Norha is interested in a particular product category (e.g., `google:Jeans`), she may be interested in the products that correspond to this category's super category (e.g., `google:Pants`).

Preconditions: The predicate of the first contextual RDF triple must be generalizable for the predicate of the second triple.

Given that `pwc:isInterestedIn` is defined as a generalizable predicate for predicate `rdfs:subClassOf` in the personal web context (PWC) taxonomy of the SmarterContext ontology, the context model presented in Figure 5.11 is an instance of the Generalization pattern.

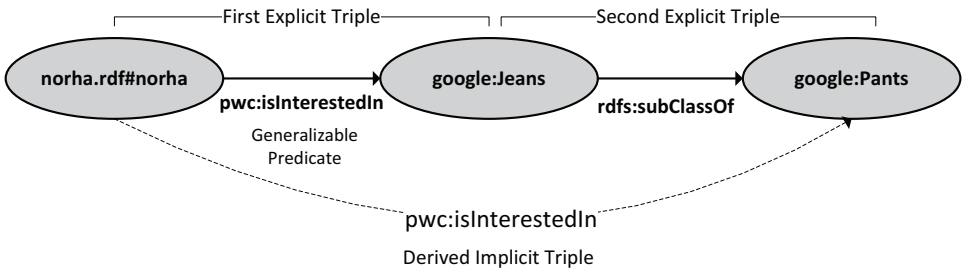


Figure 5.11: Motivating the Generalization pattern

Definition 17 (Generalizable Predicate)**.** *Let $l(p_1)$ and $l(p_2)$ be context predicates defined in, or compliant with the* SmarterContext *ontology. $l(p_1)$ is a "generalizable predicate" iff the ($l(p_1)$ rdfs:subPropertyOf gc:generalizablePredicate) triple exists in the* SmarterContext *ontology that defines $l(p_1)$. Moreover, $l(p_1)$ "is generalizable for" predicate $l(p_2)$ iff the ($l(p_1)$ gc:isGeneralizableFor $l(p_2)$) triple exists in the same ontology.*

In the model presented in Figure 5.11, the predicate of the first explicit triple, `pwc:isInterestedIn`, is a generalizable predicate for `rdfs:subClassOf`. Therefore, the triples *(pwc:isInterestedIn rdfs:subPropertyOf gc:generalizablePredicate)* and *(pwc:isInterestedIn gc:generalizableFor rdfs:subClassOf)* exist in the PWC ontology of SmarterContext.

Graph Structure: Figure 5.12 illustrates the Generalization pattern's graph structure.

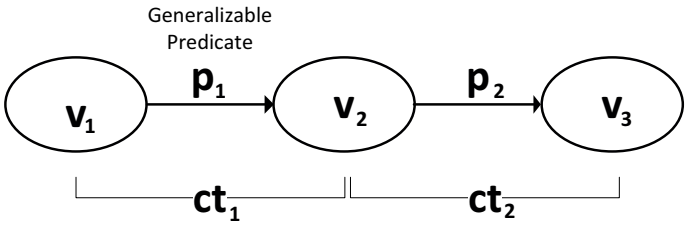


Figure 5.12: Generalization pattern's graph structure

Pattern Definition:

Definition 18 (Generalization Pattern)**.** *Let $l(p_1)$ be a generalizable predicate for predicate $l(p_2)$; G a contextual RDF graph; and v_1, v_2, v_3 labeled vertices in G,*

where $l(v_1) \neq l(v_2) \neq l(v_3)$ *correspond to URIs of either context entities or context types. The* Generalizable pattern is defined as the subset of contextual RDF triples $\{ct_1, ct_2\} \in G$, such that $ct_1 = (l(v_1)\ \ l(p_1)\ \ l(v_2))$ and $ct_2 = (l(v_2)\ \ l(p_2)\ \ l(v_3))$ (cf. Figure 5.12).

The context model presented in Figure 5.11 is an instance of the Generalization pattern, since `pwc:isInterestedIn` is a generalizable predicate for predicate `rdfs:subClassOf`. This instance of the Generalizable pattern is defined as the set of triples ct_1=*(norha.rdf#norha pwc:isInterestedIn google:Jeans)* and ct_2=*(google:Jeans rdfs:subClassOf google:Pants).*

Applicability: The Generalization pattern allows the identification of implicit contextual facts from two explicit contextual RDF triples where the predicate of the first triple is generalizable for the predicate of the second triple. That is, a context fact defined by the generalizable predicate between the subject of the first triple and the object of the second one is derivable from an occurrence of the Generalization pattern. Formally, $G[(l(v_1)\ \ l(p_1)\ \ l(v_2)), (l(v_2)\ \ l(p_2)\ \ l(v_3))] \models (l(v_1)\ \ l(p_1)\ \ l(v_3))$, iff triples [*(*$l(p_1)$ *rdfs:subPropertyOf gc:generalizablePredicate), (*$l(p_1)$ *gc:isGeneralizableFor* $l(p_2)$*)*] are defined in the SmarterContext ontology.

In smarter commerce scenarios, this pattern is useful to derive contextual facts from pairs of contextual RDF triples where the object of the second triple is a general case of the object of the first triple (e.g., Pants is a general category for Jeans), or the two objects are related to each other through context predicates that denote relationships such as similarity, complementarity, or composition (e.g., Necklaces and Earrings are related to each other). In Fig 5.13, the context fact "Norha is interested in Pants" is a semantic consequence derived from this instance of the pattern.

Pattern-based Reasoning Functions:

ExistsGeneralization$(l(v_1), l(v_2), l(p_1), l(p_2), G)$ Given two vertx labels $l(v_1)$, and $l(v_2)$ with $l(v_1) \neq l(v_2)$, two predicates $l(p_1)$ and $l(p_2)$ with $l(p_1)$ defined as a generalizable predicate for $l(p_2)$, and a context model G, this function evaluates whether the Generalization pattern applies between context entities $l(v_1)$ and $l(v_2)$ in G.

In the context model presented in Figure 5.11, the Generalization pattern applies between `norha.rdf#norha` and `google:Pants`, since `pwc:isInterestedIn` is a

generalizable for `rdfs:subClassOf`.

FindGeneralizationObjects$(l(v), l(p), G)$ Given a vertex's label $l(v)$, a generalizable predicate $l(p)$, and a context model G, this function looks for every pair of contextual RDF triples with $l(p)$ as a generalizable predicate for the predicate of each second triple, and $l(v)$ as the subject of each first triple, and returns a set of vertices that correspond to the objects of the second triples.

Having a graph G as the context model presented in Figure 5.11, the application of the function *FindGeneralizationObjects(norha.rdf#norha,pwc:isInterestedIn,G)* returns a unit set defined by the product category `google:Pants`.

Example: Consider the following partial view of an RDF context model for user Norha.

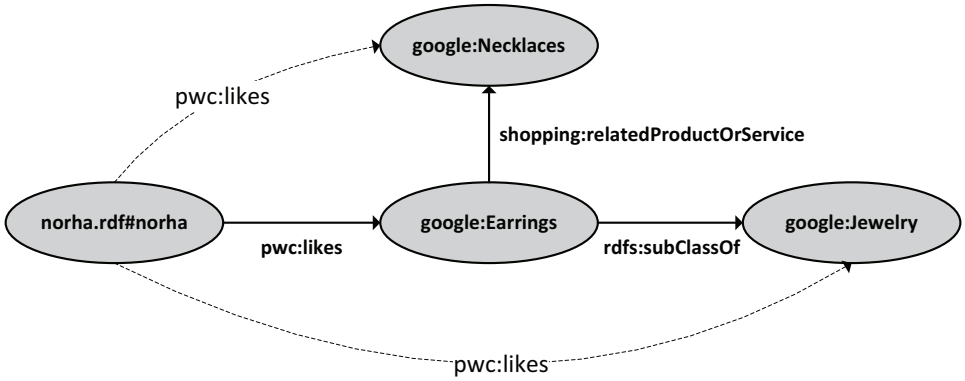


Figure 5.13: A context model with two instances of the Generalization pattern

The contextual RDF graph presented in Figure 5.13 is composed of the following explicit contextual RDF triples:

1. *(norha.rdf#norha pwc:likes google:Earings)*,
2. *(google:Earings rdfs:subClassOf google:Jewelry)*, and
3. *(google:Earings shopping:relatedProductOrService google:Necklaces)*.

In this example, predicate `pwc:likes` is generalizable for both `rdfs:subClassOf` and `shopping:relatedProductOrService`. Therefore, the context model presented in Figure 5.13 contains two instances of the Generalization pattern.

Besides knowing that "Norha likes earrings," the application of the *Exists-Generalization* function answers questions about Norha's preferences such as "does Norha like jewelry?" and "does Norha like necklaces?" For this example, the application of the *ExistsGeneralization(norha.rdf#norha, pwc:likes, $l(p_2)$, l(v), G)* function with $l(p_2)$=*rdfs:subClassOf, l(v)=google:Jewlery,* and $l(p_2)$=*shopping:relatedProductOrService, l(v)=google:Necklaces* returns `true` in both cases.

For the same contextual RDF graph, the application of the *FindGeneralizationObjects(norha.rdf#norha, pwc:likes, G)* returns `google:Necklaces` and `google:Jewelry` as the set of products that Norha likes.

The application of the Generalization pattern to the context model presented in Figure 5.13 allows the derivation of two contextual facts, "Norha likes necklaces" and "Norha likes jewelry."

Delegation Pattern

Intent: The Delegation pattern represents an implicit context fact (i.e., implicit contextual RDF triple) derivable from two contextual RDF triples connected through a common context subject known as the "delegator," and where the predicate of the first explicit triple is defined as a "delegable predicate" for the predicate of the second explicit triple which must be a "delegation predicate." The subject, predicate, and object of the derived triple corresponds to the object of the second explicit triple (delegatee), the predicate of the first explicit triple (delegable predicate), and the object of the first triple (delegator), respectively (cf. Figure 5.14). That is, the common subject (delegator) delegates the predicate of the first triple (delegable predicate) to the object of the second triple (delegatee), which acts now as the subject of the derived triple.

Motivation: Figure 5.14 presents a simple context model that states two explicit contextual facts connected through the common subject `norha.rdf#norha`, "Norha is interested in technology books" and "Norha is a colleague of Tatiana." Since this contextual RDF graph is an instance of the Delegation pattern, the implicit context fact "Tatiana is interested in technology books" is derivable from this model (cf. the triple associated with the dashed arc in Figure 5.14).

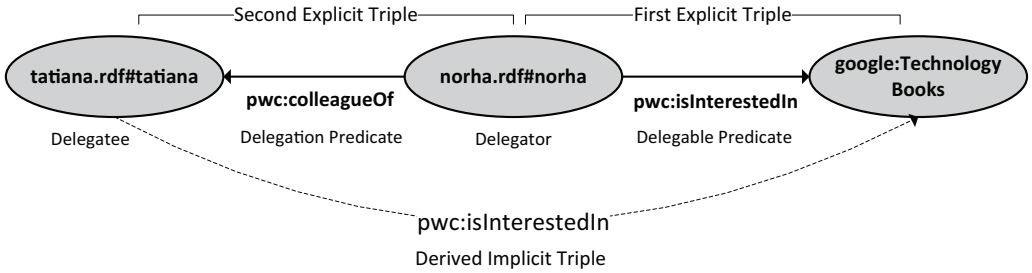


Figure 5.14: Motivating the Delegation pattern

Preconditions: The SmarterContext ontology must define the predicates of the first and second explicit triples as "delegation" and "delegable" predicates, respectively. Moreover, the ontology must define the predicate of the first explicit triple as a predicate "delegable for" the predicate of the second contextual RDF triple. Since the SmarterContext ontology defines predicate `pwc:isInterestedIn` as a delegable predicate for `pwc:colleagueOf`, and predicate `pwc:colleagueOf` as a delegation predicate, the context model presented in Figure 5.14 corresponds to an instance of the Delegation pattern.

Definition 19 (Delegation Predicate)**.** *Let $l(p)$ be a context predicate defined in, or compliant with the* SmarterContext *ontology. $l(p)$ is a "delegation predicate" iff it is an object subproperty, as defined in RDFS, of the gc:delegationPredicate* SmarterContext *object property type. That is, the ($l(p)$ rdfs:subPropertyOf gc:delegationPredicate) triple exists in the ontology that defines $l(p)$.*

Definition 20 (Delegable Predicate)**.** *Let $l(p_1)$ be a context predicate and $l(p_2)$ be a delegation predicate both defined in, or compliant with the* SmarterContext *ontology. $l(p_1)$ is a "delegable predicate" for $l(p_2)$ iff the triples ($l(p_1)$ rdfs:subPropertyOf gc:delegablePredicate) and ($l(p_1)$ gc:isDelegableFor $l(p_2)$) exist in the ontology that defines predicate $l(p_1)$.*

The context model presented in Figure 5.14 corresponds to an instance of the Delegation pattern. Predicate `pwc:colleagueOf` is a delegation predicate, and `pwc:isInterestedIn` is a delegable predicate for `pwc:colleagueOf` in the Personal Web Context (PWC) module of the SmarterContext ontology. That is, the module that defines `pwc:colleagueOf` and `pwc:isInterestedIn`.

Graph Structure: Figure 5.15 illustrates the Delegation pattern's graph structure.

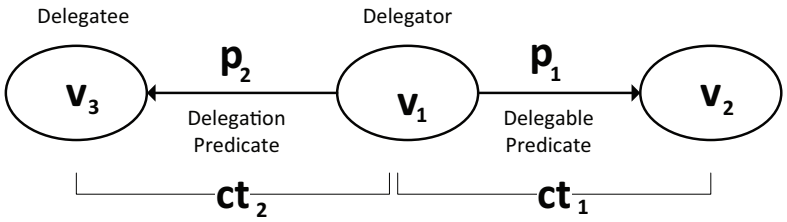


Figure 5.15: Delegation pattern's graph structure

Pattern Definition:

Definition 21 (Delegation Pattern)**.** *Let $l(p_1)$ be a context predicate delegable for a delegation context predicate $l(p_2)$; G a contextual RDF graph; and v_1, v_2, v_3 labeled vertices in G, where $l(v_1) \neq l(v_2) \neq l(v_3)$ correspond to URIs of either context entities or context types. v_1 is named the "delegator subject," and v_3 is named the "delegatee subject." The* Delegation pattern *is defined as the subset of contextual RDF triples $\{ct_1, ct_2\} \in G$, such that $ct_1 = (l(v_1)\ \ l(p_1)\ \ l(v_2))$ and $ct_2 = (l(v_1)\ \ l(p_2)\ \ l(v_3))$ (cf. Figure 5.15).*

Since predicate `pwc:colleagueOf` is a delegation predicate, `pwc:isInterestedIn` is a predicate delegable for `pwc:colleagueOf`, and ct_1=*(norha.rdf#norha pwc:isInterestedIn google:TechnologyBooks)* and ct_2=*(norha.rdf#norha pwc:colleagueOf tatiana.rdf#tatiana)*, the context model presented in Figure 5.14 is an instance of the Delegation pattern.

Applicability: The Delegation pattern allows the identification of implicit contextual facts that are derivable from a pair of contextual RDF triples joined by a common subject, where the predicate of the first triple must be delegable for the predicate of the second triple. A context fact, defined by the object of the second triple, the delegable predicate, and the object of the first triple is derivable from an occurrence of the Delegation pattern. Formally, $G[(l(v_1)\ \ l(p_1)\ \ l(v_2)), (l(v_1)\ \ l(p_2)\ \ l(v_3))] \models (l(v_3)\ \ l(p_1)\ \ l(v_2))$, iff triples [*(*$l(p_2)$ *rdfs:subPropertyOf gc:delegationPredicate), (*$l(p_1)$ *rdfs:subPropertyOf gc:delegablePredicate), (*$l(p_1)$ *gc:isDelegableFor* $l(p_2)$*))*] are defined in the SmarterContext ontology.

The context fact "Tatiana is interested in technology books" is a semantic consequence from the instance of the pattern depicted in Figure 5.14. In smarter commerce scenarios, the Delegation pattern exploits social context relationships (i.e., context

predicates denoting social relationships) to derive contextual facts about users' shopping preferences.

Pattern-based Reasoning Functions:

ExistsDelegation$(l(v_1), l(v_2), l(v_3), l(p_1), l(p_2), G)$ Given three vertex labels $l(v_1), l(v_2)$, and $l(v_3)$, two context predicates $l(p_1)$ and $l(p_2)$, and a context model G, this function evaluates whether the Delegation pattern applies in G for $l(v_1)$ as the delegate subject, $l(v_3)$ as the delegatee subject, $l(v_2)$ as the object of the derived triple, and $l(p_1)$ as a delegable predicate for $l(p_2)$.

Using the contextual RDF graph presented in Figure 5.14, the function *ExistsDelegation* with $l(v_1) = norha.rdf\#norha$, $l(v_2) = google{:}technologyBooks$, $l(v_3) = tatiana.rdf\#tatiana$, $l(p_1) = pwc{:}isInterestedIn$, and $l(p_2) = pwc{:}colleagueOf$ returns `true`.

FindDelegatees$(l(v_1), l(v_2), l(p), G)$ Given a delegator subject $l(v_1)$, an object $l(v_2)$, a delegable predicate $l(p)$, and a context model G, this function looks for all the occurrences of the Delegation pattern, and returns the set of delegatees (objects of the first triple in each occurrence of the pattern).

Having G as the contextual RDF graph presented in Figure 5.14, the function *FindDelegatees(norha.rdf#norha, google:TechnologyBooks, pwc:isInterestedIn, G)* returns `tatiana.rdf#tatiana` as the unit set of delegatees for $l(p)$ in G.

FindDelegatedObjects$(l(v), l(p), G)$ Finds all the instances of the Delegation pattern in G where $l(v)$ is the delegatee subject, and $l(p)$ is the delegable predicate, and returns a set of vertices defined by the objects of the second triple in each occurrence of the pattern.

The application of the function *FindDelegatedObjects(tatiana.rdf#tatiana, pwc:isInterestedIn, G)* to the context graph presented in Figure 5.14 returns `google:TechnologyBooks` as the set of delegated objects.

Example: Consider the following partial view of an RDF context model for user Norha.

The contextual RDF graph presented in Figure 5.16 is composed of the following explicit contextual RDF triples:

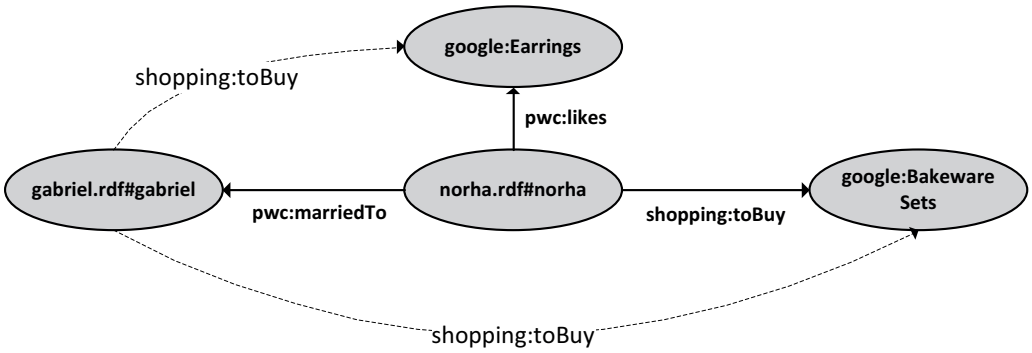


Figure 5.16: A context model with two instances of the Delegation pattern

i. *(norha.rdf#norha shopping:toBuy google:BakewareSets)*,

ii. *(norha.rdf#norha pwc:marriedTo gabriel.rdf#gabriel)*, and

iii. *(norha.rdf#norha pwc:likes google:Earrings)*.

The predicate `shopping:toBuy` defines a product as part of the user's shopping list. Since `shopping:toBuy` and `pwc:likes` are predicates delegable for the context predicate `pwc:marriedTo`, from this set of explicit triples it is possible to derive that "bakeware sets" and "earrings" could be products to buy in Gabriel's shopping list.

The function *ExistsDelegation* is useful to answer questions such as "are bakeware sets or earrings products in Gabriel's shopping list?" Even when these products are not defined as "products to buy" in Gabriel's shopping list, the contextual facts that include "Bakeware Sets" and "Earrings" as product categories that Gabriel could buy are semantic consequences from the application of the Delegation pattern to the model presented in Figure 5.16.

The function *FindDelegatees* answers questions such as "who is interested in buying products that are relevant to Norha?" This interest in other users' products is defined by social relationships such as the context predicates `pwc:marriedTo`, `pwc:colleagueOf`, and `pwc:friendOf`.

Finally, the function *FindDelegatedObjects* is useful to answer questions such as "what are the products a particular user socially related to Norha could be interested in buying?" The application of this function to the model presented in Figure 5.16 returns "Bakeware Sets" and "Earrings" as the set of products that could be included in Gabriel's shopping list.

5.4 Chapter Summary

This chapter presented our SmarterContext context reasoning engine (CoRE), which supports the inference of implicit contextual facts from explicit contextual data modeled in the form of contextual RDF graphs compliant with our SmarterContext ontology.

Situation-aware smart software (SASS) systems use environmental information to both decide whether or not to adapt their behavior, and provide context-aware functionalities. Thus, context management infrastructures must support cognitive functions to reason about high level context abstractions with the goal of inferring implicit contextual facts from sensed contextual data. Moreover, *spatial control* is a feature to represent and reason about context information efficiently (cf. Figure 2.5). Therefore, it is key to limit reasoning spaces to avoid performance degradation. Nevertheless, the management of trade-offs between expressiveness and performance is always an issue. On the one hand, ontology-based knowledge representation approaches, such as OWL and even OWL-Lite, expose performance limitations when reasoning on large data sets [HKR09]. On the other hand, pure RDFS approaches lack semantic expressiveness for context reasoning [HKR09, MM04, W3C04b]. An appropriate balance between expressiveness and performance is crucial to be able to reason about situations on high amounts of contextual data. To balance this trade-off, we investigated the application of computational biology algorithms and techniques, based on subgraphs identification and data partitioning methods, to the mining of contextual facts [PSM+11, MAB02]. As a result, we proposed our *structural context patterns* to substitute OWL-based inferences and provide general templates for users to specify domain-specific reasoning rules that can be processed by our SmarterContext CoRE. Our context patterns can be applied effectively not only to the analysis of individual context models, but also to the analysis of multiple context spheres (e.g., to correlate shopping preferences from personal context models of members of a same social network).

The next chapter presents the first contribution related to self-adaptivity of SASS systems, our framework for evaluating quality-driven self-adaptive software systems. This framework provides valuable foundations for maintaining the effectiveness of runtime context monitoring along the self-adaptation process, and for the assessment and assurance of adaptation mechanisms.

Chapter 6

A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems

This chapter presents our fourth contribution, our framework for evaluating quality-driven self-adaptive software systems [VMT+11c]. An important part of this dissertation focuses on guaranteeing the relevance of runtime monitoring with respect to changes in monitoring requirements in situation-aware smarter software (SASS) systems. To preserve the relevance of runtime monitoring in self-adaptation, it is crucial to identify adaptation properties, goals and corresponding metrics, and from them to infer monitoring requirements. Moreover, monitoring strategies (i.e., context gathering, processing and provisioning components) must be synthesized and deployed at runtime. In this way, monitoring infrastructures can be adapted dynamically when required by changes in requirements. Since dynamic context monitoring plays a relevant role in the assurance of self-adaptive systems, a more general motivation to develop this framework was the need to validate adaptation mechanisms to ensure that self-adaptive systems perform properly and users can trust them.

We proposed our framework with the goal of tackling the lack of well characterized adaptation properties and goals in the state-of-the-art self-adaptive systems, as well as of supporting the evaluation of not only existing approaches but also new self-adaptivity solutions. In our evaluation framework adaptation properties are specified explicitly and driven by quality attributes, such as those defined by researchers at Software Engineering Institute (SEI) [BKLW95]. Our framework provides (i) a set

of dimensions useful to classify self-adaptive systems; (ii) a compendium of adaptation properties to be observed in control loop implementations (in terms of the controller and the managed system); (iii) a mapping of adaptation properties to quality attributes to evaluate adaptation properties; and (iv) a set of quality metrics to evaluate adaptation properties and quality attributes. Metrics are crucial to derive monitoring requirements and to establish an explicit mapping between monitoring conditions and adaptation concerns. To define adaptation properties we analyzed existing self-adaptive approaches and investigated properties used in control theory. We then established a mapping between adaptation properties and software quality attributes. Finally, we identified a set of metrics used to evaluate software quality attributes. The mapping of adaptation properties to quality attributes and their corresponding quality metrics constitute our framework.

Borrowing properties and metrics from control theory and re-interpreting them for self-adaptive software was not a trivial task. On the one side, the semantics of the concepts involved in adaptation for control theory are different than those for self-adaptive software. On the other side, existing self-adaptive software approaches do not generally address adaptation properties explicitly. Yet another important challenge is that, in general, self-adaptive software systems are nonlinear systems.

Metrics to evaluate feedback control systems depend on the properties that result from the relationship between the control objectives, the target system's measured outputs, the disturbances affecting the system and how the target system is considered in the adaptation strategy [HDPT04]. Our re-interpretation of these properties and metrics resulted from the representative approaches that we analyzed in our survey on self-adaptation. From the identified relationship and the analysis of these strategies we identified two main dimensions to classify and evaluate self-adaptive software. These dimensions arise from the way the strategy, *structural* or *behavioral*, addresses (i) the target system, and (ii) the controller itself.

This chapter is organized as follows. Section 6.1 presents our proposed model to characterize and classify self-adaptive systems. Section 6.2 presents our catalog of adaptation properties and the way they can be mapped to quality attributes to derive metrics that allow the identification of monitoring requirements and the assessment of self-adaptive approaches. Section 6.3 summarizes and concludes the chapter.

6.1 A Characterization Model for Self-Adaptive Software

This section presents our model to characterize self-adaptive software, which consists of eight analysis dimensions, and provides a foundation for evaluating self-adaptive systems. For each of these analysis dimensions, the model considers a set of standardized classification options. These options resulted from combining classification attributes from recognized authoritative sources (e.g., SEI) with those found in the set of papers that we analyzed. In particular, the set of options for the analyzed quality attributes as observable adaptation properties was identified mainly using the taxonomy proposed by an SEI study [BKLW95]. This taxonomy provides a comprehensive characterization of software quality attributes, their concerns, factors that affect them and methods for their evaluation. We applied our model to the characterization of the approaches analyzed in our survey, which was summarized in Section 2.5.1. Tables C.1 and C.2 detail the application of our characterization model to the filtered set of surveyed approaches. For each analysis dimension of the model, we include the relevant options available from control theory, as follows.

Adaptation goal. This is the main reason for the system or approach to be self-adaptive. Adaptation goals are usually defined through one or more of the self-* properties, the preservation of specific quality of service (QoS) properties, or the regulation of non-functional requirements in general.

Reference inputs. Are the concrete and specific set of values, and corresponding types, that are used to specify the state to be achieved and maintained in the managed system by the adaptation mechanism, under changing conditions of system execution. Reference inputs are specified as (a) single reference values (e.g., a physically or logically-measurable property); (b) some form of contract (e.g., quality of service (QoS), service level agreements (SLA), or service level objectives (SLO); (c) goal-policy-actions; (d) constraints defining computational states (according to the particular proposed definition of *state*); or even (e) functional requirements (e.g., logical expressions as invariants or assertions, regular expressions).

Measured outputs. Are the set of values, and corresponding types, that are measured in the managed system. Naturally, as these measurements must be compared to the reference inputs to evaluate whether the desired state has been achieved, it should be possible to find relationships between these inputs and outputs. Furthermore, we consider two aspects on the measured outputs: how they are specified and

how monitored. For the specification, the identified options are (a) continuous domains for single variables or signals; (b) logical expressions or conditions for contract states; and (c) conditions expressing states of system malfunction. For monitoring, the options are (a) measurements on physical properties from physical devices (e.g., CPU temperature); (b) measurements on logical properties of computational elements (e.g., request processing time in software or CPU load in hardware); and (c) measurements on external context conditions (e.g., user localization or weather conditions).

Computed control actions. These are characterized in the monitor, analyze, plan, execute, and knowledge (MAPE-K) loop context, and in particular by the nature of the output of the adaptation planner or controller [IBM06]. This output affects the managed system to have the desired effect. The computed control actions can be (a) continuous signals that affect behavioral properties of the managed system; (b) discrete operations affecting the computing infrastructure executing the managed system (e.g., host system's buffer allocation and resizing operations; modification of process scheduling in the CPU); (c) discrete operations that affect the processes of the managed system directly (e.g., processes-level service invocation, process execution operations—halt/resume, sleep/re-spawn/priority modification of processes); and (d) discrete operations affecting the managed system's software architecture (e.g., managed system's architecture reconfiguration operations). The nature of these outputs is related to the extent of the intrusiveness of the adaptation mechanism with respect to the managed system and defines the extent of the adaptation mechanism to exploit the knowledge about either the structure or the behavior of the managed system in the adaptation process.

System structure. Self-adaptive systems have two well-defined subsystems (although possibly indistinguishable): (i) the adaptation controller and (ii) the managed system. One reason for analyzing controller and managed system structures is to identify whether a given approach implements the adaptation controller embedded within the managed system. Another reason is to identify the effect that the separation of concerns in these two subsystems has in the achievement of the adaptation goal. We grouped the analyzed approaches into two sets: (i) those modeling the structure of the managed system to influence its behavior by modifying its structure; and (ii) those modeling the managed system's behavior to influence it directly. We consider the behavior model and the structure model as part of the managed system's structure. The identified options for the controller structure are variations of the MAPE-K

loop with either behavioral or structural models of the managed system: (a) feedback control, that is, a MAPE-K structure with a fixed adaptation controller (e.g., a fixed set of transfer functions as a behavior model of the managed system) [HDPT04]; (b) adaptive control: a MAPE structure extended with managed system's reference or identification models of behavior (e.g., tunable parameters of controller for adaptive controllers: model reference adaptive control (MRAC) or model identification adaptive control (MIAC)) [NB97, DH02b]; (c) reconfigurable control: MAPE-K structure with modifiable controller algorithm (e.g., rule-based software architecture reconfiguration controller). For the managed system structure, the identified options are: (a) non-modifiable structure (e.g., monolithic system); and (b) modifiable structure with/without reflection capabilities (e.g., reconfigurable software components architecture). It is worth noting that not all options for system structure can be combined with any options for computed control actions. For instance, discrete operations affecting the computing infrastructure executing the managed system could be used to improve the performance of a monolithic system, whereas discrete operations for affecting this managed system's software architecture would not make sense.

Observable adaptation properties. By adaptation property we mean a quality (or characteristic) that is particular to a specific adaptation approach or mechanism. A quality can be a specific attribute value in a given state or a characteristic response to a known stimulus in a given context. Thus, observable adaptation properties are properties that can be identified and measured in the adaptation process. Given that we distinguish between the controller and the managed system in any self-adaptive system, we analyze observable adaptation properties also in both the controller and the managed system. The identified properties for the controller are (a) stability; (b) accuracy; (c) settling-time; (d) small-overshoot; (e) robustness; (f) termination; (g) consistency (in the overall system structure and behavior); (h) scalability; and (i) security. For the managed system, the identified properties result from the adaptation process: (a) behavioral/functional invariants; and (b) quality of service conditions, such as performance (i.e., latency, throughput, capacity); dependability (i.e., availability, reliability, maintainability, safety, confidentiality, integrity); security (i.e., confidentiality, integrity, availability); and safety (i.e, interaction complexity and coupling strength).

Proposed evaluation. For the analyzed approaches, we used this element to identify the strategies proposed to evaluate themselves. Among the most used evaluation mechanisms are the execution of tests in real or simulated execution platforms,

and the illustration with example scenarios.

Identified metrics. Correspond to metrics that were used to measure the adaptation's variables of interest in the analyzed approaches.

6.2 From Adaptation Properties and Goals to Monitoring Requirements

Our framework supports the evaluation of a self-adaptive system from two aspects. The first one concerns the evaluation of desired properties for the managed system. We focused only on desired properties that correspond to quality attributes of software systems. The second one relates to desired properties for the controller of the adaptation process. For the identification of desired properties for the managed system, we based our analysis on the taxonomy of quality attributes for software systems proposed by SEI researchers [BKLW95]. For properties related to the controller, we based our analysis on the *SASO properties* identified by Hellerstein *et al.* in the application of control theory to computing systems [HDPT04], and other properties identified in self-adaptive software systems surveys [EER+10, LLC10, Men00, MG05]. Furthermore and taking into account that dynamic context monitoring is a major motivation in this dissertation, we classified the identified adaptation properties according to *how* and *where* they are observed. Concerning how they are observed, some properties can be evaluated using static verification techniques while others require dynamic verification and runtime monitoring. We use the term *observed* as some properties are difficult to measure, despite the fact that controllers are designed to preserve them [TCCD12]. With respect to where, properties can be evaluated on the managed system or on the controller. On the one hand, some properties to evaluate the controller are observable on the controller itself or on both the controller and the managed system; however, most properties can only be observed on the managed system. On the other hand, properties to evaluate the managed system are observable only on the managed system. In both cases, the environment that can affect the behavior of the controller or the managed system also is a factor worth of consideration.

This section presents the foundation of our evaluation framework: a set of properties and metrics for self-adaptation, where properties observable on the managed system, either to evaluate the controller or the managed system, are evaluated in terms of quality attributes. As part of our framework, we proposed a process for

evaluating self-adaptation with which software engineers should be able to (i) identify required adaptation goals (i.e., the quality attributes that drive the adaptation of the managed system); (ii) identify adaptation properties to evaluate the controller, including the properties that are observable on the controller itself, on the managed system, or on both; (iii) map quality attributes used to evaluate the managed system to properties that evaluate the controller but are observable on the managed system; (iv) define metrics to evaluate properties observable on the managed system and on the controller; and (iv) specify monitoring requirements from the metrics defined in (iv).

6.2.1 Quality Attributes as Adaptation Goals

If we intend to evaluate an adaptive software system we need to identify the motivation behind building it—its *adaptation goal*. In general, adaptation can be motivated by the need of continued satisfaction of functional and regulation of non-functional requirements under changing context conditions. Nevertheless, as most analyzed contributions focus on non-functional factors, we based our analysis on software systems whose adaptation goals are motivated by quality concerns. Moreover, characteristics of self-adaptive systems, such as self-configuring or self-optimizing, can be mapped to quality attributes. Following this idea, Salehie and Tahvildari discussed the relationships between autonomic characteristics and quality factors such as the relationship between self-healing and reliability [ST09]. The main contribution of our framework is the application of quality attributes to evaluate self-adaptive software systems, as quality attributes are commonly used to evaluate desirable properties on the managed system. More importantly, our framework includes a mapping between quality factors and adaptation properties. In fact, this introduces a level of indirection for evaluating adaptation properties that are not directly observable on the controller. In this subsection we present the definitions of the selected quality attributes introduced with corresponding citations of the contributions analyzed in our survey that address them (cf. Tables C.1 and C.2).

Performance. It characterizes the timeliness of services delivered by the system. It refers to responsiveness, that is, the time required for the system to respond to events or the event processing rate in an interval of time. Identified factors that affect performance are latency (the time the system takes to respond to a specific event [CCG+09, GCH+04, MG05]); throughput (the number of events that can be

completed in a given time interval—beyond processing rate as the desired throughput must also be observed in time sub-intervals [DC04, KCC+07, PGH+02, SLBL10, Whi05]); and capacity (a measure of the amount of work the system can perform [DC04, KCC+07, PGH+02]).

Dependability. It defines the level of reliance that can justifiably be placed on the services the software system delivers. Adaptation goals related to dependability are availability (readiness for usage [AFF+01, BG11, CCF04, LLC10, SBDP08, Whi05]); reliability (continuity of service [BG11, EER+10, FHS+06, KCC+07, LLC10, SBDP08]); maintainability (capacity to self-repair and evolve [AFF+01, CCF04, FHS+06, SBDP08]); safety (from a dependability point of view, non-occurrence of catastrophic consequences from an external perspective (on the environment) [BG11]); confidentiality (immune to unauthorized disclosure of information); integrity (non-improper alterations of the system structure, data and behavior [BG11]).

Security. The selected concerns of the security attribute are confidentiality (protection from disclosure); integrity (protection from unauthorized modification); and availability (protection from destruction [BG11]).

Safety. The level of reliance that can justifiably be placed on the software system as not generator of accidents. Safety is concerned with the occurrence of accidents, defined in terms of external consequences. The taxonomy presented in [BKLW95] includes two properties of critical systems that can be used as indicators of system safety: interaction complexity and coupling strength. In particular, interaction complexity is the extent to which the behavior of one component can affect the behavior of other components. SEI's taxonomy presents detailed definitions and indicators for these two properties.

6.2.2 Adaptation Properties

A second part of the contribution in this chapter is the identification of adaptation properties that have been used for the analyzed spectrum of adaptive systems, from control theory to software engineering, to evaluate the adaptation process. We define adaptation properties for self-adaptive software as follows. The first four properties, called *SASO properties,* correspond to desired properties of controllers from a control theory perspective [HDPT04]; note that the stability property has been widely applied in adaptation control from a software engineering perspective. The remaining properties in the list were identified from hybrid approaches. Citations included in each

property definition refer to either papers where the property was defined or examples of adaptive systems where the property is observed in the adaptation process.

Stability. The degree in that the adaptation process will converge towards the control objective. An unstable adaptation will indefinitely repeat the controlling action with the risk of not improving or even degrading the managed system to unacceptable or dangerous levels. In a stable system, responses to a bounded input are bounded to a desirable range [AFF+01, FHS+06, LSA+00, Men00, PGH+02].

Accuracy. This property is essential to ensure that adaptation goals are met, within given tolerances. Accuracy must be measured in terms of how close the managed system approximates to the desired state (e.g., reference input values for quality attributes) [CCG+09, SLBL10].

Short settling time. The time required for the adaptive system to achieve the desired state. The settling time represents how fast the system adapts or reaches the desired state. Long settling times can bring the system to unstable states. This property is commonly known as recovery time, reaction time, or healing time [CCF04, HDPT04, KCC+07, LSA+00, Men00].

Small overshoot. The utilization of computational resources during the adaptation process to achieve the adaptation goal. Managing resource overshoot is important to avoid the system unstability. This property expresses how well the adaptation performs under given conditions—the amount of resources used in excess to achieve a required short settling-time before reaching a stable state [AFF+01, CCF04, KCC+07, LSA+00, PGH+02].

Robustness. The managed system must remain stable and guarantee accuracy, short settling time, and small overshoot even if the managed system state differs from the expected state in some measured way. Also, the adaptation process is robust if the controller is able to operate within desired limits even under unforeseen conditions [DC04, Men00].

Termination (of the adaptation process). In software engineering approaches, the planner in the MAPE-K loop typically produces discrete controlling actions to adapt the managed system (cf. Section 6.1), such as a list of component-based architecture operations. The termination property guarantees that this list is finite and its execution will finish, even if the system does not reach the desired state. Termination also is referred as deadlock-free execution, meaning that, for instance, a reconfigurable adaptation process must avoid adaptation rules with deadlocks among them [EER+10, TCCD12].

Consistency. This property aims at ensuring the structural and behavioral integrity of the managed system after performing an adaptation process. For instance, when a controller's adaptation plan is based on dynamic reconfiguration of software architecture, consistency concerns are to guarantee sound interface bindings between components (e.g., component-based structural compliance) and to ensure that when a component is replaced dynamically by another one, the execution must continue without affecting the function of the affected component. These concerns help protect the application from reaching inconsistent states as a result of dynamic recomposition [MG05]. Léger *et al.* define this property alongside atomicity, isolation and durability to complete the *ACID* properties found in transactional systems for guaranteeing reliability in transactions [LLC10]: *(i) Atomicity*, either the system is adapted and the adaptation process finishes successfully or it is not finished and the adaptation process aborts. If an adaptation process fails, the system is returned to its previous consistent state; *(ii) isolation*, adaptation processes are executed as if they were independent. Results of unfinished adaptation processes are not visible to others until the process finishes. Results of aborted or failed adaptation processes are discarded; and *(iii) durability*, the results of a finished adaptation process are permanent: once an adaptation process finishes successfully, the new system state is made persistent. In case of major failures (e.g., hardware failures), the system state can be recovered.

Scalability. The capability of a controller to support increasing demands of work with sustained performance using additional computing resources. For instance, scalability is an important property for the controller when it must evaluate an increased number of conditions in the analysis of context. As computational efficiency is relevant for guaranteeing performance properties in the controller, controllers are required to avoid the degradation of any of the operations of the adaptive process in any situation [AFF+01, DC04, FHS+06].

Security. In a secure adaptation process, not only the target system but also the data and components shared with the controller are required to be protected from disclosure (confidentiality), modification (integrity), and destruction (availability) [BKLW95].

Table 6.1 lists the set of adaptation properties defined in our framework. For each property the table indicates the applicable verification mechanism (i.e., dynamic, static, or both), and where to observe the property (i.e., managed system, controller, or both).

Table 6.1: Classification of adaptation properties

Adaptation Property	Property Verification Mechanism	Where the Property is Observed
Stability	Dynamic	Managed system
Accuracy	Dynamic	Managed system
Settling Time	Dynamic	Both
Small Overshoot	Dynamic	Managed system
Robustness	Dynamic	Controller
Termination	Static	Controller
Consistency	Both	Managed system
Scalability	Dynamic	Both
Security	Dynamic	Both

6.2.3 Mapping Adaptation Properties to Quality Attributes

Once the adaptation goal and adaptation properties have been identified, the following step maps the properties of the controller, which are observable on the managed system, to quality attributes on the managed system. Table 6.2 presents a general mapping between adaptation properties and quality attributes. These quality attributes refer to attributes on both the controller and the managed system depending on where the corresponding adaptation properties are observed. According to Tables 6.1 and 6.2, SASO properties (i.e., stability, accuracy, settling time and small overshoot) can be verified at runtime by observing performance, dependability and security factors in the managed system.

With respect to *stability*, Océano, the dynamic resource allocation system that supports SLAs for peak loads with an order of magnitude of difference, addresses stability based on dependability (i.e., availability and maintainability), and performance (i.e., throughput and capacity—scalability) [AFF+01]. In a similar way, the controller proposed by Parekh *et al.* also addresses stability as an adaptation property to guarantee desirable performance levels (i.e., throughput and capacity) [PGH+02]. They applied an integral control technique to construct a transfer function that models the system and the way the behavior of the managed system is affected by the controller. Baresi and Guinea also addressed stability by proposing a self-recovery system where service oriented architecture (SOA) business processes recover from disruptions of functional and non-functional requirements to avoid catastrophic events

Table 6.2: Mapping properties to quality attributes

Adaptation Property	Quality Attributes	
Stability	Performance	Latency Throughput Capacity
	Dependability	Safety Integrity
	Security	Integrity
Accuracy	Performance	Latency Throughput Capacity
Settling Time	Performance	Latency Throughput
Small Overshoot	Performance	Latency Throughput Capacity
Robustness	Dependability	Availability Reliability
	Safety	Interaction Complexity Coupling Strength
Termination	Dependability	Reliability Integrity
Consistency	Dependability	Maintainability Integrity
Scalability	Performance	Latency Throughput Capacity
Security	Security	Confidentiality Integrity Availability

(safety) and improper system state alterations (integrity), guaranteeing readiness for service (availability) and correctness of service (reliability) [BG11].

Concerning *accuracy*, the MOSES framework proposed by Cardellini *et al.* uses adaptation policies in the form of directives to select the best implementation of the composite service according to a given scenario [CCG+09]. MOSES adapts chains

of service compositions based on service selection using a multiple service strategy. It has been tested with multiple adaptation goals to observe the behavior of the adaptation strategy in terms of its accuracy (i.e., how close the managed system reaches the adaptation goal). Solomon *et al.* also address accuracy in their self-optimizing mechanism for business processes [SLBL10]. They used a simulation model to anticipate performance levels and make decisions about the adaptation process. For this, a tuning algorithm keeps the simulation model accurate. This algorithm compensates for the measurement of actual service time to increase the accuracy of simulations by modeling errors, probabilities and inter-arrival times. Then, it obtains the best estimate for these data such that the square root of the difference between the simulated and measured metrics is minimized.

Appleby *et al.* measure settling time in terms of the time required for deploying a new processing node including the installation and reconfiguration of all applications and data repositories in Océano [AFF+01]. Similarly, White *et al.* evaluated settling time in terms of the average response time required for autonomic EJBs to adapt [Whi05]. Candea's approach applied a recursive strategy that reduces mean time to repair (MTTR) by means of recovering minimal subsets of failed system components. If localized and minimal recovery is not enough, their approach recovers larger subsets progressively [CCF04].

With respect to the last SASO property, *small overshoot*, the control-based approach to ensure SLOs proposed by Parekh *et al.* addresses it by avoiding that control values (e.g., MAXUSERS) are set to values that exceed their legal range. They used root-locus analysis to predict the valid values of the maximum number of users. This approach divides the valid range of values into three regions in order to decide when the control values reach undesirable levels. Based on empirical studies, they analyzed properties of the transfer function to predict the desired range of values [PGH+02].

Robustness is addressed in the self-management approach for balancing system load proposed by Dowling and Cahill [DC04]. They aimed at realizing a robust controller by implementing an adaptation mechanism via decentralized agents that eliminate centralized points of failure.

Termination can be verified using dynamic and static mechanisms. Ehrig *et al.* proposed a self-healing mechanism for a traffic light system to guarantee continuity of service (reliability) by self-recovering from predicted failures (integrity) [EER+10]. They addressed termination by statically checking that self-healing rules are deadlock-free, in such a way that the self-repairing mechanism never inter-blocks traffic lights

in the same road intersection.

Scalability also is an adaptive property in the K-Components system [DC04]. For this, self-management local rules of the agents support evolving capabilities. Another approach where scalability is addressed as an adaptation property is Madam, the middleware proposed by Floch *et al.* for enabling model-based adaptation in mobile applications [FHS+06]. Scalability is a concern in Madam for several reasons. First, its reasoning approach might result in a combinatorial explosion if all possible variants are evaluated; second, the performance of the system might be affected when reasoning on a set of a concurrently running applications competing for the same set of resources. They proposed a controller where each component (e.g., the adaptation manager) can be replaced at runtime to experiment with different analysis approaches for managing scalability.

Security was not addressed as an adaptation property by any of the self-adaptive systems analyzed in our survey. However, we propose the use of SEI's definition of security as a quality attribute and its corresponding quality factors to evaluate security on the controller [BKLW95]. As presented in Table 6.1, security of the controller should be evaluated independently of the managed system. This means that ensuring security at the managed system does not guarantee security in the controller.

6.2.4 Adaptation Metrics

Adaptation metrics provide the way of evaluating adaptive systems with respect to particular concerns of the adaptation process [Men00, RWvM10]. Thus, metrics provide a measure to evaluate desirable properties. For instance, metrics to evaluate control systems measure aspects concerning the SASO properties (i.e., stability, accuracy, settling time, and small overshoot). To characterize the evaluation of adaptive systems, we analyzed the variety of self-adaptive software systems to identify adaptation properties (i.e., at the managed system and the controller) that were evaluated in terms of quality attributes. Just as evaluating most properties is impossible by observing the controller itself, we propose the evaluation of these properties by means of observing quality attributes on the managed system. To identify relevant metrics, we characterized a set of factors that affect the evaluation of quality attributes such as memory usage, throughput, response time, processing rate, mean time to failure, and mean time to repair [LSA+00, BKLW95]. These factors are essential when con-

sidering the metrics to evaluate properties on both the controller on the managed system [RWvM10].

Cardellini *et al.* evaluated performance and reliability in MOSES using the following metrics: expected response time (Ru) (the average time needed to fulfill a request for a composite service); expected execution cost (Cu) (the average price to be paid for a user invocation of the composite service); and expected reliability (Du) (the logarithm of the probability that the composite service completes its task for a user request [CCG+09]).

Appleby *et al.* measured dependability factors (e.g., availability) and performance factors (e.g., scalability in terms of throughput and capacity) in Océano using the following metrics: active connections per server (the average number of active connections per normalized server across a domain); overall response time (the average time it takes for any request to a given domain to be processed); output bandwidth (the average number of outbound bytes per second per normalized server for a given domain); database response time (average time it takes for any request to a given domain to be processed by the back-end database); throttle rate (T) (a percentage of connections disallowed to pass through Océano on a customer domain); admission rate (the complement of the domain throttle rate $(1-T)$); and active servers (the number of active normalized-servers which service a given customer domain [AFF+01]).

Average response time is a common metric used to evaluate performance in several adaptive approaches, such as the framework to develop autonomic EJB applications proposed by White *et al.* [Whi05]. Another example is K-Components, which optimizes system performance based on a load balancing function on every adaptation contract that uses a cost function to calculate its internal load cost and the ability of its neighbors to handle the load [DC04]. This cost function is defined as the addition of the advertised load cost and internal cost of the component (i.e., calculated as the estimated cost to handle a particular load type).

Parekh *et al.*, in their control-based approach to achieve performance service level objectives, used the length of the queue of the in-progress client requests as the metric to control the offered load (the load imposed on the server by client requests) [PGH+02]. Baresi and Guinea also proposed a metric to control reliability on the adaptation of BPEL business processes [BG11], based on the number of times a specific method responds to within two minutes over the total number of invocations. Moreover, they defined a KPI based on this metric, such that reliability must be greater that 95% over the past two hours of operation.

Kumar *et al.* defined a business value key performance indicator (KPI) in terms of factors, such as the priority of the user accessing the information, the time of day the information is being accessed, and other aspects that determine how critical the information is to the enterprise [KCC+07]. For this, they used a utility function as a combination of some of these factors: $utility(e_{gj-k}) = f(\sum d_{ni}, min(b_n i), b_{gj-k})$, where $i|e_{ni} \in M(e_{gj-k})$. The business utility of each edge (e_{gj-k}), which represents data streams between operators that perform data transformations, is a function on the delay d_{ni}, the available bandwidth $b_n i$ of the intervening network edges e_{ni}, and the required bandwidth b_{gj-k} of the edge e_{gj-k}.

Candea *et al.* evaluated availability in terms of mean time to recover $(MTTR)$ [CCF04]. For this, they defined two metrics: availability $(A = MTTF/(MTTF+MTTR))$ and downtime of unavailability $(U = MTTR/(MTTF+MTTR))$, where $MTTF$ is the mean time for a system or subsystem to fail (i.e., the reciprocal of reliability), $MTTR$ is the mean time to recover, and A is a number between 0 and 1. U can be approximated to $MTTR/MTTF$ when $MTTF$ is much larger than $MTTR$. Similarly, Sicard *et al.* defined a metric for availability in terms of $MTTR$ [SBDP08].

The last columns of Tables C.1 and C.2 summarizes the identified evaluation methods and metrics to assess self-adaptive systems. Although these metrics are directly related to the measurement of quality factors, we argue that such metrics are useful for evaluating adaptation properties based on our proposed mapping between quality attributes and adaptation properties. The approach by Reinecke *et al.* [RWvM10] supports our hypothesis. Their metric measures the ability of a self-adaptive system to adapt. They argue that adaptivity can be evaluated using a meta-metric named *payoff* which can be defined in terms of any performance metric to measure the effectiveness of the adaptation process. That is, the optimal adaptive system is characterized by the fact that its adaptation decisions are always optimal (i.e., always yield the optimal payoff). To apply their metric it is necessary to (i) identify the adaptation tasks, (ii) define one or more performance metrics on these tasks (i.e., these metrics should reflect the contribution of these tasks toward the adaptation goal), (iii) define a payoff metric in terms of the performance metrics, and (iv) apply the metric by observing it in the system's performance through runtime monitoring.

6.3 Chapter Summary

This chapter presented our framework for evaluating quality-driven self-adaptive software systems. Our framework defines a set of characterization dimensions and adaptation properties applicable to the evaluation of self-adaptation. We derived the characterization dimensions, presented in Section 6.1, from our foundational reference model: the feedback loop from control theory. Our framework classifies adaptation properties, presented in Section 6.2 into two groups: those concerning the desired properties of the target system, and those concerning the desired properties of the adaptation mechanism. To define the first group, we proposed the use of quality attributes. To define the second group, we borrowed the SASO properties from control theory and re-interpreted them to make them applicable to software-systems. Our framework also establishes a mapping between adaptation properties and quality attributes in Section 6.2.3. Therefore quality attributes and their corresponding metrics provide the means to assess self-adaptive software systems and thus to derive context monitoring requirements.

The next chapter presents our proposed reference model for the design of highly dynamic self-adaptive systems. In our reference model: (i) adaptation properties and goals constitute the control objectives that not only drive the adaptive behavior of the target system, but also the dynamic behavior of adaptation mechanisms and context monitoring infrastructures; (ii) quality attributes and corresponding metrics provide the means to identify context monitoring requirements and to map them to adaptation concerns.

Chapter 7

Dynamico: A Reference Model for Improving Self-Adaptivity

Most implemented approaches reported in the literature to self-adaptation assume adaptation goals and monitoring infrastructures as immutable, thus constraining their applicability to systems whose context awareness is restricted to static monitors. However, for many systems it is not advisable to discard unexpected context changes and dynamic changes in adaptation goals and user requirements, such as SLA renegotiation at runtime. Therefore, separation of concerns, dynamic monitoring, and runtime requirements variability are critical for satisfying system goals under highly changing environments.

To cope with these levels of dynamics, static monitoring infrastructures cannot guarantee the relevance of context monitoring thus affecting the effectiveness of self-adaptation. Therefore, situation-aware smarter software (SASS) systems must be designed in such a way that monitoring infrastructures can adapt themselves at runtime to guarantee the monitoring process even though the execution environment, the system and its requirements may change continuously.

This chapter presents our fifth contribution, Dynamico (DYNamic Adaptation, MonItoring and Control Objectives model), our reference model for engineering context-based self-adaptive software composed of three types of feedback loops [VTM+13]. Each of the feedback loops modeled by Dynamico manages each of the three levels of dynamics that we characterize for self-adaptation: (i) the control objectives feedback loop, (ii) the target system adaptation feedback loop, and (iii) the dynamic monitoring feedback loop. As a reference model (i.e., a standard decomposi-

tion of a known kind of problems into distinguishable parts, with functionalities and control/data flow that are well defined [BCK03]), Dynamico provides self-adaptive system designers with a reference to decide whether the objectives, the system, or the monitoring infrastructure must be adapted at runtime. In this sense, our reference model can be used to check if these dimensions are being considered in the designs. Moreover, it defines the elements and functionalities, as well as the control and data interactions to be implemented, not only among the feedback loop elements, but also among the three types of feedback loops. In addition, our characterization of the latter interactions allows our reference model to be applied partially, that is omitting any of its feedback loops, targeting self-adaptive systems where supporting changes in any of the three levels of dynamics is a crucial requirement.

In light of this, we argue that, in order to regulate the satisfaction of adaptation goals and managed application's requirements continuously, (i) each of the feedback loop elements and their interactions must be independently analyzable; and (ii) the monitoring elements must be able to process the different kinds of information that the varying context can produce appropriately. Dynamico is inspired by classical control theory and the autonomic element proposed by IBM researchers [KC03]. With this reference model we contribute to the design of self-adaptive software by making its instances consider these aspects explicitly: (i) the achievement of adaptation goals and their usage as the reference control objectives; (ii) the separation of control concerns by decoupling the different feedback loops required to satisfy the reference objectives as context changes; and (iii) the specification of context management as an independent control function to preserve the contextual relevance with respect to internal and external context changes. Dynamico together with our evaluation framework (cf. Chapter 6) provide the answer to our research question Q3: *How do we design SASS systems to maintain the relevance of context-awareness with respect to changing system goals?* Dynamico helps improve self-adaptivity by controlling the dynamicity of adaptation goals, adaptation mechanisms, and monitoring mechanisms, and the way they affect each other.

To present our Dynamico reference model we organized this chapter as follows. Section 7.1 presents a motivation example that is used throughout the chapter to explain our reference model. This example is related to our case study on dynamic SOA governance where we improve the dynamic capabilities of software systems through dynamic context monitoring. Section 7.2 presents fundamental concepts that have shaped the engineering of self-adaptive software, and from which we distill our refer-

ence model. Section 7.3 presents our proposed reference model including the feedback loop interactions. Section 7.4 discusses the governance of feedback loop interactions in Dynamico. Section 7.5 presents variations admitted by our reference model. Finally, Section 7.6 summarizes the chapter.

7.1 Improving the Dynamic Capabilities of Software Systems

This section presents a motivational example based on our case study on dynamic SOA governance. In this case study, we exploit self-adaptation mechanisms at runtime to manage service-level agreements (SLAs), and ensure quality of service (QoS) requirements in service-oriented systems. This case study is a good exemplar for validating our research on dynamic context management given that in SOA and cloud-based systems QoS is highly affected by and dependent on context information. On the one hand, SLAs may be violated at any time during system execution due to changes in the situation of relevant context entities such as computational infrastructure components (i.e., internal context), and users (i.e., external context). On the other hand, as businesses and users' requirements evolve continuously, contracted QoS conditions (i.e., adaptation goals) may be frequently re-negotiated, thus affecting the effectiveness of monitoring and adaptation mechanisms.

7.1.1 Dynamic SOA Governance

Software-as-a-Service (SaaS) is one of the business models in cloud computing environments. SaaS provides customers with several benefits such as maintenance and evolution supported by the cloud provider, high availability, pay-per-use, and low operational costs. Suppose an SaaS cloud provider, specialized in large-scale e-commerce platforms, is interested in governing the efficiency of the service-oriented infrastructure with the goal of optimizing operational costs. Assume that to guarantee low operation costs and thus contracted conditions, performance governance has been initially defined as the adaptation goal. For this, a performance SLA defines a service level objective (SLO) to guarantee efficiency of at least 90% for a particular service (e.g., *ProcessingPurchaseOrder*). The metric associated with the SLO is the *time behavior metric (TB)* proposed by Lee *et al.* [LLCK09]. We used this metric as an efficiency measure based on the processing time of service interfaces. Let us assume

that initially the *ProcessingPurchaseOrder* is composed only of one interface, thus we express the service efficiency as:

$$TB = \frac{ProcessingPurchaseOrder\ interface\ execution\ time}{total\ ProcessingPurchaseOrder\ service\ invocation\ time} \tag{7.1}$$

The denominator, *total ProcessingPurchaseOrder service invocation time*, represents the total time it takes for the service to respond after the corresponding request. The numerator, *ProcessingPurchaseOrder interface execution time*, indicates the time consumed for processing a given interface functionality. *ProcessingPurchaseOrder* is composed only of one task defined initially as one interface implementation, thus the numerator is the processing time required for executing that individual task (i.e., $total\ ProcessingPurchaseOrder\ service\ invocation\ time - waitingtime$). TB is in the range $[0, 1]$, where higher values indicate a better measure of performance in terms of time efficiency. Finally, suppose that an action guarantee, defined as part of the SLA, will trigger a self-optimizing feature that performs an online architectural reconfiguration to improve the system's efficiency and capacity.

7.1.2 The Need for Dynamic Context Monitoring

Using DYNAMICO, runtime SOA governance can be optimized by supporting adaptive monitoring strategies to address changes in monitoring requirements. Variations in monitoring requirements can be generated by changes in either the governance objectives (i.e, adaptation goals), the target system, the adaptation mechanism, or relevant context entities. The following two use cases illustrate the need for supporting dynamic monitoring, to preserve the context-awareness of the adaptation mechanism upon changes in the target system (i.e., changes in internal context entities) and adaptation goals.

Use Case 1: changes in internal context entities. Suppose that due to self-adaptation, the *ProcessingPurchaseOrder* service is replaced by a set of distributed services intended to enlarge the order processing capacity of the e-commerce platform. Consequently, the efficiency metric presented in Equation (7.1) must be applied to every new service interface. With traditional static monitoring mechanisms, the governance of the performance SLA is compromised as the monitoring infrastructure was originally implemented to monitor the time efficiency of the *ProcessingPurchaseOrder*

service interface only. The monitoring of the new interfaces is not supported without manually implementing the required sensors and monitors. This implies that every time monitoring conditions or the set of context entities to be monitored change, the monitoring instrumentation must be adjusted manually. Moreover, the effectiveness in performing these changes depends on the effectiveness in reporting them. Using Dynamico, the SOA governance infrastructure is able to deal with changes in monitoring requirements at runtime. Once the new services for purchase order processing are deployed, adaptation mechanisms will trigger the adaptation of the monitoring strategy to monitor the new service interfaces. Our monitoring infrastructure exposes autonomous capabilities to configure and deploy new sensor interfaces and monitoring conditions at runtime.

Use Case 2: changes in adaptation goals. Suppose now that the initial SLA is re-negotiated. A new service level objective (SLO) on throughput is added to the efficiency SLO defined originally as the contracted condition of the performance SLA (cf. Equation (7.1)). The new throughput SLO defines two different throughput levels, depending on the applicable context situation, as summarized in Table 7.1.

Table 7.1: Adding a new throughput SLO to the existing SLA

Throughput SLO of the Performance SLA		
Throughput level	Monitoring Condition	Relevant Context Entities
Medium load	No. of likes on an offer $\leq$ 200,000	A special offer on a social network
Highest peak load	Is Black Friday or Christmas season?	Day of the year

The original monitoring infrastructure is implemented so that the initial performance SLA supports only the monitoring of the individual *ProcessingPurchaseOrder* interface. Once the SLA is re-negotiated, the adaptation mechanism is no longer effective as the new monitoring requirements imposed by the new throughput SLO are not supported. Dynamic changes in the monitoring infrastructure may occur at different levels. They may imply either the deployment of new sensors and new monitoring condition algorithms, or the modification of existing monitoring thresholds and conditions. In any case, without supporting changes in monitoring strategies

at runtime, the adaptation mechanisms must be adjusted manually to ensure their relevance with respect to new adaptation goals. In this example, the new throughput SLO detailed in Table 7.1 will trigger the adaptation of the existing monitoring strategy. Two new sensor interfaces and corresponding monitoring conditions will be added to the monitoring infrastructure based on the new SLA specification. With these new components, the context manager can monitor the acceptance of a special offer placed on a social network integrated into the e-commerce platform, and the season. In both cases, the monitored information is used to anticipate the expected system load and thus adapt the e-commerce platform to modify its capacity according to the context situation.

7.2 Design Drivers in the Engineering of Self-Adaptive Software

We proposed Dynamico to guide the design of self-adaptive software systems addressing the separation of concerns between the monitoring process, the adaptation controller, and the management of control objectives (adaptation goals). This separation of concerns is crucial for governing the consistency between adaptation mechanisms and control objectives, while preserving the relevance of context monitoring of the adaptation mechanism. In light of this, we concluded that a loose-coupling schema is preferable to a tight-coupling one for the integration and communication among the feedback loop elements. However, we retain the idea of composing instances of feedback loops similarly as specified by the generic hierarchical structure described in the autonomic computing reference architecture (ACRA) proposed by IBM researchers (cf. Section 2.4) [IBM06]. Finally, while the autonomic manager, as an implementation of the feedback loop, is the architecture driver for ACRA, our architecture drivers are the independent MAPE-K loop elements, their explicit interactions, and the separation of these elements in three main groups, as follows.

7.2.1 The Three Levels of Dynamics

We identify three levels of dynamics that must be controlled in the engineering of context-driven self-adaptive software systems: (i) the management of changing control objectives, (ii) the dynamic behavior of the adaptation mechanism controlling the target system, and (iii) the management of dynamic context information. Each

of these levels of dynamics plays an important role in governing the dynamic nature of the other two levels. In the case of the first level, as business goals and corresponding control objectives that must drive the behavior of the adaptive system evolve continuously, context monitoring mechanisms (the third level of dynamics), and adaptation controllers (the second level) are required to change accordingly. Furthermore, the management of control objectives may be affected as a result of monitored observations at the third level of dynamics. For instance, whenever the system identifies that even though the adaptation mechanism is performing properly, control objectives may be reviewed to modify the adaptation and/or monitoring mechanism due to changes in context situations.

These three levels of dynamics are illustrated using the motivation scenario described in Section 7.1. The first level, the management of changing control objectives, corresponds to the software instrumentation required to identify changes in adaptation goals. In our example, these changes correspond to the re-negotiation of the performance SLA by adding a new throughput SLO to the initial efficiency SLO. The second level, the dynamic behavior of the adaptation mechanisms, refers to the capability of adaptation strategies to adapt according to changes in either adaptation goals, or context situations. The adaptation mechanism in the scenario does not expose dynamic behavior. That is, the adaptation strategy is always the same. The third level, the management of dynamic context information, refers to the instrumentation required to support changes in monitoring strategies at runtime. In the application example, the dynamic reconfiguration of the monitoring strategy is triggered by two different situations. In the first case, new sensor interfaces are deployed at runtime to monitor the new service interfaces that have been added with the new set of distributed services for processing purchase orders (cf. Section 7.1.2, Use Case 1). In the second case, new sensor interfaces and monitoring conditions are deployed dynamically due to changes in adaptation goals (i.e., the new throughput SLA). The negotiation of a new throughput SLO requires from the monitoring infrastructure to keep track of two new context entities, a special offer placed on a social network and the day of the year (cf. Table 7.1).

7.3 Our Proposed Reference Model

Reference models serve as starting points for software architecture and high-level design specifications. Bass *et al.* define a reference model in software engineering as

a standard decomposition of a known kind of problems into clearly distinguishable parts [BCK03]. Each of these parts has assigned a well defined functionality, and the data flow among these parts is explicitly specified.

Following this definition and based on our design drivers (cf. Section 7.2), we distill in our reference model the characteristics that have been commonly discussed and used in other representative research in the engineering of self-adaptive software. We started by considering the general feedback control loop block diagram presented in our research approach (cf. Figure 1.3). In this diagram, the target system to be controlled, its controller and corresponding transducers are represented as rectangles. The elements for setting the reference input (set point) and perform the comparison against the system measurements are combined in a crossed circle. This block diagram reflects the relative simplicity of the "autonomous" but independent elements used in control engineering. This simplicity hides the very specific and natural electro-mechanical properties (e.g., resistance, capacitance, and inductance) of these elements. In contrast, in the MAPE-K model the elements are interdependent and their functions are specified in a general way. Concerning the characteristics of the different control strategies, control theory takes advantage of exactly the particular complex properties of the matter that constitutes both the controller elements, as well as the system to be controlled. In the case of software artifacts, even though they lack the physical properties analyzed in control engineering, these artifacts are given particular properties of behavior by their particular design. Nonetheless, and because of this, it is practically impossible to generalize them.

Therefore, given the characteristics of software systems (i.e., the systems to be controlled), we find that the combination of a general specification for the common elements of both feedback-loops and MAPE-loops together with a loose coupling scheme, are the best options for Dynamico. Figure 7.1 captures these decisions, which represents the general component of our reference model. This diagram clearly results from the merging of the classical feedback-loop and the MAPE loop model (cf. Figure 1.3 and Figure 2.13, respectively).

7.3.1 Addressing Separation of Concerns

Analyzing Figure 7.1 from both the control theory and software architecture perspective, for a software system (*target system*) to become effectively context-driven self-adaptive, it should incorporate at least three subsystems: (i) a *control objectives*

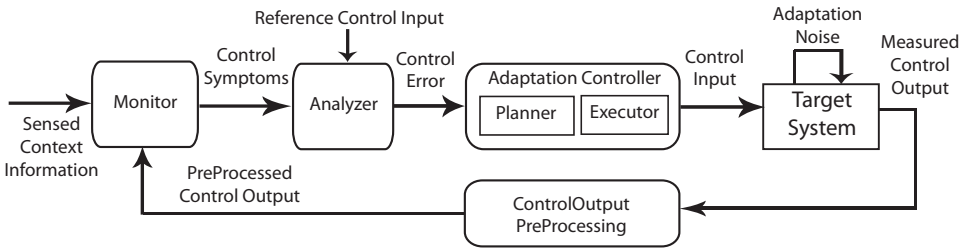


Figure 7.1: General components of DYNAMICO. Feedback control block diagram with explicit functional elements and corresponding interactions to control dynamic adaptation in a software system

manager, (ii) an *adaptation controller mechanism*, and (iii) a *context manager* or *monitoring infrastructure controller mechanism*. This design separates the concerns with respect to the three levels of dynamics we proposed as design drivers for the engineering of context-driven self-adaptive systems: (a) the regulation of the target system's functional and non-functional requirements satisfaction; (b) the continuous accomplishment of adaptation goals and the preservation of the target system's properties under changing conditions of execution; and (c) the relevance of the context monitoring infrastructure according to the varying execution environment (dynamic context monitoring). This separation of concerns leads us to abstract the block diagram presented in Figure 7.1 into the block diagram presented in Figure 7.2. In this diagram, which constitutes our reference model as such, each of the three feedback loops, the *control objectives feedback loop* (CO-FL), the *adaptation feedback loop* (A-FL), and the *monitoring feedback loop* (M-FL), is an instance of the model depicted in Figure 7.1.

The identification of these subsystems as independent feedback loops allows us to independently analyze, design, implement, and assess the instrumentation required to address the complexity of changing requirements at each of the three levels of dynamics. In this way, and depending on the nature of the adaptive system, this instrumentation can be easily temporal and spatial distributed and maintained. In addition, the entire software system would be less affected by the computational effort of each of the three subsystems. The separation of concerns made explicit by the DYNAMICO model is particularly crucial for cases such as the cloud-based e-commerce platform presented in our application scenario. In this example, the automatic reconfiguration of the monitoring strategy would not be feasible without having the context manager as an independent implementation of the adaptation mechanism. In the same way, the explicit control of changes in SLAs requires separate

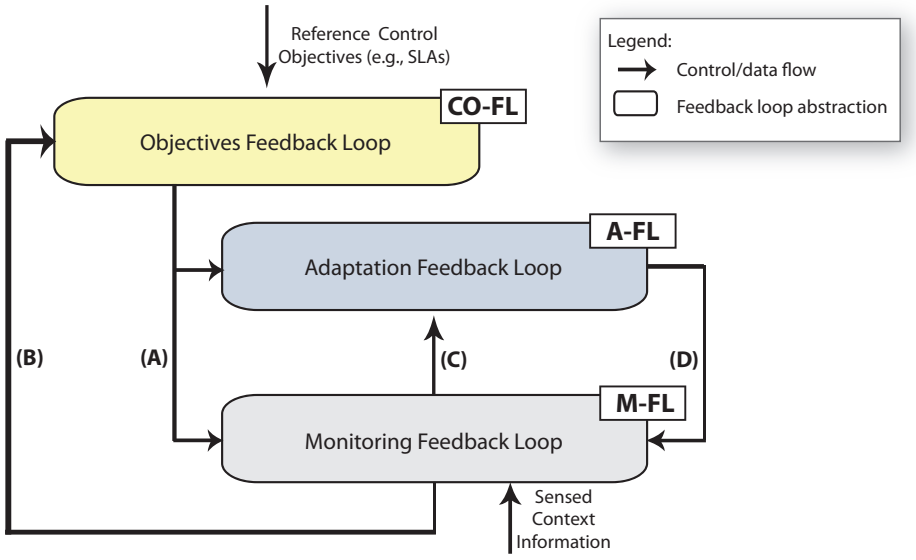


Figure 7.2: The three levels of dynamics in context-driven self-adaptive software. The control objectives feedback loop, (CO-FL), controls changes in adaptation goals and monitoring requirements to ensure their fulfilment. The adaptation feedback loop, (A-FL), controls the adaptive behavior of the target system and the adaptation mechanism, according to control objectives and taking into account monitored context events. The dynamic monitoring feedback loop, (MFL), manages context information for preserving context relevance of the adaptation mechanism. Labels (A), (B), (C) and (D) highlight the control/data flow among the feedback loops.

instrumentation. In the case where a dynamic adaptation mechanism is necessary, having a self-contained adaptation strategy (i.e., planner and executor) will contribute to the preservation of desired properties.

By applying the separation of concerns introduced in our reference model, it is possible to support three different types of adaptation, depending on the different interactions implemented among the feedback loops: *preventive, corrective* and *predictive*. In preventive adaptation, the dynamic monitoring feedback loop notifies the adaptation feedback loop about context events (*context symptoms*) that, even when they are causing no effects yet in the target system behavior, they eventually will. This is the case for the monitoring condition that evaluates the number of *likes* of an offer placed on a social network. As the offer becomes popular, that is, the number of likes is close to 200,000, a predictive adaptation process can be started to take the system to its medium-load capacity (cf. Table 7.1). Consequently, even though the adaptation subsystem has not detected any disturbances yet for triggering adaptation in the target system, based on this context information, it can minimize the risks of

the goal satisfaction to be violated by performing a system adaptation in advance.

Corrective is the usual type of adaptation that takes place when monitoring mechanisms supporting the adaptation feedback loop detect that adaptation goals are no longer satisfied. In our application example this can occur when the monitoring feedback loop identifies an SLO violation in either the efficiency of the *ProcessingPurchaseOrder* service(s), or the expected minimum number of purchase orders processed per unit of time (cf. Table 7.1). Any of these situations requires from the adaptation controller to perform another, perhaps more aggressive, system reconfiguration, or to apply restrictive mechanisms of use to prevent the system from collapsing before a new adaptation is performed.

Predictive adaptation takes advantage of both, historical information to anticipate risks of goal violation, as well as the identification of plausible symptoms that provide evidence to necessitate adaptation eventually. These symptoms may be presented in the form of patterns of correlated events that potentially become significant advice for adaptation. An example of this latter case in our application scenario is the detection of a low but constant degradation of the *ProcessingPurchaseOrder* service efficiency around significant dates, but without reaching the critical levels that trigger corrective adaptation. Using this historical information, the dynamic monitoring feedback loop can trigger an alert event to indicate or notify the operators that the negotiated performance SLA should be reviewed to keep the system operation in a safe state.

Finally, it is worth noting that in Figure 7.2 despite this separation of concerns, the control objectives feedback loop (i.e., CO-FL in the figure), the adaptation feedback loop (i.e., A-FL, including the target system), and the dynamic monitoring feedback loop (i.e., M-FL) together, also constitute a feedback loop. Figure 7.3 presents the detailed view of the reference model where each level of dynamics is designed as an instance of the general feedback loop with explicit components required for controlling the self-adaptation in software systems.

7.3.2 The Control Objectives Feedback Loop

In Dynamico, the regulation of requirements satisfaction and the preservation of adaptation properties are objectives controlled through the collaboration of the A-FL and the M-FL. We define requirements and adaptation properties as system variables to be controlled. We refer to these variables as *control objectives* and *adaptation goals* interchangeably. These requirements can be functional and non-functional, and the

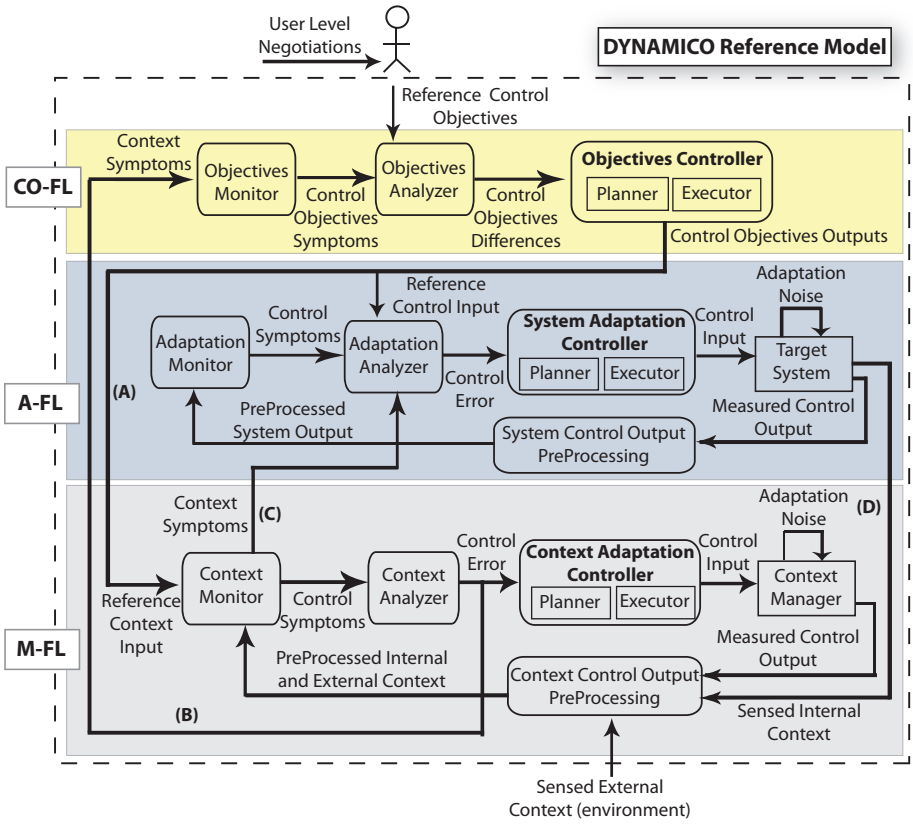


Figure 7.3: DYNAMICO with a detailed view of the controllers for the three levels of dynamics.

target system must satisfy them, depending for this on the adaptive capabilities of the overall system. Adaptation properties refer to the properties that are inherent in self-adaptive software, and thus, all adaptation mechanisms should expose these properties (cf. Chapter 6). As mentioned in Section 7.2, these control objectives are subject to change by user-level (re)negotiations at runtime and therefore must be addressed in a consistent and synchronized way by the adaptation mechanism and the context manager. There may be several causes for these changes. In a first case, service level agreements with dependencies on context situations can imply changes in control objectives at runtime. In our application example, this is the case of the throughput SLO (cf. Table 7.1). This SLO defines two different thresholds. The medium load threshold is applicable to those cases where special product offers are placed online (e.g., on a social network integrated to the business e-commerce platform). After placing the offer, it must be monitored to apply preventive adaptation with the goal of adjusting the system capacity according to the popularity of the of-

fer. Popular offers are expected to affect the e-commerce platform load considerably. Similarly, time context must be monitored to keep track of the shopping seasons to apply preventive adaptation to guarantee the system operation when the system load reaches its highest point (cf. Table 7.1). In another case, when the system is in execution, the initial SLA conditions can be re-negotiated. An instance of this case occurs in the second use case of our application example. After the contracted services for the e-commerce platform have been in production, a new throughput SLA is added to the efficiency SLO agreed initially. Both the throughput and efficiency SLOs are managed explicitly as the control objectives for the adaptive system. Thus, both reference inputs, the A-FL reference control input, and the M-FL reference context input, should be derived automatically from changes in control objectives and fed into the corresponding feedback loops, as illustrated by interaction (A) in Figure 7.3. All of these changes between SLOs and SLAs, which are treated as changes in reference inputs, are governed by the CO-FL.

Nonetheless, this explicit management of control reference inputs has two important implications: (i) it is required to model and express the corresponding properties quantitatively in terms of quality attributes, and (ii) it is necessary to have a mechanism to measure and update these reference inputs at runtime whenever they change. Our evaluation framework, presented in Chapter 6 concerns these two implications. Concerning the second implication, the dynamic adaptation of control reference inputs (control objectives) is addressed by *closing* the CO-FL. In the context of the main loop (the one composed of the three feedback loops), the A-FL receives symptoms from the M-FL through interaction (C), which in turn adapts its behavior according to changes in control objectives to guarantee monitoring relevance along the adaptation process. Under more dynamic scenarios, the A-FL controller may be required also to change its adaptation strategy according to changes in control objectives. Furthermore, as an important concern in service provision is the fulfillment of SLAs as specified in contracts, a plausible way to express and manage these reference goals quantitatively is through *contract management* and its explicit modeling.

7.3.3 The Adaptation Feedback Loop

The adaptation feedback loop, A-FL, serves as a guarantor for regulating the target system's requirements satisfaction and preserving the adaptation properties. Recalling our application example, the efficiency and throughput SLOs represent system's

requirements. Due to the changing nature of SLAs and context situations, the satisfaction of these requirements depends on the adaptive capabilities of the e-commerce platform. Among the adaptation properties applicable to the adaptation mechanism of the application example are settling time, small overshoot, stability, and reconfiguration termination. In particular, settling time, the time it takes for the adaptation mechanism to complete the e-commerce platform reconfiguration, is crucial to guarantee the contracted conditions. Our framework for evaluation quality-driven self-adaptive systems provides a comprehensive catalog of adaptation properties and corresponding quality attributes and metrics (cf. Chapter 6).

A-FL follows the separation of concerns criteria. In turn, these criteria conform to the general protocol of control theory, which relies on quantitative expressions to measure the error in the controlled system variables, and respective reference control inputs for these variables. The A-FL gathers these measurements continuously from the target system through context monitors. These monitors notify control symptoms for adaptation to the A-FL analyzer, which determines whether a system adaptation is required (cf. analyzer in Fig. 7.1). The simplest case for this occurs when the measured variables under control, compared to their corresponding reference control inputs, indicate that some control objective is no longer satisfied. Whenever it is relevant, the A-FL analyzer notifies this fact with the corresponding information to the system adaptation controller. With this information, the planner element selects a strategy to adapt the system for it to re-establish the fulfillment of the violated control objective. A possible result of this strategy is to compute and send a list of system architecture reconfiguration actions to the executor (e.g., a set of distributed services to replace the original *ProcessingPurchaseOrder* service). The executor translates these actions to the specific runtime platform and executes them in the target system, thus closing the main control loop. DYNAMICO and its A-FL can take advantage of any strategy to perform the target system adaptation.

7.3.4 The Context Monitoring Feedback Loop

The role of the monitoring feedback loop, M-FL, as an independent feedback control loop is crucial for improving self-adaptivity given the dynamic nature of context information. In a *context-based self-adaptive system*, a context manager must be able to make decisions based on past, current and foreseeable future states of context. It must analyze context symptoms and facts to support the system adaptation and the

management of control objectives. Moreover, the monitoring mechanism must adapt itself to support new context management requirements as the common control objectives are re-negotiated, or the adaptive system evolves. For instance, the context manager for the application example must be able to deploy new context management instrumentation. In the first use case, the deployment of the new set of distributed services, caused by the adaptation of the e-commerce platform, will trigger the deployment of a new set of time behavior sensing interfaces to keep track of the new services' peformance. In the second use case, the re-negotiation of the performance SLA will trigger the deployment of the monitoring infrastructure required to keep track of two new types of context information, the shopping season (i.e., time context according to our general SmarterContext ontology presented in Chapter 4, and the special product offer (i.e., artificial context).

The M-FL in Figure 7.3 represents a context manager that supports dynamic monitoring. The reference context inputs correspond to the reference context management objectives derived from the CO-FL reference control objectives. In the application example these reference context management objectives define the context monitoring requirements, which are derived from the metrics defined in the performance SLA. Context monitors are in charge of gathering primary context information from the internal and external environment, and the correlation of this information to infer either, context symptoms that can affect the target system adaptation process (provided to the A-FL through interaction (C) in Figure 7.3), or control symptoms to decide about the context manager adaptation. This information is pre-processed by the context control output preprocessing element to generate numeric observables from physical and logical sensors, and producing comparable measures by performing basic transformations on them.

The context analyzer performs the context handling process required for the context adaptation controller to decide about adapting the monitoring strategy, and for the CO-FL to decide about changing the system objectives (interaction (B)), as demanded by the current state of the environment and the self-adaptive system requirements. The change of control objectives can be performed fully- or semi-automatically, depending on whether it is necessary to re-negotiate the contracts, and consequently, for the user to intervene. The context adaptation controller is responsible for defining and executing the adaptation plan for the context manager, according to its adaptation strategy.

Finally, the measured control output and the target system's internal context are

used to ensure the context manager goals, thus supporting the system adaptation process and the management of the system control objectives.

The explicit identification of adaptation properties with corresponding quality attributes and metrics, as supported by the catalog that we proposed as part of our evaluation framework, allows us to identify context monitoring requirements. In our dynamic SOA governance motivation example, control objectives correspond to the SLOs defined as part of the performance SLA. We specify SLAs in the form of RDF/XML documents whose semantics are defined by SmarterContext ontologies that we created for characterizing the entities of control objectives and monitoring strategies. Each SLO corresponds to a quality factor (e.g., capacity), which in turn is measured through one or more metrics. Each metric is composed of one or more sensing variables and one metric expression that involves all the variables. Each sensing variable is mapped to a specific context type and a physical sensor. We synthesize monitoring strategies at runtime from the metrics defined in the control objectives specification, the SLA in this case. Each sensing variable maps to a sensing interface (i.e., sensor service), whereas metric expressions map to monitoring conditions (i.e., the monitoring logic of context monitors). Upon the definition of new SLAs or the re-negotiation of existing ones, the M-FL analyzer depicted in Figure 7.3 calculates changes in monitoring requirements. Then, the planner element of the M-FL generates the adaptation plan that will modify the monitoring strategy by deploying new, or modifying existing sensing interfaces and monitoring conditions. Finally, the M-FL executor performs the adaptation of the monitoring infrastructure.

7.3.5 Feedback Loop Interactions

In Dynamico, not only are the three described feedback control loops well separated, but also the elements within each feedback loop. However, even though control loops are designed independently of each other, they must operate cooperatively to achieve the overall system objectives.

As depicted in Figs. 7.2 and 7.3, to regulate the satisfaction of the control objectives, Dynamico specifies four interactions among its three feedback loops. These interactions are labeled (A), (B), (C) and (D) in Fig. 7.3. We classify interactions (A) and (B) as *indirect interactions* because they are realized through the CO-FL, whereas interactions (C) and (D) as *direct interactions* due to their direct connections between the M-FL and the A-FL.

Interaction (A) provides the reference context input (i.e., context manager requirements) for the context manager (M-FL) to (i) maintain its relevance with respect to the actual context situation and contracted conditions; and (ii) decide on context management strategies. In the application example, reference context inputs correspond to the context management requirements derived from the SCA specification.

Interaction (B) enables the control objectives manager (CO-FL) to decide about the changes in the control objectives, whenever the M-FL detects that, given the current context, the current set of control objectives should be adjusted or re-negotiated dynamically. Common control objectives are crucial for governing the interactions between the A-FL and the M-FL. We specify common control objectives in the form of contracts, machine readable SLAs specified in RDF/XML, to infer both adaptation and context monitoring objectives. Thus, a context management infrastructure (i.e., M-FL) must be able to infer, from contracts and common control objectives, the context management reference inputs (monitoring requirements), and corresponding monitoring strategies.

Interaction (C) is triggered by context symptoms that are identified and sent from the M-FL context monitor to the A-FL analyzer. These context symptoms, which can be manifested as groups of events presented with different characteristics, are important for decision making in the A-FL. The communication mechanism and the information associated with these symptoms depend on the type of adaptation the system is supporting (i.e., preventive, corrective or predictive). For example, for a predictive adaptation, the M-FL could trigger symptomatic events in advance about whether or not to perform a future adaptation. For a preventive adaptation, the M-FL also sends symptoms, but the adaptation is performed immediately. In contrast, for corrective adaptation, symptoms are either, pushed by the M-FL or pulled by the A-FL depending on who recognizes the need for adaptation (the context manager or the adaptation controller).

Interaction (D) represents the flow of internal context sensed by the M-FL from the adaptive system. Monitoring of internal context information is necessary to assess the system consistency after an adaptation. Moreover, by analyzing internal context information that characterizes the current state of system properties, the M-FL could provide useful information to understand the relationship between context symptoms, achievement of system goals, and the preservation of adaptation properties.

7.4 Governing and Controlling Feedback Loop Interactions

According to our reference model, an adaptive system is defined as a collection of cooperating feedback loops that ensure the achievement of the system objectives under changing context conditions. However, DYNAMICO can be combined with other models for adaptive systems. In particular, the IBM architectural blueprint provides the ACRA model to orchestrate control loops hierarchically for autonomic systems [IBM06, KC03]. Combined with this model, DYNAMICO supports the distribution of functions in a more fine-grained level, that is, at the feedback-loop elements level. More extensive use of knowledge bases, as the ones proposed for the MAPE-K loop, should also facilitate interactions among control loops. Such knowledge bases store historical information such as symptoms, as well as internal and external contextual facts required by the analyzers in any of the three types of control loop. Moreover, these persistence mechanisms help fine-tune contracts and policies to achieve the control objectives. It is worth noting that having common control objectives enable the three control loops to reason consistently about the system goals, and to determine the coordinated control actions on each of them.

Having common control objectives is important to govern the interactions among the feedback loops. Figure 7.4 illustrates DYNAMICO abstracted as a control objectives feedback loop. The A-FL (adaptation mechanism), the M-FL (context manager), and the core controlled target system are abstracted as a whole managed *(super target) system*. This managed system is governed by the CO-FL according to changes in contracted conditions. Reference control objectives (i.e., contracts) are fed into the system through direct user intervention. Changes in these objectives can result from re-negotiations or from context symptoms received through interaction (B) (cf. Fig. 7.2).

According to Fig. 7.4, whenever the objectives change as a result of symptoms received from the context manager, the control objectives monitor perceives these symptoms as symptoms of changes in the current set of control objectives. Then, the CO-FL analyzer makes decisions on the necessity of producing a new set of reference control inputs. If applicable, the CO-FL controller produces a new set of reference control inputs and reference context inputs to be sent to the adaptation mechanism and the context manager, respectively. The measured control objectives feed the system back with information about the achievement of the system control goals.

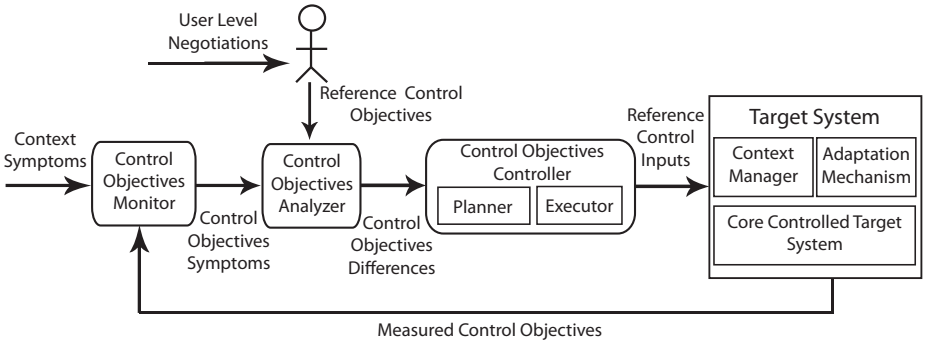


Figure 7.4: DYNAMICO abstracted as a feedback loop for governing changes in control objectives.

Finally, if the control objectives change as a result of a re-negotiation, the user is responsible for providing the control objectives analyzer with the new SLAs, and their corresponding SLOs and context monitoring requirements.

7.5 Possible Dynamico Variations

To deal with out-of-kilter environmental behaviors or perturbations, the control community has developed several variations to modify the control function, such as the Model Reference Adaptive Control (MRAC) and the Model Identification Adaptive Control (MIAC) mechanisms [DH02b, NB97]. The main difference between MRAC and MIAC is how the reference model is defined—in MIAC directly inferred from the running process, whereas in MRAC pre-computed using a mathematical model.

These variations also are applicable to DYNAMICO. Both variations can be realized, for instance, using a rule-based or policy-based reconfiguration approach in the planner element of the system adaptation controller, as illustrated in Fig. 7.5. In this figure, the CO-FL is represented by the control objectives manager. The adjustment mechanism detects, through the measured control output, whether the target system is facing an out-of-kilter environmental perturbation (e.g., an unusual high number of online shoppers during the Black Friday season), or the adaptation strategy is far from being effective. If this is the case, it modifies either the system adaptation planner (in the adaptation mechanism), or the context adaptation planner (in the context manager), depending on the situation.

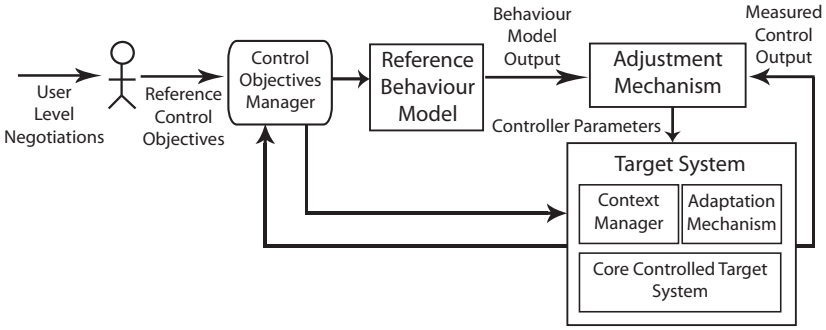


Figure 7.5: Reference model variation for supporting adaptive feedback control loops with reference behavior models.

7.6 Chapter Summary

This chapter presented DYNAMICO, our reference model for engineering highly dynamic adaptive software systems. This kind of systems must deal with and effectively respond to highly dynamic contexts of execution, by evaluating their own behaviour at runtime and reconfiguring themselves whenever the satisfaction of system requirements can be compromised.

A highly dynamic context is characterized by (i) expected and unexpected changes in context conditions such as user location (in mobile software clients), network access point, service throughput and load (on the server side), time and calendar dates, and even user interests associated with specific locations and special dates; (ii) dynamic changes in adaptation goals and user requirements, such as re-negotiation of QoS levels for specific services; and (iii) other sensible changes that affect the satisfaction of system requirements, such as unauthorized intrusions or faults. In addition, all of these changes are assumed as natural requirements to be satisfied by the self-adaptive system at runtime.

DYNAMICO helps cope with this kind of dynamic requirements by defining three types of feedback loops. Each of these feedback loops manages each of the three levels of dynamics that we characterized for self-adaptation: (i) the control objectives feedback loop, for managing changes in adaptation goals and user requirements; (ii) the target system adaptation feedback loop, to deal with changes addressable directly at the target system level; and (iii) the dynamic monitoring feedback loop, to manage changes that require the deployment of different or additional monitoring infrastructures to those already configured for execution, thus maintaining its relevance with respect to the changing adaptation goals. As a reference model, DYNAMICO

reconciles the many visions and contributions of different approaches for the development of self-adaptive software systems, whether they hide or exhibit the elements of feedback control loops. Nonetheless, our reference model emphasizes the visibility of these control elements and constitutes a guide to design self-adaptive systems in which the system goals, the target system itself, or the monitoring infrastructure must be adapted—assuming this is a crucial requirement for the system to be developed. Depending on these requirements, the model can be applied as a whole, with its three feedback loops, or partially, involving only a subset of them.

The next chapter presents our proposal to include V&V tasks explicitly in the adaptation process with the goal of ensuring the achievement of the self-adaptive system's goals and the preservation of adaptation properties, for instance as defined in our catalog of adaptation properties presented in Chapter 6.

Chapter 8

A Model for Explicit Runtime V&V Tasks in Self-Adaptive Systems

Software validation and verification (V&V) ensures that software products satisfy user requirements and meet their expected quality attributes throughout their life cycle. Adaptive software systems modify their behavior to guarantee the satisfaction of functional and non-functional requirements under evolving and possibly uncertain execution environments. While high levels of adaptation and autonomy provide new ways for software systems to operate in highly dynamic environments, developing certifiable V&V methods for guaranteeing the achievement of self-adaptive software goals is one of the major challenges facing the entire research field. This chapter presents our sixth contribution, our reference model to include V&V operations explicitly in the adaptation process for ensuring the achievement of software self-adaptation goals. This chapter also presents our analysis of fundamental challenges and concerns for the development of V&V methods and techniques that provide certifiable trust in self-adaptation. Both of these contributions are valuable to help V&V researchers advance this field.

To explain the concepts presented in this chapter, we use a variation of the dynamic SOA governance scenario described in Chapter 7. In this example, a performance SLA has been negotiated in terms of three throughput service level objectives (SLOs) to guarantee three different levels of system capacity in a cloud-based e-commerce platform: normal, medium and high load. These SLOs are observed on

the bottleneck-operation of the system, *ProcessingPurchaseOrders*, and measured in terms of number of transactions per time unit. A *normal capacity* is required for a regular load of the shopping platform. A *medium capacity* is required when special offers are placed on social networks promoting them. A *high capacity* must be guaranteed to deal with the highest peak load of the platform caused by shopping seasons such as "Black Friday".

As in the original scenario, the *adaptation goal* for this dynamic service-oriented infrastructure is the contracted system capacity, in terms of the performance SLA. *Short settling time* and *consistency* correspond to the *adaptation properties* to be preserved. According to our catalog of adaptation properties presented in Chapter 6, settling time is defined as the time required for the adaptive system to achieve the desired state. Consistency guarantees the structural and behavioral integrity of the managed system with respect to the respective service component architecture (SCA) integrity constraints after its adaptation.

Use Case 1: Controlling the Elasticity of the E-Commerce Platform. As efficiency is a major concern, the capacity of the system must be either increased, or decreased according to the context situations that determine the expected load of the system. To accomplish this, context monitors must keep track of the popularity of special offers placed on social networks, as well as the day of the year to determine the applicable shopping season.

Use Case 2: Re-negotiating Adaptation Goals at Runtime. After the e-commerce platform has been in operation, a new set of SLOs is added to the performance SLA. These new SLOs define different thresholds of response time that must be guaranteed according to the classification of the e-commerce platform's customers. Customers are classified as regular and premium users. A particular *maximum response time* threshold applies to regular customers. For premium customers, the maximum response time must correspond to 90% of the threshold defined for regular customers. Response time thresholds can be re-negotiated at runtime.

This chapter is organized as follows. Section 8.1 introduces the *viability zone* of situation-aware smart software (SASS) systems, concept that is crucial for the assurance of self-adaptation. Section 8.2 discusses the *what, where* and *when* of self-adaptive systems assurance. Section 8.3 presents our proposed reference model to

make V&V tasks explicit in self-adaptation. Section 8.4 discusses runtime V&V enablers required for the assurance of adaptive systems. Section 8.5 summarizes the chapter.

8.1 The Viability Zone

We define the *viability zone* of a SASS system as the set of possible system states in which the system operation is not compromised [ABSP11], that is, the set of states where the system's requirements and desired properties (i.e., adaptation goals) are satisfied. Viability zones can be characterized in terms of relevant context attributes and corresponding desired values. These context attributes correspond to either measurements of internal variables of the target system or the adaptation mechanism, or environmental variables whose variations can take the system outside its viability zone. Viability zones are N-dimensional. Therefore, a particular SASS system may have more than one associated viability zone (e.g., one for each adaptation goal). The global viability zone of a system thus results from the composition of these partial viability zones. Moreover, existing viability zones can be added, replaced or adjusted by adding or removing variables of interest at runtime.

In the case of our application example, the initial viability zone is defined in terms of the performance SLA, and the three throughput SLOs (normal, medium and high capacity). These three SLOs constitute three different levels of system capacity that can been interpreted as three viability sub-zones. The variables that characterize the e-commerce platform's viability zone correspond to the actual throughput of the *ProcessingPurchaseOrders* operation, the popularity of special offers placed in a social network (including whether an offer has been placed), and the shopping season, all of them to be monitored at runtime. Seasons are characterized in three groups: regular, medium (e.g., Valentine's Day), and high seasons (e.g., Christmas and Black Friday). Another associated viability zone in this example is used to control the short settling-time adaptation property. This zone is defined by a single-variable that is monitored to keep track of the time the e-commerce platform takes to reconfigure the system to obtain the desired throughput. Furthermore, after the re-negotiation of the performance SLA, a new viability zone must be computed at runtime to control the response time SLOs as defined in Use Case 2.

8.1.1 V&V under Viability Zone Dynamics

The viability zone can change with context changes, as opposed to the solution space concept, which is assumed to be fixed. In effect, the viability zone of a target system under adaptation constantly varies along adaptation dimensions. These variations take place every time the adaptation operation modifies either the target system architecture (e.g., adding or removing components and connectors) or the controller itself (e.g., modifying its parameters or replacing the control algorithm), thus introducing new, or removing existing variables and associated domain types.

Therefore, not only are runtime V&V methods required to cope with the viability zone dynamics problem, but these V&V methods also need to be automatically generated according to the modifications that result from dynamic adaptation. Thus, to extend the V&V coverage of the expanded viability zone, runtime models are required for the incremental derivation of software artifacts for V&V monitoring and checking.

In the aforementioned example, understanding its viability zone dynamics is crucial for the self-adaptive e-commerce platform V&V tasks. In fact, the adaptation mechanism together with its V&V tasks can be interpreted as an optimization problem, where the optimal solution is chosen among those within the viability zone, based on the system capacity policies, as proposed by Balasubramanian *et al.* [BDM+11]. First, transitions between viability sub-zones are associated with an adaptation policy (adaptation strategy). For instance, when the system is approaching the threshold between a lower and a higher load—going from the lower to the higher, the corresponding adaptation task must be triggered to increase the system processing capacity accordingly (e.g., by deploying new components for scalable processing). Similarly, the system capacity must be reduced when it goes from a higher load to a lower one. In both cases, as the software component structure is modified as a result of the adaptation, the consistency property (cf. our catalog of adaptation properties in Section 6.2.2) must be verified at runtime on the resulting system. For example, by checking the structural conformance of the software architecture with respect to the SCA specification. Second, changes in viability zones (e.g., changes in variables' thresholds, and addition or replacement of variables in adaptation dimensions) may affect not only the adaptation strategy, but also the monitoring infrastructure, since these changes are caused by changes in adaptation goals. Finally, runtime V&V tasks aim to keep the adaptive system inside its viability zone, even when viability zones are subject to changes at runtime. The way how V&V tasks contribute to achieve

this goal depends on the nature of the system and its requirements. For instance, for safety-critical applications, runtime V&V must check if the system will trespass the boundaries of its viability zone as a result of an adaptation, before instrumenting it in the running system. In those cases where self-adaptation is interpreted as an optimization problem, V&V tasks can be used both before the adaptation and after it. Before the adaptation, to restrict the alternatives to consider to those within the viability zone. After the adaptation, to ensure that the solution is satisfying the new requirements under possibly changed context situations.

8.2 The What, Where and When Questions of Runtime V&V

We identified the underlying V&V questions in the domain of self-adaptive systems as *what*, *where*, and *when* to validate.

8.2.1 What: Requirements and Adaptation Properties

The answer to the what question concerns the identification of adaptation goals (e.g., non-functional requirements of the target system) and adaptation properties (e.g., desired characteristics of the adaptation mechanism). Adaptation properties and goals correspond to control objectives specifications in our DYNAMICO reference model (cf. Section 7.3.2). Explicit adaptation goals and properties are crucial for the implementation of V&V tasks for self-adaptation, and the identification of the metrics required to decide about the assurance of the system. Moreover, having an explicit mapping between adaptation goals and properties, and relevant context is required to ensure the coherence between V&V tasks and the relevant context variables that characterize the system's viability zone.

8.2.2 Where: Separation of Concerns

We distinguish two system levels in self-adaptive systems: the target system to be dynamically adapted according to context changes, and the adaptation mechanism. For runtime V&V it is critical to understand the extent of the separation of these two levels. This separation of concerns allows us to characterize, investigate, and analyze

V&V research problems for self-adaptive software effectively, by focusing specifically on the respective concerns of each level.

Although the discussion in this chapter is applicable to both feedback and feedforward control in computing systems [HDPT04], we focus on feedback control since runtime V&V depends on online measurements from the target system and the adaptation mechanism. That is, measured outputs are important for making adaptive system quality decisions at runtime. Moreover, as feedforward control takes also environmental disturbances—external context—into account, subsequently we use the terms feedback loop and control loop interchangeably. Following the feedback loop abstraction from the V&V perspective, the target system is an open loop for which the adaptation mechanism provides the elements to close the loop. In other words, the target software system itself is unaware of both context conditions and self-performance, with respect to the satisfaction of its own functional and non-functional (context-dependent) requirements. Thus, given that the objective of V&V is to guarantee the quality of a system, and this quality is expressed as the fulfillment of its requirements, in self-adaptive systems V&V tasks must be incorporated as part of the adaptation loop. This implies that, in addition to the common context monitoring elements considered in feedback adaptation loops, additional components dedicated to verification and testing of the target system itself are required. At runtime, these components could, for instance, perform partial and incremental model checking on the next most probable states with respect to the current system state. Referred to our application example, the property to be verified could be the structural conformance of the reconfigured software application, with respect to the SCA structural constraints.

In addition, the separation of concerns between the target system and the adaptation mechanism implies different possible V&V interactions among these two system levels. Each of these interactions affects, in different ways, the ultimate goal of self-adaptation: the continued and effective operation of the target system services under varying context conditions. Of course, a general requirement is that the adaptation mechanism executes as unobtrusively and independently as possible from the target system. This observation has two implications. First, the target system functionalities must execute uninterruptedly for as long as possible while the adaptation mechanism performs the required adaptations on these functionalities. Moreover, the target system is expected to remain functional even if the V&V fails (i.e., if it indicates that the new system state is invalid). This implies that the adaptation mechanism must

also run without interruptions. Second, unavailability of the adaptation mechanism should not cause unavailability of the target system. However, at some point, it is reasonable to expect that the adaptation mechanism, and even the target system itself, will require a shut down for maintenance or correcting system failures.

8.2.3 When: V&V in the Adaptation Process

Traditional V&V strategies involve checking and testing before system deployment under presumably well-defined conditions of system operation. This process of checking and testing is often automated using model checking, theorem proving, and testing tools. For context-dependent requirements, traditional V&V activities and certification techniques, designed to be applied before system deployment on fully specified requirements, are neither sufficient nor applicable. On the one hand, these formal V&V methods are often too expensive to be executed regularly at runtime when the system adapts due to their time and space complexity. On the other hand, context-dependent variables are unbound at design time, but bound at runtime. Thus, performing V&V on these variables at runtime is valuable to reduce the verification space significantly, even when the self-adaptive system's viability zone varies with context changes. From this perspective, it is crucial to determine precisely when in the adaptation process these V&V operations are to be performed to guarantee the system properties and prevent unsafe operation.

In addition, the considerations discussed in the previous subsection (i.e., the where) require the analysis of at least the following questions with respect to *when* to perform V&V tasks:

i. What properties can be exclusively verified at design time (executing neither the target system nor the adaptation mechanism)?

ii. What properties can be exclusively verified or tested at system configuration time?

iii. What properties can be exclusively verified or tested at runtime?

iv. What properties can be verified or tested either at design time, configuration time, or at runtime?

The answers to these questions are highly interdependent. For example, an approach aimed at verifying stability (what)—a behavioral adaptation property of the

adaptation mechanism—may require the assessment of performance quality factors such as latency, throughput and capacity. These factors assume runtime (when) monitoring on the target system (where). Stability is defined as the convergence of the subject system behavior toward a desired state. Moreover, many of the design concerns, such as availability, performance, survivability, fault tolerance and security, are highly interdependent and evolve at discrete points in time. It is critical to separate these concerns at design as well as at runtime.

In our application example, the performance SLA and its SLOs (throughput and response time—cf. Use Case 1 and Use Case 2 in the introduction of this chapter), as well as the settling time and SCA structural conformance properties constitute the *what* to validate. Regarding the *where* question, throughput and response time must be observed on the target system, settling time must be observed on both the adaptation mechanism and the target system, whereas the SCA structural conformance, on the target system. Finally, concerning the *when* question, V&V tasks to ensure these requirements and properties must be performed at runtime.

In the following section we give some answers to the *when* and *where* questions by extending the feedback-loop elements with V&V responsibilities.

8.3 Making V&V Explicit in the Self-Adaptation Loop

So far, we have analyzed four key V&V drivers for self-adaptive systems that pose major research challenges for self-adaptive systems research communities relate: (i) the viability zone and its dynamics; (ii) what to validate and verify, and its dependency on context information; (iii) where to validate—closely related to the separation of concerns between the target system and the adaptation mechanism; and (iv) when to perform V&V in self-adaptive systems with respect to the adaptation loop.

To advance the assurance of self-adaptive systems, we argue for the integration of runtime V&V tasks in the adaptation process. Accordingly, this section presents our proposal for making V&V tasks explicit in the elements of feedback adaptation loops, as for example in the MAPE-K loop [KC03]. Moreover, we discuss runtime V&V enablers (i.e., requirements at runtime, models at runtime, and dynamic context monitoring), which provide effective support to materialize V&V assurances for self-adaptation. Our proposal, depicted in Figure 8.1, clearly answers *when* and

where concrete V&V tasks can be implemented in the adaptation loop, using these enablers. The V&V enablers—dashed boxes in this figure—also provide a guide for other SEAMS-related research communities to contribute with runtime V&V methods for self-adaptive systems. With this proposal we contribute to the convergence of these research communities towards the realization of suitable assurance mechanisms for self-adaptive systems.

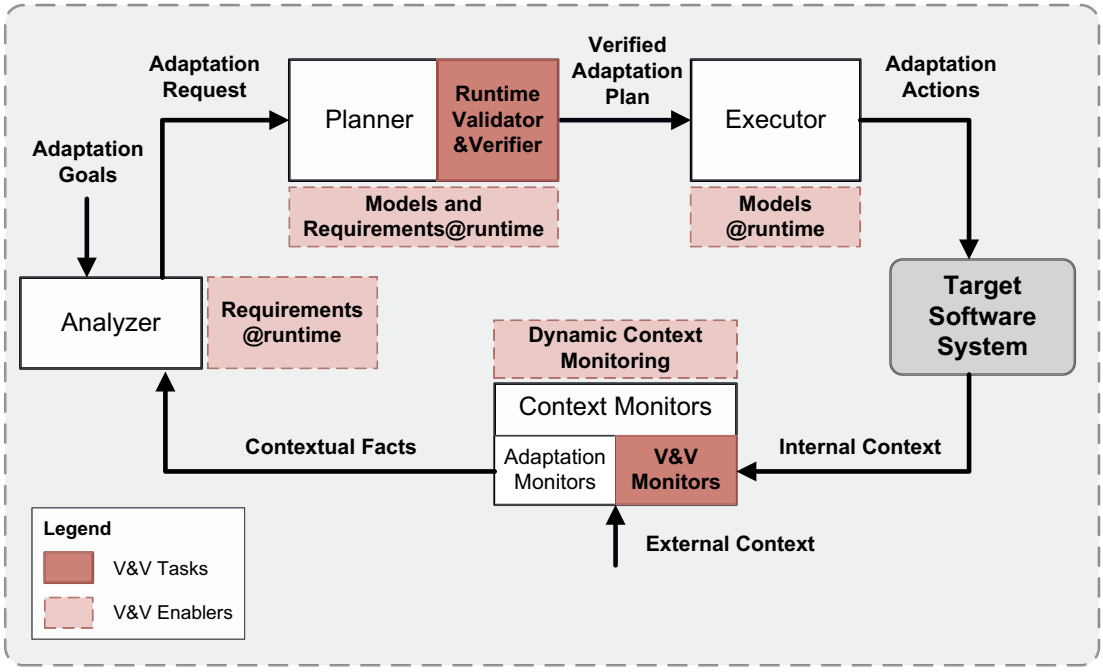


Figure 8.1: Runtime V&V tasks made explicit in the self-adaptation loop

Applying this proposal to our application example, we use requirements at runtime to represent machine-readable specifications of the performance SLA, and its throughput and response-time SLOs. In this way, runtime validators and verifiers can have access to the requirements and properties defined as adaptation goals that must be ensured by the adaptation process. Then, models at runtime can be used to represent both, the software architecture of the e-commerce platform to be adapted, the adaptation strategies, and the context monitoring strategies. Throughout the adaptation process, planners, runtime validators and verifiers, and executors can use SCA models to manipulate and adapt the system's software architecture, as well as to verify properties such as the SCA structural conformance. Similarly, we use models at runtime to represent the information gathered by context monitors as contextual RDF graphs. We exploit this form of representing context information to characterize

the e-commerce platform state with respect to its viability zone.

We identify two particular elements in the adaptation loop that initiate runtime V&V tasks: *runtime validators & verifiers* (associated with the `Planner` element), and *V&V monitors* (associated with `Context Monitors` elements).

8.3.1 The Runtime Validator and Verifier

The responsibility of the *runtime validator & verifier* elements is to verify each of the outputs (i.e., adaptation plans) produced by the adaptation planner with respect to the properties of interest. The instrumentation of an adaptation plan on a given system execution state implies a change of the system state. Thus, the verification of these properties can be performed before or after instrumenting the plan.

In the case of our application example, concerning the SCA structural conformance property, the produced reconfiguration plans modify the target system's software architecture to obtain a new software structure to satisfy the agreed SLOs. To prevent execution failures, these plans must be verified, *before* instrumenting them, in such a way that the resulting structures satisfy the SCA integrity constraints defined in the standard (e.g., components, connectors, wires, and bindings). However, if the adaptation plan is for affecting the performance SLO, the corresponding verification should be performed *after* its instrumentation, with the new system's performance measurements. Moreover, on the new target system structure, partial and incremental verification also could be performed *in advance* on the most probable states that are immediately adjacent to the one generated by the adaptation plan. These states could be computed with statistical approaches such as the proposed in [SG06]. In addition, similar verification could be performed on the controller algorithm, if this is object of adaptation, such as in self-tuning control approaches [DH02a, EGMT09].

Nonetheless, different performance and synchronization issues between the executor and the runtime validator & verifier elements may appear. An example of this occurs when considering the previously introduced *in advance* partial and incremental V&V of the structural conformance property. In this case, the idea is to perform V&V not only on the state produced by the adaptation plan, but also on the most probable states that can immediately follow it, as a result of further adaptation processes. Thus, runtime V&V elements could verify the property of interest on these states either *at the same time*, or *after* instrumenting the plan to reach the produced state. In other words, if the function computing the next most probable states is correct, the

system structure to be obtained with the produced adaptation plan had to be verified in the previous adaptation. Alternatively, the execution of this "advance" runtime verification can be delayed, and even scheduled for later execution, for instance if this verification is highly time-consuming and can compromise the performance SLO of the target system.

Finally, in those cases in which the system state is represented and maintained explicitly (i.e., having a stateful representation via, for example, reflection or models at runtime), and the system is modified by an adaptation, this explicit state has to be transformed or updated accordingly.

8.3.2 The V&V Monitors

V&V monitors are responsible for monitoring and enforcing the V&V tasks performed by the runtime validator & verifier elements. Referring to the V&V tasks assigned to the runtime validator & verifier elements in our example, we could use the V&V monitors to perform the aforementioned "advance" runtime verification. As outlined in the previous subsection, this is a verification task that can be scheduled by the *runtime validator & verifier* elements for later execution, to be performed on the most probable states to the current one in execution.

8.3.3 Assurance of Runtime V&V Tasks

Derived from the previous discussion, we identify the following questions, which pose additional challenges for ensuring the effectiveness of V&V tasks.

What if V&V fails or provides a negative answer? To prevent the target system from reaching inconsistent states and avoid catastrophic situations, one first strategy is to guarantee the atomicity property in the adaptation process, as defined in our catalog of adaptation properties (cf. Section 6.2.2). That is, to guarantee that the adaptation process is an atomic operation that finishes and successfully modifies the target system, or it fails and the target system is left unmodified in its previous safe state. The verification of the atomicity and termination properties is a challenging problem, given that they should be guaranteed internally by the planner, and possibly requiring interactions with the executor.

How can we validate "snapshots and transitions between states without affecting the target system? V&V tasks must not affect the desired behaviour of the adaptive system. Therefore, we identify another kind of properties—properties of runtime V&V methods, intended to support the safe integration of traditional V&V techniques and mechanisms into the adaptation loop. These properties include *sensitivity, isolation, incrementality, and composability.*

As stated by González *et al.*, sensitivity and isolation refer to the level of runtime validation that a particular self-adaptive system can support [GPG09]. On the one hand, sensitivity defines the degree to which V&V tasks (e.g., runtime testing operations) interfere with the running target system. That is, the degree to which runtime V&V may affect the satisfaction of system requirements and adaptation goals. Instances of factors that can affect runtime test sensitivity are (i) component state—not only because runtime validation tasks are influenced by the actual state of the system, but also because the state of the system can be altered as a result of V&V operations; (ii) component interactions—as the runtime testability of a component may depend on the testability of the components it interacts with; (iii) resource limitations—because runtime V&V may affect non-functional requirements on the target system, such as performance at undesirable levels; and (iv) availability—as runtime validation can be performed depending on whether testing tasks require exclusive usage of components with high availability requirements.

On the other hand, they also define isolation as the means to counteract runtime test sensitivity. Techniques for implementing test isolation include (i) state separation (e.g., blocking the component operation while testing takes place or performing testing on cloned components); (ii) interaction separation (e.g., blocking component interactions that may be propagated due to results of test invocations); (iii) resource monitoring (e.g., indicating that testing must be postponed due to resource unavailability); and (iv) scheduling (e.g., planning V&V executions when the target system and involved components are less used).

8.4 Runtime V&V Enablers

Runtime V&V techniques for self-adaptation require special support to deal with the dynamic nature self-adaptive systems in the assurance of adaptation goals. We classify this support into three main categories, as follows:

i. Enablers for the management of adaptation properties and requirements at runtime;

ii. Enablers for the exploitation of models at runtime; and

iii. Enablers for dynamic context monitoring.

Clearly, these categories correspond to selected challenges of the *Models@runtime* [BBF+10] and *Requirements@runtime* [SBW+10] communities, rather than research challenges of V&V communities. Nevertheless, given that runtime V&V tasks for self-adaptive systems rely on this support, with this categorization we aim to provide valuable guidance, not only for V&V researchers to understand the support that runtime V&V for self-adaptation requires, but also for self-adaptive systems related researchers to visualize how could they attack runtime V&V challenges.

8.4.1 Requirements and Adaptation Properties at Runtime

The first category of support required for runtime V&V concerns the specification of *what* is to be validated and verified. That is, the specification of the adaptation properties and system requirements the adaptation process must guarantee. For this, software engineers can use our catalog of adaptation properties mapped to quality attributes (cf. Chapter 6). In either case, V&V methods and techniques must determine whether the software product satisfies its requirements, especially after performing adaptation operations. These requirements and properties, expressed using different notations and formalisms, constitute the actual reference specifications for V&V tasks to accomplish their mission. Thus, requirements and adaptation goals must be available as machine-processable specifications (cf. Requirements@runtime in Fig. 8.1) to be used by adaptation analyzers, monitors, validators and verifiers. Furthermore, to minimize the impact of runtime V&V tasks on the adaptive system, support for tracing changes on requirements and properties also is required to identify what to validate and verify incrementally.

8.4.2 Models at Runtime

Having machine-processable models at runtime of the target system provides adaptation controllers, monitors, validators and verifiers with up-to-date structural and behavioral representations of the target system, and the relationships between these

representations and adaptation properties and goals. Recalling our application example, after the renegotiation of the performance SLA (cf. Use Case 2 in the introduction of this chapter), the runtime representations of the system and its requirements must change accordingly. That is, a new requirement is added to the context-driven SLA specification, as well as the corresponding monitoring strategy, using a contextual RDF graph. As a result, not only adaptation components, but also V&V tasks and monitors will have up-to-date representations of the new goals that must be ensured, and the corresponding context entities to be monitored.

Model Evolution

Having an explicit representation of the target system, the properties to be preserved, and the relationships between these properties and adaptation mechanisms is critical for the assessment of self-adaptive systems at runtime. At design time, models provide a meta-level representation of these concerns. At execution time, instances of these design time models, models at runtime, provide up-to-date representations of the system to V&V operations. These online representations support decision making on the preservation of desired properties. Since self-adaptive systems are continuously changing, the effectiveness of runtime V&V tasks depends on the timely coherence between the actual system state and its runtime models. Model evolution support is therefore required to preserve the coherence of runtime models with respect to the system and its environment.

Model evolution for self-adaptive systems can borrow relevant ideas from control-based approaches. These approaches include model reference adaptive control (MRAC) and model identification adaptive control (MIAC) [DH02a, MKS09]. MRAC and MIAC not only separate adaptation models from adaptation controllers, but also V&V models from V&V tasks. As illustrated in Figure 8.2, MRAC and MIAC enable a basic level of model evolution by modifying the adaptation and V&V models at runtime. The main difference between MRAC and MIAC, from the perspective of runtime V&V, is that in MRAC changes in models are controlled by users, whereas in MIAC changes in models are managed by executors as defined by runtime validators and verifiers. Changes in models (cf. label ChM: Change model) may cause changes not only in adaptation controllers, but also in V&V tasks (cf. labels SCh: Send changes, and AC: Adapt controllers). Therefore, runtime support is required to adapt runtime validators and verifiers accordingly (cf. label AC-VV). Changes

in V&V models could be triggered by adaptation mechanisms. In any case, these changes must be subject to V&V operations. From a software engineering perspective, the probabilistic approach to model synchronization proposed by Epifani *et al.* constitutes a good MIAC approach to model evolution [EGMT09].

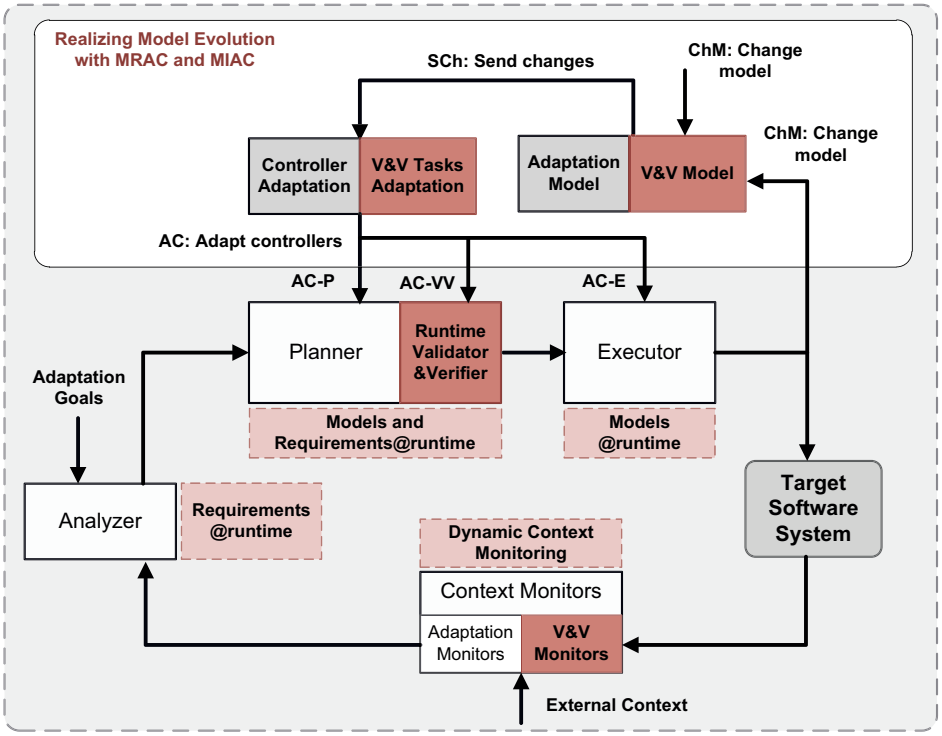


Figure 8.2: Realizing model evolution with MRAC and MIAC in runtime V&V

Different model evolution mechanisms can be applied depending on the modeling technique used. For example, to synchronize UML models with corresponding systems, the model-driven engineering community provides model transformation techniques applicable at runtime [VNH+10]. In systems relying on probabilistic models, synchronization of models is realized by changing the model's parameters at runtime. One key challenge in probabilistic models is to synchronize the measured probabilities with the probabilities used in the model [EGMT09].

8.4.3 Dynamic Context Monitoring

The third category of V&V support corresponds to runtime context monitoring. Context monitoring is crucial to optimize the assessment of dynamic software systems as the effectiveness of V&V methods is highly dependent on the information provided by

context sources. These context information sources must be consistent with the actual system adaptation properties and requirements. Thus, for V&V tasks to succeed in the assessment of a self-adaptive system, it must understand the situations of relevant context entities and their implications for the preservation of system properties and requirements, even when these requirements and properties vary over time. Dynamic context monitoring for supporting the runtime V&V of self-adaptive software has been a major motivation for this dissertation.

From the perspective of runtime V&V, runtime monitoring must support context representation and monitoring to characterize the system's state with respect to its viability zone, taking into account the dynamic nature of viability zones. Regarding context representation, operational specifications of context information must be able to represent semantic dependencies among properties and requirements to be satisfied, V&V strategies, and the environmental situations that have impact on the system behaviour and the assessment tasks. Hence, an important challenge refers to context representation such that these specifications can adapt dynamically, according to changes in V&V concerns. Our SmarterContext solution addresses this challenge since it supports the modification of contextual RDF models at runtime.

With respect to context management, an important challenge is the instrumentation of monitoring infrastructures with dynamic capabilities to deploy new monitoring strategies at runtime according to changes in V&V concerns (e.g., to deploy new sensing interfaces and context monitors based on changes in adaptation goals and properties dynamically). The dynamic capabilities of our SmarterContext infrastructure, enabled by our Dynamico reference model (cf. Chapter 7), address this requirement. SmarterContext deploys new sensing interfaces and context monitors upon changes in control objectives specifications, that is, adaptation properties and goals in the form of V&V concerns. In our application example, the monitoring infrastructure must be adapted at runtime to deploy new monitoring strategies derived from the new response time SLO that resulted from contract re-negotiation (cf. Use Case 2 in the introduction of this chapter).

Runtime monitoring could help alleviate issues concerning the application of traditional V&V techniques at runtime. An instance of such issues is the state explosion problem inherent in model checking techniques. In adaptive software systems, the uncertainty of the execution environment, the dynamic nature of system requirements, and the continuous adaptation of systems exacerbate the problem. From this perspective, we argue that if we are able to characterize the current state of a system at

a specific time during its execution, the number of system states to be checked could be significantly smaller. At design time many variables are free or not bounded, thus all of their possible significant values must be checked. In contrast, at runtime, variables are bound using the actual system state and the situation of relevant environmental (internal and external) entities. In other words, the number of possible states for the system to maintain within its viability zone is considerably reduced by the current and next most probable context situations. This is precisely where *context monitoring* plays a crucial role in the assessment of self-adaptive software systems. Nevertheless, due to the uncertainty inherent in dynamic software systems, it is infeasible to specify context requirements in advance exhaustively. Moreover, since context is evolving over time, monitoring requirements—entities to be monitored and monitoring conditions—also are continuously evolving. Therefore, the application of traditional V&V techniques to the assessment of self-adaptive systems at runtime depends on the dynamic capabilities of the runtime monitoring instrumentation.

8.5 Chapter Summary

This chapter presented our proposal to make V&V tasks explicit in the adaptation process, and discussed key challenges for the development of certifiable runtime V&V methods that can certify adaptation mechanisms. Our reference model is the first contribution that has been proposed to address the integration of runtime V&V methods as concrete tasks to be performed by elements of the adaptation process.

To propose the reference model presented in this chapter, we identified and discussed key factors and challenges to consider when tailoring existing V&V methods, or developing new ones, to be applied at runtime in self-adaptive systems; analyzed runtime assessment concerns from the perspective of *when* in the adaptation process, and in which of the two parts of an adaptive system (i.e., the *where*)—the target system or the adaptation mechanism—the V&V tasks must be implemented and performed; and proposed different possibilities to integrate these V&V methods as responsibilities for the adaptation process elements. To enable this integration, we analyzed how to exploit some of the foundational ideas developed by research communities related to self-adaptive software to support the application of V&V methods at runtime.

With this reference model and the discussions presented in this chapter we provide researchers from various runtime V&V communities with research avenues that

can shape the development of certifiable assurance techniques, as required for the engineering of trustworthy self-adaptive systems.

The next chapter presents the last contribution of this dissertation, the implementation of the SmarterContext dynamic infrastructure, including its context reasoning engine, and how it supports runtime self-adaptation to address the changing nature of monitoring requirements.

Chapter 9

Implementation of the SmarterContext Infrastructure

This chapter presents design and implementation details of SmarterContext and summarizes the functionalities of its reasoning engine and the components of its generic architecture. Both the SmarterContext generic architecture and reasoning engine (CoRE) are extensible to different application domains of situation-aware smart software (SASS) systems. We implemented SmarterContext to: (i) support changes in context management strategies through the self-reconfiguration of both the architecture of its monitoring infrastructure and the business logic of its monitoring conditions. These changes may imply the deployment of new context gatherers and context processing components, or the modification of existing monitoring conditions; (ii) support adaptive reasoning through the addition of new context types, semantic web rules, and pattern-based rules at runtime, without requiring the manual deployment of new software components; and (iii) assist users in gathering, exploiting, maintaining and controlling the access to their personal context information.

This chapter is organized as follows. Section 9.1 presents the SmarterContext generic architecture and how it was derived from our Dynamico reference model. Section 9.2 explains the implementation and reasoning mechanisms of the SmarterContext CoRE, and presents its optimized version intended to improve processing time when reasoning on large context repositories. Section 9.3 explains Surprise, the SmarterContext module that realizes user-driven privacy and security for context data stored in PCSs. Finally, Section 9.4 summarizes and concludes the chapter.

9.1 Realizing Dynamic Context Monitoring

Dynamic context monitoring corresponds to the third level of dynamics addressed by our DYNAMICO reference model as one of the design drivers in the engineering of self-adaptive systems (cf. Figure 7.2). This section explains how we applied DYNAMICO to the implementation of our dynamic context management solution.

9.1.1 The SmarterContext Generic Architecture

The SMARTERCONTEXT architecture and its realization are based on the *assembly model specification* for SCA version 1.0 [OSOA07]. SCA defines a programming model for building software systems based on SOA design principles. It provides a specification for both the composition and creation of service components. SCA is independent of implementation technologies and communication mechanisms. For the implementation of components, SCA supports several programming languages and environments. For communication mechanisms, SCA is compliant with various communication and service access technologies such as web services, messaging systems, and remote procedure calls (RPC). *SCA implementations* support the realization and execution of SCA-based architectures. Examples of these implementations are Apache Tuscany,[1] Fabric3,[2] IBM WebSphere [SCLM10], and FraSCAti [SMF+09]. We experimented the implementation of SMARTERCONTEXT using two of these SCA technologies: IBM WebSphere and FraSCAti. We used Java as the programming technology and web services as the communication mechanism.

Figure 9.1 presents the legend for the SCA artifacts used in the architectures presented in this chapter. *Components* are the basic artifacts that implement the program code in SMARTERCONTEXT. *Services* are the interfaces that expose functions to be consumed by other components. *References* enable components to consume services. *Composites* provide a logical grouping for components. *Wires* interconnect components within a same composite. In a composite, interfaces provided or required by internal components can be *promoted* to be visible at the composite level. *Properties* are attributes modifiable externally, and are defined for components and composites. *Composites* are deployed within an *SCA domain* that generally corresponds to a processing node.

[1] http://tuscany.apache.org
[2] http://www.fabric3.org

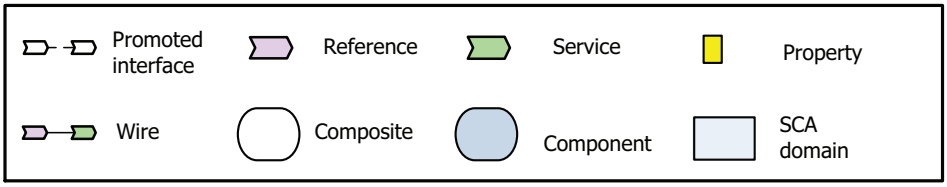


Figure 9.1: SCA artifacts legend

Figure 9.2 presents the general architecture of SmarterContext. `SmarterContext` is the main composite of the architecture and includes four general components. The first component, `SmarterContextGUI`, has three functions. First, it allows users to define and modify personal context information, and privacy and security policies. Second, it enables system administrators to define and modify context reasoning rules. Third, it provides an abstraction of the first level of dynamics in Dynamico (cf. bottom right corner frame in Figure 9.2), the control objectives feedback loop (CO-FL), to allow system administrators to modify system goals (control objectives) at runtime. These changes in control objectives generate changes in the monitoring requirements that must be addressed by SmarterContext dynamically.

Components `ContextManager` and `DynamicMonitoringInfrastructure` implement the management of the context information life cycle. `ContextManager` includes components to integrate context information into context repositories, to maintain and dispose existing contextual data, as well as, for introspection purposes, to update the inventory of components managed dynamically. `DynamicMonitoringInfrastructure` corresponds to the adaptive part of the monitoring infrastructure, is controlled by the monitoring feedback loop, and implements the context gathering, processing and provisioning tasks. The components related to these tasks can be distributed on third party processing nodes, that is, on the computational infrastructure of context consumers and providers. `MonitoringF-Loop` includes the components that implement the feedback loop in charge of controlling the dynamic behavior of our context management infrastructure.

Besides the `SmarterContext` composite, the architecture presented in Figure 9.2 includes other three composites: `AdaptationMiddleware`, `ContextProvider`, and `ContextConsumer`. `ContextProvider` and `ContextConsumer` represent the third parties that exchange context information with SmarterContext. The `AdaptationMiddleware` composite is an abstraction of QoS-CARE [TCCD12, Tam12]

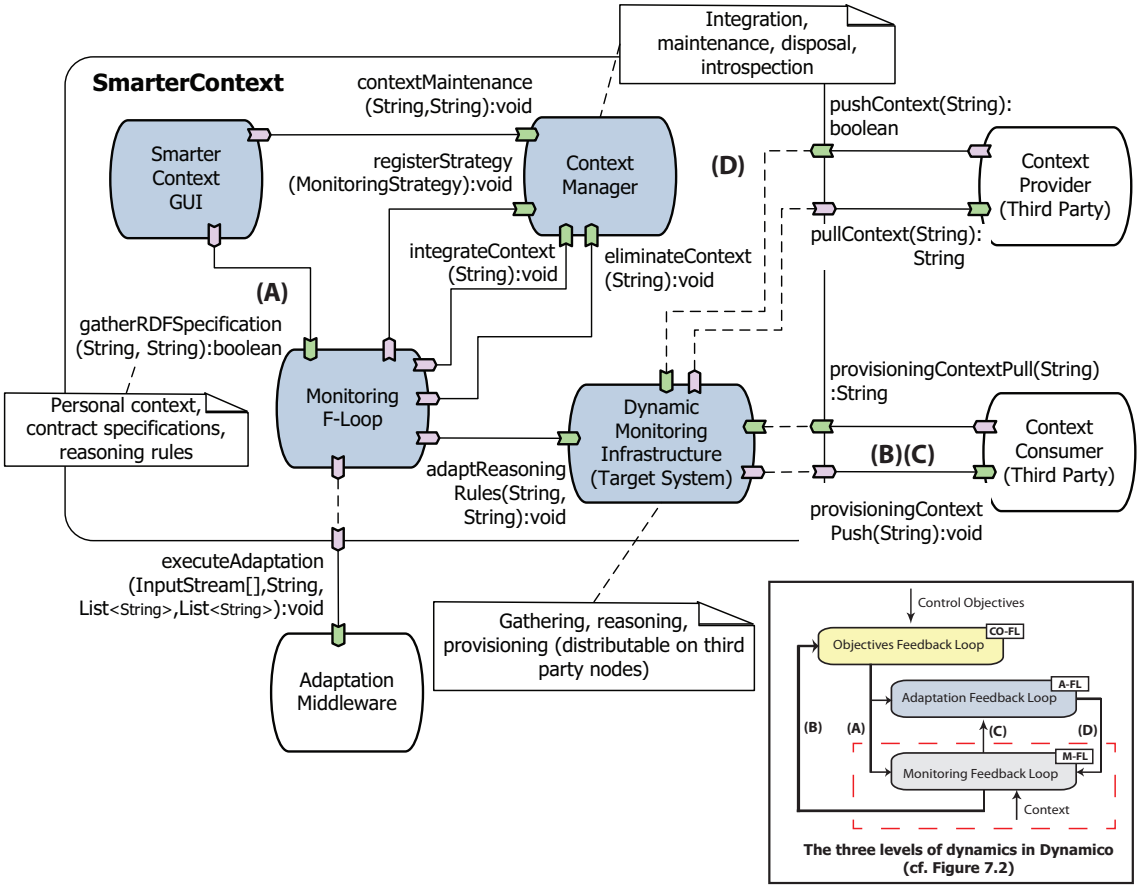


Figure 9.2: SmarterContext general architecture. The bottom right corner frame depicts the three levels of dynamics controlled in Dynamico. The `SmarterContext` composite realizes the third level of dynamics of Dynamico (i.e., the monitoring feedback loop) by defining three components: `MonitoringF-Loop`, `Context Manager` and `DynamicMonitoringInfrastructure`. The `AdaptationMiddleware` composite represents the middleware that supports the dynamic reconfiguration of the SmarterContext infrastructure. Composites `ContextProvider` and `ContextConsumer` correspond to third parties that obtain context from and provide context to SmarterContext.

and FraSCAti [SMF+09], which constitutes the middleware that supports the dynamic architectural reconfiguration of our context manager.

Labels (A), (B), (C) and (D) in Figure 9.2 correspond to communication links that realize the interactions among the three levels of dynamics defined in Dynamico (cf. Labels (A), (B), (C) and (D) in the frame at the bottom right corner of the same figure). Label (A) corresponds to the interaction between the objectives and monitoring feedback loops that enables our SmarterContext implementation to receive the specifications of control objectives and their changes. Labels (B) and (C) represent the interactions that allow SmarterContext to notify context monitoring facts to

the system objectives manager and the adaptation mechanism, respectively. Both the system objectives manager and adaptation mechanism are abstracted as context consumers in this architecture (cf. composite `ContextConsumer` in Figure 9.2). Label (D) represents the interaction through which the monitoring infrastructure gathers context information about the situation of the adaptive system.

9.1.2 Services in the SmarterContext Generic Architecture

`MonitoringF-Loop` Component

- *gatherRDFSpecification:String* × *String* → *boolean.* This service provides the monitoring feedback loop with a control objectives specification that can be either personal context(including third parties to be included as context consumers and/or providers) in the case of user-centric SASS systems, system objectives (e.g., QoS contracts) that will drive the adaptation process and context manager, and reasoning rules to be added or deleted from the CoRE. *Parameters:* a String value with the control objectives specification in XML/RDF; a String that indicates the nature of the received specification. *Return value:* a boolean that indicates whether the received XML/RDF specification is SmarterContext compliant.

`ContextManager` Component

- *contextMaintenance:String* × *String* → *void.* This service enables users to maintain the context information stored in context spheres. *Parameters:* a String value with the context specification in XML/RDF; a String that indicates whether the maintenance operation corresponds to context modification, which includes changes in privacy and security privileges granted to third parties, or context disposal.

- *registerStrategy:MonitoringStrategy* → *void.* This service provides the `ContextManager` component with a new context monitoring strategy that must be registered for architecture introspection purposes. *Parameters:* a `MonitoringStrategy` object that includes the corresponding control objectives and SCA composite specifications.

- *integrateContext:String* → *void.* This service allows users to add third parties authorized as context consumers and/or providers. *Parameters:* a String with

an XML/RDF specification that includes the third parties to be integrated with the corresponding privacy and security policies.

- *eliminateContext:String → void.* This service allows users to eliminate third parties that were authorized as context consumers and/or providers previously. *Parameters:* a String with an XML/RDF specification that includes the third parties to be eliminated.

`DynamicMonitoringInfrastructure` Component

- *adaptReasoningRules:String × String → void.* This service allows the modification of reasoning rules in the SmarterContext CoRE. *Parameters:* a String with an XML/RDF specification of the reasoning rules to be updated; a String that indicates whether the operation type is an addition, deletion or modification of rules.

- *pushContext:String → boolean.* This service allows the gathering of context information from context providers using a push mechanism. *Parameters:* a String with the gathered context information in XML/RDF. *Return value:* a boolean that indicates whether the received XML/RDF specification is SmarterContext compliant.

- *provisioningContextPull:String → String.* This service allows context consumers to obtain context information from the SmarterContext infrastructure using a pull mechanism. *Parameters:* a String with an XML/RDF that specifies the information of the context consumer including, when required, the public key for accessing personal contextual data. *Return value:* a String that corresponds to the provisioned contextual data in XML/RDF.

Services of Composites External to SmarterContext

`AdaptationMiddleware` Composite

executeAdaptation:InputStream[ ] × String × List<String> × List<String>
→ void. This service allows the executor of the context management loop to invoke the adaptation mechanism provided by the adaptation middleware. *Parameters:* an array of `InputStream` objects that contains .class files with the implementations of the components to be deployed dynamically; a String that contains either the specification

of the SCA composite that defines the components to be deployed, or an XML/RDF model that specifies changes in the set of context reasoning rules; a generic list of String with the identification of the sensors' SCA references (endpoints of context providers connected to SmarterContext gathering services); and a generic list of String with the identification of the SCA context gathering services that will be consumed by the sensors' references.

`ContextProvider` Composite

pullContext:String → *String.* This service allows SmarterContext to gather context information using a pull mechanism. *Parameters:* a String with the identification of the entity related to the context information to be gathered (e.g., a user). *Return value:* a String that corresponds to the gathered contextual data in XML/RDF (e.g., the product categories user Norha added to her wish list).

`ContextConsumer` Composite

provisioningContextPush:String → *void.* This service allows SmarterContext to provide context information to third parties using a push mechanism. *Parameters:* a String that corresponds to the provisioned contextual data in XML/RDF (e.g., Norha's list of favorite locations).

9.1.3 The SmarterContext Monitoring Feedback Loop Architecture

The implementation of the context monitoring feedback loop (the third level of dynamics in our Dynamico reference model) enables our monitoring infrastructure with dynamic capabilities to adapt (i) the set of context reasoning rules supported by the CoRE, (ii) the context monitoring logic that evaluates gathered context against monitoring conditions, and (iii) the context gathering and provisioning infrastructure.

Figure 9.3 presents the `monitoring feedback loop (M-FL)` composite, which is the concrete realization of the `MonitoringF-Loop` component depicted in Figure 9.2. The services exposed by composite `M-FL` were explained in Section 9.1.2 as services of the `MonitoringF-Loop` component. The initial component of our monitoring feedback loop is `ContextMFLMonitor` (cf. Figure 9.3). This component receives the control objectives (CObj) specification in XML/RDF format from the user, creates an

`RDFSpecification` object from the received specification, looks for a previous version of this COb specification, stores the new COb specification in its knowledge base, and finally provides component `ContextMFLAnalyzer` with two `RDFSpecification` objects that represent the new and former COb specifications.

The second component of the `M-FL` composite is `ContextMFLAnalyzer`, which is in charge of analyzing changes in COb specifications, and specifying these changes in an RDF model. After analyzing changes in COb specifications this component invokes `ContextMFLPlanner` and provides it with the new COb specification and the model that specifies the changes. If there is not previous COb specification, `ContextMFLAnalyzer` simply provides `ContextMFLPlanner` with the new COb specification and a null Model.

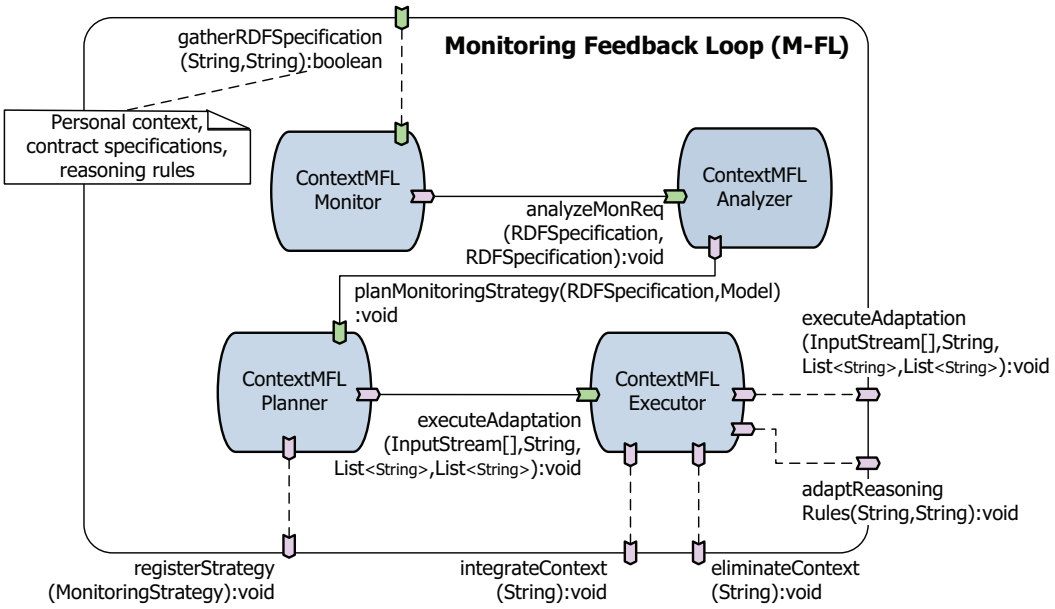


Figure 9.3: Context monitoring feedback loop architecture. This architecture corresponds to the third level of dynamics in Dynamico.

The third component of composite `M-FL` is `ContextMFLPlanner`. This component is in charge of synthesizing new monitoring strategies as well as changes in existing ones. We define monitoring strategies as an object that contains (i) a set of implementation files for the SCA components to be deployed, the specification of the corresponding SCA composite, and two lists that specify SCA services and corresponding references. These services and references allow the connection of sensors exposed by third parties to context gatherers exposed by the SmarterContext infrastructure, and context providers gatherers exposed by third parties to context

providers exposed by the SmarterContext infrastructure; or (ii) a set of context reasoning rules to be added or deleted from the SmarterContext CoRE.

The last component of composite `M-FL` is `ContextMFLExecutor`. This component invokes the services that will trigger the adaptation of the context monitoring infrastructure. The SmarterContext monitoring infrastructure can be adapted at runtime by either (i) changing the set of context reasoning rules, (ii) modifying the monitoring logic, (iii) deploying new context sensors, and context gathering, monitoring and provisioning components.

Table 9.1 summarizes the self-adaptive capabilities of SmarterContext. The first column refers to the type of change in COb specifications that may trigger the adaptation of SmarterContext; the second column corresponds to the semantics of the reference input that feed the monitoring feedback loop through component `ContextMFLMonitor` (cf. Figure 9.3); the third column presents the type of control action used to affect the target system (i.e., the monitoring infrastructure); the fourth column describes the effect obtained on the context monitoring infrastructure after performing the adaptation process. The self-adaptive capabilities that address changes in the set of reasoning rules and in the authorized context providers and consumers concern our case study on situation-aware smarter shopping. The self-adaptive capabilities implemented to address the addition or renegotiation of system objectives (e.g. QoS contracts) concern our case study on dynamic SOA governance.

9.1.4 Services in the Monitoring Feedback Loop Composite

`ContextMFLMonitor` Component

This component exposes only one service that corresponds to the *gatherRDFSpecification:String* $\times$ *String* $\rightarrow$ *boolean* service explained in Section 9.1.2.

`ContextMFLAnalyzer` Component

analyzeMonReq:RDFSpecification $\times$ *RDFSpecification* $\rightarrow$ *void.* This service analyzes changes in COb specifications. *Parameters:* an `RDFSpecification` object that corresponds to the new COb specification; an `RDFSpecification` object that corresponds to the previous version of the new COb specification. The second parameter can be a null value to indicate that there is not a previous version of this COb specification.

Table 9.1: Self-Adaptive Capabilities of SmarterContext

Changes in CObs Specifications	Reference Inputs	Control Actions	Adaptation Effects
Addition/deletion of reasoning rules	RDF specifications of reasoning rules	Parameters (i.e., RDF rules) affecting the behavior of SmarterContext	Modified reasoning capabilities of the CoRE
Addition/deletion of context providers and/or consumers	RDF models representing personal context	Discrete operations affecting the SmarterContext software architecture	Changes in the set of deployed context sensing, gathering, and provisioning components
Addition or renegotiation of system objectives	RDF models representing system objectives with explicit monitoring requirements	Parameters (i.e., arithmetic and logic expressions) affecting the behavior of SmarterContext	Changes in existing monitoring logic
		Discrete operations affecting the SmarterContext software architecture	Changes in the set of deployed context sensing, gathering, monitoring and provisioning components

`ContextMFLPlanner` Component

planMonitoringStrategy:RDFSpecification × *Model* → *void.* This service synthesizes a new monitoring strategy based on a new or modified CObs specification. *Parameters:* an `RDFSpecification` object that corresponds to the CObs specification; an RDF `Model` object that specifies changes in monitoring requirements (this parameter can be null).

`ContextMFLExecutor` Component

executeAdaptation:InputStream[ ] × *String* × *List<String>* × *List<String>* → *void.* This service invokes the adaptation mechanism provided by the QoS-CARE/FraSCAti framework or by the SmarterContext reasoning engine, depending on whether the adaptation concerns the reconfiguration of the monitoring infrastructure or changes in the set of reasoning rules. *Parameters:* an array of `InputStream` objects with the

implementations of the components to be deployed dynamically; a String representation of the SCA composite, a generic list of String with the identification of the sensors' SCA references (endpoints of context providers connected to our SmarterContext gathering services); and a generic list of String with the identification of the SCA context gathering services that will be consumed by the sensors' references.

9.2 Implementation of the Context Reasoning Engine (CoRE)

The SmarterContext CoRE infers implicit contextual facts in the form of RDF triples from explicit contextual triples stored in contextual RDF graphs. The first version of the SmarterContext CoRE reasons on contextual RDF graphs using RDFS and OWL-Lite rules, as well as user-defined rules at different levels of the SmarterContext ontology. In Chapter 5 we explained these rules and how they support the inference of implicit contextual facts in SmarterContext. We implemented this version of the SmarterContext CoRE using Jena[3] [CDD+04] and two of its supported semantic web rule engines: the inference engine provided with the Jena framework, and Pellet [SPG+07a].

This section explains how we implemented context models in the form of contextual RDF graphs available at runtime, and the artifacts of the version of the SmarterContext CoRE that uses semantic web rules to infer context information from contextual RDF graphs.

9.2.1 Implementation of Contextual RDF Graphs

Contextual RDF Graphs are models used in SmarterContext to represent, at runtime, information about relevant context entities, the relationships among them, and/or their monitoring strategies. We implemented contextual RDF graphs using Jena *models*, which are graph-based data structures that store a set of statements. These statements correspond to contextual RDF triples in SmarterContext. We defined *contextual RDF graphs* and *contextual RDF triples* in Section 5.1.3.

SmarterContext uses the three types of models supported by Jena, namely *RDF model*, *ontology model*, and *inference model*. We use RDF models to create

[3] http://jena.apache.org

and manipulate contextual RDF graphs; ontology models to access and manipulate ontologies represented in RDF; and inference models to access the inference capabilities of the reasoning engine and represent inferred contextual data. Package `com.hp.hpl.jena.rdf.model` provides the core interfaces and classes for programming RDF models and inference models. RDF models correspond to raw contextual data, whereas inference models includes context inferences. Package `com.hp.hpl.jena.ontology` provides the interfaces and classes for working with ontologies.

9.2.2 The SmarterContext CoRE Artifacts

Figure 9.4 presents an abstraction of the SmarterContext CoRE implementation,[4] as an instantiation of the Jena inference architecture. White boxes correspond to Jena artifacts, whereas gray boxes represent the instances of these artifacts that implement the SmarterContext reasoner. Type `ModelFactory` allows the access to the inference capabilities of reasoners. `InfGraph` allows the instantiation of reasoning models which are associated with RDF models and ontology models instantiated from types `Model` and `OntModel`, respectively, and to a `Reasoner` object. `InfGraph` objects allow the manipulation of inferred context information. Models, including RDF and ontology models, allow the implementation of contextual RDF graphs that contain the gathered contextua data (i.e., explicit contextual facts provided by users or gathered from context sources). Contextual RDF graphs, explicit and inferred, constitute personal context spheres (PCS). For the SmarterContext CoRE to infer context information, the inferred context graph must be associated with explicit contextual data graphs and to the corresponding instance of the SmarterContext reasoner. An instance of type `Reasoner` is bound to a set of base assertions depending on the type of inference to be supported (e.g., RDFS, OWL-Lite, OWL-Full). For the SmarterContext implementation of this dissertation we used RDFS and OWL-Lite. Ontology definitions are bound to reasoners in the form of schemas. User-defined reasoning rules can be bound to reasoners to extend their inference capabilities. The SmarterContext ontology and its compliant vocabularies provide the schema for the SmarterContext reasoner. SmarterContext reasoning rules such as the ones explained in Section 5.2.2 extend the inference support provided by RDFS and

[4]This figure is an extension of the figure that depicts the Jena inference machinery at http://jena.apache.org/documentation/inference/index.html

OWL to reason about context information in specific domains.

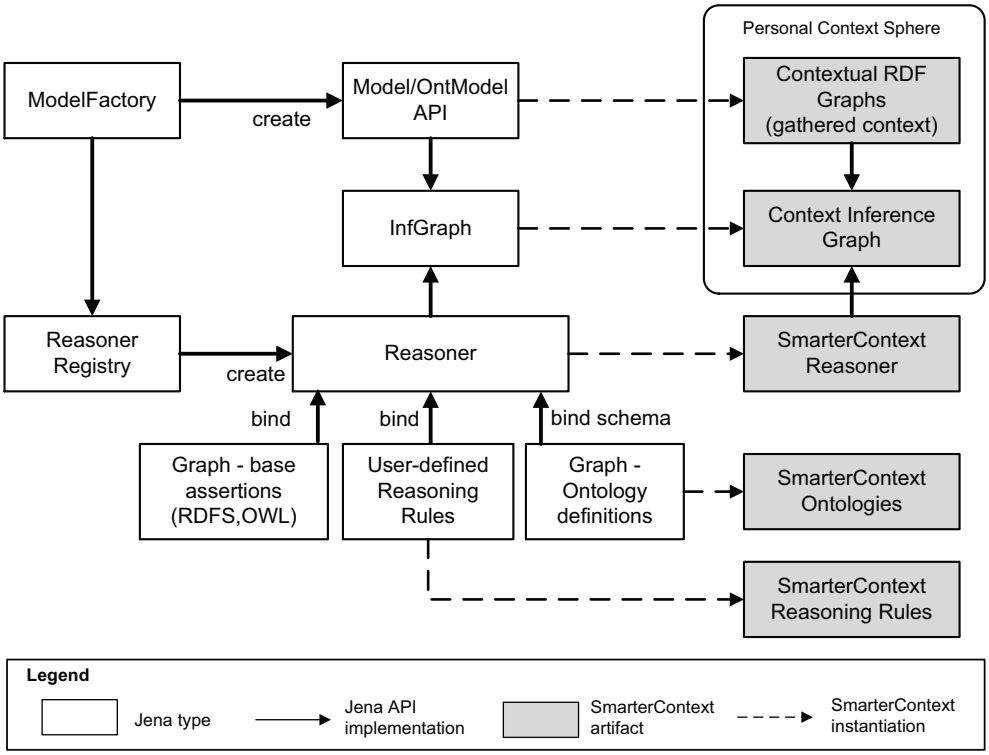


Figure 9.4: An abstraction of the SmarterContext CoRE implementation. Jena types correspond to artifacts offered by the Jena framework. SmarterContext artifacts correspond to our extension for context reasoning.

9.2.3 Optimization of the SmarterContext Reasoning Engine

With the goal of improving performance when reasoning on high volumes of context information, we proposed, in Section 5.3.1, a set of contextual patterns. These patterns are RDF subgraphs that provide templates for defining domain-specific rules that allow the inference of contextual facts that may be commonly found in the corresponding application domains. The catalog of contextual patterns we proposed in this dissertation focuses on contextual facts for our situation-aware smarter shopping case study.

This optimized version of the SmarterContext CoRE relies on efficient data structures rather than on OWL-based reasoning. In this way, the SmarterContext CoRE can infer implicit contextual facts (implicit RDF triples) using RDFS only, that is without requiring the complexity added by OWL assertions. This implementation improves context processing time significantly. Our approach is as follows:

1. *Data Structures.* A partitioning data structure associated with each context pattern supported in the corresponding application domain must be created for each contextual RDF graph (i.e., PCSs and RDF vocabularies integrated into the SmarterContext ontology) involved in the context reasoning space. Section 5.3.1 describes the context patterns proposed to support context reasoning in this version of our SmarterContext CoRE.

2. *Partitioning and Indexing.* Each contextual RDF graph model must be traversed to partition contextual triples according to context patterns and create indices for each predicate type associated with the pattern. For this, the partitioning process, which is performed off-line, checks whether the context predicate of each contextual RDF triple in the model is related to any of the object properties defined in the SmarterContext ontology as predicate types associated with applicable context patterns (cf. Table 5.1). When one such association is found, the corresponding triple is stored in the context predicate index of the context pattern data structure.

3. *Context Inference.* Implicit contextual facts are discovered by applying pattern-based reasoning functions to the contextual RDF triples stored in the corresponding pattern's indices. Section 5.3.1 explained and exemplified the functions supported by the catalog of patterns proposed in this dissertation, including the way SmarterContext uses them to infer contextual facts.

The implementations of these steps are detailed as follows.

Data Structures for Context Pattern Partitioning and Indexing

Figure 9.5 presents the data structures used to maintain the indices of context predicates associated with context patterns. For existing context repositories, these indices must be created off-line. However, they can be maintained easily at runtime. We defined these data structures at three different levels, and create them for each contextual RDF graph that is within the inference space used by the SmarterContext CoRE. The first level corresponds to the *context patterns* data structure. This data structure contains the context patterns applicable to the problem domain. Each context pattern includes a reference to the data structure defined at the second level, the *pattern predicates* data structure. This data structure contains the predicate types associated with the context pattern definition. For example, the

`gc:transitivePredicate` predicate type associated with the SmarterContext *transitivity* pattern (cf. the catalog of patterns presented in Sect. 5.3.1). Each predicate type contains the references to the first data structure defined at the third level. The third level of data structures, *triples*, allows the indexing of triples within the contextual RDF graph. This level involves three data structures: *model predicates, model subjects* and *model objects*. The *model predicates* data structure contains the context predicates of the triples in the contextual RDF graph that are related to the predicate types associated with the pattern definition. Each of these predicates references the lists of subjects of the triples associated with the corresponding predicate in the context model. The last data structure contains the list of objects associated with each subject through the predicate that correspond to the index.

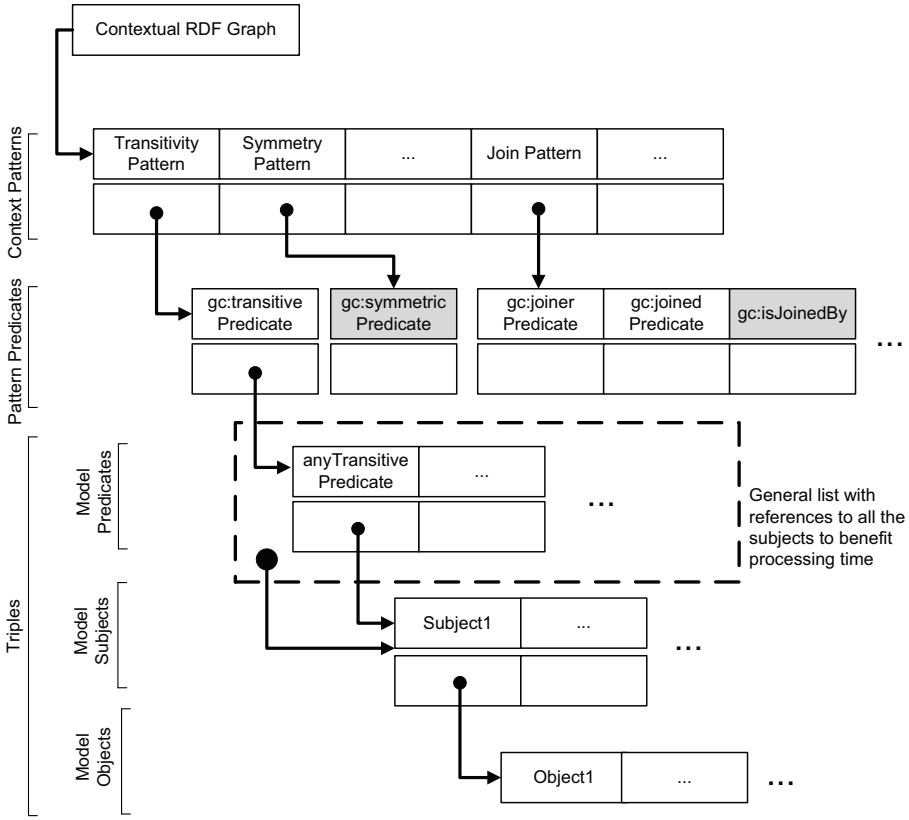


Figure 9.5: Data structures for context pattern partitioning and indexing. The data structures at the *triples* level store triples of the contextual RDF model that are associated with a pattern predicate type (cf. *pattern predicates* level), and through this predicate to a context pattern (cf. *context patterns* level). A general list at the level of model predicates defines references to all the subjects associated with the indexed triples of the model.

Partitioning and Indexing

The goal of the partitioning process is to group the triples of the model that are related to context patterns into clusters. These clusters correspond to the entries of the *context patterns* data structure (cf. the first level depicted in Figure 9.5). Using the catalog of context patterns proposed in this dissertation, we group model triples based on the following predicates: `gc:transitivePredicate`, `gc:symmetricPredicate`, `gc:isInverseOf`, `gc:joinerPredicate`, `gc:joinedPredicate`, `gc:isJoinedBy`, `gc:generalizablePredicate`, `isGeneralizableFor`, `gc:delegablePredicate`, `gc:delegationPredicate`, `gc:isDelegableFor`. These are the predicates associated with the definitions of the patterns. The partitioning process is performed in two ways depending on whether the predicate of the triple to be classified is a subtype of a predicate type associated with a context pattern (e.g., `gc:locatedIn` is a subtype of `gc:transitivePredicate`), or the latter is an object property (predicate) applicable to the triple predicate (e.g., `gc:isInverseOf` is an object property for `pwc:parentOf` since the triple (`gc:isInverseOf, pwc:parentOf, pwc:childOf`) exists in the SmarterContext ontology). In any case, to benefit processing time, besides the lists of subjects and objects related to each index we define, for each context model, a general list that contains the references to all the subjects associated with the indexed triples of the model (cf. dashed section in Figure 9.5).

To illustrate the partitioning and indexing process, consider the partial graph-based view of user Norha's PCS and vocabulary `geo` presented in Figure 9.6(a) and 9.6(b). Norha's PCS involves two SmarterContext vocabularies, `google` and `geo`. `google` characterizes product and service categories, whereas `geo` defines geographical locations. The partitioning and indexing process must be applied to the user's PCS and the two vocabularies.

Suppose a context consumer requests context information about Norha's location. Since the location context predicates defined in the SmarterContext ontology relate to context patterns *Transitivity* and *Join*, the engine will use the indices associated with these context patterns to reason about the user's location.

Figures 9.7 and 9.8 depict the result of the partitioning and indexing process for the contextual RDF graphs presented in Figure 9.6. Norha's PCS is associated with one index defined for the `gc:locatedIn` context predicate (cf. highlighted index in Figure 9.7). This index is used in the partition of both *Transitivity* and *Join* pattern. Since `gc:locatedIn` is a subtype of both `gc:transitivePredicate` and

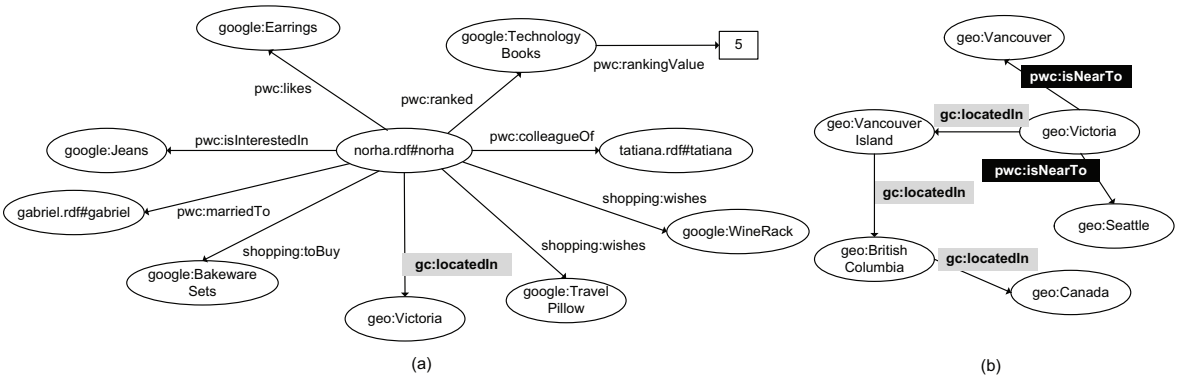


Figure 9.6: A partial graph-based view of (a) user Norha's PCS and (b) `geo` vocabulary. The graph involves two context vocabularies, `google` and `geo`. The highlighted context predicates correspond to predicates related to context patterns used to reason about location context.

`gc:joinerPredicate`, this index is composed of the triple represented by the arc highlighted in gray in Figure 9.6(a). Notice that there are no indices associated with predicate `gc:joinedPredicate`, meaning that Norha's PCS has no triples whose predicates are defined as subtypes of this pattern predicate.

The partial view of the `geo` vocabulary's contextual graph (cf. Figure 9.6(b)) is associated with the partitions and indices presented in Figure 9.8. The first index contains the triples associated with the arcs highlighted in gray in Figure 9.6(b). That is, the triples whose predicate is a subtype of `gc:transitivePredicate` and `gc:joinerPredicate`—`gc:locatedIn` in this case. The second index corresponds to the triples represented by the arcs highlighted in black in the same figure. These are the triples whose predicate is a subtype of `gc:joinedPredicate`—`gc:isNearTo` in this example.

Context Inference

In SmarterContext, context reasoning with structural context patterns consists of inferring implicit contextual facts from explicit triples specified in contextual RDF graphs, by applying pattern-based reasoning functions on the data stored in indices. For example, to infer the locations where the user is located, the SmarterContext CoRE applies the function *FindTransitivityClosure((l(v),l(p),G)*, defined for the *Transitivity* pattern in the catalog of patterns presented in Section 5.3.1, with *l(v)=norha.rdf#norha*, *l(p)=gc:locatedIn*, and *(G=norha.rdf* $\bowtie$ *geo.rdf)*, where

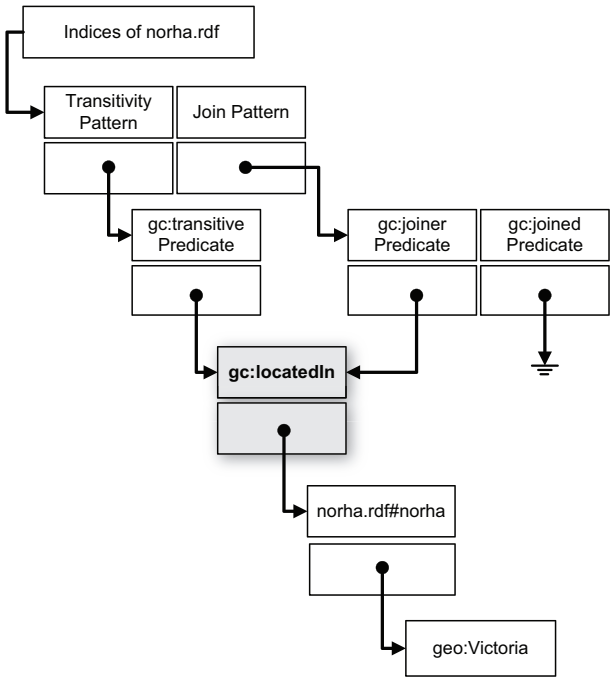


Figure 9.7: Indices of Norha's PCS (cf. Figure 9.6(a)) for the context patterns related to location context. One partition, with its corresponding context predicate indices, was created for each context pattern involved in the inference process: *Transitivity* and *Join* in this case.

$\bowtie$ represents the join of the *norha.rdf* and *geo.rdf* models through the resource `geo:Victoria` of their corresponding `gc:locatedIn` index. Figure 9.9 illustrates this process for the contextual RDF graphs presented in Figure 9.6. The inferred context triples allow us to conclude that user Norha is not only located in Victoria—as stated explicitly in her PCS—but also on Vancouver Island, British Columbia, and in Canada.

Similarly, by combining the `gc:locatedIn` index of Norha's PCS with the `gc:isNearTo` index of vocabulary `geo`, the SmarterContext CoRE will infer that the user is located near Vancouver and Seattle (cf. the *Join* pattern's functions defined in the catalog presented in Section 5.3.1).

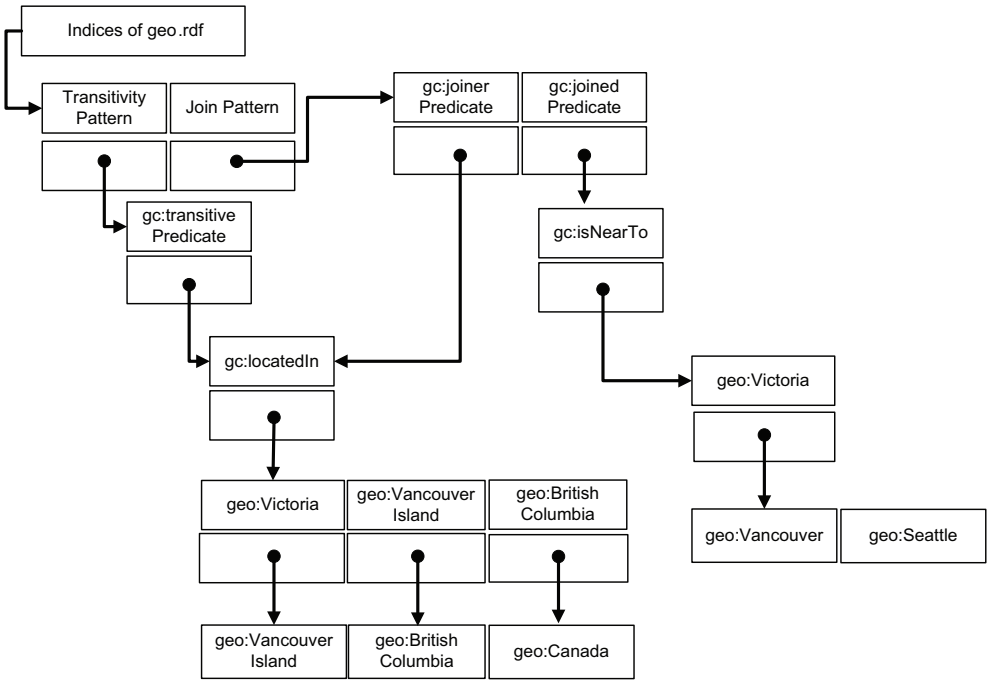


Figure 9.8: Indices of the `geo` vocabulary (cf. Figure 9.6(b)). One partition with its corresponding context predicate indices was created for each context pattern. For explanation purposes, subject `geo:Victoria` is depicted twice. However, since the elements of indices are Java objects with corresponding references, each element is instantiated only once for the indices of a same context model.

9.3 Realizing User-Controlled Privacy and Security

This section presents implementation aspects of SURPRISE (Smartercontext UseR PRIvacy and SEcurity), our solution to guarantee privacy and security for personal context information stored in PCSs. The three key features supported by our privacy and security solution for RDF data are: *dynamic selectivity, dynamic granularity* and *partial encryption* (cf. Section 3.3). With these features, SURPRISE (i) allows users to configure access permissions to their sensitive personal information to third parties, selectively and with different levels of granularity; (ii) supports changes in these configurations at runtime to add or remove third parties or permissions; and (iii) realizes partial encryption to share non-sensitive data with not explicitly authorized third parties, while protecting user identity. From the perspective of privacy and security we classify contextual data into two types: *sensitive* and *non-sensitive* contextual data. Sensitive context must be protected against unauthorized disclosure,

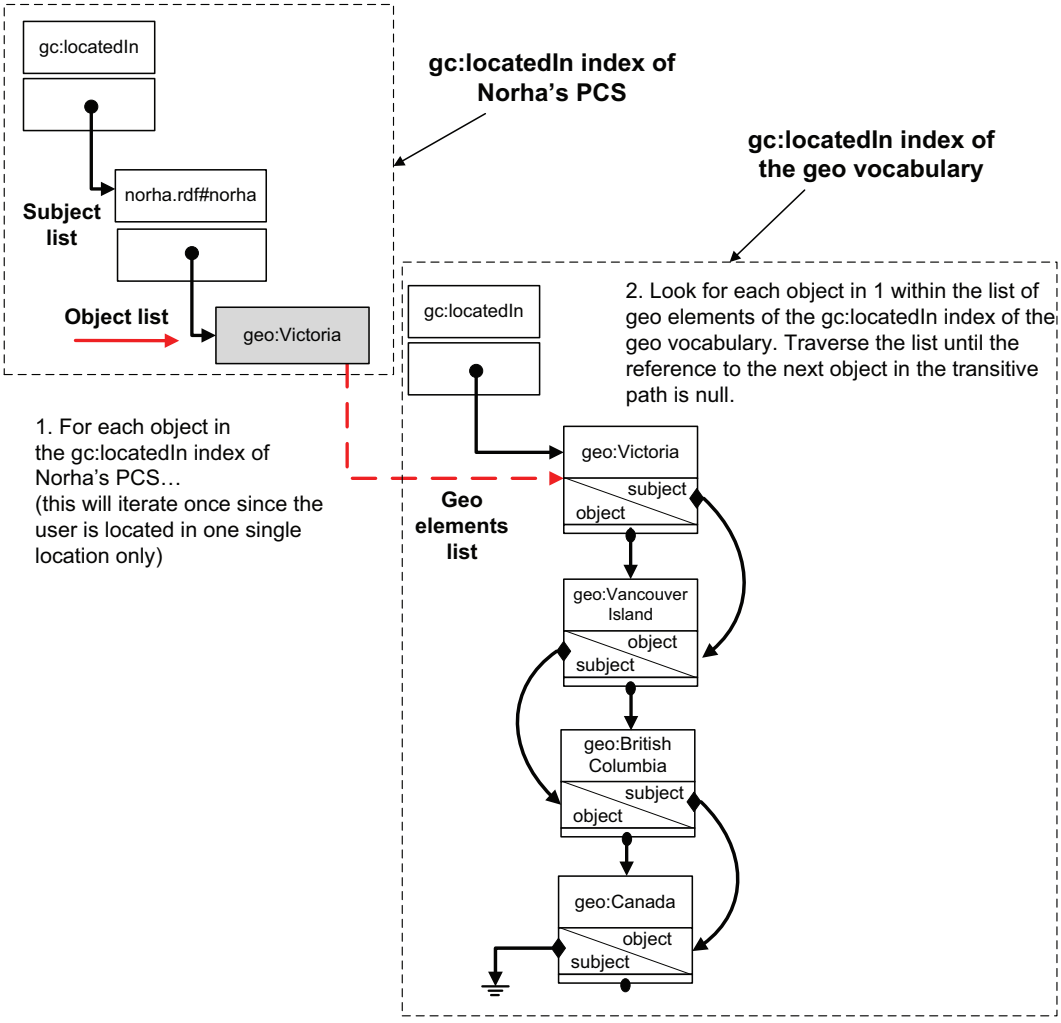


Figure 9.9: Context inference based on the *Transitivity* pattern. Gray boxes represent the triple object elements of the corresponding index. Note that each element in the Geo elements list may act as either a subject or an object.

non-sensitive data can be disclosed as long as user identities are protected.

One of the most representative partial encryption approaches for RDF data is *Partial RDF Encryption (PRE)* [Gie05, Gie06]. We selected PRE as a baseline for the implementation of our solution for three main reasons. First, it is a sound solution for partial encryption. Second, both its specification and implementation are publicly available. Third, it is based on Jena [CDD+04], the semantic web framework that we used to implement the SmarterContext engine.

9.3.1 Modified Features of PRE in Surprise

Even though PRE supports partial encryption, its PREPolicy encryption policy is insufficient to address dynamic selectivity and dynamic granularity as defined in SmarterContext. Considering the partial view of Norha's PCS depicted in Figure 9.10, this means that with PRE only, we could reason on Norha's non-sensitive contextual data because it supports partial encryption, but the owner of this information can neither grant different privilege levels to different third parties nor select particular context types to be shared with a particular third party. To address these requirements, we extended PRE in three ways. First, we modified the PREPolicy structure and semantics to enable the definition of policies that support dynamic selectivity and dynamic granularity for SmarterContext (hence called PREPolicySC). Second, we modified the PRE encryption containers (ECs) to support dynamic selectivity efficiently. Finally, we extended the PRE4J API to provide the functionality required to support the redefined encryption policies and ECs. Appendix D provides further details about these modifications.

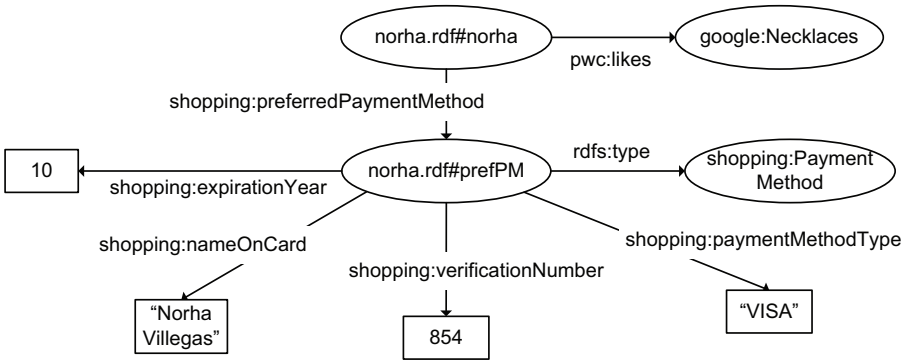


Figure 9.10: A partial view of Norha's PCS with sensitive contextual data. The triples related to the user's payment method correspond to sensitive context information.

9.3.2 Redefinition of PRE Policies in Surprise

The most important aspect of our policy redefinition is its semantics, that is, what the policy represents in terms of RDF encrypted data when applied to PCSs. To support policies in context management, we defined PREPolicySC files (privacy and security policies in SmarterContext) based on the PREPolicy files defined in PRE.

To illustrate this aspect better, consider the application of encryption policies in our situation-aware smarter shopping case study. Figure 9.10 depicts Norha's

preferred payment method, represented in her PCS as an RDF sub-graph composed of seven triples. Norha's PCS, depicted partially in the same figure, has an extra triple that represents non-sensitive data, in particular the fact that she likes necklaces.

Informally, the application of the corresponding PREPolicy policy to Norha's PCS yields the RDF graph presented in Figure 9.11(a), whereas the application of the PREPolicySC policy produces the RDF graph depicted in Figure 9.11(b). Notice that the triple related to her product preferences remains unencrypted.

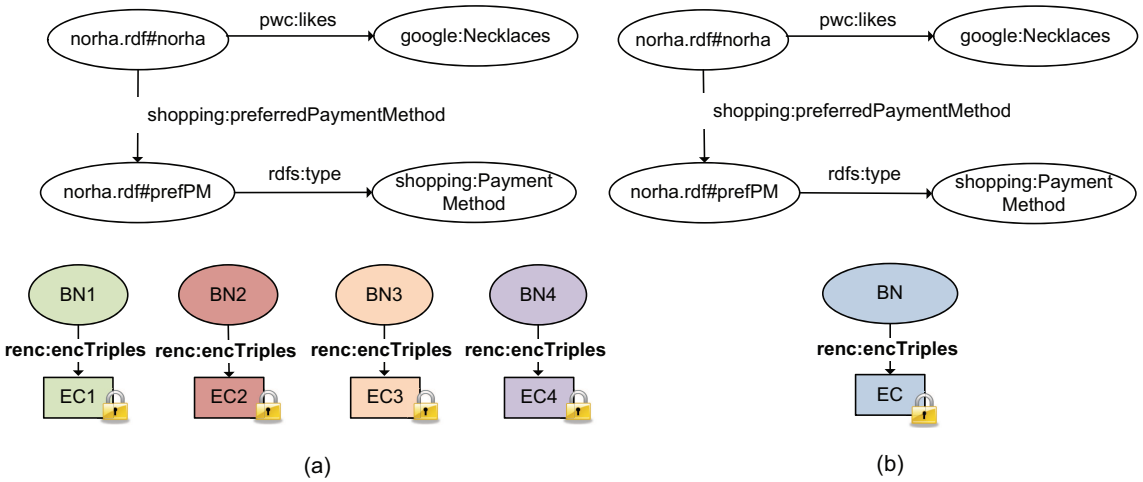


Figure 9.11: Encryption semantics in (a) PRE and (b) Surprise

In the graphs presented in Figure 9.11:

- $\texttt{BN}_i$ is a blank node;
- `renc:encTriples` is the predicate that identifies encrypted triples; and
- $\texttt{EC}_i$ is the encryption container that encapsulates the encrypted RDF elements.

Figure 9.12(a) depicts the main elements of privacy and security policies in Surprise (PREPolicySC policies). Figure 9.12(b) illustrates a PREPolicySC pattern for encrypting personal context information related to preferred payment methods in our case study. Triple patterns, such as the ones presented in Figure 9.12(b), allow the identification of contextual data to be encrypted in users' PCSs. Policies in Surprise specify these patterns. It is important to clarify that these patterns are completely unrelated to the patterns for context reasoning to which we referred in the previous section.

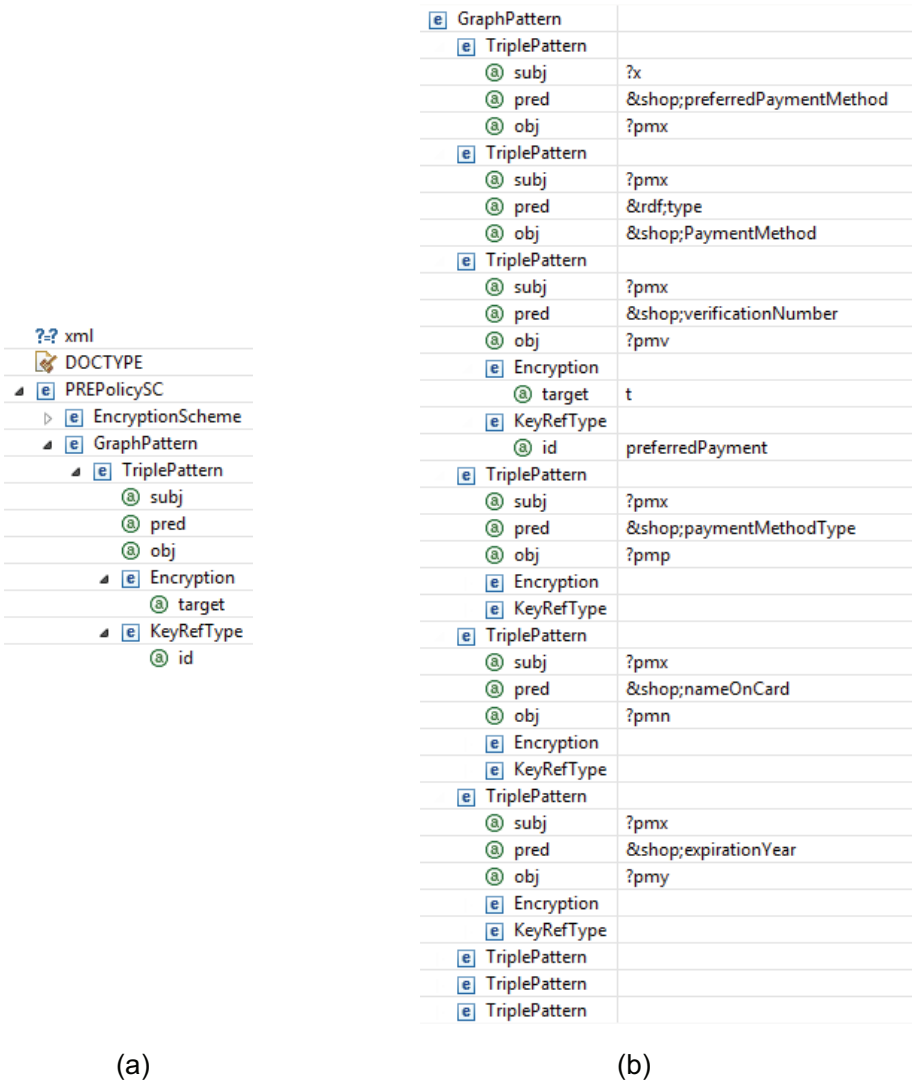


Figure 9.12: (a) The main elements of our PREPolicySC specification. (b) Example of a PREPolicySC `GraphPattern` for encrypting `preferredPaymentMethod` triples in our situation-aware smarter shopping case study.

9.3.3 New Features

Graph Patterns for SmarterContext sensitive data

Encryption patterns are domain-specific. For our case study on situation-aware smarter shopping, we identified six sensitive context types from the categories defined in the SmarterContext ontology. These types are related to user locations, payment methods, personal identification and other personal data such age, gender and marital status.

Storage of Encryption Keys in the PCS

In SmarterContext users control the sharing of the contextual data stored in PCSs. This implies the granting of context exchange privileges to third parties and the definition of encryption policies for sensitive context types. For this, we implemented the mechanisms to store in the SmarterContext infrastructure the public keys of all the third parties authorized by the user as context consumers and/or providers. Then, these public keys are linked in the user's PCS to the sensitive context types that each authorized third party can access. Public keys also are added to ECs. Whenever the user decides to stop sharing a particular sensitive context type with a third party, the corresponding key references are deleted from the PCS, and the encryption key from the EC.

The semantics of privileges granted to third parties in user PCSs is supported by the SmarterContext ontology and its Surprise module as follows. For each third party integrated by the user, SmarterContext creates a new triple in the PCSs with subject the URI that identifies the user, predicate `pwc:hasIntegrated`, and object the URI that identifies the third party. Then, each third party is described by a set of triples using the following data property predicates: `sprise:receiveSCData`, `sprise:sendSCData`, `sprise:hasPublicKey`, and `sprise:hasAccess`. The domain of all these properties corresponds to the `pwc:PWESite` type. `sprise:receiveSCData` and `sprise:sendSCData` have range boolean and define whether third parties can send and receive, only send, or only receive contextual data to and from the user's PCS. `sprise:hasPublicKey` defines a string value with the identifier of the third party's public key stored in the SmarterContext infrastructure's key repository. Finally, `sprise:hasAccess`, whose range includes any context type defined in the SmarterContext ontology, defines the set of context categories that the third party can access.

9.4 Chapter Summary

This chapter presented design and implementation details of the most important components of our SmarterContext infrastructure and its reasoning engine. We implemented SmarterContext as a self-adaptive system whose architecture is derived from our Dynamico reference model. In particular, the SmarterContext architecture described in this chapter presents a realization of the first and third

levels of dynamics characterized by DYNAMICO, namely *control objectives feedback loop* and *context monitoring feedback loop*, respectively. Besides the architectural details of SMARTERCONTEXT, this chapter also presented implementation details of the SMARTERCONTEXT reasoning engine, including an optimized version that uses pattern-based indices, and SURPRISE, our approach to control privacy and security of personal contextual data.

Both the SMARTERCONTEXT architecture and reasoning engine presented in this chapter can be extended to be applied in specific domains. As part of the validation of our contributions, in the next chapter we present the application of SMARTERCONTEXT to dynamic SOA governance and online shopping, the application domains of the case studies conducted as part of this dissertation.

Chapter 10

Evaluation

This chapter presents the evaluation of several aspects of the contributions related to this dissertation: Section 10.1 presents a qualitative analysis on how personal context spheres (PCS) and SmarterContext have the potential to enable innovative business models in the smarter commerce realm. Key aspects in these new business models are the empowerment of users as the controllers of their personal context, and the exploitation of this information to improve value creation and revenue generation for shoppers, retailers and cloud infrastructure and information service providers. Section 10.2 evaluates aspects of our contributions related to the improvement of user quality of experience (QoE) in user-centric SASS systems. This section starts with an evaluation of the applicability and feasibility of SmarterContext for realizing user-centric SASS applications in the e-commerce domain. This evaluation involves our PCS model, our SmarterContext ontology, our context reasoning engine (CoRE), and our Dynamico reference model. This section also presents SmarterDeals, a deal recommendation system that exploits users' changing personal context information to deliver highly relevant offers of products and services. We developed SmarterDeals to validate quantitatively the level of improvement of user QoE in terms of the accuracy of deal recommendations, when SmarterContext is integrated into situation-aware applications. Section 10.3 focuses on the applicability and feasibility of SmarterContext, and in particular of our Dynamico reference model, to improve self-adaptivity in business-centric applications. This section demonstrates how, with several of the contributions presented in this dissertation, it is possible to synthesize changes in monitoring strategies from changes in control objectives (i.e., changes in contracted conditions), and implement them dynamically by adapting either the monitoring logic or the software architec-

ture of SmarterContext. Section 10.4 presents a theoretical and experimental evaluation of the efficiency of SmarterContext CoRE. The experimental results presented in this section demonstrate the practical applicability of the SmarterContext CoRE to reason using large-scale repositories of context information at runtime. Finally, Section 10.5 summarizes the chapter.

10.1 E-commerce Value Creation with Personal Context Spheres

Suppose a week ago, Norha, our online buyer who lives in Victoria (BC), moved to Toronto (ON) for six months. However, she still receives emails with discount coupons for groceries for Victoria stores. Suppose another shopper, Hausi, whose kids are now adults, is annoyed with special offers related to children's books, to which he subscribed years ago, every time he logs into his favorite online book retailer. The experiences of these frustrated online shoppers prove that value creation, for sellers and buyers, is insufficiently exploited by existing e-commerce business models due to a lack of knowledge about buyers' *changing* situations and preferences.

We identified two main reasons for this lack of awareness about buyers' situations in existing e-commerce solutions. First, users have no control over the personal information they provide to existing e-commerce applications. Therefore, they avoid sharing information that could be useful to understand their shopping needs. Second, the information gathered about users by a particular seller remains within the boundaries of this seller's e-commerce application. Thus, the information gathered from the interactions between the user and this web site cannot be exploited by other online stores that offer related product and services. We can solve these two issues by managing personal context information stored in PCSs using SmarterContext.

User interactions in the smart internet are valuable sources of context information for value creation in e-commerce applications. Sellers can use personal context to enhance the shopping experiences of buyers thus creating value and generating revenue. For this, SmarterContext tracks user web interactions to gather and exploit meaningful personal context information. In contrast to existing e-commerce business models, SmarterContext can help develop new ones where users control the access to their personal context and personal data is shared among multiple sellers.

This section presents a qualitative evaluation of the value creation potential of selected features of SmarterContext using the information laws by Moody and Walsh [MW99]. To perform this evaluation, we analyzed how personal context information stored in PCSs can support value generation for buyers, sellers, and cloud infrastructure and information service providers. We argue that SmarterContext can help realizing new e-commerce business models where buyers have more pleasant and effective shopping experiences; sellers increase their revenue by delivering recommendations and offers of products and services that customers are really willing to buy; and cloud infrastructure and information service providers, enabled by the information stored in PCSs, accrue new opportunities to generate revenue based on business-to-business (B2B) models. For example, by maintaining the SmarterContext software infrastructure required on the seller side, by providing up-to-date context information from customers' PCSs, and by selling services based on information analytics enabled by PCSs.

10.1.1 Enabling new E-Commerce Business Models with SmarterContext

To discuss how our approach contributes to the realization of the smarter commerce's vision stated in Section 2.1.5, we focus on one of the main components of business models, *value proposition*. The value proposition of a business model defines how the company's products or services satisfy customers' expectations. From a buyer's perspective, traditional e-commerce models center their value propositions on personalization and customization of product and service offerings, price and time savings, as well as convenience. From a seller's point of view, value proposition commonly means revenue generation and market opportunity.

Context models maintained in PCSs constitute important sources of value proposition in e-commerce business models supported by smarter commerce. Key features of our user-driven context management solution that have the potential to trigger lucrative changes in existing e-commerce business models include:

- The access to personal context is controlled fully by its owner, the buyer. The privacy and security of personal contextual data is guaranteed even at the PCS's provider side. This means, private personal information in PCSs is encrypted even for the infrastructure provider. Moreover, context consumers authorized

by the user, such as online retailers integrated into the user's PCS, have access only to the information that they receive from the SmarterContext infrastructure. In other words, context consumers have access only to the information that is actually relevant to support them in optimizing the buyer's shopping experience, and for which provisioning has been authorized by the user. Moreover, the user can eliminate personal context information, as well as context consumers and providers when desired. SmarterContext addresses these user-controlled privacy and security requirements through its Surprise module 9.3.

- Simple user web interactions such as "liking", "tagging", and "ranking" constitute the vehicle to discover and gather relevant context information.
- The management of the context information life cycle includes automatic and user-controlled context disposal to deal with the perishable nature of this information.
- SmarterContext supports the addition and deletion of context types to and from PCSs without requiring manual programming or deployment of software components. This feature is key to address adaptive aggregation in users' relevant context types and interactions.
- Context information is gathered throughout the buyer's entire web experience. Therefore, personal context information stored in PCSs can be exploited by any application authorized by the user, in contrast to traditional models where personal information is gathered and exploited by individual online sellers. In these traditional models users have limited possibilities for exploiting their own information.

10.1.2 Exploiting the Value of Context through SmarterContext

Consider the following scenario. In contrast to online shopping experiences, with in-store physical experiences the layout of physical shelves and the distribution of their products cannot adjust automatically based on the current context situation and preferences of the customer just arriving to a particular aisle at the store. Furthermore, since past physical interactions between buyers and products were not recorded, it is

impractical to understand how the past experiences of buyers can affect their current purchase decisions. Similarly, it is infeasible to know other products the shopper could have considered for purchase during this shopping experience. In contrast, SmarterContext makes it possible to define business models that implement such personalization levels. Indeed, SmarterContext enables SASS e-commerce applications with the infrastructure required to understand not only the interactions of the buyer with products in the department store's online catalog, but also the buyer's social network, past shopping experiences, personal context, and alternative products.

The internet of things (IoT) provides unique virtual representations of objects (things) using Internet standards [MSPC12]. Bucherer and Uckelmann discussed several ways of exploiting the value of information in business models based on the IoT [BU11]. Their discussion focused on the identification of value creation opportunities given the seven laws of information proposed by Moody and Walsh [MW99]. We used Bucherer's and Uckelmann's analysis of these seven laws of information to discuss how SmarterContext supports value creation for buyers, sellers and infrastructure and information service providers in smarter commerce business models supported by the smart internet.

First Law: Information is (infinitely) shareable without loss of value

SmarterContext facilitates the sharing of information about buyers' situations and product and service offers. It enables the creation of new business models based on the provisioning of the context information stored in PCSs. Although the vision of SmarterContext is to empower users with full control of their personal information, by integrating business applications into PCSs, users authorize context sphere infrastructure providers to share their personal context with these applications. Therefore and since information is monetizable, SmarterContext enables fabulous business opportunities for infrastructure and information service providers. For example, a company such as IBM managing Norha's PCS could charge companies such as Target, Sears and Walmart for integrating them into the user's PCS and for providing them with Norha's context information.

Second Law: The value of information is directly proportional to its use

Smarter Commerce businesses are continuously looking for innovative ways to exploit customers' information gathered from business analytics applications and Web 2.0 ap-

plications such as social networks, wikis, and blogs. By deploying these applications in the smart internet it is possible to leverage the amount and quality of this information. However, value creation based on this data depends on the capabilities of businesses to be aware of the existence of this information. In the smart internet, web entities are not only virtual representations of physical entities, but also sources of meaningful information, generally implicit, about users' situations and needs. SmarterContext supports value creation for businesses and users by making this context information explicit and available through PCSs. For this, SmarterContext identifies relevant context information from the interactions of buyers with web entities, reasons about this information to infer implicit contextual facts that improve the knowledge about buyers' intents and situations, and provides this information to sellers effectively.

In our situation-aware smarter commerce case study, SmarterContext supports online retailers in increasing their revenue generation by providing them with relevant and up-to-date context information about user Norha, such as her shopping list, current and preferred locations, and product preferences. Then, businesses can exploit this personal context information to recommend products and services according to the buyer's situation. Finally, since infrastructure providers act as the mediators in this information flow, SmarterContext enables them to generate revenue based on the provisioning of personal context information to businesses. This context information can be exploited by businesses not only to provide individual shoppers with relevant offers, but also to optimize business analytics (BA). BA constitutes another source of revenue generation for infrastructure providers based on the provisioning of anonymous information about users' purchase behaviors from PCSs. In this way, context-driven business analytics can improve not only marketing decisions, but also operational decisions such as the assurance of the e-commerce infrastructure. For example, by analyzing the preferences of users currently logged into a retailer's e-commerce application, it is possible to anticipate the expected system load and thus manage the infrastructure resources accordingly.

Third Law: Information is perishable and depreciable over time

Most frequent online buyers, such as Hausi, the user who still receives children's book offers even though his kids are now adults, have experienced frustration due to irrelevant product and service offers received in their email in-boxes or presented as online advertisements. Indeed, the potential of the smart internet for value cre-

ation and revenue generation is highly dependent on its capabilities to manage the information life cycle. On the one hand, historical information is key in smarter commerce business models to predict trends in the behavior of customers. On the other hand, since the situations of people and businesses change continuously, having no mechanisms to manage the relevance of information over time affects the relationship between businesses and customers negatively. As a result, the management of the context information life cycle, including the disposal of out-of-date or obsolete information, was a critical driver in the design and implementation of SmarterContext. For this, SmarterContext implements two main mechanisms. The first one is the empowerment of the user to dispose context information that is no longer relevant. For example, Hausi can manually eliminate the *children's book* product category from the list of favorite products in his PCS. The second one is the automatic disposal, permanent or temporal, of context information. Automatic permanent disposal of context information occurs when SmarterContext deletes the categories that have exceeded their maximum persistence time from Hausi's context sphere. Automatic temporal disposal of context information occurs when SmarterContext marks context information as no longer relevant, but without eliminating it from the user's PCS, because it may be useful again in the future.

Fourth Law: The value of information increases with accuracy

SmarterContext provides two mechanisms to improve the accuracy of information obtained from smart internet platforms. The first mechanism is its reasoning engine. SmarterContext reasons about gathered raw context information to infer implicit contextual facts, thus enhancing accuracy by enriching the quantity and quality of information available about users.

Since users are the ones who certainly know best what their needs and preferences are, the second mechanism is the integration of the user in the context management loop. SmarterContext gathers context information from the interactions of users with web entities, and empowers users to control the management of their personal context information life cycle.

In smarter commerce scenarios, sellers and buyers benefit from this accuracy by improving revenue generation and user quality of experience (QoE), respectively. User QoE is highly improved because buyers receive personalized offers of relevant products and services. As a result shopping experiences are more pleasant and effective. For

example, user Norha receives accurate offers when she arrives at West Edmonton Mall due to the context information about her location and shopping preferences provided by SmarterContext to the smarter commerce application in her mobile device. Consequently, revenue generation improves since the probability of transforming offers in purchases increases, and the time required by Norha to make purchase decisions decreases.

Fifth Law: The value of information increases when combined with other information

SmarterContext optimizes the value of information stored in PCSs by reasoning about context along several dimensions. For example, by combining two types of context information relevant to user Norha, product and service preferences, and current location, SmarterContext improves the relevance of delivered offers, thus optimizing value creation for this buyer and revenue generation for the seller. Moreover, context information is gathered along the entire web experience, rather than from an individual web application only.

Sixth Law: More information is not necessarily better

Filtering, personalization, customized information feeds, and pre-processing are effective mechanisms to reduce the overload that can produce excessive amounts of irrelevant information [BU11]. These are central aspects in the management of the context information life cycle with SmarterContext. As users' web interactions are the means used by SmarterContext to discover relevant context information, the most important filter is the user. Consequently, SmarterContext gathers only context information that is relevant to the user. Furthermore, the SmarterContext reasoning engine provides context information based on the requirements of the applications integrated into the user's PCS. Interesting revenue generation opportunities based on the provision of information services arise for infrastructure providers from the filtering and reasoning capabilities of SmarterContext.

Seventh Law: Information is not depletable

Rather than depletable, information is self-generating since its analysis leads to more information. Business analytics (BA) results could be greatly improved with SmarterContext. SmarterContext generates new knowledge by combining

different sources of information in the smart internet. BA helps businesses identify and analyze trends and patterns, and anticipate events. This information is crucial to leverage decision making in the process of optimizing business goals and creating value for customers. In retailing, and particularly in online shopping, BA provides useful information to improve the relevance of product and service offers with respect to customer preferences. The goal is to satisfy customer expectations while improving the business income. Despite a range of products available for BA in e-commerce, existing approaches take advantage only of information gathered from the interaction between the buyer and the particular seller's shopping web system. In contrast, SmarterContext can help optimize the results of BA by gathering relevant context about the buyer's behavior throughout the entire shopping experience including other relevant shopping sites. Instead of analyzing customer preferences using traditional business intelligence dimensions, context types can be exploited as a new set of dimensions that are highly relevant for the understanding of customer expectations in smarter commerce business models.

10.2 Improving User Quality of Experience

The general applicability of the contributions of this dissertation demonstrates the impact potential of our research to strategic business sectors and society in general. This section demonstrates the applicability of SmarterContext to user-centric situation-aware smart (SASS) software systems in e-commerce domains. For this, we explain how we instantiated our SmarterContext generic architecture, presented in Section 9.1, to realize dynamic context management in our situation-aware smarter shopping case study.

10.2.1 Applying SmarterContext to User-Centric SASS E-Commerce Applications

Figure 10.1 presents an abstract view of the SmarterContext solution applied to the management of dynamic context in e-commerce. Users (cf. Label 1) register into our SmarterContext infrastructure (cf. Label 3) to create PCSs (cf. Label 5). At any time, users can register web applications (e.g., retailing applications, cf. Label 2), which are compliant with SmarterContext, into their PCSs. Afterwards, SmarterContext monitors the interactions of its users with these applications,

according to privacy and security policies defined by users. From these interactions, SmarterContext gathers and stores contextual data into PCSs. By reasoning on these data, SmarterContext exploits PCSs to help retailers optimize user shopping experiences. Personal context information is stored in PCSs using linked data repositories. Doted arrows represent the integration of context providers and consumers into user PCSs. PCSs are part of the context management cloud infrastructure (cf. Label 3) together with the services for context gathering, reasoning and provisioning. These services are instantiated from the SmarterContext generic architecture that was explained in Section 9.1. Users can maintain their context information through the front-end application (cf. Label 4).

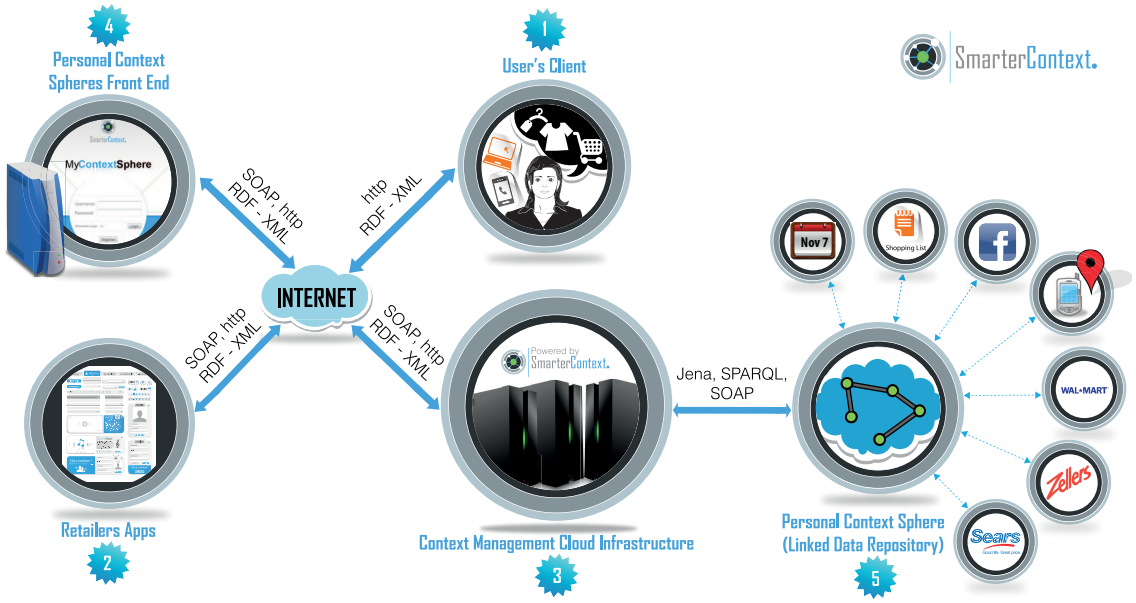


Figure 10.1: An abstract view of the SmarterContext solution for e-commerce

Figure 10.2 depicts the software architecture for the case study on situation-aware smarter shopping. This architecture is deployed on four SCA domains (cf. Labels 1–4). The first one, *personal context sphere application server*, hosts the context sphere application that allows users to maintain their context information, and control the access to their PCSs. The second one, *client device*, contains a SmarterContext browser extension that discovers third parties compliant with SmarterContext, and assists users in the integration of these applications as context consumers and/or providers into PCSs. The third one, *third party application server*, contains the RDF sensors that enable third

parties to exchange context information with the SmarterContext infrastructure. The fourth one, *SmarterContext Server*, contains the composites that define the core of the SmarterContext infrastructure (i.e., `monitoring feedback loop (M-FL)`, `ContextManager`, and `DynamicMonitoringInfrastructure`), and the `AdaptationMiddleware` composite. These composites were explained in Section 9.1.3.

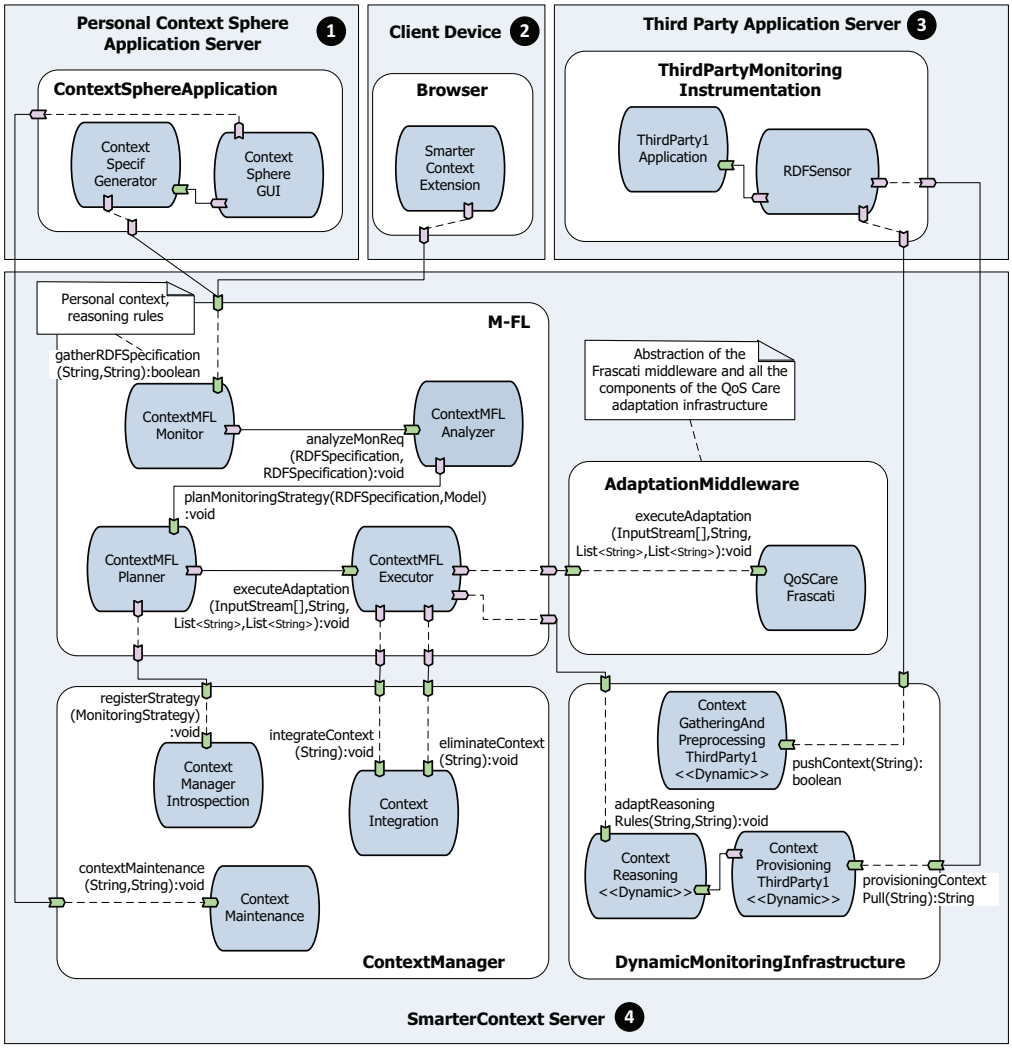


Figure 10.2: The SmarterContext architecture for the smarter shopping case study

10.2.2 Realizing Self-Adaptation

The instantiation of the SmarterContext generic architecture for the case study on situation-aware smarter shopping supports *behavioral* and *architectural* self-adaptation to address changes in reasoning rules, and the addition or deletion of

third parties at runtime (cf. first and second rows in Table 9.1).

Behavioral Self-Adaptation

Control actions in behavioral self-adaptation are any type of signals or parameters that affect the behavior of SmarterContext. In the shopping case study behavioral self-adaptation is realized by changing the set of reasoning rules that the SmarterContext CoRE uses to infer implicit contextual facts from the information gathered from user web interactions. These rules can be pure semantic web deduction rules (cf. Section 5.2.2) or instances of our structural context patterns (cf. Section 5.3.1). This adaptation mechanism is invoked by component `ContextMFLExecutor` in the `M-FL` composite through the service *adaptReasoningRules* exposed by the `ContextReasoning` component of the `DynamicMonitoringInfrastructure` composite. These services were explained for the SmarterContext generic architecture in Section 9.1.2. Predicates involved in context reasoning rules relate to context types that correspond to the domains and ranges of context relationships. Therefore, rules can be classified into context types through predicates. In this way SmarterContext is able to map new context reasoning rules to the context requirements of a particular third party. For example, if an online shopping store is interested in location context, the engine applies rules whose domain and/or range are related to the `gc:LocationContext` type.

Structural Self-Adaptation

As presented in Table 9.1, when users add or delete context consumers and providers, control actions in the form of discrete operations affect the SmarterContext architecture. This adaptation process causes changes in the set of deployed context gathering, and provisioning components (i.e., `ContextGatheringAndProcessing` and `ContextProvisioning` components of the `DynamicMonitoringInfrastructure` composite). Control actions for structural adaptation in SmarterContext are realized through the *executeAdaptation* service that is exposed by the QoS-CARE/FraSCAti adaptation middleware. In the architecture depicted in Figure 10.2, component `RDFSensor` must be already deployed in the third party side (cf. Label 3). However, these sensors could be deployed dynamically having the middleware also available at this side.

10.2.3 Improving the Accuracy of Deal Recommendations with SmarterContext

Daily-deal applications are popular implementations of online advertising strategies that offer products and services to users based on their personal profiles. Current implementations are effective but can frustrate users with irrelevant deals due to stale profiles. To exploit these applications fully, deals must become smarter and situation-aware. This subsection presents SmarterDeals, our deal recommendation system that exploits users' changing personal context information to deliver highly relevant offers [EVMT12]. SmarterDeals relies on recommendation algorithms based on collaborative filtering, and SmarterContext. SmarterContext provides SmarterDeals with up-to-date information about users' locations and product preferences gathered from their past and present web interactions. We implemented SmarterDeals as part of our situation-aware smarter shopping case study. SmarterDeals allowed us to validate the level of improvement of user QoE in terms of the accuracy of deal recommendations, when SmarterContext is integrated into situation-aware applications. Our goal with this experiment was to evaluate the accuracy of a deal recommendation algorithm, proposed by us, that uses personal context maintained by SmarterContext. We validated this algorithm against two baseline approaches: traditional user-based collaborative filtering [AT05], and the approach used by Bell and Koren [Kor08] in the Netflix competition [BYV07]. In contrast to these approaches, our algorithm uses context information from different users to improve the accuracy of daily-deal recommendations. Furthermore, since SmarterContext infers implicit contextual facts from explicit context observations, our recommendation algorithm also takes into account categories that could potentially be relevant to the user, even when the user has not marked them explicitly as relevant in her profile. Appendix E introduces collaborative filtering and explains the two approaches that we used as baselines.

The results of our experiments demonstrate the value of personal context managed with SmarterContext in e-commerce domains. For many deal categories the accuracy of SmarterDeals is between 3% and 8% better than the approaches we used as baselines. For some categories, and in terms of multiplicative relative performance, SmarterDeals outperforms related approaches by as much as 173.4% and 37.5% on average.

A Compelling Application to Evaluate SmarterContext: Groupon

Groupon[1] delivers daily coupons based on the personal information registered by the user during the sign-up process. This information corresponds to the user's gender, age, and favorite locations and deal categories. Users can edit their personal information at any time through Groupon web or mobile applications. Groupon allows users to share deal recommendations by email as well as broadcast them to their social networks.

Even though Groupon has been effective in the accomplishment of its business goal, the current implementation of its daily-deal application can frustrate users with irrelevant deals due to stale profiles. Groupon delivers offers of products and services by taking into account only the information registered by the user during the sign-up process. Nevertheless, most of this information gets out-of-date quickly. In daily-deal applications location and preferred deal categories are types of highly dynamic context information. With respect to the user's preferred locations, Groupon delivers deals related to the whole set of registered locations, which can include different cities. This practice lacks location-aware filtering mechanisms thus compromising the effectiveness of delivered coupons. For example, for users who are frequent travelers, daily deals must be delivered taking into account the users' current location, even if this location is not part of the user's list of favorite locations. Regarding the list of preferred deal categories, sending coupons using only the information that was registered during the sign-up process is ineffective, since the relevance of deal categories is highly dependent on changing context information such as location or time. Hence, categories that could have been relevant yesterday, may no longer be relevant today nor in the near future. For example, a user whose kids are children may be interested in children's books today, but probably not in a few years from now. In Groupon, users must change their personal information and preferences to preserve the relevance of received offers.

Daily-Deals with SmarterDeals and SmarterContext

Figure 10.3 presents an overview of SmarterDeals. Our application is composed of two main artifacts: the *recommendation engine*, and the *filtering and personalization module*. For SmarterContext to provide SmarterDeals with personal context information, users must integrate SmarterDeals into their PCSs. After completing this prerequisite, SmarterContext provides SmarterDeals with personal

[1] http://www.groupon.com

context information about the user's product and service preferences, and locations.

Our recommendation engine exploits context information about the user's product or service preferences to predict daily-deal categories relevant to the user as follows. In the first step our recommendation algorithm correlates similarities among users based on the *Pearson Correlation Coefficient* (PCC) [SM95]. PCC ranges from -1, which indicates a negative correlation, to +1, which indicates a positive correlation between two users. A value of 0 indicates no correlation. Users who have a PCC equal to or greater than 0.7 are considered similar enough in our approach.[2] In the second step, our algorithm aggregates the ratings of product or service categories given by users similar to the user who will receive the recommendation, to predict the rating of the corresponding product or service category. SmarterDeals decides whether a product or service category is relevant to a user using the predicted rating of the corresponding category. In this case study ratings range from 1 to 5, where 1 indicates the lowest level of relevance and 5 the highest. Our algorithm recommends categories with predicted ratings equal to or greater than 4. Once the recommendation engine has predicted the list of potential relevant categories, in the third step SmarterDeals uses Groupon's application programming interface (API) to provide users with business daily-deal offers filtered by the user's locations and preferences.

SmarterDeals, supported by SmarterContext, improves the relevance of daily product and service recommendations delivered to users by exploiting:

- up-to-date product and service categories gathered from web interactions performed by the users throughout their web experiences, and
- up-to-date information about current and preferred users' locations.

Simulating Contextual Data with the Yelp Data Set

PCSs store personal contextual data in the form of RDF graphs. To validate our recommender system with real data, we used the Yelp academic data set [Yel04].

The Yelp data set includes 271,418 ratings that 65,411 real users have given to 6,900 local businesses. Ratings range from 1 to 5, and local businesses have been tagged with one or many of the 365 product and service categories defined by Yelp. We used the information of the Yelp data set to create 65,411 RDF graphs. Each

[2] Any value between 0.5 and 1 can be used to represent strong association between two variables. We considered 0.7 as a suitable measure for this case study. https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php

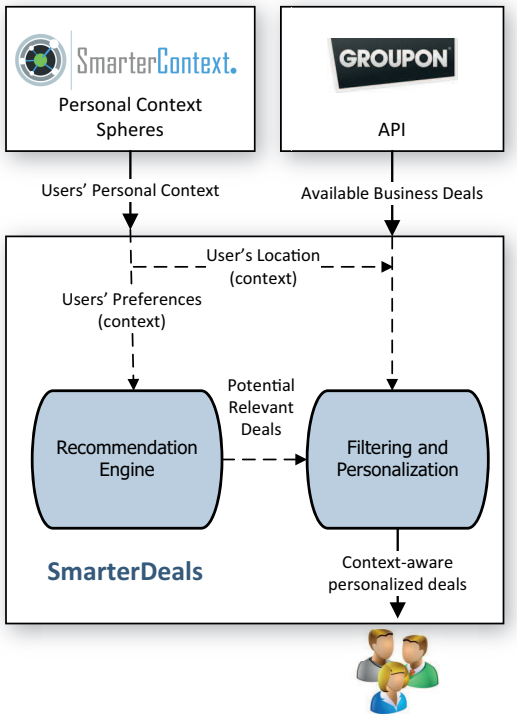


Figure 10.3: SmarterDeals: our context-aware recommender system

graph simulates the context sphere of a user. Since in the Yelp data set users have rated businesses instead of products or services, we obtained favorite users' product and service categories from the categories associated with the businesses rated by these users. Yelp businesses are located in 67 different cities across North America. Users' locations were obtained from the locations of the businesses reviewed by each user. Nevertheless, the set of cities related to businesses in Groupon differs from the set of cities related to businesses in Yelp. Thus, to use Groupon's API in SmarterDeals, for each location associated with a user in Yelp, we considered all of the nearby locations in Groupon. Therefore, if Groupon is unavailable in the user's relevant locations, we can still deliver Groupon deals related to nearby locations. For this, we extended the SmarterContext `geo`[3] vocabulary that defines geographical locations to include the locations used by Groupon and Yelp.

Groupon defines 633 product and service categories classified into 18 general categories whereas, the Yelp data set contains 433 product or service categories with no hierarchies. 284 of these categories are exactly equal to those in Groupon. Thus, to recommend deals based on the product and service categories defined by Groupon,

[3]http://smartercontext.org/vocabularies/rdf/geo.rdf

we mapped manually the remaining 149 Yelp categories into similar GROUPON categories. SMARTERCONTEXT classifies products and services using the Google product taxonomy [Goo12]. For this case study, we extended this taxonomy by creating a complementary ontology, the `deals` ontology,[4] from the set of Yelp product and service categories mapped to GROUPON deal categories.

Since some Yelp users have a very small number of ratings, we reduced the Yelp data set by considering only users who have at least 20 ratings. The reduced data set has 58,069 ratings given to 313 product or service categories by 1,683 users. These 313 categories, now mapped into GROUPON categories, belong to 17 parent categories. This set of 17 parent categories was further reduced to 14 parent categories by eliminating categories with less than 50 ratings, which we considered as not statistically significant.

Our Recommendation Algorithm

Our recommendation algorithm is a variation of *adjusted collaborative filtering (ACF)* (cf. Appendix E), that applies to hierarchies of item categories, where rating prediction is based on item categories that belong to the same parent category. That is, we calculate similarities among items whose immediate category belongs to the same immediate super-category. We hypothesized that by partitioning items according to their parent categories it is possible to improve the accuracy of recommendations. This is because aspects such as the sensitivity of the parent category may affect the ratings given by users. For example, users may rate products or services related to "Health & Fitness" more carefully than those classified as "Nightlife". In this case study, we partitioned the deal categories to be recommended according to the parent categories defined by GROUPON.

Figure 10.4 illustrates our approach. Instead of having an overall average rating μ, we calculate average ratings for each parent category P. Moreover, we compute the user's observed deviation b_u for each parent category P. Thus, instead of having a unique b_u per user, we have a b_{uk} for each parent category related to the items rated by the corresponding user, as follows:

[4] http://smartercontext.org/vocabularies/rdf/dealcategories.owl

$$b_{uk} = \frac{\sum_{i \in R_k(u)} (r_{ui} - \mu_k - b_i)}{\lambda_3 + |R_k(u)|}$$

Equation 10.1. User deviation for P_k

In Equation (10.1), $R_k(u)$ denotes the set of items rated by user u into parent category P_k, and μ_k denotes the average of ratings given to items classified into parent category P_k (cf. the section of the matrix that corresponds to the horizontal arrow in the upper part of Figure 10.4).

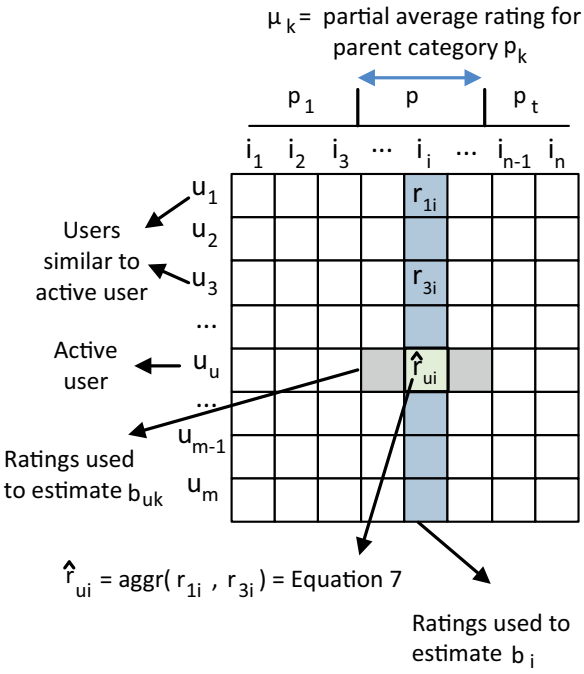


Figure 10.4: Our recommendation algorithm for daily-deals

Similarity between users is calculated using PCC with 0.7 as the threshold (cf. Equation (E.1) in Appendix E) and b_i is calculated as in ACF (cf. Equation (E.5)). Finally, the unknown rating $\hat{r}_{ui}$ given by the active user u to item i is predicted with the aggregate function that is used in ACF (cf. Equation (E.7)), with $b_{ui} = \mu_k + b_{uk} + b_i$, with $i \in P_k$ (cf. Figure 10.4).

Validation Results

We implemented the evaluation process for SmarterDeals as follows. For each existing r_{ui} rating, we created a new version of the Yelp dataset, $Y_{-r_{ui}}$, with r_{ui} removed. Then, we applied our approach and the two approaches we used as baselines to predict the deleted rating r_{ui}. The predicted rating $\hat{r}_{ui}$ may or may not be the same as r_{ui} (the known rating). In general, we are interested to see whether the error $e_{ui} = r_{ui} - \hat{r}_{ui}$ is small. We repeat this procedure for each existing rating r_{ui}, i.e. we ran the algorithms as many times as there are ratings in the dataset. This exhaustive evaluation gave us a precise picture of the quality of each approach. To measure the effectiveness of each approach we used *root mean squared error (RMSE)* as defined in Equation (10.2), a widely accepted metric to assess the accuracy of the values predicted by a model or an estimator with respect to the values actually observed [HKTR04].

$$RMSE = \sqrt{\sum_{(u,i) \in TestSet} \frac{(e_{ui})^2}{|TestSet|}}$$

Equation 10.2. Root mean squared error

Since our approach relies on partitions based on parent categories, we designed our tests as follows: First, we predict the rates for every single parent category independently. Second, we compute the RMSE for each parent category. Third, we apply the procedure for every recommendation technique to predict deleted ratings.

Table 10.1 presents our validation results. Column *Parent Category* contains the 14 Groupon parent categories included in the reduced Yelp data set. Columns *Classic CF (C)*, *ACF (A)*, and SmarterDeals *(S)* present the RMSE for the two baselines—the traditional user-based collaborative filtering method and ACF approach, and our context-driven approach, respectively. Column *(AiC)*, calculated as a percentage $(C - A)/C$, corresponds to the improvement of A over C. That is the improvement of the error measure (RMSE) of ACF with respect to the traditional user-based recommendation method. Similarly, Column *(SiC)* represents the improvement of SmarterDeals (S) over the traditional method (C), and is calculated as a percentage $(C - S)/C$. Column *SiC-AiC* compares the improvement of S over C with respect to the improvement of A over C. Finally, Column *Relative Per-*

formance (RP), calculated as a percentage $(SiC - AiC)/SiC$, represents the relative improvement of our approach with respect to the ACF approach. Figure 10.5 presents the improvement in terms of accuracy of the ACF approach (AiC), and our approach (SiC), with respect to the traditional user-based collaborative filtering method (C). Figure 10.6 presents the relative performance of SmarterDeals with respect to the ACF approach.

Table 10.1: Validation results for SmarterDeals.

Parent Category	Classic CF (C)	ACF (A)	SmarterDeals (S)	(AiC)	(SiC)	SiC-AiC	Relative Performance (RP)
Automotive	1.16	1.15	1.12	1.4%	3.9%	2.5%	173.4%
Financial Services	1.67	1.60	1.55	3.9%	6.8%	2.9%	74.8%
Real Estate	1.74	1.68	1.64	3.1%	5.3%	2.3%	73.2%
Health & Fitness	1.09	1.02	0.99	6.2%	9.3%	3.0%	48.4%
Beauty & Spas	1.17	1.08	1.04	7.4%	11.0%	3.5%	47.1%
Education	1.16	1.11	1.09	4.5%	6.6%	2.1%	45.5%
Travel	0.97	0.91	0.89	6.1%	8.4%	2.3%	36.8%
Food & Drink	0.95	0.87	0.85	8.1%	10.3%	2.2%	26.7%
Arts and Entertainment	0.93	0.86	0.84	7.2%	9.0%	1.9%	26.2%
Restaurants	0.95	0.87	0.86	8.7%	9.8%	1.1%	12.7%
Shopping	1.60	1.55	1.55	3.1%	3.3%	0.2%	7.6%
Nightlife	0.85	0.76	0.75	11.6%	12.2%	0.6%	5.3%
Professional Services	0.90	0.78	0.80	12.6%	10.3%	-2.3%	-18.5%
Public Services & Government	1.11	0.98	1.03	11.8%	7.8%	-4.0%	-34.0%
Average	1.16	1.09	1.07	6.8%	8.1%	1.3%	37.5%

Our approach is about 8.1% more accurate than classic CF and 1.3% than ACF. For about half of the categories, SmarterDeals is more than 2% better than ACF.

To put these results in perspective, it is important to point out that the Netflix competition, which carried a 1 million dollar prize, was about improving the RMSE compared to the recommender system of Netflix by 10%.[5]

In terms of multiplicative relative performance, for some categories the accuracy of SmarterDeals is much better than ACF's (e.g., for Automotive our approach outperforms ACF by 173.4%, and 37.5% on average).

There are two parent categories from the 14 included in the tests, *Professional Services* and *Public Services and Government*, for which our method did not do very well. This seems attributable to the broad definition of these two categories.

[5]Most of the teams got improvements by less than 2%.

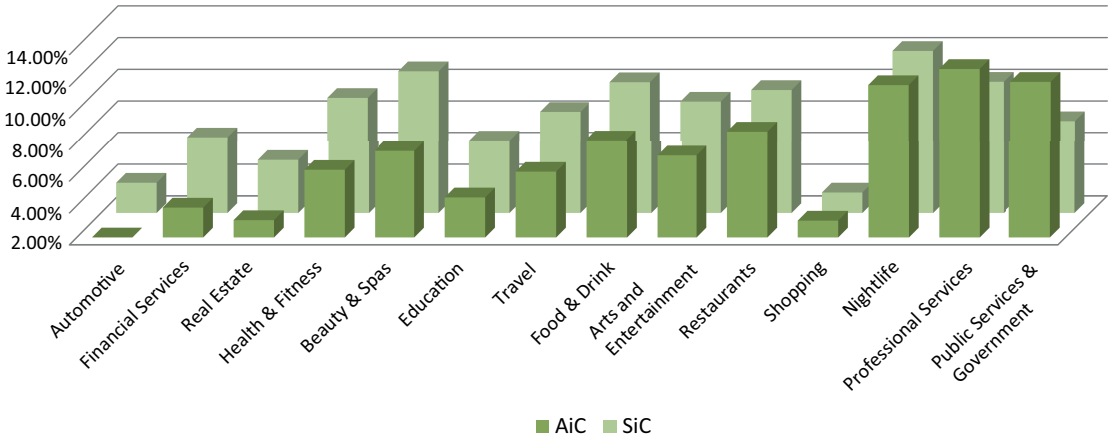


Figure 10.5: Improvement of ACF (AiC), and SmarterDeals (SiC) with respect to classic CF (C)

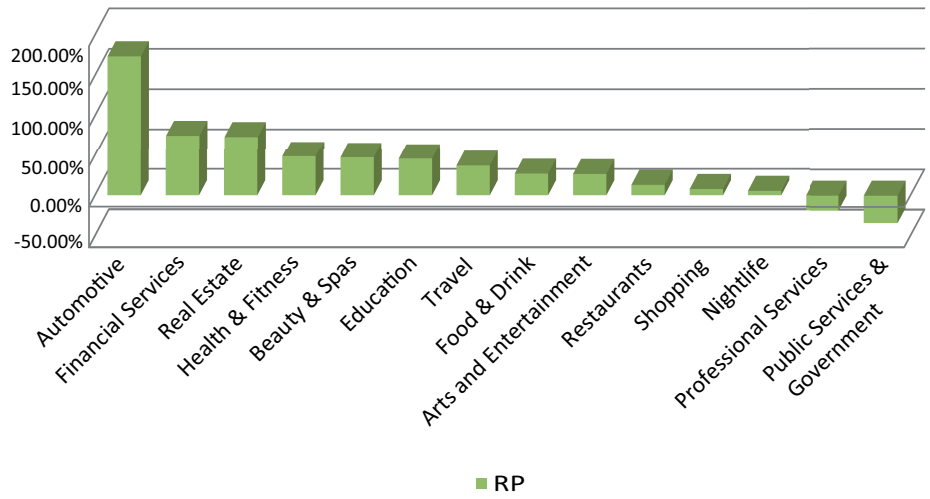


Figure 10.6: Relative performance of SmarterDeals (S) with respect to ACF (A)

10.3 Improving Self-Adaptivity

Chapter 7 presented Dynamico, our reference model for engineering context-driven self-adaptive software systems. Dynamico models three types of feedback loops that

correspond to the three levels of dynamics that we characterized for adaptive software, namely the *control objectives* feedback loop, the *target system adaptation* feedback loop, and the *monitoring* feedback loop. The dynamics among these three levels must be controlled at runtime to manage the coherence of both the adaptation process and context monitoring mechanisms with respect to system goals (i.e., control objectives).

SmarterContext provides a sound solution for the third feedback loop defined in Dynamico, the context monitoring feedback loop. This section demonstrates the applicability of our dynamic context monitoring solution, which was derived from Dynamico, to business-centric domains. In particular, to our dynamic SOA governance case study where a service-oriented target system is adapted at runtime to satisfy changing QoS contracts. For this, SmarterContext adapts itself to support the addition/deletion of both quality attributes and monitoring conditions defined in SLAs.

10.3.1 Applying SmarterContext to Dynamic SOA Governance

We used Dynamico to design and implement an adaptation mechanism that adapts an e-commerce SOA application to guarantee the satisfaction of a performance SLA contracted between a retailer and a cloud infrastructure provider. *Throughput*, defined as the time spent to process a purchase order request ($ms/request$), is the quality factor defined for the initial version of the SLA. In light of this system goal, the adaptation mechanism (e.g., the QoS-CARE/FraSCAti framework [Tam12]) will reconfigure the e-commerce application by deploying new purchase order processing components when the processing capacity must be augmented. Later, the performance SLA is re-negotiated by adding *capacity* as a new quality factor. The capacity contracted conditions state that the way of delivering the catalog of products varies according to the client's bandwidth. That is, depending on the bandwidth, product descriptions can include: (i) text and a thumbnail of the original catalog image, (ii) text, a thumbnail, and the original image, or (iii) text, a thumbnail, the original image, and a descriptive video of the product. Figure 10.7 depicts the instantiation of the SmarterContext generic architecture for this scenario. Notice that the architecture of the `M-FL` composite is the same as the architecture of the `M-FL` composite for the smarter shopping case study (cf. Figure 10.2). The only variable components are the context gatherers and handlers (`ContextReasoning` components in the shop-

ping case study, and `ContextMonitoring` components in this case study), which are deployed dynamically and according to their specifications for the application domain. This demonstrates the general applicability of SmarterContext.

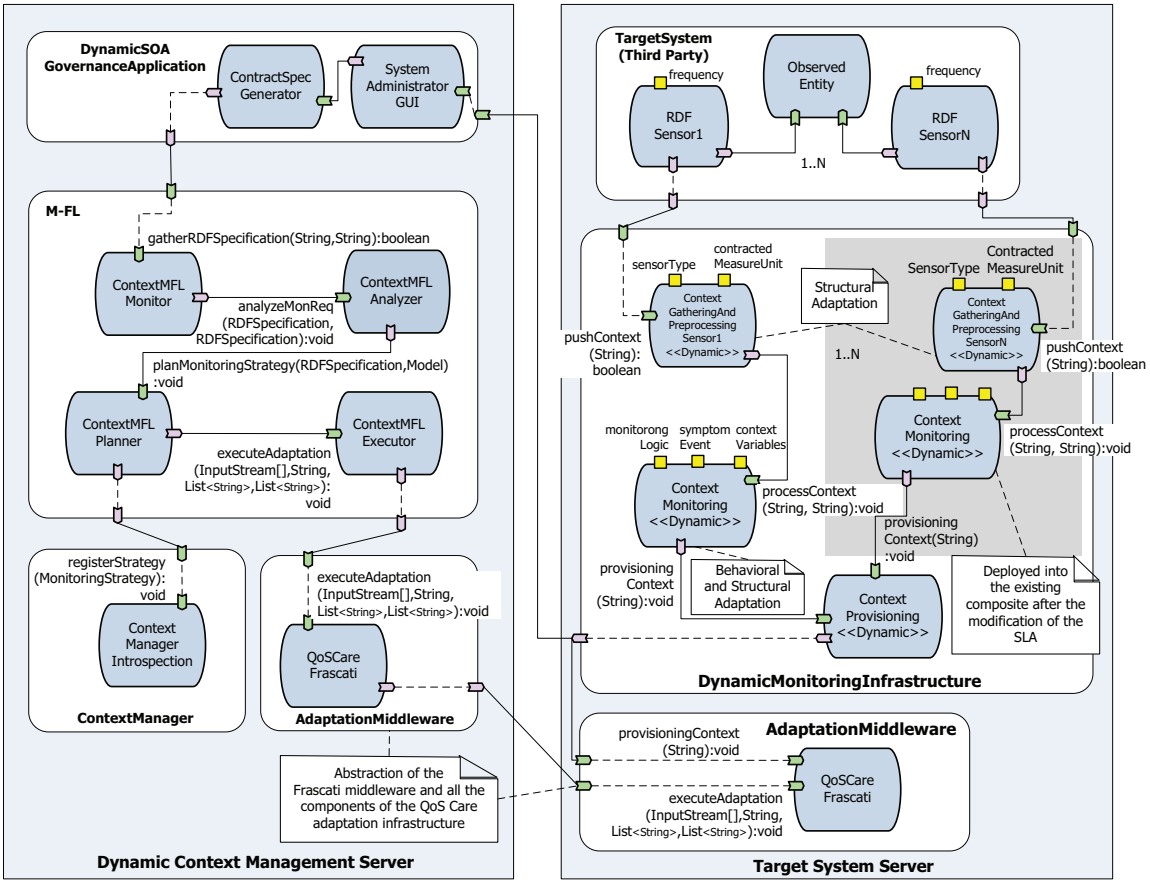


Figure 10.7: The SmarterContext arquitecture for the dynamic SOA governance case study

Composite `DynamicSOAGovernanceApplication` corresponds to the first level of dynamics in Dynamico. The components of this composite, `SystemAdministratorGUI` and `ContractSpecGenerator`, allow system administrators to register new and modify existing SLAs, and to generate the control objectives (COb) specifications that feed the other two levels of dynamics (i.e., the adaptation and monitoring feedback loops).

Composites `AdaptationMiddleware`, deployed on the dynamic context manager server and the target system server to allow the dynamic deployment of monitoring components, and `ContextManager` were explained in Section 9.1.1. Composite `M-FL`,

which implements the feedback loop that control the adaptation of the monitoring mechanism, was explained in Section 9.1.3.

Composite `DynamicMonitoringInfrastructure` contains the components deployed dynamically to monitor the contracted conditions of the SLA. Figure 10.7 highlights, within the `DynamicMonitoringInfrastructure` composite, the `ContextGatheringAndPreprocessing` and `ContextMonitoring` components that are deployed to monitor the new bandwidth quality factor that is added after renegotiating the performance SLA.

10.3.2 Realizing Dynamic Monitoring Strategies

In DYNAMICO, the satisfaction of system goals and adaptation properties (control objectives) is regulated by the adaptation and monitoring feedback loops (i.e., second and third levels of dynamics in self-adaptive systems), whereas changes in system goals are regulated by the control objectives feedback loop (i.e., first level of dynamics).

To control the relevance of adaptation and monitoring mechanisms with respect to system goals (i.e., control objectives), it is necessary to model control objectives and map them explicitly to monitoring requirements. These models must be manipulable at runtime.

Control objectives (CObj) specifications allow SMARTERCONTEXT to (i) synthesize monitoring strategies for implementing the monitoring mechanism required to support the adaptation process, and (ii) identify changes in existing monitoring strategies when existing system control objectives change or new ones appear.

In our dynamic SOA governance case study, CObj specifications correspond to SLAs that not only define the contracted service level objectives (SLO) and corresponding metrics, but also specify monitoring conditions associated with metrics, sensing interfaces and guarantee actions.

Control Objectives Specifications

Figure 10.8 depicts the *control objectives* ontology for QoS contracts in SMARTERCONTEXT. This ontology allows the specification of control objectives (e.g., SLAs in our dynamic SOA governance case study) mapped to elements of both monitoring strategies and adaptation mechanisms represented by entities derived from the *context monitoring strategy (cms)* ontology. Even though cardinalities are not usually specified in RDF graph-based representations, we represent them for explanation

purposes.

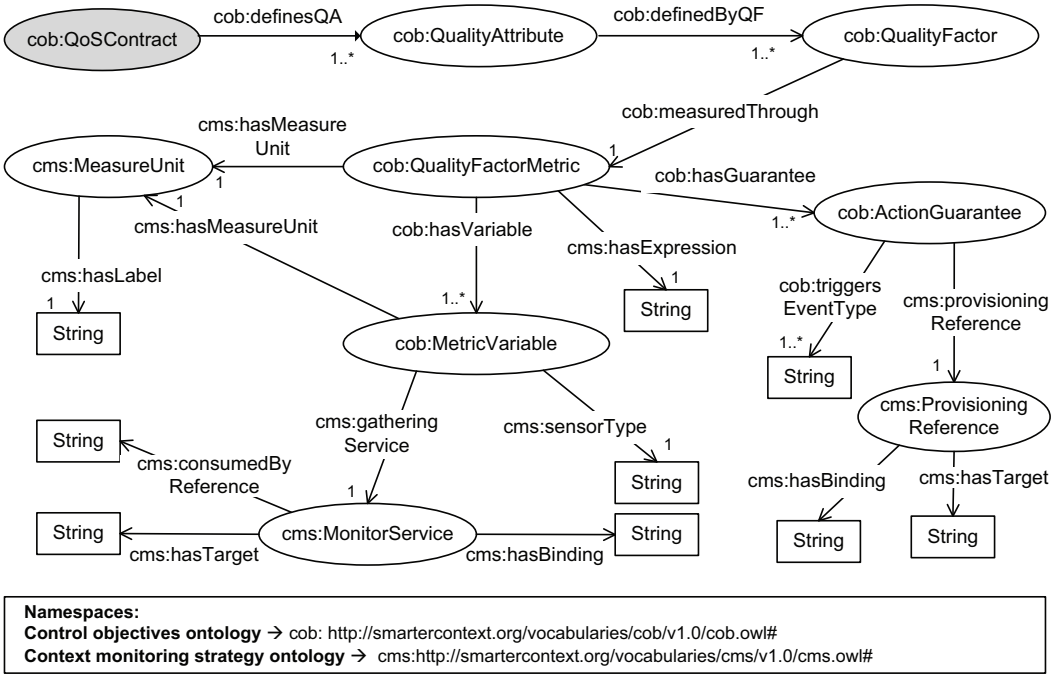


Figure 10.8: The control objectives (CObj) ontology

The highlighted node represents the root type of any quality-driven CObj specification, `cob:QoSContract`, which corresponds to an SLA in the case of SOA governance. Our ontology for the specification of QoS contracts, particularly SLAs in service-oriented architecture environments, is based on the characterization of SLAs contributed by researchers from *Software Engineering Institute (SEI)* [BLM08]. Their SLA characterization supports our vision on what must be explicitly defined in control objectives specifications. According to them, a properly specified SLA must include, among others, (i) the metrics to be collected, (ii) who will collect these metrics and how, and (iii) the actions to be taken when the service does not meet the contracted conditions. In CObj specifications supported by SmarterContext (cf. Figure 10.8), these aspects correspond to (i) monitoring conditions, (ii) monitoring services, and (iii) action guarantees associated with adaptation events or notifications to system administrators.

An SLA in a SmarterContext CObj specification must include at least one quality attribute. Quality attributes in SmarterContext are instances of type `cob:QualityAttribute` and represent measurable qualities that can be observed automatically through a sensing interface. Each quality attribute is composed of at

least one quality factor. Quality factors are instances of type `cob:QualityFactor` and correspond to quality concerns or properties that allow the measurement of quality attributes. In SmarterContext CObs specifications, each quality factor must be associated with at least one metric. Metrics are instances of type `cob:QualityFactorMetric` and define the variables (`cob:MetricVarible`), the evaluation expression (`String`), the measure unit (`cms:MeasureUnit`), and the action guarantee (`cob:ActionGuarantee`) required to control the preservation of the contracted quality attributes. A metric must be associated with one and only one evaluation expression, and one unit of measurement, and at least one action guarantee. Action guarantees are instances of type `cob:ActionGuarantee` and specify the event types, with corresponding endpoints, to be triggered when the contracted conditions are violated.

The mapping between control objectives and monitoring strategies is realized through the association of elements of type `cob:MetricVariable` to instances of type `cms:MonitorService`. `cms:MonitorService` objects provide the mechanisms to collect the metrics required to assess the contracted qualities. Each metric variable must be associated with one and only one sensor type, and one and only one monitoring service. The type `cms:MonitorService` allows the specification of the URI of the reference that will consume the monitoring interface when the gathering is based on a pushing mechanism (i.e., the source sends the sensed data to the SmarterContext infrastructure). Similarly, `cms:MonitorService` allows the specification of the URI of the target reference or binding when the gathering mechanism is realized through a pulling mechanism (i.e., the SmarterContext infrastructure pulls the sensed data from the source).

The mapping between control objectives and adaptation mechanisms is realized through the specification of action guarantees. Each object of type `cob:ActionGuarantee` is associated with an event type that corresponds to the symptom used by the adaptation mechanism analyzer to decide about adaptation actions to be taken. Action guarantees also specify an object of type `cms:ProvisioningReference` that defines the binding or target URI to be consumed. Besides adaptation actions, action guarantees can imply notifications to system administrators.

Figure 10.9 represents, partially, a CObs specification for the performance SLA that resulted from the first negotiation in our case study.

Namespace `qa:` corresponds to the vocabulary that characterizes quality at-

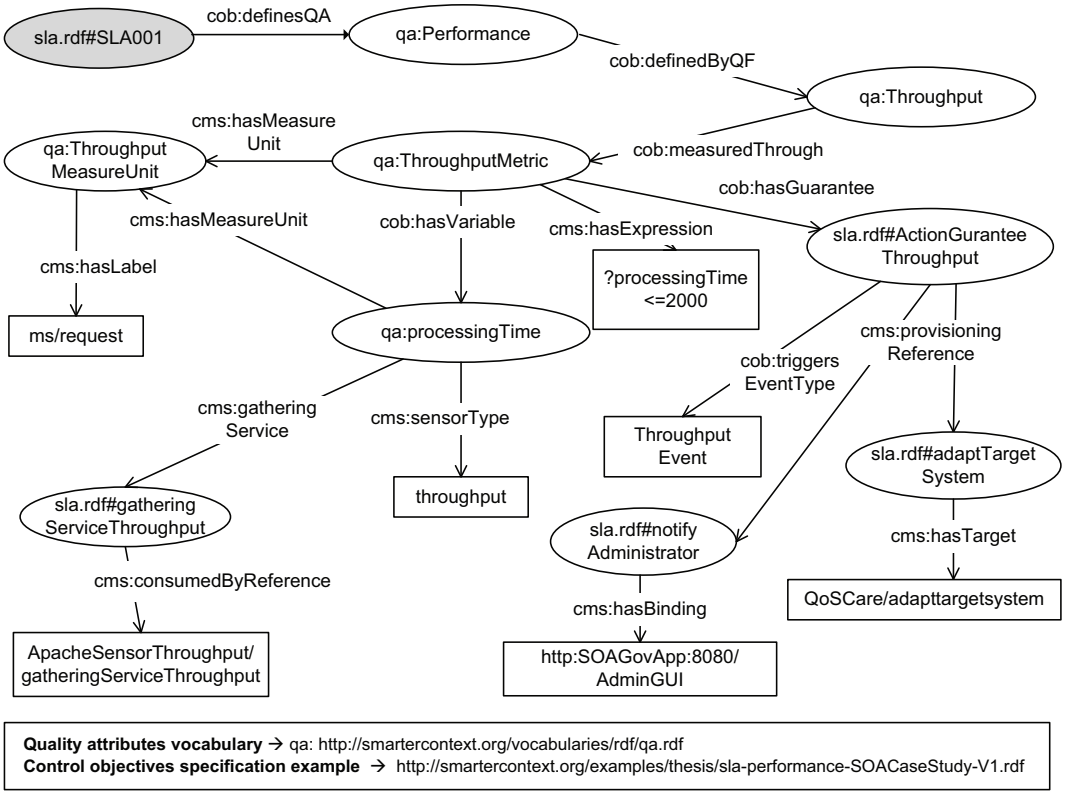


Figure 10.9: A control objectives specification example for the throughput quality attribute defined in the first negotiation of the performance SLA.

tributes mapped to quality factors in our dynamic SOA governance case study. This version of the performance SLA defines a throughput quality factor, measured through a throughput metric (`qa:ThroughputMetric`) that is composed of a single variable (`qa:processingTime`). This variable is involved in the metric expression $?processingTime \leq 2000$, measured in terms of *ms/request* (as defined in the element `qa:ThroughputMeasureUnit`) and associated with a `cms:MonitorServiceType` identified as `sla.rdf#gatheringServiceThroughput`. The action guarantee associated with the throughput quality factor (`sla.rdf#ActionGuaranteeThroughput`) is associated with two provisioning references. The first one, `sla.rdf#adaptTargetSystem` is to invoke the service in charge of activating the adaptation process. The second one, `sla.rdf#notifyAdministrator`, is to inform business administrators about the violation of the contracted throughput conditions.

Synthesizing Monitoring Strategies at Runtime

In our SmarterContext solution monitoring strategies can be generated dynamically from COb specifications such as the one depicted in Figure 10.9. A monitoring strategy is defined as `DynamicMonitoringInfrastructure` composite (cf. the architecture depicted in Figure 10.7) that specifies components for context gathering, pre-processing, monitoring, and provisioning. These strategies are dynamic because SmarterContext supports at runtime the modification of the monitoring logic, and, enabled by an architectural adaptation middleware, the deployment of new context management components.

Figure 10.10 illustrates, for our dynamic SOA governance scenario, the generation of the `DynamicMonitoringInfrastructure` composite (cf. the highlighted composite in the same figure) from the COb specification presented in Figure 10.9. The RDF subgraphs associated with elements `qa:ThroughputMetric` and `sla.rdf#ActionGuaranteeThorughput` constitute the foundational elements for generating the `DynamicMonitoringInfrastructure` composite. The dotted connectors associate elements of the COb specification with the corresponding architectural artifact. For example, the connector labeled with number 1 indicates that component `ContextMonitoring` is dynamically generated from metric `qa:ThroughputMetric`.

Changes in COb specifications are supported in a similar manner. When an existing SLA is renegotiated, the `DynamicSOAGovernanceApplication` generates a new COb specification. Then, the SmarterContext monitoring feedback loop (`M-FL`) gathers this specification, analyzes whether it corresponds to renegotiated SLA, and if so, generates a new plan with consists of the set of new context gathering and monitoring components to be deployed. Finally, the `M-FL` executes the plan to deploy the new components.

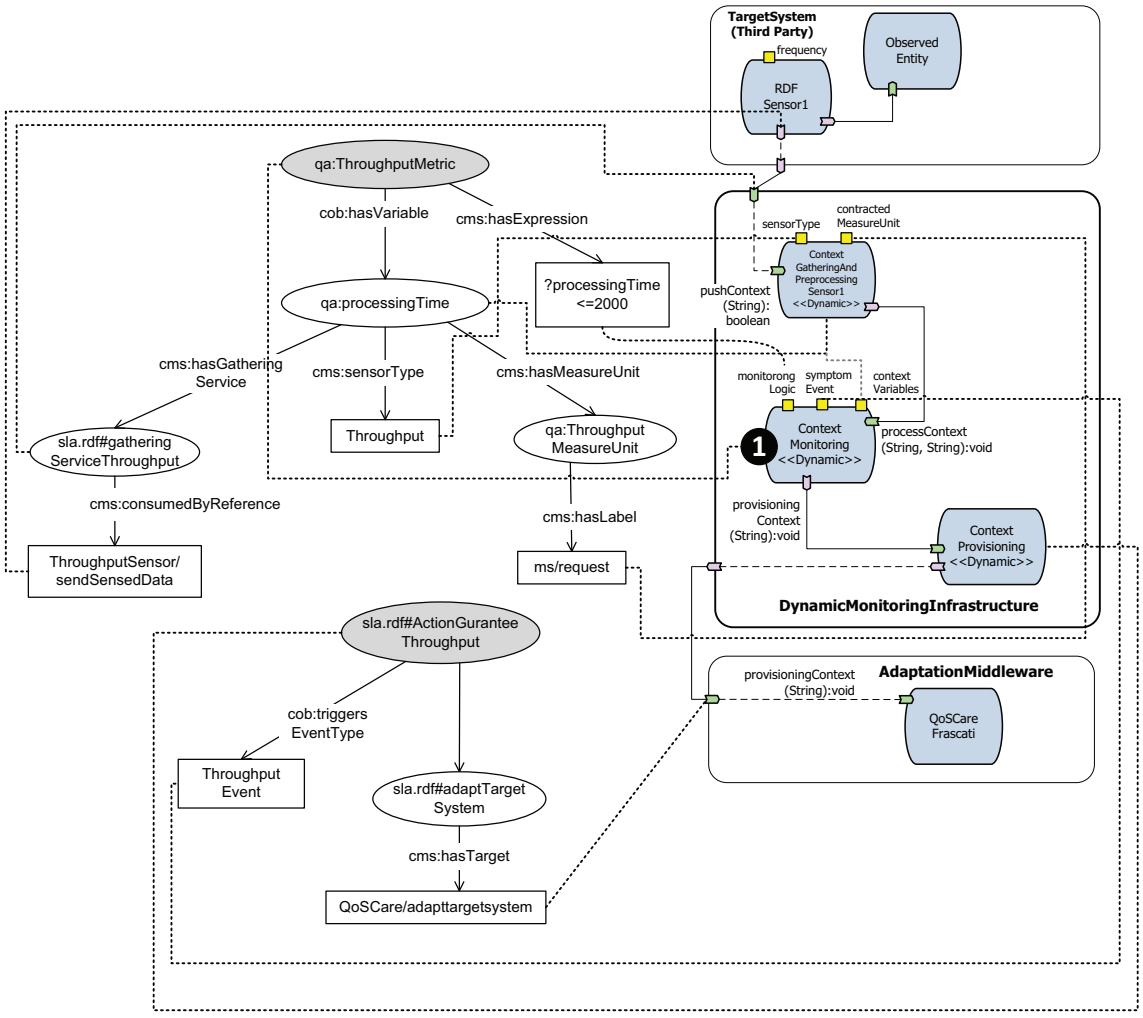


Figure 10.10: Synthesizing monitoring strategies dynamically. The dashed connectors associate the element from the CObs specification to the corresponding architectural artifact in the architecture.

Table 10.2 presents the general mapping between elements of the SmarterContext CObs ontology and artifacts of monitoring strategies. This mapping enables the generation of monitoring strategies and deployment of the corresponding monitoring infrastructure at runtime.

10.3.3 Experimental Evaluation

SmarterContext realizes the dynamic monitoring feedback loop specified by Dynamico. QoS-CARE realizes the adaptation framework to reconfigure both the dynamic monitoring infrastructure of SmarterContext and the target system. In this section we present a comparative evaluation of our implementation against the

Table 10.2: Mapping between elements of the COb ontology and SCA artifacts that constitute context monitoring strategies.

	Context Monitoring Strategy	
COb Entity	Monitoring SCA Artifact	SCA Type
`cob:QoSContract`	`DynamicMonitoring Infrastructure`	composite
`cob:QualityFactorMetric`	`ContextMonitoring`	component
Value of `cms:hasExpression`	`monitoringLogic`	component property
`cob:MetricVariable`	`ContextVariables`	component property
	`ContextGatheringAnd Preprocessing`	component
Value of `cms:sensorType`	`sensorType`	component property
Value of `cms:hasLabel`	`contractedMeasure Unit`	component property
`cms:MonitorService`	`pushContext(String): boolean`	service/interface
Value of `cms:consumedByReference`	reference in the third party sensor	reference
`cob:ActionGuarantee`	`contextProvisioning`	component
Value of `cms:hasTarget` or `cms:hasBinding`	`provisioningContext (String):void`	service/interface
Value of `cob:triggers EventType`	`symptomEvent`	component property

highly cited Rainbow/Znn.com, the self-adaptive implementation developed by Garlan *et. al* [GCH+04]. For this evaluation we conducted several experiments to analyze the feasibility and effectiveness of DYNAMICO to improve context-awareness of SASS solutions derived from it. Our case study focuses on a dynamic SOA governance scenario where Znn.com is used as a service-oriented target system to be adapted in order to guarantee service level agreements (SLA), which are renegotiated at runtime. Given that Znn.com was explicitly developed to evaluate the effectiveness of Rainbow [CGS09], we also use it as our target system for comparability purposes.

On one side, the evaluation results demonstrate the feasibility of our reference

model in terms of the *settling time* (i.e., the time required to adapt a monitoring strategy upon changes in control objectives) of our Dynamico implementation. On the other side, these results demonstrate the effectiveness of using Dynamico to improve context-awareness in SASS solutions. Finally, based on the findings of our comparative evaluation, we analyze the suitability of Znn.com as a benchmark target system to compare self-adaptation implementations.

10.3.4 Evaluation Criteria

Cheng *et. al* used the following three factors for evaluating the effectiveness of Rainbow to adapt Znn.com: (i) performance, measured in terms of the time required to perform the adaptation (i.e., called settling time in control theory [HDPT04]), and the runtime overhead caused by executing Rainbow with Znn.com as the adaptation mechanism; (ii) the engineering effort required to use Rainbow to add self-adaptive capabilities to Znn.com; and (iii) the capability of Rainbow to maintain system quality attributes under changing conditions of execution. The following sections present the results of our case study evaluation using these factors.

10.3.5 Performance

To evaluate the performance of our Dynamico implementation we realized the dynamic SOA governance scenario with several platform configurations. The different hardware and software configurations were based on Intel i3@2.4Ghz processors with 4 GB of RAM running GNU/Linux Fedora 16 with non-relevant services and applications shut down. For the context monitoring infrastructure we used SmarterContext v1.5,[6] whereas for the SCA platform with autonomous reconfiguration capabilities we used QoS-CARE v1.3[7] and FraSCAti v1.4[8] with Java 1.6.0_23 with 128 MB of RAM.

We measured the performance of the Dynamico implementation in terms of (i) the settling time of both the adaptation of the target system and the adaptation of the monitoring infrastructure, and (ii) the overhead caused by adding our Dynamico implementation in the execution of Znn.com. On the one hand, from the execution of the client tester that changes the SLA in the CO-FL, we obtained the settling

[6] Available from http://gforge.icesi.edu.co/svn/smartercontext
[7] Available from https://scm.gforge.inria.fr/svn/scesame/qos-care
[8] Available from svn://svn.forge.objectweb.org/svnroot/frascati

time and overhead measurements for the adaptation of the monitoring infrastructure (cf. Table 10.3). The first row in this table indicates that the CO-FL, which keeps track of changes in control objectives and sends COb specifications to the M-FL, takes 698 ms for the first scenario (i.e., SLA-v1 requiring one throughput gatherer and one throughput processor) and 732 ms for the second (i.e., SLA-v2 requiring the addition of two bandwidth gatherers and two bandwidth monitors—for monitoring upload and download capacity). The second row depicts the timings for the M-FL execution, which analyzes and synthesizes the adaptation plan of the monitoring infrastructure. The third row indicates the timings for instrumenting the synthesized monitoring infrastructure (i.e., deploying and binding the context processors and gatherers). The total settling times, and the measured overhead of Dynamico, which is practically negligible in both scenarios, represent an acceptable cost for having these two additional feedback loops for adapting the monitoring infrastructure.

Table 10.3: Average settling time and overhead (ms) of the monitoring infrastructure adaptation for SLA-v1 and SLA-v2

	SLA-v1 [ms]	SLA-v2 [ms]
CO-FL	698	732
M-FL	21	29
Adaptation Instrumentation	1,131	1,579
Total settling time	1,850	2,340
Overhead	3	3

It is worth noting that Rainbow has no support for adapting the monitoring infrastructure of the adaptation mechanism. Thus, Rainbow and Dynamico cannot be compared in this regard. However, throughout this section we have demonstrated that the CO-FL and M-FL, the two additional feedback loops specified by Dynamico, improve the context awareness of our adaptation mechanism.

On the other hand, from the execution of the client tester that changes the Znn.com execution conditions to induce its adaptation, we obtained an average settling time of 84 ms and an overhead of 2 ms. Nonetheless, the comparative analysis of this settling time presented several difficulties. First, it makes no sense to disaggregate this measurement, given that we used the same strategy as Rainbow to adapt Znn.com. That is, by modifying parameters of the configuration file of Apache, for example to vary the number of server thread pool. Even though this kind of adaptation

is argued to be architectural, because it makes Apache to configure internally, a different number of server threads (i.e., software components in its internal architecture), it can be argued that it is parametric. This is because the adaptation mechanism modifies a parameter (a single value) that affects the target system behavior, not its actual software structure. Of course, as Znn.com is a PHP monolithic application without introspection capabilities, modifying its software structure is impractical. Second, the settling time of Rainbow to perform a target system adaptation was measured for a videoconference system in a different paper (i.e., [GCH+04]) than the one that used Znn.com. In [GCH+04], the reported measurement on the adaptation settling time is 2,100 ms and 2,700 ms for two different scenarios.

Despite of the aforementioned difficulties to perform a comparative analysis, the obtained results demonstrate the feasibility of implementing our reference model, considering the complexity of combining three different types of feedback loops in the adaptation mechanism.

10.3.6 Engineering Effort

Measuring the engineering effort to develop a software system, on an absolute scale and independently of human factors, is well known to be a challenging task. This often means that engineering efforts of different development teams cannot be compared.

To evaluate the effort of enabling DYNAMICO to adapt Znn.com with that of Rainbow, we considered the tasks to be developed in similar conditions. These tasks are (i) development and testing of three types of required sensors (12 h in DYNAMICO vs. 49 h in Rainbow), (ii) development and testing of adaptation scripts (8h vs. 21h), and (iii) architecture (target system-adaptation mechanism) deployment configuration (11 h vs. 24 h), for a total of 31 h vs. 94 h.[9] The ratio of the development effort DYNAMICO/Rainbow is around 1/3 and is observed in all of the evaluated factors, approximately. Given the available information, we found two possible explanations for this difference. The first is the degree of separation of concerns between the target system and the adaptation mechanism. DYNAMICO provides our implementation not only with a well defined structure of subsystems, each addressing different levels of dynamics, but also with the characterization of the interactions among them. Having clear these two aspects is fundamental to maintain separated the addressed concerns and functionalities of each subsystem. As a consequence, the visibility of the different

[9]The Rainbow adaptation effort data are based on [GCH+04].

feedback loops is maintained from architecture design to code, which favors the maintainability and reusability of our adaptation mechanism. The second is the chosen model for the system architecture, which is SCA in our DYNAMICO implementation, and ACME in Rainbow. Our implementation uses SCA interfaces as the communication means among all of the system components, which helps to maintain well defined limits among the subsystems. In any case, the obtained measurements show that the effort required to tailor our DYNAMICO implementation to adapt Znn.com is significantly lower than tailoring Rainbow for the same goal, and definitely lower than developing self-adaptive capabilities in Znn.com from scratch.

10.3.7 Maintaining System Quality Attributes

Znn.com is a small PHP monolithic application, executed in Apache. This fact has an unfortunate implication for the evaluation of the capability of any adaptation mechanism to maintain Znn.com's quality attributes. Indeed, the possibilities for adapting Znn.com/Apache are restricted to the strategy implemented by Rainbow, that is, by changing some parameters of the Apache configuration file and restarting it. Therefore, any adaptation mechanism required to maintain the same quality attributes maintained by Rainbow in Znn.com would necessarily use this strategy. As a consequence, and given that our DYNAMICO implementation adapts Znn.com using exactly the described strategy, it would obtain the same evaluation than Rainbow regarding this factor.

Nonetheless, by virtue of the improved context awareness obtained by the two extra feedback loops specified in DYNAMICO, CO-FL and M-FL, our adaptation mechanism is able to maintain quality attributes that depend on context information not sensed by the current monitoring infrastructure. Thus, the reliability of the decision-making process for the preservation of quality attributes is also improved.

10.4 Efficiency of the SmarterContext Reasoning Engine

This section presents the evaluation, in terms of efficiency, of the structural pattern-based version of our SmarterContext reasoning engine (CoRE). Reasoning about context information stored in personal context spheres (PCS) at runtime is a key requirement for improving user quality of experience (QoE) in user-centric situation-aware smart software (SASS) systems. Context representation in SmarterContext relies on semantic web ontologies and vocabularies. Therefore, context reasoning can be realized using semantic web engines that implement inference capabilities based on RDFS and OWL assertions. Context reasoning in the first version of our SmarterContext CoRE relies on RDFS and OWL-Lite description logics (cf. Section 5.2). For this implementation we used Pellet [SPG+07a], an open source general OWL reasoner,[10] as the reasoning engine, and Jena as the programming framework.

Despite their powerful expressiveness, the practical applicability of description logics and their reasoning engines is challenged by their performance when dealing with large data repositories [HKR09] such as PCSs. Consequently, to reason about dynamic context at runtime we need alternative approaches that exhibit a favorable trade-off between expressivity and scalability. Our context reasoning approach based on structural context patterns is one such alternative. Thus, our second version of the SmarterContext CoRE defines structural patterns in the form of contextual RDF subgraphs that allow us to replace the OWL-Lite properties used in the first version: `owl:TransitiveProperty`, `owl:SymmetricProperty`, and `owl:inverseOf`. These patterns are then used as templates to define context reasoning rules. Context predicates (i.e., RDFs properties and instances of them) involved in pattern-based reasoning rules are associated with indices implemented by SmarterContext to optimize the context reasoning process (cf. Section 9.2.3). That is, for each context model, each pattern-related predicate has an index that contains all the triples with this predicate. The construction of these indices constitutes the preprocessing stage of our context reasoning solution.

This section demonstrates the practical applicability of the optimized implementation of the SmarterContext CoRE. Our evaluation compares the performance of our optimized pattern-based implementation of the SmarterContext CoRE

[10] http://clarkparsia.com/pellet

with the performance of Pellet.

10.4.1 Worst Case Running Time Analysis

Even though there are several benchmarks of semantic web reasoning engines (e.g., [DCtTdK11, WLL+07]), they usually focus on experimental performance analyses rather than algorithmic analyses of their computational complexities. As a result, we found no explicit evidence of Pellet's time complexity analysis neither in these benchmarks nor in Pellet's seminal papers [SPG+07a, SPG+07b]. However, given that Pellet is a description logic reasoning engine, and description logic engines often have exponential running time [HKR09], we hypothesize that Pellet's worst case time complexity is exponential.

Computing the transitive closure of context predicates is a common and expensive context reasoning operation. For example, from the context fact "user Norha is located in Victoria" a retailer could be interested in knowing the other locations the user is located. These locations are inferred by calculating the transitive closure of predicate `gc:locatedIn` with `geo:Victoria` as the source node. This example is explained in more detail in Section 9.2.3 (cf. Figure 9.9). In a more involved case, a retailer may be interested in knowing all the products or service categories that a shopper could be interested in buying from the fact that she "is interested in" particular products. The inference of these preferences implies the computation of the transitive closure of predicate `rdf:subClassOf` for the products the user is interested in.

Contextual RDF graphs are digraphs. The worst case running time of the transitive closure algorithm on a digraph is polynomial of degree three [TR02]. Pellet reasons on RDF graphs. Therefore, the running time to compute the transitive closure of a contextual RDF graph using Pellet is $O(n^3)$. We improve this running time by partitioning context data using the predicate indices associated with context patterns. Because of this preprocessing, we can infer context facts faster than general reasoning engines. In particular, the running time to compute the transitive closure of a context predicate using our pattern-based approach (cf. Section 5.3) is $O(n^2)$.

Algorithm Analysis

According to the SmarterContext ontology, predicates `pwc:isInterestedIn`, `pwc:likes`, `shopping:toBuy`, and `shopping:wishes` are all generalizable for pred-

icate `rdfs:subClassOf`. Moreover, `rdfs:subClassOf` is a transitive predicate. Therefore, the SmarterContext CoRE can apply the generalization pattern together with the transitivity pattern to infer those products that can be recommended to the user from the preferences stored as explicit contextual RDF triples in her PCS. The application of these patterns for domain-specific predicates such as `pwc:isInterestedIn` is possible because of pattern-based rules defined by experts on the application domain.

Suppose a retailer wants to know all the products user Norha could potentially be interested in. The SmarterContext CoRE infers these products by applying patterns generalization and transitivity. Using the the generalization pattern we infer that if Norha is interested in a particular product category, she could potentially be interested in all the product categories that are super-classes of that product category. This is because according to the pattern definition, predicate `pwc:isInterestedIn` is generalizable for predicate `rdfs:subClassOf`. Figure 10.11 presents partially (a) the context model of the user with two triples whose objects (target nodes) correspond to product categories the user is interested in, and (b) the context model of the product and service categories vocabulary, which contains selected product categories of the ontology with their inheritance relationships. Predicate `rdfs:subClassOf` is a transitive predicate (cf. gray boxes in Figure 10.11(b)). To infer the product categories the user could potentially be interested in, the SmarterContext CoRE calculates the transitive closure of predicate `rdfs:subClassOf` for each product she is interested in (cf. gray nodes), according to the information in her context model (cf. Figure 10.11(a)).

Figure 10.12 depicts, partially, the index of the generalizable predicate `pwc:isInterestedIn` for Norha's context model. This index contains all the triples from the user's PCS that are related to this predicate. Each object (target node of the triple) corresponds to a resource that represents a product or service category that the user has marked as interesting along her web experience. We name this list `userProductList`. The size of this index is given by the number of triples in the user's PCS that are related to the corresponding predicate, `pwc:isInterestedIn` in this example. In the worst case, this size is equal to the number of RDF triples in the user's PCS. However, this is not a common case since a user's PCS contains several context dimensions besides product and service category interests (e.g., social connections, location context, or calendar events).

Figure 10.13 depicts, partially, the index of predicate `rdfs:subClassOf` for the

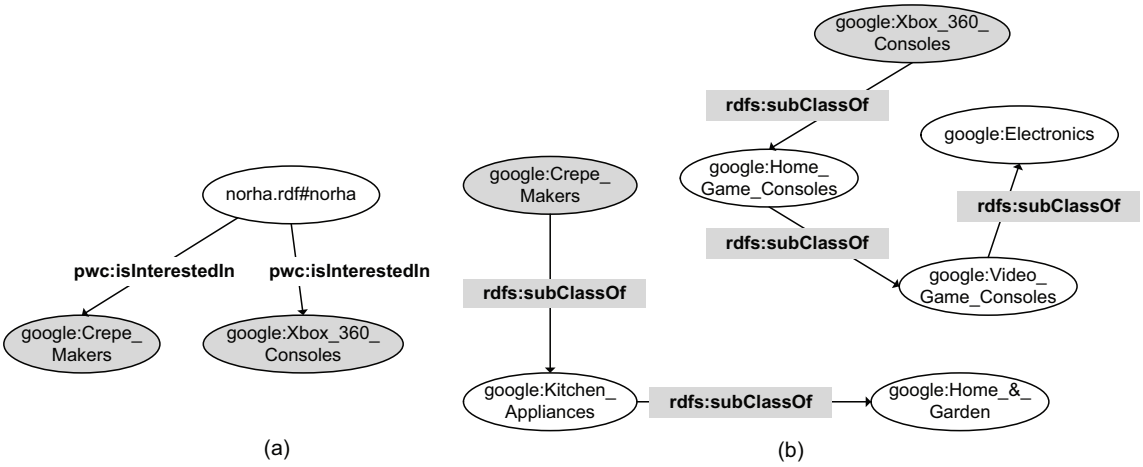


Figure 10.11: A partial graph-based view of (a) user Norha's PCS and (b) `google` vocabulary

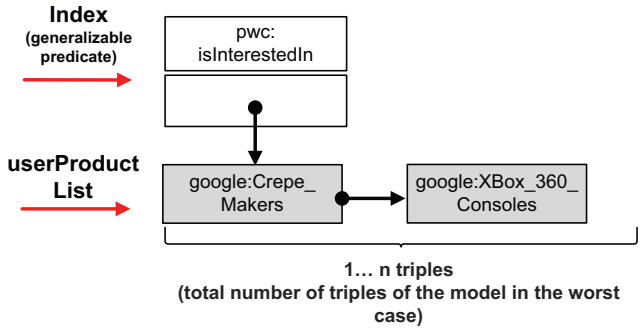


Figure 10.12: Indices of generalizable predicates for a user's PCS

product and service category vocabulary. The size of each index corresponds to the number of triples related to the corresponding predicate in the vocabulary of products. Elements of the index are stored in a hash data structure where each element defines two references: a reference to the element that corresponds to the object in the triple (i.e., each element may act as both subject and object—cf. the legend of the Figure 10.13), and a reference to the next element in the list. We call this list `vocabularyProductList`. In the worst case the size of this list is bounded by the total number of triples in the RDF model, the `google` vocabulary in this example. Our implementation of indices in SmarterContext, as explained in Section 9.2.3, consists of data structures that represents triples in the form of lists of subjects and objects. These lists contain objects that represent context entities (i.e., RDF resources) and implement several references, according to the pattern-based predicates they relate to. Each object is instantiated only once for the indices of a

same context model.

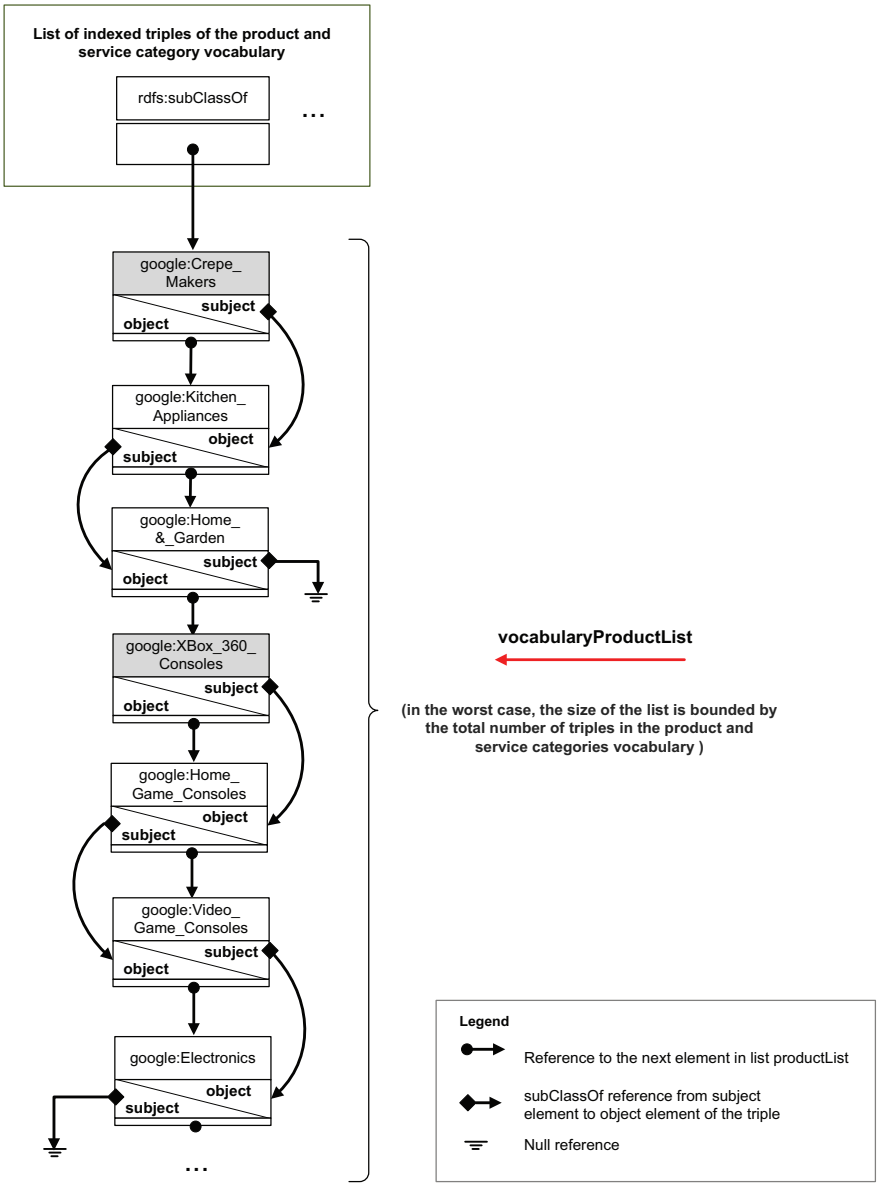


Figure 10.13: `rdfs:subClassOf` index for the products and service categories vocabulary. The highlighted elements of the list correspond to the product categories the user is interested in.

Algorithm 1 depicts the pseudo-code of the algorithm used by the SmarterContext CoRE to infer contextual data using the generalization and transitivity patterns. Note that the algorithm refers to the indices presented in Figures 10.12 and 10.13.

Algorithm 1 Inferring products the user could potentially be interested in

Input: $userProductList$
Input: $vocabularyProductList$
1: $productsOfInterestList \leftarrow \emptyset$
2: $pRef \leftarrow null$
3: $sRef \leftarrow null$
4: $productCategory \leftarrow null$
5: **for** each $productCategory \in userProductList$ **do**
6: // *cf. Figure 10.12*
7: $pRef \leftarrow vocabularyProductList.getProdCategory(productCategory)$
8: // *cf. Figure 10.13*
9: **if** $pRef \neq null$ **then**
10: **while** $pRef.getObject() \neq null$ **do**
11: $sRef \leftarrow pRef.getObject()$ // *to obtain the reference of the object that represents the next parent category*
12: $productsOfInterest.add(sRef)$
13: $pRef \leftarrow sRef$
14: **end while**
15: **end if**
16: **end for**
17: **return** $productsOfInterest$

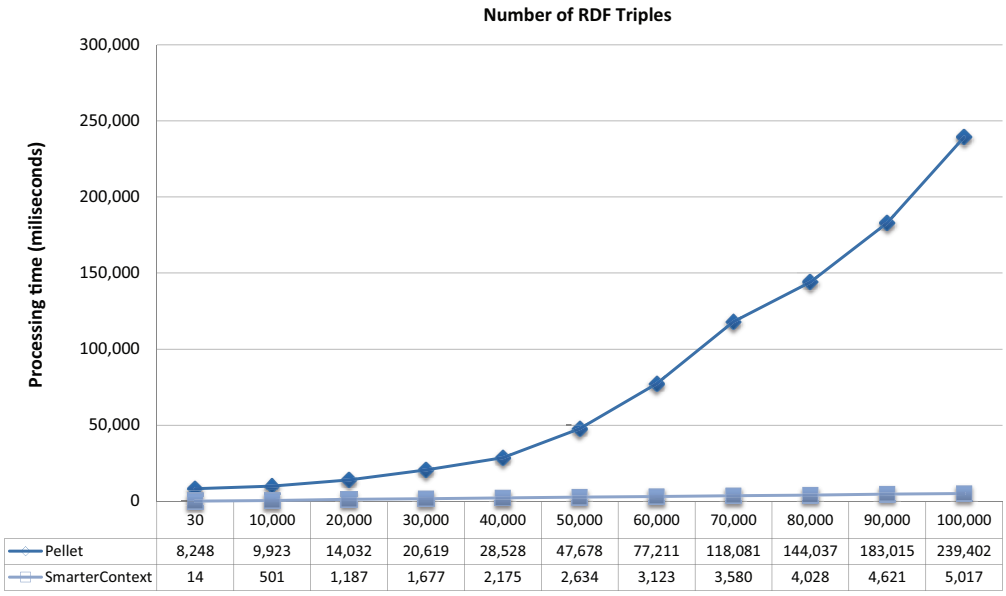
It is straightforward to see that the SmarterContext algorithm for the generalization pattern has a running time of $O(nm)$ where n is the number of indexed triples for the user's PCS and m is the number of indexed triples for the product and service categories vocabulary. Assuming $n \geq m$, the worst case running time of this algorithm is $O(n^2)$. The first `for` loop iterates over `userProductList` (cf. Figure 10.12). At each iteration, this loop looks for the corresponding product category in `vocabularyProductList` (cf. Figure 10.13). If the product is found in `vocabularyProductList`, the second `while` loop iterates over this list from the found product category until there is no more objects in its transitive path. This `while` loop performs three operations: a get operation to obtain the product to be recommended, the addition of this product to `productsOfInterestList`, and a change of reference to advance towards the next subject in the transitive path of `rdfs:subClassOf`.

The time complexity analysis of this algorithm is as follows. The initialization operations in lines 1–4 are executed in constant time $O(1)$. The running time of the `for` loop in line 5 is $O(n)$ according to the number of triples in the index of predicate `pwc:isInterestedIn` of the user's PCS. Each iteration of this loop performs a hash search over the vocabulary list that takes $O(1)$ time (cf. line 7), an evaluation of the returned product category that takes $O(1)$ time (cf. line 9), and performs an internal `while` loop starting at line 10 that iterates at most $O(n)$ times. The operations within the `while` loop (cf. lines 11–13) take constant time $O(1)$. From this analysis, the worst case running time of the SmarterContext CoRE algorithm for the application of the generalization pattern is $n^2 + 3n + 1$ which is $O(n^2)$. Therefore, based on the running time of SmarterContext, we can conclude that our pattern-based context reasoning algorithm is more efficient than general reasoning algorithms based on descriptive logic approaches such as Pellet.

Experimental Analysis

To analyze the efficiency of the SmarterContext CoRE we conducted several experiments to compare its processing time with respect to the processing time of the Pellet reasoning engine, which is a good representative of the description logic approaches. The results of these experiments corroborate our algorithmic analysis of the running time of SmarterContext.

In these experiments we performed the same reasoning operations with Pellet and SmarterContext while measuring the processing time. The size of the reasoning space of each tests is given by the number of triples in the user's PCS and in the vocabularies involved in the reasoning operation (e.g., `geo` and `google`). The size of a vocabulary is generally fixed or has low variation in terms of both the frequency of changes and the number of triples. On the contrary, the size of a PCS is highly variable. Figure 10.14 plots the processing times against processed RDF triples for SmarterContext and Pellet for the reasoning operation that returns all the product and service categories a user could be interested in. We varied the size of the reasoning space by adding 10,000 RDF triples each time. The graph shows that the growth rate for Pellet is considerably higher than for SmarterContext. We also measured the usage of memory and found no difference with respect to the space complexity of these approaches.



	30	10,000	20,000	30,000	40,000	50,000	60,000	70,000	80,000	90,000	100,000
Pellet	8,248	9,923	14,032	20,619	28,528	47,678	77,211	118,081	144,037	183,015	239,402
SmarterContext	14	501	1,187	1,677	2,175	2,634	3,123	3,580	4,028	4,621	5,017

Figure 10.14: Processing time of SmarterContext versus Pellet to obtain all the product and service categories user Norha could be interested in.

10.4.2 Improving Efficiency with Structural Context Patterns and Indices

The implementation of indices for preprocessing the predicates associated with structural context patterns improves the processing time of our SmarterContext CoRE considerably. While the size of the reasoning space for engines such as Pellet is affected by the size of the entire contextual RDF model, the size of the reasoning space for the SmarterContext CoRE is determined by the number of indexed triples only. Table 10.4 illustrates this aspect. The first column in this table corresponds to the number of triples in the user's PCS. The second and third columns correspond to the average processing time that it takes to calculate the transitive closure for predicate `gc:locatedIn` with Pellet and SmarterContext, respectively. The number of indexed triples that the SmarterContext CoRE processes is small with respect to the size of the entire context repository. Moreover, the number of indexed triples remains fixed in this case because (i) the user is located in one place only (i.e., there is only one and only one triple with the indexable predicate `gc:locatedIn` in the user's PCS, at a specific time), and (ii) the hierarchy of geographical locations in the `geo` vocabulary never changes.

Another important aspect for the efficiency of the SmarterContext CoRE is

Table 10.4: Average processing time of the transitive closure for predicate `gc:locatedIn`

Number of Triples	Pellet (miliseconds)	SmarterContext (miliseconds)
30	70	25
10,000	1,446	25
20,000	3,554	25
30,000	6,217	25
40,000	10,119	32
50,000	16,621	34
60,000	23,900	37
70,000	31,483	28
80,000	45,323	31
90,000	73,817	42
100,000	108,833	39

that with the usage of structural context patterns our reasoning approach eliminates the complexity added by OWL ontologies. Semantic web reasoning engines such as Pellet infer knowledge from both ABox and TBox statements [HKR09]. In description logics, ABox statements represent assertional knowledge about instances, whereas TBox statements correspond to terminological knowledge (i.e., schemas). To infer knowledge using OWL ontologies, semantic web reasoning engines reason on both types of statements ABox and TBox. On the contrary, our structural context patterns approach applies to ABox statements only. Of course, this implies less expressiveness for our solution. Nevertheless, we showed that the reasoning capabilities of our engine are expressive enough for realizing dynamic context monitoring effectively in user-centric SASS systems within the e-commerce domain.

SmarterContext indices must be created off-line when a large repository is indexed by the first time (e.g., when a new vocabulary is added to the system). However, given the dynamic nature of context, indices must be modified at runtime. In particular, the indices associated with PCSs. The addition of triples to an existing index is an operation that takes constant time. Whenever a new triple is inserted into the user's PCS and the predicate of this triple is associated with one or more context patterns, this process only involves finding the index associated with this predicate within the list of indices of the context model and inserting the subject (with the corresponding object) into the list of subjects of the indexed predicate.

Finally, since structural context patterns define the semantics of the Smarter-

CONTEXT CORE, the addition of new patterns will require the manual implementation of the corresponding pattern-based reasoning logic.

10.5 Chapter Summary

This chapter presented the validation results of several aspects related to the contributions presented in this dissertation. Using our two case studies, situation-aware smarter shopping and dynamic SOA governance, we analyzed and evaluated our contributions from the perspective of their potential to enable new business models in e-commerce domains, their feasibility to be implemented in both user-centric and business-centric SASS systems, their effectiveness in improving user QoE and self-adaptivity in dynamic software systems, and their scalability to work effectively in large-scale environments.

The validation of research contributions such as the ones presented in this dissertation is an expensive and challenging process. First, the implementation of meaningful experiments is limited by the lack of real contextual data. SMARTERCONTEXT supports several context dimensions, however it is difficult to find available data sets that allow us to simulate all these dimensions. Using the Yelp data set we were able to test the impact of exploiting context about user locations and preferences. Second, there is a lack of effective simulators applicable to our research. Therefore, the validation process in this dissertation required a high investment of resources to implement not only the software artifacts directly related to our contributions, but also the target systems that we used in our demos and experiments. Besides the SMARTERCONTEXT monitoring infrastructure, its privacy module SURPRISE, and the SMARTERCONTEXT reasoning engine, we implemented (i) several e-commerce applications compliant with SMARTERCONTEXT, (ii) SMARTERDEALS and its recommendation engine, and (iii) the service-oriented system used as target system in the dynamic SOA governance case study. Finally, the exhaustive validation of frameworks and reference models requires their application to many different systems in different domains. This process may take several years and requires the participation of a broad community of researchers and practitioners.

Chapter 11

Summary and Conclusions

This chapter summarizes this dissertation by revisiting our main research challenges, addressed goals, and contributions. It also presents some limitations that we have identified for the application of SmarterContext in industrial settings. Finally, this chapter concludes this dissertation with a discussion of future work.

11.1 Dissertation Summary

Everyday activities of humans are increasingly depending on software technologies. This increasing dependency poses enormous challenges for software engineers to design, implement, deliver and maintain software applications that can support effectively the interactions of dynamic groups of users, stakeholders, organizations, and hardware and software infrastructures that coexist within highly variable environments. These software systems are exposed to continuous changes in their requirements and are affected by environmental situations. In cases where these environmental situations are highly dynamic and uncertain, these systems must adapt themselves at runtime to satisfy their context-dependent requirements. We call these systems situation-aware smart software (SASS) systems—the research subject of this dissertation.

The fundamental motivation behind our research concerns the improvement of (i) situation-awareness and user quality of experience (QoE), and (ii) the dynamic capabilities of SASS systems. We categorized these systems as user-centric, such as those in our situation-aware smarter shopping case study, or business-centric such as those in our dynamic SOA governance case study. The research problem addressed

in this dissertation was the dynamic management of context information to improve the relevance of SASS systems' context-aware capabilities with respect to changes in their requirements and execution environment.

11.1.1 Addressed Challenges and Goals

We classified our research challenges and thus our contributions into two groups: *context-awareness* (or dynamic context management) and *self-adaptivity*. The challenges that we addressed in this dissertation are summarized as follows:

Dynamic Context Management (context-awareness)

CH1. The impossibility of fully specifying context entities and their monitoring requirements at design time.

CH2. Context monitoring requirements change continuously. Thus, context monitoring strategies must support the addition and deletion of context types and monitoring conditions at runtime—having the user in the context management loop.

Self-Adaptivity

CH3. The engineering of software systems with explicit control of the dynamicity of adaptation goals, adaptation mechanisms, and monitoring infrastructures, and the way they affect each other in the adaptation process. This is to preserve the effectiveness of context monitoring requirements and thus of self-adaptation.

CH4. The assessment of adaptation mechanisms at runtime to determine whether an undesirable system state could be reached as a result of a faulty adaptation process.

To address these challenges, we accomplished the following goals:

G1. To propose a context representation mechanism to model dynamic context information and corresponding monitoring requirements in any application domain.

G2. To propose an adaptive context management solution that, assisted by the user, supports context gathering, reasoning, provisioning and disposal under changing context monitoring requirements.

G3. To characterize adaptation properties and system goals for guiding the specification of context monitoring requirements in quality-driven self-adaptive software systems.

G4. To propose a reference model for engineering adaptive software that helps improve self-adaptivity, by controlling the dynamicity of adaptation goals, adaptation mechanisms and monitoring mechanisms, and the way they affect each other in the adaptation process.

G5. To investigate the application of dynamic context monitoring to the runtime V&V of self-adaptive software.

G6. To implement a proof-of-concept of the adaptive context management solution for both case studies: situation-aware smarter shopping, and dynamic SOA governance.

11.1.2 Contributions

This subsection summarizes our contributions. Contributions *C1–C3* are related to dynamic context management, and address the improvement of user QoE and situation-awareness in user-centric SASS systems. Contributions *C4–C6* are related to self-adaptivity, and address the improvement of the dynamic capabilities of self-adaptive systems. Contribution *C7* addresses both dynamic context management and self-adaptivity.

C1: The Personal Context Sphere

To contribute user-centric and user-controlled contextual models that are useful to improve situation awareness and user QoE we proposed the *Personal Context Sphere (PCS)*. In abstract terms, a PCS is a context model that contains context information about web entities relevant to the user's tasks in a particular application domain. In concrete terms, a PCS is a distributed repository provided by a third party (e.g., a cloud infrastructure provider) that stores both the context information of a user gathered along the user's entire web experience, and the context consumers and providers with which the user shares this information. We conclude that our PCS approach is the first approach that provides a user-centric model to allow user-driven integration while observing the principles of the smart internet and the personal web. To the best of our knowledge, our PCS model and the SmarterContext infrastructure provide a unique solution that demonstrates a comprehensive realization of the smart

internet principles (i.e., a user-centric model for instinctive interactions, sessions for users and their mocs, and collaborative and collective web interactions), and the personal web conceptual principles (i.e., the user as the center and controller of web integration, web integration based on the semantics of the user's context, the web working on behalf of the user to manage personal context, and the exploitation of context information inferred from social interactions among PCSs).

This contribution addresses goal *G1* because it provides an effective abstraction to model personal context information and its management requirements in user-centric SASS systems. It also addresses goal *G2* because PCSs constitute the fundamental concept that enables SmarterContext to assist users in the management of the life cycle of their personal context information.

C2: The SmarterContext Ontology

SASS systems require context models to represent the relevant aspects of entities that affect the interactions between users and systems, as well as the relationships between users and these entities. Ontologies are useful to describe concepts and the relationships among them. Therefore, ontology-based models are natural mechanisms to represent context information since context is a specific kind of knowledge. The SmarterContext ontology is a suitable mechanism for context representation and reasoning in SASS systems. It provides the mechanisms for the formal specification of the semantics of contextual data from both user-centric and business-centric perspectives. Furthermore, an important modeling feature for realizing user-centric interactions and services in SASS systems is knowledge sharing. Our SmarterContext ontology not only allows the implementation of runtime context models required to manage context dynamically, but also the interchange of context information among heterogeneous and distributed sources and consumers.

Our SmarterContext ontology addresses goals *G1* and *G2* because it provides an effective mechanism to model dynamic context, its monitoring requirements, and monitoring strategies. We demonstrated the general applicability of our ontology through the two case studies conducted in this dissertation.

C3: The SmarterContext Reasoning Engine

To address goal *G2*, in particular context reasoning, we proposed a context reasoning engine based on the SmarterContext ontology. Our approach to context reason-

ing supports the inference of implicit contextual facts from explicit contextual data specified in context models compliant with the SmarterContext ontology. We proposed an alternative approach to OWL ontologies to infer contextual facts that otherwise could not be inferred using RDFS only. This was to eliminate the unnecessary complexity added by the extra expressiveness provided by description logics. We demonstrated the practical applicability of the SmarterContext reasoning engine when dealing with large-scale context repositories, and its effectiveness to improve the accuracy of recommender systems.

C4: A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems

To advance towards the systematic assessment of self-adaptive solutions, we proposed a framework that defines a set of characterization dimensions and adaptation properties applicable to the evaluation of self-adaptation mechanisms. We derived these characterization dimensions from our foundational reference model: the feedback loop from control theory. In this framework we classified adaptation properties into two groups: those concerning the desired properties of the target system, and those concerning the desired properties of the adaptation mechanism. We derived the first group of properties from quality attributes, and the second group from the SASO properties of control theory (i.e. *stability, accuracy, short settling time* and *small overshoot*). We re-interpreted these control theory properties to make them applicable to software-systems. Our framework also establishes a mapping between adaptation properties and quality attributes. Therefore quality attributes and their corresponding metrics provide the means to assess self-adaptive software systems and thus to derive context monitoring requirements.

Our evaluation framework contributed to address goal *G3* because from adaptation properties and goals SmarterContext can derive context monitoring requirements. It also contributed to address goal *G5* because it provides a useful starting point to identify assurance criteria required to validate and verify self-adaptive systems at runtime.

C5: DYNAMICO—A Reference Model for Designing Self-Adaptive Systems

Most implemented approaches to self-adaptation assume adaptation goals and monitoring infrastructures as non-mutable, thus constraining their applicability to systems whose context awareness is restricted to static monitors. However, for many systems it is not acceptable to discard unexpected context changes and dynamic changes in adaptation goals and user requirements, such as SLA re-negotiation at runtime. Therefore, separation of concerns, dynamic monitoring, and runtime requirements variability are critical for satisfying system goals under highly changing environments. Goal *G4* of this dissertation concerned the design of self-adaptive systems that are compliant with these critical features. To contribute towards the fulfilment of this goal, we proposed our Dynamico reference model. Dynamico provides a guide to design and implement SASS systems that must deal with and effectively respond to highly dynamic contexts of execution, by evaluating their own behaviour at runtime and reconfiguring themselves whenever the satisfaction of system requirements can be compromised. Dynamico helps cope with the dynamics inherent in self-adaptivity by defining three types of feedback loops that correspond to the three levels of dynamics that we have identified for self-adaptation: (i) the control objectives feedback loop, for managing changes in adaptation goals and user requirements; (ii) the target system adaptation feedback loop, to deal with changes addressable directly at the target system level; and (iii) the dynamic monitoring feedback loop, to manage changes that require the deployment of different or additional monitoring infrastructures to those already configured for execution, thus maintaining its relevance with respect to the changing adaptation goals. The main contribution of our reference model is the separation of concerns required to deal with the three levels of dynamics in self-adaptation. As a reference model, Dynamico reconciles the many visions and contributions of different approaches for the development of self-adaptive software systems, whether they hide or exhibit the elements of feedback control loops. Nonetheless, our reference model emphasizes the visibility of these control elements and constitutes a guide to design self-adaptive systems in which the system goals, the target system itself, or the monitoring infrastructure must be adapted—assuming this is a crucial requirement for the system to be developed.

We designed and implemented SmarterContext using Dynamico as the reference model. With this and with our case study on dynamic SOA governance, we

demonstrated the applicability of DYNAMICO.

C6: A Model for Explicit Runtime V&V Tasks in Self-Adaptive Systems

We proposed a reference model to make V&V tasks explicit in the adaptation process, and discussed key challenges for the development of certifiable runtime V&V methods that can certify adaptation mechanisms in the achievement of their adaptation goals. Certifiable V&V methods and tools are critical for the success of autonomous, autonomic, smart, self-adaptive and self-managing systems. To the best of our knowledge, our reference model is the first proposal to address the integration of runtime V&V methods as concrete tasks to be performed by elements of the adaptation process. With this reference model, we are pioneers in the SEAMS and Models@Run.Time research communities on the design of self-adaptive systems that include runtime V&V instrumentation as an inherent feature of the adaptation process. Our proposal for making V&V tasks explicit in the adaptation loop provides solid starting points for V&V researchers from other communities to deploy different techniques and methods for improving the trustworthiness of self-adaptive and self-managing systems.

This contribution addresses goal *G5* because it allows us to understand the crucial role that dynamic context monitoring plays in the realization of runtime V&V for self-adaptive software systems.

C7: Implementation of the SmarterContext Infrastructure

The last contribution of this dissertation is the implementation of the general architecture of our SMARTERCONTEXT solution and its reasoning engine, and their realization for our case studies on situation-aware smarter shopping and dynamic SOA governance. This contribution not only addresses goal *G6* but also most of the other goals of this dissertation. This contribution allows us to demonstrate the practical applicability of our DYNAMICO reference model and our solution to dynamic context management.

11.2 Limitations

This section discusses aspects that may constitute limitations on the application of SMARTERCONTEXT to an industrial setting in the short-term. Most these aspects

correspond to challenges inherited by SmarterContext from the research communities related to its underlying technologies. Therefore, these aspects are not only limitations but also sources of interesting collaboration opportunities for our research.

Semantic Interoperability of Context Information

Semantic interoperability is a requirement to manage the context information life cycle with SmarterContext. The application of SmarterContext to user-centric SASS applications relies on the transmission of contextual data with unambiguous and shared meaning between PCSs and web and mobile applications, as well as web services that act as context consumers and/or providers. For this, these third party applications must be compliant with SmarterContext in the sense that they must be enabled to (i) exchange RDF messages with the SmarterContext infrastructure, (ii) process these messages to gather meaningful context provided by SmarterContext, and (iii) provide SmarterContext with contextual data from the interactions of users with them. To realize this interoperability, it is necessary that these third parties "speak the same language" than SmarterContext. This is, these applications and SmarterContext must use the same ontologies and vocabularies to understand, unambiguously, the semantics of context information.

The semantic interoperability of context information may constitute a limitation because for its realization it is necessary that all the third parties that interact with SmarterContext use the same vocabularies to categorize context entities. In SmarterContext we can classify semantic interoperability as high-level and low-level interoperability. High-level interoperability refers to the foundational ontology of SmarterContext (i.e., the *gc* ontology) and its domain-specific extensions (e.g., the *pwc* ontology for the personal web and the *shopping* ontology for e-commerce). High-level interoperability can be achieved through direct negotiation between a specific third party and SmarterContext. The real problem is low level interoperability since it involves the multiple and diverse vocabularies used to represent entities that constitute contextual data across different web applications. For example, different retailers may use different vocabularies with variable concepts to characterize product and service categories. This is known as the semantic heterogeneity problem faced by computer systems. In the semantic web this problem is known as *ontology matching* [ES07] and is being widely investigated by semantic web researchers.

Middleware Support for Architectural Adaptation

SmarterContext supports two types of self-adaptation: behavioral (i.e., by changing the thresholds of variables in monitoring conditions, by changing the reasoning rules of the CoRE, or by injecting new monitoring logic into context monitoring components), and architectural (i.e., by adding or removing sensors, as well as context gathering and monitoring components). SmarterContext is a service-oriented system that follows the service component architecture (SCA) specification [OSOA07]. SCA provides a model for composing applications based on service-oriented architecture principles. To realize architectural self-adaptation SmarterContext uses the QoS-CARE/FraSCAti framework [Tam12]. FraSCAti is an open-source implementation of the SCA standard [SMF+09, SMR+12]. QoS-CARE [Tam12] is an SCA layer for dynamic reconfiguration that runs on top of FraSCAti. Besides FrasSCAti, there are other SCA implementations such as Apache Tuscany,[1] Fabric3,[2] and IBM WebSphere [DRS+07]. As part of the research conducted for this dissertation, we investigated the implementation of SmarterContext using IBM Websphere technologies. Despite the wide applicability of IBM Websphere in industry, it currently provides no support for the dynamic deployment of new SCA components as required by SmarterContext. Thus, on the one hand the industrial adoption of the SCA standard is still limited. On the other hand, there exists no SCA middlewares for self-adaptation that are mature enough to be applicable in industry. QoS-CARE and FraSCAti may require several more years to reach this level of maturity. Therefore, the full exploitation of the dynamic capabilities of SmarterContext depends on the evolution of these self-adaptation technologies. However, SmarterContext can exploit behavioral adaptation without requiring these emerging technologies by changing either reasoning rules or monitoring logic dynamically as discussed in Section 10.2.2.

The Expressiveness versus Performance Trade-off

SmarterContext supports both OWL-based and structural pattern-based reasoning. The OWL version of the SmarterContext CoRE exploits all the expressiveness provided by description logics. It is highly dynamic in the sense that its application to different domains implies only the extension of its foundational ontolo-

[1] http://tuscany.apache.org
[2] http://www.fabric3.org

gies and context reasoning rules. As a result, no manual coding is required to apply SmarterContext in a new domain. The problem with this version is its scalability and performance which limit the applicability of the reasoning engine at runtime when dealing with large-scale context repositories. In contrast, the pattern-based version of the SmarterContext CoRE has excellent performance but is less expressive than the OWL version. Even though we applied this version to many representative reasoning scenarios of our situation-aware smarter commerce case study, we still need to validate its expressiveness for other domains and for e-commerce in an industrial setting. Moreover, the application of the pattern-based version in new domains may require the definition of new context-patterns, and the manual implementation of the corresponding reasoning logic.

Privacy and Confidentiality

We had many opportunities of getting valuable feedback from potential users of SmarterContext at different venues. A major concern from the perspective of users is privacy and confidentiality. Users claim to be worried about providing their sensitive contextual data to a third party in charge of maintaining their PCSs. However, what most users have not realized yet is that they already have not only shared their personal contextual data with several web applications, but transferred the control of their information to the owners of these applications. The vision of SmarterContext is different. In SmarterContext users are the owners and controllers of their personal context. SmarterContext allows users to define policies to control both the third parties authorized to exchange personal context with SmarterContext and the types of contextual data than can be shared. Not even the cloud provider that hosts PCSs would have access to personal contextual data since encryption is also controlled by users. Moreover, users can modify their privacy and security policies at any time. The privacy angst seems to arise from users being scared about controlling their personal contextual data or being comfortable with the existing models. This requires a change of perspective by users for realizing the vision of SmarterContext in the e-commerce realm.

Barriers from Existing E-Commerce Business Models

We also had opportunities to introduce the SmarterContext vision to retailers. Even though in all cases they recognized the potential of SmarterContext to

improve value and revenue, in most cases they were not entirely convinced that sharing information about their own customers with other retailers is a good idea. In our opinion, this is the greatest barrier against the adoption of SmarterContext in the e-commerce industry. Because the information of users belongs mainly to the most powerful retailers, the ones that have the biggest participation in the market. To overcome this limitation it is necessary to change the perspective of these companies as well as a redefinition of their business models. For this, they must acknowledge the shopper as the real center of the business model. Indeed, with the proliferation of mobile, ubiquitous and social computing customers have instant access to an enormous amount of relevant information about products, offers, prices and companies. This has radically changed the way people make purchasing decisions. SmarterContext proposes a win win model where all the actors benefit from the responsible exploitation of personal context information. Retailers compliant with SmarterContext will not only share their knowledge about customers with other retailers, but also obtain valuable information from their customer's entire web experience. The real challenge for businesses will be now to exploit this information to improve their value proposition.

11.3 Future Work

This dissertation concludes with a presentation of selected future work opportunities derived from our research.

Control Science: Foundational Science for Runtime V&V Methods

Control science can be defined as a systematic way to study certifiable V&V methods and tools to allow humans to trust decisions made by self-adaptive systems. In a 2010 report, Dahm identified control science as a top priority for the US Air Force (USAF) science and technology research agenda for the next 20 years [Dah10]. Certifiable V&V methods and tools are critical for the success of autonomous, autonomic, smart, self-adaptive and self-managing systems. One systematic approach to control science for adaptive systems is to study V&V methods for the mechanisms that sense the dynamic environmental conditions and the target system behavior, and act in response to these conditions by answering the questions *what*, *when* and *how* to adapt. We identify the management of viability zones as an important component of control

science. We define the viability zone of a SASS system as the set of possible system states in which the system operation is not compromised. Viability zones can be characterized in terms of relevant context attributes and corresponding desired values. Therefore, they change according to changes in context situations. A particular SASS system may have more than one associated viability zone, which can be added, replaced or adjusted by adding or removing variables of interest at runtime. Changes in viability zones affect the self-adaptive system at its three levels of dynamics. Therefore, managing viability zones at runtime is a crucial requirement for the assurance of self-adaptive systems. We are interested in investigating mathematical models that can be applied together with dynamic context management, and software engineering techniques to develop control science towards certifiable trust in self-adaptation.

Runtime Models for the Assurance of Self-Adaptive Systems

Runtime models have been recognized as important enablers for the assurance of self-adaptive systems. We identified three subsystems that are key in the design of effective context-driven self-adaptation: the control objectives manager, the adaptation controller, and the context monitoring system. These subsystems represent three levels of dynamics in self-adaptation that can be controlled through three feedback loops, i.e., the control objectives, the adaptation, and the monitoring feedback loops, respectively. We argue that runtime models provide abstractions that are crucial to support the feedback loops that control these three levels of dynamics. From this perspective, models at runtime could be developed specifically for each level of dynamics to support the control objectives manager, adaptation controller, and the monitoring system. At the control objectives level, models at runtime represent requirements specifications subject to assurance in the form of functional and non-functional requirements. At the adaptation level, models at runtime represent states of the managed system, adaptation plans and their relationships with the assurance specifications. At the monitoring level, models at runtime represent context entities, monitoring requirements, as well as monitoring strategies and their relationships with assurance criteria and adaptation models. Most importantly, runtime models at these levels must have efficient and effective methods of interaction between them because changes in requirement specifications may trigger changes at both the adaptation and the monitoring levels, and the associated runtime models. Similarly, changes in adaptation models may imply changes in monitoring strategies or context entity

models. In any case, runtime models at the adaptation and monitoring levels must maintain an explicit mapping to the models defined at the control objectives level which specify the requirements. We are interested in investigating runtime modeling techniques suitable for the assurance of SASS systems at the three levels of dynamics of self-adaptation.

Personalized Web Tasking using SASS Systems

The vision of the smart internet concerns the engineering of smart systems and smart interactions that can assist users in accomplishing daily life concerns effectively and efficiently. For this, the engineering of smart internet applications requires frameworks, models, and techniques to personalize web-tasking. We are interested in applying our experience with SmarterContext to the investigation of innovative SASS systems that allow the personalization of web-tasking in the smart internet. For this, an important requirement is to provide effective mechanisms to automate repetitive and ordinary tasks while hiding the complexity of these mechanisms from the user. Hence, personalized web-tasking with situation-aware smart applications concerns the design and implementation of situation-aware self-adaptive software applications that understand the user situation and their execution environment, to deliver functionalities that assist people in performing tasks with a minimum effort while maximizing user satisfaction. We are interested in investigating the development of a software-engineering frameworks to guide the design and implementation of situation-aware smart software systems for personalized web-tasking, including user-driven task automation, in the smart internet.

SmarterContext: From Research to Real Applications

We will continue investigating the development of middlewares for self-adaptation through the integration of SmarterContext with the QoS-CARE/FraSCAti framework. With this, our goal is to provide researchers and practitioners with an implementation that supports the validation of self-adaptive software, hopefully in industrial settings. We also want to investigate the application of SmarterContext to other domains such as health-care and recommender systems. Given the user-centric nature of our approach, we argue that the personal context sphere is a natural solution for the electronic health record problem. Moreover, dynamic context management can benefit recommender systems tremendously. E-commerce as well as health-care

provide fabulous opportunities to exploit context data in the improvement of the accuracy of user-centric recommendations. Last but not least, we want to continue the development of SmarterContext to implement it as part of a real e-commerce solution. For this, we are conducting a feasibility study to create a startup company.

Glossary

Adaptation feedback loop (A-FL)

The second level of dynamics in the DYNAMICO reference model. A software controller that implements an adaptation mechanism that can expose self-adaptive behavior.

Adaptation property

A desirable characteristic of an adaptation approach or mechanism. Adaptation properties are usually characterized in terms of quality attributes and SASO properties.

Context information

Any information that is useful to characterize the state of individual environmental entities and the relationships among them.

Context-awareness

Same as situation-awareness. The capability of a system to gather and process information from its environment to understand the situation of external and internal entities that can affect the accomplishment of its goals. [ADB+99].

Contextual RDF graph

An Resource Description Framework (RDF) graph whose nodes and arcs correspond to either types derived from the SMARTERCONTEXT ontology or values defined in vocabularies compatible with this ontology.

Contextual RDF triple

An RDF triple that represents a context fact or context observation.

Control action

The means to affect (adapt) the target system to obtain the desired effect.

Control objectives feedback loop (CO-FL)

The first level of dynamics in the DYNAMICO reference model. A control-based software mechanism that governs changes in system goals at runtime.

Dynamic context information

Same as dynamic context. Refers to any piece of environmental information subject to unforeseeable changes that occur while the context-aware system executes.

Dynamic monitoring feedback loop (M-FL)

The third level of dynamics in the DYNAMICO reference model. A control-based software mechanism that governs changes in context monitoring requirements at runtime to preserve context-awareness in adaptation processes.

Feedback loop

Same as closed loop. Refers to a foundational model in control theory that is used to automate the control of dynamic systems. Feedback loops support the management of uncertainty by adjusting the behavior of the target system with the goal of counteracting disturbances from the environment [HDPT04].

Linked data (LD)

A common semantic web framework based on RDF for sharing data across the web [BLHH+06].

MAPE-K loop

The feedback loop implemented by the *autonomic element*, which was proposed by the autonomic computing initiative. The goal of this feedback loop is to support the management of computational resources by *monitoring* and *analyzing* their environment and operation to *plan* and *execute* maintenance actions automatically. The information flow among the components of the loop is supported by a *knowledge base* [KC03].

Ontology

In computer science, an ontology is a description of knowledge in a domain of interest. Ontologies are machine-readable specifications with formally defined semantics [HKR09].

Ontology class

Represents an entity type defined in the vocabulary of a knowledge domain [HKR09]. In the SmarterContext ontology classes correspond to context types.

Ontology individual

Individuals are instances of ontology classes [HKR09]. In SmarterContext an example of a context instance or individual is the favorite location of a user.

Ontology property

Same as predicate and context predicate in vocabularies based on the SmarterContext ontology. Represents either a relationship between two entities or an attribute that characterizes an entity (a link between an entity and a data value, e.g., the age of a user) [HKR09].

OWL-Lite

The less expressive language of the semantic web OWL family of languages which are used to support knowledge representation and reasoning in web environments [W3C04d].

Personal context sphere (PCS)

A context model that allows the specification of context information that is relevant to a user as well as privacy policies. PCSs also support the integration of third party applications that act as context consumers and providers of personal context data.

Personal web (PW)

An implementation of the smart internet that focuses on the user as the center of web integration [Ng10].

RDF triple

A string representation of an arc and is defined as a set of three sequences of characters that identify three RDF resources: the label of the source vertex known as the subject, the label of the arc known as the predicate, and the label of the target vertex known as the object [MM04].

Relevant context entity

An environmental object that can affect the operation of the system. A context situation refers to the information that describes the states of a set of relevant context entities at a particular point in time

Resource description framework (RDF)

A general purpose language for representing information in the web [MM04].

Self-adaptivity

The capability of a system to modify itself, at runtime, according to environmental changes that can affect its expected behavior. [CLG+09].

Semantic web (SW)

An extension of the web that enables systems to smartly search, combine, and process web data based on the meaning that these data have to humans [HKR09].

Situation-aware smarter software (SASS)

A software system able to understand its environment to adapt itself with the goal of addressing changing requirements and context situations.

Smart internet

A new generation of the internet where web entities, represented by online services and content, are discovered, aggregated and delivered dynamically, automatically, and interactively according to users' needs and situations [CCNY10].

Smarter commerce

An IBM strategic initiative that places the customer in the center of the commerce business processes. Smarter commerce provides compelling applications for the research conducted in this dissertation [IBM11b].

Socio-technical ecosystem

A set of dynamic and interdependent communities of users, organizations, and computing infrastructures that coexist in complex and changing environments [NFG+06].

Target system

Same as managed system, the system intended to be self-adaptive.

Uncertainty

Refers to unforeseeable changes in environmental conditions and requirements that affect the system operation.

Viability zone

The viability zone of a SASS system corresponds to the set of system states in which the operation of the system is guaranteed.

Acronyms

A-FL Adaptation Feedback Loop

ACF Adjusted Collaborative Filtering

ACRA Autonomic Computing Reference Architecture

CF Collaborative Filtering

CMS Context Monitoring Strategy Module of the SmarterContext ontology

CO-FL Control Objectives Feedback Loop

COb Control Objectives module of the SmarterContext ontology

CoRE Context Reasoning Engine

DYNAMICO DYNamic Adaptation, MonItoring and Control Objectives model

FODA Feature-Oriented Domain Analysis

GC General Context module of the SmarterContext ontology

IoT Internet of Things

LD Linked Data

M-FL Monitoring Feedback Loop

MIAC Model Identification Adaptive Control

MOC Matter of Concern

MRAC Model Reference Adaptive Control

OWL Web Ontology Language

PCC Pearson Correlation Coefficient

PCS Personal Context Sphere

PW Personal Web

PWC Personal Web Context module of the SmarterContext

QoE Quality of Experience

QoS Quality of Service

RDF Resource Description Framework

RDFS Resource Description Framework Schema

RMSE Root Mean Squared Error

SASO Control properties: Stability, Accuracy, Settling time, Overshoot

SASS Situation-Aware Smarter Software

SCA Service Component Architecture

SEAMS Software Engineering for Adaptive and Self-Managing Systems

SLA Service-level Agreement

SOA Service-oriented application/architecture

SW Semantic Web

References[3]

[AAH09] T. Anagnostopoulos, C. Anagnostopoulos, and S. Hadjiefthymiades. An Online Adaptive Model for Location Prediction. In *Proceedings 3rd International ICST Conference Autonomic Computing and Communications Systems (Autonomics 2009)*, pages 64–78, 2009. 313

[ABSP11] J. Aubin, A. Bayen, and P. Saint-Pierre. *Viability Theory: New Directions*. Springer, Heidelberg, 2011. 181

[ACC09] Z. Abid, S. Chabridon, and D. Conan. A Framework for Quality of Context Management. In *Proceedings 1st International Conference on Quality of Context (QuaCon 2009)*, volume 5786 of *LNCS*, pages 120–131. Springer, 2009. 313

[ADB+99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings 1st International Symposium on Handheld and Ubiquitous Computing (HUC 1999)*, volume 1707 of *LNCS*, pages 304–307. Springer, 1999. 9, 37, 41, 280, 312

[AFF+01] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger. Océano - SLA Based Management of a Computing Utility. In *Proceedings 7th IFIP/IEEE International Symposium on Integrated Network Management*, pages 855–868, 2001. 62, 63, 148, 149, 150, 151, 153, 155, 319

[AT05] G. Adomavicius and A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible

[3]The numbers at the end of each bibliography item are the backward references to the pages where it was cited.

Extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005. 234

[AYG10] A. Achilleos, K. Yanga, and N. Georgalas. Context Modelling and a Context-Aware Framework for Pervasive Service Creation: A Model-Driven Approach. In *Proceedings 8th IEEE International Conference on Pervasive and Mobile Computing (PerCom 2010)*, pages 281–296. IEEE Computer Society, 2010. 313

[BBF+10] N. Bencomo, G. Blair, R. France, F. Munoz, and C. Jeanneret. Proceedings 4th international workshop on models@run.time. In *4th International Workshop on Models@run.time (MODELS 2009)*, volume 6002 of *LNCS*, pages 173–188. Springer, Heidelberg, 2010. 191

[BBH+09] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A Survey of Context Modelling and Reasoning Techniques. *Pervasive and Mobile Computing*, 6:161–180, 2009. 42, 44, 45, 47

[BCK03] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, Reading, Mass, 2nd edition, 2003. 159, 165

[BDM+11] S. Balasubramanian, R. Desmarais, H. A. Müller, U. Stege, and S. Venkatesh. Characterizing Problems for Realizing Policies in Self-Adaptive and Self-Managing Systems. In *Proceedings 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011)*, pages 70–79, New York, NY, USA, 2011. ACM. 182

[BG11] L. Baresi and S. Guinea. Self-Supervising BPEL Processes. *IEEE Transactions on Software Engineering*, 37(2):247–263, 2011. 63, 148, 152, 155, 319

[BHBL09] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data — The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009. 32, 81

[BHK98] J. S. Breese, D. Heckerman, and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings 14th con-*

ference on Uncertainty in Artificial Intelligence (UAI 1998), pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. 323, 324

[BKLW95] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock. Quality Attributes. Technical report CMU/SEI-95-TR-021, Software Engineering Institute, 1995. 141, 143, 146, 148, 150, 154

[BL06] T. Berners-Lee. Linked Data. http://www.w3.org/DesignIssues/LinkedData.html, 2006. 32, 52, 53, 82

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. http://www.ietf.org/rfc/rfc3986.txt, 2005. 52, 104

[BLHH+06] T. Berners-Lee, W. Hall, J. A. Hendler, K. O'Hara, N. Shadbolt, and D. J. Weitzner. A Framework for Web Science. *Foundations and Trends in Web Science*, 1(1):1–130, 2006. 49, 51, 281

[BLM08] P. Bianco, G. Lewis, and P. Merson. Service Level Agreements in Service-Oriented Architecture Environments. Technical Report CMU/SEI-2008-TN-021, Carnegie Mellon University Software Engineering Institute, 2008. 246

[BSG+09] Y. Brun, G. D. M. Serugendo, C. Gacek, H. M. Giese, H. Kienle, M. Litoiu, H. A. Müller, M. Pezzè, and M. Shaw. *Engineering Self-Adaptive Systems through Feedback Loops*, volume 5525 of *Lecture Notes in Computer Science*, pages 48–70. Springer-Verlag, 2009. 3, 11, 57, 62, 66

[BU11] E. Bucherer and D. Uckelmann. *10 Business Models for the Internet of Things*, pages 1–25. Springer Berlin Heidelberg, 2011. 226, 229

[Bun04] H. Bunt. Modular Partial Models: A Formalism for Context Representation. In *Proceedings 4th International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT 2004)*, volume 2680 of *LNCS*, pages 427–434. Springer, 2004. 43

[BYV07] R. M. Bell, K. Yehuda, and C. Volinsky. The BellKor Solution to the Netflix Prize. http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf, 2007. 234

[CA06] M. Chantzara and M. Anagnostou. Designing the Context Matching Engine for Evaluating and Selecting Context Information Sources. In *Proceedings 2nd International Workshop on Modeling and Retrieval of Context (MRC 2005)*, volume 3946 of *LNAI*, pages 101–117. Springer, 2006. 46, 313

[CBA09] L. Chen, M. A. Babar, and N. Ali. Variability Management in Software Product Lines: A Systematic Review. In *Proceedings 13th International Software Product Line Conference (SPLC 2009)*, pages 81–90. Carnegie Mellon University Software Engineering Institute, 2009. 311

[CCD05] J. Coutaz, J. L. Crowley, and S. Dobson. Context is Key. *Communications of the ACM (CACM)*, 48(3):49–53, 2005. 37, 48, 313

[CCF04] G. Candea, J. Cutler, and A. Fox. Improving Availability with Recursive Microreboots: a Soft-State System Case Study. *Performance Evaluation*, 56(1-4):213–248, 2004. Dependable Systems and Networks - Performance and Dependability Symposium (DSN-PDS) 2002: Selected Papers. 63, 148, 149, 153, 156, 319

[CCG+09] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. QoS-driven runtime adaptation of service oriented architectures. In *Proceedings ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 2009)*, pages 131–140, New York, NY, USA, 2009. ACM. 63, 147, 149, 152, 155, 319

[CCNY10] M. Chignell, J. Cordy, J. Ng, and Y. Yesha, editors. *The Smart Internet: Current Research and Future Applications*. Springer-Verlag, Berlin, Heidelberg, 2010. 9, 26, 28, 29, 31, 283

[CCRR02] J. L. Crowley, J. Coutaz, G. Rey, and P. Reignier. Perceptual Components for Context Aware Computing. In *Proceedings 4th International Conference on Ubiquitous Computing (UbiComp 2002)*, volume 2498 of *LNCS*, pages 117–134. Springer, 2002. 11, 312, 313

[CDD+04] J. J. Carroll, I. Dickinson, C. Dollin, A. Seaborne, K. Wilkinson, and D. Reynolds. Jena: Implementing the Semantic Web Recommendations. In *Proceedings 13th International World Wide Web Conference (WWW 2004)*, pages 74–83, 2004. 102, 107, 207, 216

[CGS09] S.-W. Cheng, D. Garlan, and B. Schmerl. Evaluating the Effectiveness of the Rainbow Self-Adaptive System. In *Proceedings 2009 Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009) ICSE Workshop*, pages 132–141, may 2009. 251

[Cha08] H. Chang. Modeling Context Life Cycle for Building Smarter Applications in Ubiquitous Computing Environments. In *Proceedings Confederated International Conferences on the Move to Meaningful Internet Systems (OTM 2005)*, volume 5333 of *LNCS*, pages 851–860, 2008. 313

[Cho07] O. Choi. A Meta Data Model of Context Information for Dynamic Service Adaptation on User Centric Environment. In *Proceedings International Conference on Multimedia and Ubiquitous Engineering (MUE 2007)*, pages 108–113, 2007. 313

[CL05] G. Chartrand and L. Lesniak. *Graphs & Digraphs*. Chapman & Hall/CRC, 4th edition, 2005. 104

[CLG+09] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, pages 1–26. Springer-Verlag, 2009. 35, 57, 61, 62, 283

[Cre09] J. W. Creswell. *Research Design – Qualitative, Quantitative and Mixed Methods Approaches*. SAGE Publications, 3rd edition, 2009. 6

[Cro03] J. L. Crowley. Context Driven Observation of Human Activity. In *Proceedings 1st European Symposium on Ambient Intelligence (EUSAI 2003)*, volume 2875 of *LNCS*, pages 101–118. Springer, 2003. 46, 312

[CRS07] D. Conan, R. Rouvoy, and L. Seinturier. Scalable Processing of Context Information with COSMOS. In *Proceedings 7th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS 2007)*, volume 4531 of *LNCS*, pages 210–224. Springer, 2007. 43

[Dah10] Dahm, W.J.A. Technology Horizons a Vision for Air Force Science & Technology During 2010-2030. Technical report, U.S. Air Force, 2010. 61, 66, 276

[DC04] J. Dowling and V. Cahill. Self-Managed Decentralised Systems using K-Components and Collaborative Reinforcement Learning. In *Proceedings 1st ACM SIGSOFT Workshop on Self-Managed Systems*, pages 39–43, New York, NY, USA, 2004. ACM. 62, 63, 148, 149, 150, 153, 154, 155, 319

[DCtTdK11] K. Dentler, R. Cornet, A. ten Teije, and N. de Keizer. Comparison of Reasoners for Large Ontologies in the OWL 2 EL Profile. *Semantic Web*, 2(2):71–87, 2011. 257

[Dey01a] A. K. Dey. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16(2-4):97–166, 2001. 312

[Dey01b] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001. 312

[DH02a] G. Dumont and M. Huzmezan. Concepts, Methods and Techniques in Adaptive Control. In *Proceedings IEEE American Control Conference (ACC 2002)*, volume 2, pages 1137–1150, Anchorage, AK, USA, 2002. 60, 188, 192

[DH02b] G. Dumont and M. Huzmezan. Concepts, Methods and Techniques in Adaptive Control. In *Proceedings 2002 American Control Conference*, volume 2, pages 1137–1150. IEEE, 2002. 145, 176

[dLGM+13] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cikic, R. Desmarais, S. Dustdar,

G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. *Software Engineering for Self-Adaptive Systems: A second Research Roadmap*, volume 7475, pages 1–32. Springer, 2013. 2, 16, 18

[DM05] A. K. Dey and J. Mankoff. Designing Mediation for Context-Aware Applications. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 12(1):53–80, 2005. 313

[DRS+07] C. Dudley, L. Rieu, M. Smithson, T. Verma, and B. Braswell. *WebSphere Service Registry and Repository Handbook*. IBM Redbooks, 1st edition, 2007. 274

[DWM08] D. Dudkowski, H. Weinschrott, and P. Marron. Design and Implementation of a Reference Model for Context Management in Mobile Ad-Hoc Networks. In *Proceedings 22nd International Conference on Advanced Information Networking and Applications (AINAW 2008)*, pages 832–837. IEEE Computer Society, 2008. 313

[EER+10] H. Ehrig, C. Ermel, O. Runge, A. Bucchiarone, and P. Pelliccione. *Formal Analysis and Verification of Self-Healing Systems*, volume 6013 of *LNCS*, pages 139–153. Springer, 2010. 63, 146, 148, 149, 153, 319

[EGMT09] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model Evolution by Run-Time Parameter Adaptation. In *Proceedings 31st International Conference on Software Engineering (ICSE 2009)*, pages 111–121. IEEE, 2009. 188, 193

[End95] M. R. Endsley. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors*, 37(1):32–64, 1995. 38

[ES07] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. 273

[ESSD08] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. *Selecting Empirical Methods for Software Engineering Research*, pages 285–311. Springer London, 2008. 6

[Euz08] J. Euzenat. Dynamic Context Management for Pervasive Applications. *Knowledge Engineering Review*, 23(1):21–49, 2008. 11

[EVMT12] S. Ebrahimi, N. M. Villegas, H. A. Müller, and A. Thomo. SmarterDeals: A Context-aware Deal Recommendation System based on the SmarterContext Engine. In *Proceedings 2012 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2012)*, pages 116–130, Riverton, NJ, USA, 2012. IBM Corp. 8, 14, 16, 20, 34, 111, 234

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999. 52

[FHS+06] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using Architecture Models for Runtime Adaptability. *IEEE Software*, 23:62–70, 2006. 62, 63, 148, 149, 150, 154, 319

[FVP+11] M. E. Frîncu, N. M. Villegas, D. Petcu, H. A. Müller, and R. Rouvoy. Self-Healing Distributed Scheduling Platform. In *Proceedings 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2011)*, pages 225–234, Washington, DC, USA, 2011. IEEE Computer Society. 16, 19

[GCB07] M. Graves, A. Constabaris, and D. Brickley. FOAF: Connecting People on the Semantic Web. *Cataloging & Classification Quarterly*, 43(3-4):191–202, 2007. 85

[GCH+04] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer*, 37:46–54, 2004. 62, 63, 147, 251, 254, 319

[GHJV97] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Longman, Inc, 2 edition, 1997. 115

[Gie05] M. Giereth. Partial Encryption of RDF Graphs. In *Proceedings 4th International Semantic Web Conference (ISWC2005)*, Heidelberg, 2005. Springer-Verlag. 216

[Gie06] M. Giereth. PRE4J - A Partial RDF Encryption API for Jena. *Academic Medicine*, 70(3):216–223, 2006. 216

[Goo12] Google Inc. The Google Product Taxonomy. http://support.google.com/merchants/bin/answer.py?hl=en&answer=160081, 2012. 97, 238

[GPG09] A. González, E. Piel, and H.-G. Gross. A Model for the Measurement of the Runtime Testability of Component-Based Systems. In *Proceedings International Conference on Software Testing Verification and Validation Workshops (ICSTW 2009)*, pages 19–28. IEEE, 2009. 190

[Gro12] Groupon. The Groupon Deal Categories. https://sites.google.com/site/groupon/apiv2/api-resources/deals, 2012. 98

[HDPT04] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004. 11, 12, 13, 56, 66, 142, 145, 146, 148, 149, 184, 252, 281

[Hep08] M. Hepp. GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In *Proceedings 16th international conference on Knowledge Engineering: Practice and Patterns (EKAW 2008)*, pages 329–346. Springer-Verlag, 2008. 85

[HGB10] R. Hebig, H. Giese, and B. Becker. Making Control Loops Explicit when Architecting Self-Adaptive Systems. In *Proceedings 2nd International Workshop on Self-Organizing Architectures (SOAR 2010)*, pages 21–28, New York, NY, USA, 2010. ACM. 11

[HIM05] K. Henricksen, J. Indulska, and T. McFadden. Modelling Context Information with ORM. In *Proceedings Confederated International Workshops on the Move to Meaningful Internet Systems (OTM 2005)*, volume 3762 of *LNCS*. Springer, 2005. 43, 313

[HIMB05] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for Distributed Context-Aware Systems. In *Proceedings Confederated International Conferences on the Move to Meaningful Internet Systems (OTM 2005)*, volume 3760 of *LNCS*, pages 846–863. Springer, 2005. 313

[HIR08] P. Hu, J. Indulska, and R. Robinson. An Autonomic Context Management System for Pervasive Computing. In *Proceedings 6th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008)*, pages 213–223, 2008. 11, 313

[HKR09] P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*, volume 33 of *Textbooks in Computing*. Chapman & Hall/CRC, 1st edition, 2009. 49, 50, 51, 54, 84, 102, 104, 140, 256, 257, 264, 282, 283

[HKTR04] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems*, 22:5–53, 2004. 240

[Hoa09] C. Hoareau. Modeling and Processing Information for Context-Aware Computing: A Survey. *New Generation Computing*, 27(3):177–196, 2009. 313

[HRH09] G. Hynes, V. Reynolds, and M. Hauswirth. Enabling Mobility between Context-Aware Smart Spaces. In *Proceedings International Conference on Advanced Information Networking and Applications Workshops (WAINA 2009)*, pages 255–260, 2009. 50, 313

[HRI07] P. Hu, R. Robinson, and J. Indulska. Sensor Standards: Overview and Experiences. In *Proceedings 3rd International Conference on Intelligent Sensors, Sensor Networks and Information (ISSNIP 2007)*, pages 485–490, 2007. 313

[HS10] A. Hotho and G. Stumme. Towards the Ubiquitous Web. *Semantic Web*, 1(1,2):117–119, 2010. 36

[Hyn09] G. Hynes. A Context Lifecycle for Web-Based Context Management Services. In *Proceedings 4th European Conference on Smart Sensing*

and Context (EuroSSC 2009), volume 5741 of *LNCS*, pages 51–65, 2009. 9, 37, 49, 313

[IBM06] IBM Corporation. An Architectural Blueprint for Autonomic Computing. Technical report, IBM Corporation, 2006. 3, 12, 58, 59, 66, 144, 163, 175

[IBM11a] IBM Corporation. IBM Synchronizes its Commerce 2.0 Strategy with Smarter Commerce Initiative. ftp://public.dhe.ibm.com/software/commerce/IDC_IBM_Synchronizes_Its_Commerce_2.0_Strategy_May2011.pdf, 2011. 33

[IBM11b] IBM Corporation. IBM's Smarter Commerce Overview. http://www.ibm.com/smarterplanet/us/en/smarter_commerce, 2011. 2, 283

[IBM12] IBM Corporation. Smarter Commerce: Redefining Business in the Age of the Customer. http://www.ibm.com/smarterplanet/global/files/us_en_us_commerce_brochure_final_new.pdf, 2012. 33

[Int11] Internet Society. Internet Society Discussion Paper: Preserving the User Centric Internet. http://www.isoc.org/pubpolpillar/usercentricity/, 2011. 35

[Kap09] G. M. Kapitsaki. Context-Aware Service Engineering: A survey. *Journal of Systems and Software*, 82(8):1285–1297, 2009. 313

[KB11] Y. Koren and R. M. Bell. *Advances in Collaborative Filtering*, pages 145–186. Springer, Boston, MA, 2011. 326

[KBLT09] M. Knappmeyer, N. Baker, S. Liaquat, and R. Tnjes. A Context Provisioning Framework to Support Pervasive and Ubiquitous Applications. In *Proceedings 4th European Conference on Smart Sensing and Context (EuroSSC 2009)*, volume 5741 of *LNCS*, pages 93–106. Springer, 2009. 313

[KC03] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003. 12, 58, 159, 175, 186, 281

[KC07] B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007. 6, 311

[KCC+07] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Middleware for Enterprise Scale Data Stream Management using Utility-driven Self-Adaptive Information Flows. *Cluster Computing*, 10:443–455, 2007. 62, 63, 64, 148, 149, 156, 320

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA): Feasibility Study. Technical Report CMU/SEI-90-TR-21, Carnegie Mellon University Software Engineering Institute, 1990. 10

[KFNM04] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Proceedings 3rd International Semantic Web (ISWC 2004)*, volume 3298, pages 229–243. Springer Berlin/Heidelberg, 2004. 86

[KH05] M. Krause and I. Hochstatter. Challenges in Modelling and Using Quality of Context (QoC). In *Proceedings 2nd International Workshop Mobility Aware Technologies and Applications (MATA 2005)*, volume 3744 of *LNCS*, pages 324–333. Springer, 2005. 45, 313

[KM07] J. Kramer and J. Magee. Self-Managed Systems: an Architectural Challenge. In *Proceedings 2007 workshop on the Future of Software Engineering (FOSE 2007)*, pages 259–268. IEEE Computer Society, 2007. 57, 66

[Kor08] Y. Koren. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model. In *Proceedings 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008)*, pages 426–434, New York, NY, USA, 2008. ACM. 234, 326

[KS07] R. Krummenacher and T. Strang. Ontology-Based Context Modeling. In *Proceedings 3rd Workshop on Context-Aware Proactive Systems (CAPS 2007)*, 2007. 43, 50

[Liu08] Q. Liu. A Novel Platform for Context Maintenance and Discovery in a Ubiquitous Environment. In *Proceedings 5th International Conference on Embedded and Ubiquitous Computing (EUC 2008)*, pages 565–570, 2008. 313

[LLC10] M. Léger, T. Ledoux, and T. Coupaye. Reliable Dynamic Reconfigurations in a Reflective Component Model. In *Proceedings 13th International Symposium on Component Based Software Engineering (CBSE 2010)*, volume 6092 of *LNCS*, pages 74–92. Springer, 2010. 62, 63, 146, 148, 150, 320

[LLCK09] J. Y. Lee, J. W. Lee, D. W. Cheun, and S. D. Kim. A Quality Model for Evaluating Software-as-a-Service in Cloud Computing. In *Proceedings 7th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2009)*, pages 261–266, Washington, DC, USA, 2009. IEEE Computer Society. 160

[LSA+00] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance Specifications and Metrics for Adaptive Real-Time Systems. In *Proceedings 21st IEEE Real-Time Systems Symposium (RTSS 2000)*, pages 13–23, 2000. 149, 154

[LZ05] S. Lei and R. Zhang. Mobile Context Modelling Using Conceptual Graphs. In *Proceedings IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, 2005 (WiMob 2005)*, volume 4, pages 131–138. IEEE Computer Society, 2005. 313

[MAB02] R. Milo, E. Al, and C. Biology. Network Motifs: Simple Building Blocks of Complex Networks. *Science*, pages 824–827, 2002. 140

[Men00] A. C. Meng. On Evaluating Self-Adaptive Software. In *Proceedings 1st International Workshop on Self-Adaptive Software*, pages 65–74, Secaucus, NJ, USA, 2000. Springer. 146, 149, 154

[MG05] A. Mukhija and M. Glinz. Runtime Adaptation of Applications through Dynamic Recomposition of Components. In *Proceedings 18th International Conference on Architecture of Computing Systems*

(ARCS 2005), pages 124–138, Innsbruck, Austria, 2005. Springer. 62, 63, 64, 146, 147, 150, 320

[MKS09] H. A. Müller, H. M. Kienle, and U. Stege. *Autonomic Computing: Now You See it, Now You Don't—Design and Evolution of Autonomic Software Systems*, volume 5413 of *LNCS*, pages 32–54. Springer, 2009. 12, 66, 192

[MM04] F. Manola and E. Miller. RDF Primer. Technical report, W3C, 2004. 15, 32, 50, 51, 52, 104, 140, 283

[Moo07] P. Moore. A Survey of Context Modeling for Pervasive Cooperative Learning. In *Proceedings 1st International Symposium on Information Technologies and Applications in Education (ISITAE 2007)*, pages K51–K56, 2007. 42, 50

[MPS08] H. Müller, M. Pezzè, and M. Shaw. Visibility of Control in Adaptive Systems. In *Proceedings 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008)*, pages 23–26, 2008. 3, 11, 57, 58

[MSPC12] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac. Internet of Things: Vision, Applications and Research Challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012. 36, 226

[MTVM12] J. C. Munoz, G. Tamura, N. M. Villegas, and H. A. Müller. Surprise: User-controlled Granular Privacy and Security for Personal Data in SmarterContext. In *Proceedings 2012 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2012)*, pages 131–145, Riverton, NJ, USA, 2012. IBM Corp. 8, 14, 19, 34, 84

[MW99] D. Moody and P. Walsh. Measuring The Value Of Information: An Asset Valuation Approach. In *Proceedings 7th European Conference on Information Systems (ECIS 1999)*, pages 496–512, 1999. 224, 226

[NB97] K. S. Narendra and J. Balakrishnan. Adaptive Control Using Multiple Models. *IEEE Transactions on Automatic Control*, 42:171–187, 1997. 145, 176

[NB09] B. M. M. K. Nigel Baker, Madiha Zafar. *Context-Aware Systems and Implications for Future Internet*, volume 0, pages 335–344. IOS Press, 2009. 36

[NCCY10a] J. Ng, M. Chignell, J. Cordy, and Y. Yesha. *Motivation*, volume 6400 of *Lecture Notes in Computer Science*, pages 3–8. Springer Berlin / Heidelberg, 2010. 26

[NCCY10b] J. Ng, M. Chignell, J. Cordy, and Y. Yesha. *Smart Interactions*, volume 6400 of *Lecture Notes in Computer Science*, pages 59–64. Springer Berlin / Heidelberg, 2010. 28

[NCCY10c] J. Ng, M. Chignell, J. Cordy, and Y. Yesha. *Smart Services*, volume 6400 of *Lecture Notes in Computer Science*, pages 173–177. Springer Berlin / Heidelberg, 2010. 29

[NCCY10d] J. W. Ng, M. Chignell, J. R. Cordy, and Y. Yesha. *Overview of the Smart Internet*, pages 49–56. Springer-Verlag, Berlin, Heidelberg, 2010. 26, 30

[NFG+06] L. Northrop, P. Feiler, R. Gabriel, J. Goodenough, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. Ultra-Large-Scale Systems—The Software Challenge of the Future. Technical report, Carnegie Mellon University Software Engineering Institute, 2006. 1, 61, 284

[Ng10] J. Ng. The Personal Web: Smart Internet for Me. In *Proceedings 2010 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2010)*, pages 330–344, Markham, ON, Canada, 2010. IBM Corp. 31, 282

[OGA+06] S. Ou, N. Georgalas, M. Azmoodeh, K. Yang, and X. Sun. A Model Driven Integration Architecture for Ontology-Based Context Modelling and Context-Aware Application Development. In *Proceedings 2nd European Conference Model Driven Architecture—Foundations and Applications (ECMDA-FA2006)*, volume 4066 of *LNCS*, pages 188–197. Springer, 2006. 313

[OMT08] P. Oreizy, N. Medvidovic, and R. N. Taylor. Runtime Software Adaptation: Framework, Approaches, and Styles. In *Proceedings 30th International Conference on Software Engineering (ICSE 2008)*, pages 899–910, 2008. 57, 61

[OSOA07] Open Service Oriented Architecture. SCA Assembly Model version 1.0. http://www.osoa.org, 2007. 198, 274

[PB05] D. Preuveneers and Y. Berbers. Adaptive Context Management Using a Component-Based Approach. In *Proceedings 5th IFIP WG 6.1 International Conference Distributed Applications and Interoperable Systems (DAIS 2005)*, volume 3543 of *LNCS*, pages 14–26. Springer, 2005. 313

[PCP07] N. Paspallis, A. Chimaris, and G. A. Papadopoulos. Experiences from Developing a Distributed Context Management System for Enabling Adaptivity. In *Proceedings 7th IFIP WG 6.1 International Conference Distributed Applications and Interoperable Systems (DAIS 2007)*, volume 4531 of *LNCS*, pages 225–238. Springer, 2007. 49, 313

[PGH+02] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using Control Theory to Achieve Service Level Objectives In Performance Management. *Real-Time Systems*, 23:127–141, 2002. 62, 63, 64, 148, 149, 151, 153, 155, 320

[PGR07] M. Park, M. Gu, and K. Ryu. Context Information Model Using Ontologies and Rules Based on Spatial Object. *Communications in Computer and Information Science*, 2:107–114, 2007. 50, 313

[PRB+08] N. Paspallis, R. Rouvoy, P. Barone, G. A. Papadopoulos, F. Eliassen, and A. Mamelli. A Pluggable and Reconfigurable Architecture for a Context-Aware Enabling Middleware System. In *Proceedings Confederated International Conferences on the Move to Meaningful Internet Systems (OTM 2008)*, volume 5331 of *LNCS*, pages 553–570. Springer, 2008. 313

[PSM+11] G. Pavlopoulos, M. Secrier, C. Moschopoulos, T. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. Bagos. Using Graph Theory to Analyze Biological Networks. *BioData Mining*, 4(1):1–27, 2011. 140

[Rei08] R. Reichle. A Comprehensive Context Modeling Framework for Pervasive Computing Systems. In *Proceedings 8th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, volume 5053 of *LNCS*, pages 281–295. Springer, 2008. 43, 50

[RHI07] R. Robinson, K. Henricksen, and J. Indulska. XCML: A Runtime Representation for the Context Modelling Language. In *Proceedings 5th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2007)*, pages 20–26. IEEE Computer Society, 2007. 313

[RIS+94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings 1994 ACM conference on Computer Supported Cooperative Work (CSCW 1994)*, pages 175–186, New York, NY, USA, 1994. ACM. 323, 324

[RWvM10] P. Reinecke, K. Wolter, and A. van Moorsel. Evaluating the Adaptivity of Computing Systems. *Performance Evaluation*, 67(8):676–693, 2010. Special Issue on Software and Performance. 154, 155, 156

[SACD08] I. Salomie, I. Anghel, T. Cioara, and M. Dinsoreanu. A Context Awareness Model Enhanced with Autonomic Features. In *Proceedings 4th International Conference on Intelligent Computer Communication and Processing (ICCP 2008)*, pages 239–246, 2008. 11, 48, 313

[SBDP08] S. Sicard, F. Boyer, and N. De Palma. Using Components for Architecture-based Management: the Self-Repair case. In *Proceedings 30th International Conference on Software Engineering (ICSE 2008)*, pages 101–110. ACM, 2008. 62, 63, 64, 148, 156, 320

[SBW+10] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. Requirements-Aware Systems. A Research Agenda for RE for Self-Adaptive Systems. In *Proceedings 18th International Requirements Engineering Conference (RE 2010)*, pages 95–103. IEEE, 2010. 191

[Sch02] A. Schmidt. *Ubiquitous Computing - Computing in Context*. PhD thesis, Computing Department, Lancaster University, UK, 2002. 39

[Sch05] A. Schmidt. A Layered Model for User Context Management with Controlled Aging and Imperfection Handling. In *Proceedings 2nd International Workshop on Modeling and Retrieval of Context (MRC 2005)*, volume 3946 of *LNCS*, pages 86–100. Springer, 2005. 313

[SCLM10] C. Sadtler, G. Crooks, J. Laskowski, and S. Mitchell. *Getting Started with Websphere Application Server Feature Pack for Service Component Architecture*. IBM Redbooks, 2010. 198

[SdSJNR+09] J. Strassner, de Souza Jos Neuman, D. Raymer, S. Samudrala, S. Davy, and K. Barrett. The Design of a Novel Context-Aware Policy Model to Support Machine-Based Learning and Reasoning. *Cluster Computing*, 12(1):17–43, 2009. 48, 313

[SG06] J. Schumann and P. Gupta. Bayesian Verification & Validation Tools for Adaptive Systems: Report on Principle of Operation and Prototypical Implementation of Bayesian Envelope Tool for Neural Networks. Technical report, National Aeronautics and Space Administration (NASA), 2006. 188

[Sha95] M. Shaw. Beyond Objects: A Software Design Paradigm Based on Process Control. *ACM Software Engineering Notes*, 20(1):27–38, 1995. 11, 58

[SHK07] N. Samaan, H. Harroud, and A. Karmouch. PACMAN: A Policy-Based Architecture for Context Management in Ambient Networks. In *Proceedings 4th IEEE Consumer Communications and Networking Conference (CCNC 2007)*, pages 497–502. IEEE Computer Society, 2007. 43, 48, 313

[SKKR01] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings 10th International Conference on World Wide Web (WWW 2001)*, pages 285–295, New York, NY, USA, 2001. ACM. 323, 324

[SLBL10] A. Solomon, M. Litoiu, J. Benayon, and A. Lau. Business Process Adaptation on a Tracked Simulation Model. In *Proceedings 2010 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2010)*. ACM, 2010. 63, 148, 149, 153, 320

[SLP04] T. Strang and C. Linnhoff-Popien. A Context Modeling Survey. In *Proceedings Workshop on Advanced Context Modelling, Reasoning and Management at Sixth International Conference on Ubiquitous Computing (UbiComp 2004)*, 2004. 42, 44

[SLPF03] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL: A Context Ontology Language to Enable Contextual Interoperability. In *Proceedings 4th IFIP WG6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS 2003)*, volume 2893 of *LNCS*, pages 236–247. Springer, 2003. 43, 50, 312, 313

[SM95] U. Shardanand and P. Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Proceedings 1995 CHI conference on Human Factors in Computing Systems (CHI 1995)*, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. 236, 323, 324

[SMF+09] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.-B. Stefani. Reconfigurable SCA Applications with the FraSCAti Platform. In *Proceedings 2009 IEEE International Conference on Services Computing (SCC 2009)*, pages 268–275, Washington, DC, USA, 2009. IEEE Computer Society. 198, 200, 274

[SMR+12] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani. A Component-based Middleware Platform for Reconfigurable Service-Oriented Architectures. *Software: Practice and Experience*, 42(5):559–583, 2012. 274

[SPA08] SPARQL Query Language for RDF. Technical report, W3C - World Wide Web Consortium, 2008. 103

[SPG+07a] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007. 102, 207, 256, 257

[SPG+07b] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. http://pellet.owldl.com/papers/sirin05pellet.pdf, 2007. 257

[SRP+05] M. Strimpakou, I. Roussaki, C. Pils, M. Angermann, P. Robertson, and M. Anagnostou. Context Modelling and Management in Ambient-Aware Pervasive Environments. In *Proceedings Workshop on Location- and Context-Awareness (LoCA 2005)*, volume 3479 of *LNCS*, pages 2–15. Springer, 2005. 43

[ST09] M. Salehie and L. Tahvildari. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4:14:1–14:42, 2009. 147

[SW09] H. R. Schmidtke and W. Woo. Towards Ontology-Based Formal Verification Methods for Context Aware Systems. In *Proceedings 7th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2009)*, volume 5538, pages 309–326. IEEE Computer Society, 2009. 50, 313

[Tam12] G. Tamura. *QoS-CARE: A Reliable System for Preserving QoS Contracts through Dynamic Reconfiguration*. Phd thesis, University of Lille 1 - Science and Technology, and Universidad de Los Andes, 2012. 199, 243, 274

[TBK99] D. P. Truex, R. Baskerville, and H. Klein. Growing Systems in Emergent Organizations. *Communications of the ACM*, 42(8):117–123, 1999. 3, 61

[TCCD12] G. Tamura, R. Casallas, A. Cleve, and L. Duchien. QoS Contract-Aware Reconfiguration of Component Architectures Using E-Graphs. In *Proceedings 7th International Workshop on Formal Aspects of Component Software (FACS 2010)*, volume 6921 of *LNCS*, pages 34–52. Springer, 2012. 61, 62, 63, 146, 149, 199, 320

[TKAZC09] C. Taconet, Z. Kazi-Aoul, M. Zaier, and D. Conan. CA3M: A Runtime Model and a Middleware for Dynamic Context Management. In *Proceedings Confederated International Conferences on the Move to Meaningful Internet Systems (OTM 2005)*, volume 5870 of *LNCS*, pages 513–530. Springer, 2009. 43

[TR02] G. M. T. and T. R. *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons, Inc., 2002. 104, 257

[TT09] V. X. Tran and H. Tsuji. A Survey and Analysis on Semantics in QoS for Web Services. In *Proceedings International Conference on Advanced Information Networking and Applications (AINA 2009)*, pages 379–385. IEEE, 2009. 61

[TVM+13] G. Tamura, N. M. Villegas, H. A. Müller, J. P. Sousa, B. Becker, M. Pezzè, G. Karsai, S. Mankovskii, W. Schäfer, L. Tahvildari, and K. Wong. *Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems*, volume 7475 of *LNCS*, pages 108–132. Springer, 2013. 8, 16, 17, 19

[VM10a] N. M. Villegas and H. A. Müller. Context-driven Adaptive Monitoring for Supporting SOA Governance. In *Proceedings 4th International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2010)*, pages 111–133. Carnegie Mellon University Software Engineering Institute, 2010. 8, 14, 16, 20, 66

[VM10b] N. M. Villegas and H. A. Müller. *Managing Dynamic Context to Optimize Smart Interactions and Services*, pages 289–318. Springer-Verlag, Berlin, Heidelberg, 2010. 2, 8, 9, 14, 18, 29, 37, 38, 42, 66

[VM13] N. M. Villegas and H. A. Müller. *The SmarterContext Ontology and its Application to the Smart Internet: A Smarter Commerce Case Study.* LNCS. Springer, 2013. In press. 8, 14, 19, 104

[VMMn+11] N. M. Villegas, H. A. Müller, J. C. Muñoz, A. Lau, J. Ng, and C. Brealey. A Dynamic Context Management Infrastructure for Supporting User-driven Web Integration in the Personal Web. In *Proceedings 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2011)*, pages 200–214, Markham, ON, Canada, 2011. IBM Corp. 8, 14, 19, 34, 66, 84, 85, 93, 104

[VMT11a] N. M. Villegas, H. A. Müller, and G. Tamura. On Designing Self-Adaptive Software Systems. *Revista S&T*, 9(18):29–51, 2011. 16, 20

[VMT11b] N. M. Villegas, H. A. Müller, and G. Tamura. Optimizing Run-Time SOA Governance through Context-Driven SLAs and Dynamic Moni-

toring. In *Proceedings 2011 IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2011)*, pages 1–10. IEEE, 2011. 8, 14, 16, 20, 66, 84

[VMT+11c] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas. A Framework for Evaluating Quality-driven Self-Adaptive Software Systems. In *Proceedings 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011)*, pages 80–89, New York, NY, USA, 2011. ACM. 8, 11, 12, 13, 16, 19, 37, 57, 62, 141

[VNH+10] T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, and B. Becker. Incremental Model Synchronization for Efficient Run-Time Monitoring. In *Models in Software Engineering, Workshops and Symposia at MODELS 2009*, volume 6002 of *LNCS*, pages 124–139. Springer, Heidelberg, 2010. 193

[VTM+13] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas. *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems*, volume 7475 of *LNCS*, pages 265–293. Springer, 2013. 8, 16, 17, 19, 35, 158

[W3C01] W3C - World Wide Web Consortium. XML Schema Part 2: Datatypes - W3C Recommendation. http://www.w3.org/TR/xmlschema-2/, 2001. 53

[W3C04a] W3C - The World Wide Web Consortium. Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 2004. 104

[W3C04b] W3C - World Wide Web Consortium. OWL Web Ontology Language Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210/#s3.1, 2004. 55, 140

[W3C04c] W3C - World Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2004. 15, 51, 53, 101, 104, 118, 122

[W3C04d] W3C - World Wide Web Consortium (W3C). OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, 2004. 15, 32, 50, 51, 53, 81, 101, 282

[Whi05] J. White. Simplifying Autonomic Enterprise Java Bean Applications via Model-driven Development: a Case Study. *The Journal of Software and System Modeling*, pages 601–615, 2005. 63, 148, 153, 155, 320

[WLL+07] T. Weithöner, T. Liebig, M. Luther, S. Böhm, F. Henke, and O. Noppens. Real-World Reasoning with OWL. In *Proceedings 4th European conference on The Semantic Web: Research and Applications (ESWC 2007)*, pages 296–310, Berlin, Heidelberg, 2007. Springer-Verlag. 257

[WMA10] D. Weyns, S. Malek, and J. Andersson. FORMS: a FOrmal Reference Model for Self-adaptation. In *Proceedings 7th international conference on Autonomic computing (ICAC 2010)*, pages 205–214, New York, NY, USA, 2010. ACM. 66

[WSB+10] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel. RELAX: A Language to Address Uncertainty in Self-Adaptive Systems Requirement. *Requirements Engineering*, 15(2):177–196, 2010. 37

[Yel04] Yelp Inc. Yelp's Academic Dataset. http://www.yelp.com/academic_dataset, 2004. 8, 236

[Yin03] R. K. Yin. *Case Study Research: Design and Methods*, volume 5. SAGE Publications, 3rd edition, 2003. 6

[ZH08] W. Zhang and K. M. Hansen. Semantic Web Based Self-Management for a Pervasive Service Middleware. In *Proceedings 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*, pages 245–254, 2008. 313

[ZLO07] A. Zimmermann, A. Lorenz, and R. Oppermann. An Operational Definition of Context. In *Proceedings 6th International and Interdisciplinary Conference on Modeling and using Context (CONTEXT 2007)*, volume 4635 of *LNCS*, pages 558–571. Springer, 2007. 9, 40, 41, 313

[ZSL05] A. Zimmermann, M. Specht, and A. Lorenz. Personalization and Context Management. *User Modeling and User-Adapted Interaction*, 15(3-4):275–302, 2005. 313

Appendix A

Context Information Survey: Methodology and Demography

This appendix summarizes our survey on context information by presenting data related to the selected references and the research communities involved in the problems of context modeling and context management. Moreover, it details the process followed in our study (i.e., bibliographic search, filtering, and classification), and presents useful information about trends in the evolution of this field of research. Most importantly, the information presented in this appendix is valuable to guide context-related research communities in leveraging existing research contributions in the design and implementation of suitable approaches for modeling and managing context information.

Methodological Aspects

Our survey was based on some of the guidelines proposed by Kitchenham and Charters in their *systematic review method* [KC07] and its application proposed by Chen *et al.* [CBA09]. The first activity of this process was our bibliographic search. Based on our set of research questions related to context definition, modeling, and management, we defined a list of keywords and search strings used for our investigation. The second step involved the filtering and classification of the results obtained from the previous activity. To determine whether or not a particular search result was related to the definition, modeling or management of context information, we fully reviewed the abstract and the contributions section of each paper. Then, the filtering was further refined by concentrating on contributions published in high-quality international

journals and conference proceedings. Finally, a few additional papers were included into the set, after the detailed review of the related work of the selected papers.

Demographic and Chronological Data

Figure A.1 illustrates the distribution of context definition, context modeling, and context management contributions from 53 papers by publication year. Before 2000, context-aware and pervasive computing were emergent areas of research. Most research efforts focused on the understanding of context and context-awareness [ADB+99]. In the next phase, from 2000 to 2003, researchers continued working on the understanding and definition of context, but also some efforts emerged dealing with the representation and exploitation of context information. Computer scientists, such as Dey [Dey01a, Dey01b], Strang [SLPF03], Crowley [CCRR02, Cro03] and colleagues, made many valuable contributions. Finally, the last two periods of time focus on addressing the challenges inherent in the representation, acquisition, handling, and provisioning of context information. In particular, the increasing number of contributions in context modeling and management can be explained with the need for providing context information to support several challenges in different research communities.

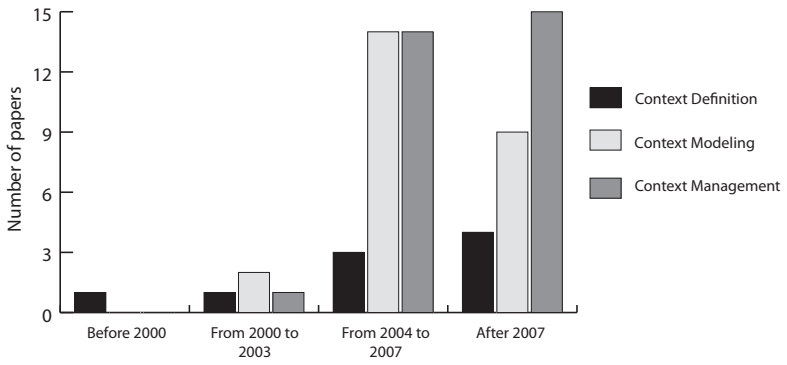


Figure A.1: Distribution of context definition, context modeling, and context management contributions by publication year.

Our survey considered the following list of papers presented by category (i.e., context definition, context modeling and context management) and in chronological order. If a paper was useful for the identification of features in more than one category, then it is listed more than once.

1. Context definition: Abowd *et al.* [ADB+99], Dey [Dey01b], Coutaz *et*

al. [CCD05], Zimmermann [ZSL05, ZLO07], Chang [Cha08], Kapitsaki [Kap09], Hynes [Hyn09], Hoareau [Hoa09].

2. Context modeling: Crowley *et al.* [CCRR02], Strang [SLPF03], Henricksen *et al.* [HIM05], Krause and Hochstatter [KH05], Lei and Zhang [LZ05], Schmidt [Sch05], Ou *et al.* [OGA+06], Park [PGR07], Robinson *et al.* [RHI07], Choi [Cho07], Salomie *et al.* [SACD08], Achilleos [AYG10], Anagnostopoulos *et al.* [AAH09].

3. Context management: Strang *et al.* [SLPF03], Krause and Hochstatter [KH05], Preuveneers and Berbers [PB05], Dey [DM05], Henricksen *et al.* [HIMB05], Chantzara and Anagnostou [CA06], Hu *et al.* [HRI07], Paspallis *et al.* [PCP07], Robinson *et al.* [RHI07], Samaan *et al.* [SHK07], Liu [Liu08], Salomie *et al.* [SACD08], Hu *et al.* [HIR08], Dudkowski *et al.* [DWM08], Paspallis *et al.* [PRB+08], Zhang and Hansen [ZH08], Strassner *et al.* [SdSJNR+09], Hynes *et al.* [HRH09], Schmidtke and Woo [SW09], Abid *et al.* [ACC09], Knappmeyer *et al.* [KBLT09]

Research Communities

Figure A.2 (a) presents the overall distribution of contributions by research community. Each paper studied in our survey was classified into one of seven research communities: *artificial intelligence and knowledge representation, autonomic computing, human-computer interaction, mobile computing and wireless networks, model driven engineering, pervasive and ubiquitous computing,* and *self-adaptive and self-organizing systems*. Most approaches from autonomic computing, mobile computing, model driven engineering, and self-adaptive and self-managing systems belong to the software engineering community. It is important to note that this information does not represent an exhaustive search of the contributions in each community. Nevertheless, the graph shows the participation of each community within the set of selected papers for this survey. It is possible to conclude that before 2000, most efforts for understanding and defining context information were made by researchers in the pervasive and ubiquitous computing domain. At the turn of the century, other research communities started to contribute to this area. From 2000 to 2003, most contributions originated from the pervasive and ubiquitous computing, human-computer interaction, and mobile computing and wireless networks communities; from 2004, many valuable contributions were added by the self-adaptive and self-organizing systems,

artificial intelligence and knowledge representation, autonomic computing and model driven engineering communities. The variety of the contributions has been generated in part by the increasing complexity of the problems studied by these research communities. Moreover, dealing with the dynamic nature of context information is a concern of diverse research communities in computer science, software engineering, engineering in general, health information sciences, and social sciences in general.

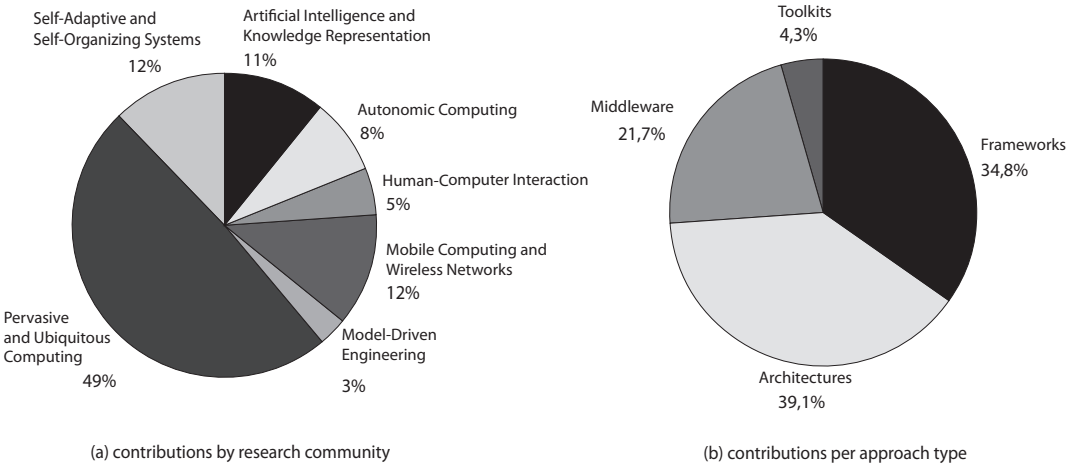


Figure A.2: Overall distribution of contributions by (a) research community, and (b) approach type

Context Modeling and Context Management Features

In this survey, the set of contributions related to context modeling consists of 25 papers while the set for context management includes 30. Figure A.2 (b) depicts the distribution of context management contributions per approach type. Figure A.3 (a) presents the distribution of modeling contributions for each top level feature detailed in Section 2.2.3. Generally, early approaches focused on the representation of entities and situations—a fundamental problem in context modeling. Figure A.3 (b) displays the distribution of contributions for a set of selected management features considered important for supporting smart interactions and smart services.

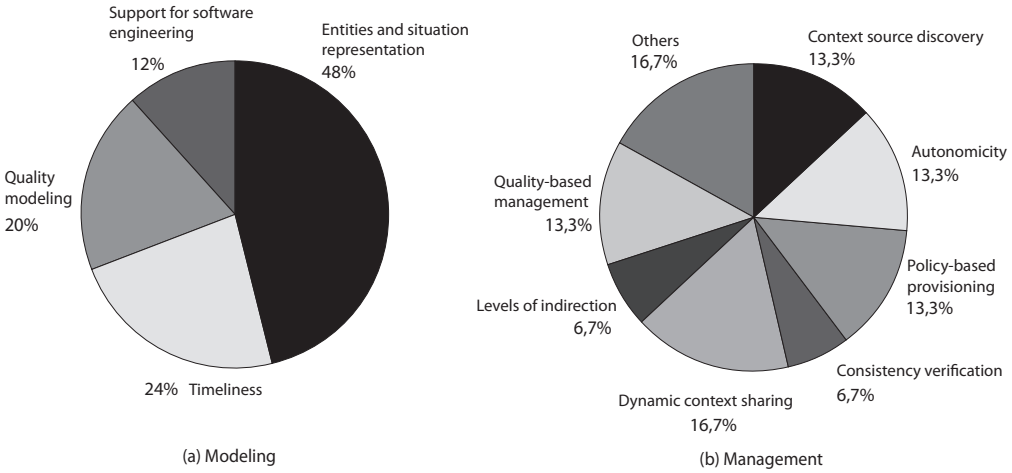


Figure A.3: Distribution of contributions by (a) context modeling features, and (b) selected features of context management

Appendix B

A Personal Context Sphere Example

This appendix presents a partial view of the PCS for user Norha in our situation-aware smarter shopping case study. Each triple in Table B.1 below represents an RDF statement, contextual facts in SmarterContext, defined by a subject, a predicate, and an object.

The World Wide Web Consortium[1] (W3C) provides an RDF validation service[2] useful to visualize small RDF graphs. It is possible to visualize this partial view of user Norha's context model by copying the RDF/XML content to the "Check by Direct Input" field of the validator, or by entering the URL of the model (`http://smartercontext.org/examples/thesis/norha.rdf`) in the field "Check by URI" of the validator.

[1] http://www.w3.org
[2] http://www.w3.org/RDF/Validator

Table B.1: RDF triples that define the personal context sphere for user Norha

#	Subject	Predicate	Object
1	google:XBox_360_Consoles	rdf:type	shopping:ProductService Category
2	gabriel.rdf#gabriel	rdf:type	gc:HumanEntity
3	google:Earrings	rdf:type	shopping:ProductService Category
4	geo:Victoria	rdf:type	gc:GeoLocation
5	deals:Gyms_&_ Fitness_Centers	rdf:type	shopping:ProductService Category
6	google:Electric_Grills	rdf:type	shopping:ProductService Category
7	google:Tennis_Shoes	rdf:type	shopping:ProductService Category
8	http://www.wem.ca	gc:geoLocation Classification	"Place"^^xsd:string
9	http://www.wem.ca	rdf:type	gc:PhysicalLocation
10	http://www.wem.ca	rdf:type	gc:PhysicalLocation
11	norha.rdf#norha	pwc:isInterestedIn	deals:Gyms_&_ Fitness_Centers
12	norha.rdf#norha	pwc:hasIntegrated	http://www.sears.ca
13	norha.rdf#norha	shopping:toBuy	google:Electric_Grills
14	norha.rdf#norha	gc:locatedIn	http://www.wem.ca
15	norha.rdf#norha	pwc:marriedTo	gabriel.rdf#gabriel
16	norha.rdf#norha	rdf:type	pwc:User
17	norha.rdf#norha	pwc:likes	google:XBox_360_Consoles
18	norha.rdf#norha	pwc:preferredLocation	geo:Victoria
19	norha.rdf#norha	shopping:wishes	google:Earrings
20	norha.rdf#norha	pwc:hasIntegrated	http://www.target.ca
21	norha.rdf#norha	pwc:hasIntegrated	http://www.walmart.ca.en
22	norha.rdf#norha	shopping:toBuy	google:Tennis_Shoes
23	norha.rdf#norha	pwc:colleagueOf	tatiana.rdf#tatiana
24	http://www.walmart.ca/en	rdf:type	pwc:PWESite
25	http://www.sears.ca	rdf:type	pwc:PWESite
26	http://www.target.ca	rdf:type	pwc:PWESite

Appendix C

Characterizing Self-Adaptive Software Systems

The tables in this appendix summarize the application of our evaluation framework for quality-driven self adaptive systems, the fourth contribution related to this dissertation that was described in Chapter 6, to the approaches analyzed in our survey on self-adaptive software systems.

Table C.1: Characterization of the self-adaptation approaches analyzed in our survey

Approach	Adaptation Goal	Reference Inputs	Measured Outputs	Control Actions	System Structure	Adaptation Properties	Evaluation/Metrics
Appleby *et al.* Océano [AFF+01]	Self-managing Self-optimizing	Contracts: SLAs	SLOs/Logical properties of computational elements	Discrete operations affecting the comput. infrastructure	Adaptive control/Modifiable structure with reflection	Stability, settling time, small overshoot, scalability/Scalability, Dependability: availability, maintainability	Settl. time: perform. adapt.process./Active servers, connections, resp. time, bwidth, throttle and admission rates
Baresi and Guinea [BG11]	Self-recovery	Contracts: SLAs, funct. req.	SLOs/Logical properties of computational elements	Discrete operations affecting the managed system's process	Adaptive control/Non-modifiable structure	None for the controller/Behav., Dependab.: safety, integrity, availab.,reliab.	Functional and reliability/ $Reliability \geq 0.95 \frac{ResponseFrequency}{TimeUnit}$
Candea *et al.* Microreboots [CCF04]	Self-recovery	Contracts: SLOs-QoS	Malfunction conditions/Logical properties of computational elements	Discrete operations affecting the managed system's process	Adaptive control/Modifiable structure with reflection	Overshoot, settling time/Dependability: availability	Recovery/Availability $A = \frac{MTTF}{MTTF+MTTR}$ Downtime $U = \frac{MTTR}{MTTF+MTTR}$
Cardellini *et al.* MOSES [CCG+09]	QoS preser.	Contracts: QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's process	Reconfig. ctrl./Modif. struct.reflect.	Accuracy/ Perform.: latency, Depend.:reliab., cost.	Accuracy adap.strat./ Response time (Ru), execution cost (Cu), expected reliab. (Du)
Dowling and Cahill K-Components [DC04]	Self-managing	Contracts: SLOs-QoS	SLOs/Logical properties of computational elements	Discrete operations affecting the managed system's software architecture	Reconfigurable control/Modifiable structure with reflection	Robustness, scalability/ Performance: throughput, capacity	None/ Load cost of components
Ehrig *et al.* [EER+10]	Self-healing	Goal actions	Malfunction conditions/Logical properties of computational elements	Discrete operations affecting the managed system's process	Adaptive control/Non-modifiable structure	Termination/ Dependability:reliability	Formal verification of properties. Running example/ None
Floch *et al.* MADAM [FHS+06]	QoS preserv. Self-configuring	Contracts: SLAs-SLOs-QoS	SLOs/Logical properties of computational elements	Discrete operations affecting the managed system's software archiecture	Adaptive control/Modifiable structure with reflection	Scalability/Dependab.: reliability, maintainability	Scalability Simulation environment/ None
Garlan et al. Rainbow [GCH+04]	Self-repairing	Contracts: SLAs-SLOs-QoS	SLOs/Logical properties of computational elements	Discrete operations affecting the managed system's software architecture	Adaptive control/Modifiable structure with reflection	None for the controller/Performance: latency	Effectiveness. Running example/ Latency (not explicitly defined)

Table C.2: Characterization of the self-adaptation approaches analyzed in our survey (cont.)

Approach	Adaptation Goal	Reference Inputs	Measured Outputs	Control Actions	System Structure	Adaptation Properties	Evaluation/Metrics
Kumar *et al.* MWare [KCC+07]	Self-manag. self-config. self-optim.	Contracts: QoS; policy actions	SLOs/Logical properties of computational elements and external context	Discrete operations affecting the managed system's software architecture	Adaptive control/Modifiable structure with reflection	Settling time, overshoot/ Performance: throughput, capacity	Real scenarios/ Business utility function
Léger *et al.* [LLC10]	QoS preservation Self-configuring	Constraints: computational states	Malfunction conditions/Logical properties of computational elements	Discrete operations affecting the managed system's software architecture	Reconfigurable control/Modifiable structure with reflection	Consistency: atomicity, isolation, durability/ Dependability:availability, reliability	Performance. Running example/ None
Mukhija and Glinz CASA [MG05]	QoS preservation Self-config.	Contracts: QoS	SLOs/Logic. properties of comput. elem. ext. context	Discrete oper. managed system's soft. architecture	Reconfig. ctrl./Modif. structure reflection	Consistency/ Performance	Performance. Running examp./ None
Parekh *et al.* [PGH+02]	QoS preservation	Single reference value	SLOs/Logical properties of computational elements	Continuous signals affecting behavioral properties	Feedback control/ Non-modifiable structure based on math.model	Stability, overshoot/ Performance: throughput, capacity	Lotus Notes Running example/ Offered load
Sicard *et al.* [SBDP08]	Self-manag. self-healing	Constraints: computational states	Not explicit monitoring	Discrete operations affecting the managed system's software architecture	Feedback control/Modifiable structure with reflection	None for the controller/ Dependability: reliability, availability	Performance. Simulations/ Availability $A = MTTR$
Solomon *et al.* [SLBL10]	Self-optimization	Contracts: QoS	SLOs/Logical properties of computational elements	Discrete oper. managed system's soft. architecture	Adaptive control/Modifiable structure with reflection	Accuracy/ Performance	Accuracy. Running example/ None
Tamura *et al.* QoSCare [TCCD12]	QoS preservation	Contracts: SLOs-QoS	SLOs/Logical properties of comput. elem.	Discrete oper. managed system's soft. architecture	Reconfigurable control/Modif. struct.reflect.	Termination, consistency/ QoS properties, dependability: reliability	Formal verification of properties. Running example/ None
White *et al.* [Whi05]	Self-manag. self-healing self-protect.	Contracts: SLOs-QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's process	Adaptive control/Non-modifiable structure	Settling time/ Performance:throughput, dependability: availab.	Empirical. Development effort/ Average response time

Appendix D

Modified Features of PRE in Surprise

Table D.1 summarizes the modifications implemented in Surprise with respect to public keys, policies, encryption containers and support for changes at runtime (cf. Column 1 in Table D.1), features that relate directly to the privacy and security requirements in SmarterContext: *dynamic selectivity, dynamic granularity* and *partial encryption*.

Table D.1: Modified features of PRE in SURPRISE

Feature	PRE	SURPRISE	SURPRISE additions to PRE
Public Keys	Integrated in the PREPolicy definition (specified in their policy language (PRE-PL) but not implemented in PRE4J).	Defined at the user level. Stored in SMARTERCONTEXT and associated with users' PCSs.	User encryption keys deleted from the PREPolicySC file and stored in user's PCS to allow indirect association between encryption keys and sensitive context types. This is to address dynamic selectivity.
Policies	One PREPolicy for each sensitive context type. Multiple queries and iterations required when encrypting multiple context types.	One PREPolicySC supports the specification of multiple sensitive context types to be encrypted. Only one query and iteration required to encrypt multiple context types.	Sensitive context types defined as a `KeyRefType` child element in the `TriplePattern` elements of the policy to support the encryption of multiple sensitive context types. This is to address dynamic granularity.
	Triple patterns associated with public keys.	Triple patterns associated with sensitive context types.	
Encrypt. Containers	One EC required for each triple. Single EC unsupported for encrypting triple sets obtained from the application of policies.	One EC for all the triples in the same sensitive context type. Single EC supported for encrypting triple sets obtained from the application of triple patterns defined in policies.	Sensitive context triples grouped according to their context type defined in the SMARTERCONTEXT ontology. This is to address efficiency.
Support for Changes at Runtime	Public keys cannot added to or deleted from an existing EC at runtime.	The `Encryptedkey` slot is modified without affecting other slots in the EC, when adding or removing keys at runtime.	The encryption process was simplified to modify only the `KeyInfo` slots to add or remove public keys dynamically. This is to address dynamic selectivity.
	The addition of new encrypted triples into an existing EC at runtime is unsupported.	New encrypted triples can be added into an existing EC without decrypting the stored data.	The EC meta-data was modified to add the context type of the encrypted data into the clear section of the EC to allow the addition of new encrypted triples without decrypting the existing data. This is to address dynamic granularity.

Appendix E

Collaborative Filtering

Collaborative filtering is a recommendation technique in which users receive recommendations of items that have been positively rated by other people with similar preferences [SKKR01]. The goal of recommendation methods based on collaborative filtering is to predict the unknown rating that a user may give to an item by considering the ratings given to that item by other users.

E.0.1 Calculating Similarities

Our approach exploits similarities among users to recommend product and service categories. The similarity level between a pair of users is calculated based on similar ratings and preferences. Collaborative filtering techniques based on similarities among users are known as *user-based collaborative filtering* techniques [BHK98, RIS+94, SM95]. The similarity between a pair of users can be calculated using different similarity measures such as *correlation*-based (cf. Equation (E.1)) [SM95, RIS+94] and *cosine*-based (cf. Equation (E.2)) [BHK98, SKKR01].

$$sim(u,u') = \frac{\sum_{i \in I_{uu'}} (r_{ui} - \bar{r}_u)(r_{u'i} - \bar{r}_{u'})}{\sqrt{\sum_{i \in I_{uu'}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in I_{uu'}} (r_{u'i} - \bar{r}_{u'})^2}}$$

Equation E.1. Correlation-based user similarity

In these equations r_{ui} and $r_{u'i}$ are the ratings given to item i by users u and u', respectively. $I_{uu'}$ is the set of items co-rated by both users u and u'. In Equation (E.2)

$$sim(u,u') = cos(\vec{u},\vec{u'}) = \frac{\vec{u}.\vec{u'}}{||\vec{u}||_2 \times ||\vec{u'}||_2} = \frac{\sum_{i \in I_{uu'}} r_{ui} r_{u'i}}{\sqrt{\sum_{i \in I_{uu'}} r_{ui}^2} \sqrt{\sum_{i \in I_{uu'}} r_{u'i}^2}}$$

Equation E.2. Cosine-based user similarity

each user is defined as a vector of ratings. In this case the similarity between two users is measured by computing the cosine of the angle between the corresponding two vectors. Once similarities between users are calculated, we can consider the top-N similar users, or users having similarities greater than a desired threshold (i.e., 0.7 for this case study) as the users most similar to a given user.

E.0.2 Rating Prediction

The most important step in collaborative filtering is the prediction of the rating that a particular user would give to an item. A common approach to predict the value of an unknown rating r_{ui} given by user u to item i is the use of an aggregate function of the ratings given to item i by users similar to u. Equation (E.3) presents three different aggregate functions commonly used for rating prediction in collaborative filtering systems [BHK98, RIS+94, SKKR01]. Equation (E.3)(a) is known as *simple average*, Equation (E.3)(b) as *weighted sum*, and Equation (E.3)(c) as *adjusted weighted sum*. Multiplier k is used as a normalizing factor and usually is defined as $k = 1/\sum_{u' \in \hat{U}} |sim(u,u')|$, with $\hat{U}$ as the set of users similar to user u. The average rating of user u in Equation (E.3)(c) is defined as $\bar{r}_u = (1/|S_u|)\sum_{i \in S_u} r_{ui}$, with S_u as the set of all items rated by user u [BHK98].

E.0.3 Baseline Approaches

To evaluate our approach we used two well known recommendation methods as baselines. The first baseline approach is the traditional user-based collaborative filtering method [RIS+94, SM95]. Consider the user-ratings matrix presented in Figure E.1. Rows correspond to users and columns to items (e.g., product categories). Consequently, each cell represents the rating given by a particular user to the corresponding product category. The goal is to predict the unknown rating given by the active user

$$r_{ui} = \frac{1}{N} \sum_{u' \in \hat{U}} r_{u'i}$$

(a) Simple average

$$r_{ui} = k \sum_{u' \in \hat{U}} sim(u, u') \times r_{u'i}$$

(b) Weighted sum

$$r_{ui} = \bar{r}_u + k \sum_{u' \in \hat{U}} sim(u, u') \times (r_{u'i} - \bar{r}_{u'})$$

(c) Adjusted weighted sum

Equation E.3. Rating prediction

u to product category i. That is, to calculate $\hat{r}_{ui}$ represented by the highlighted cell. The first step is to find the users that are similar to the active user. Similarity between users is calculated using the Pearson Correlation Coefficient (PCC) method (cf. Equation (E.1)). Using a threshold of 0.7, we found u_1 and u_3 as the users similar to the active user. To predict the unknown rating this approach uses weighted sum as the aggregation function (cf. Equation (E.3)(b)). This function aggregates the ratings given to product category i by the users similar to the active user, ratings r_{1i} and r_{3i}.

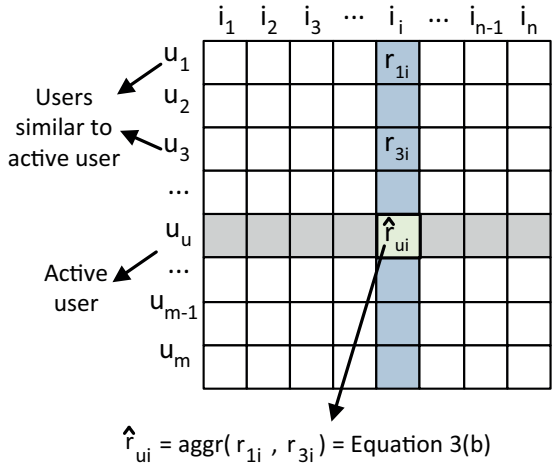


Figure E.1: Traditional user-based collaborative filtering approach

The second baseline approach is the one used by Koren and Bell (winners of the Netflix prize) [Kor08, KB11]. They argue that collaborative filtering data are affected by systematic tendencies for some users to give higher ratings than others, and for some items to be better rated than others. To tackle these effects, they adjust collaborative filtering using *baseline predictors*. A baseline predictor for an unknown rating $r_{u,i}$ denoted by b_{ui} encapsulates these effects that do not involve user-item interaction, and is calculated as:

$$b_{ui} = \mu + b_u + b_i$$

Equation E.4. Baseline predictor

μ is the average rating over all of the items rated by all of the users, and b_u and b_i indicate the observed deviations of user u and item i from the average, respectively:

$$b_i = \frac{\sum_{u \in R(i)} (r_{ui} - \mu)}{\lambda_2 + |R(i)|}$$

Equation E.5. Item deviation

$$b_u = \frac{\sum_{i \in R(u)} (r_{ui} - \mu - b_i)}{\lambda_3 + |R(u)|}$$

Equation E.6. User deviation

Detailed explanations regarding the calculation of b_i and b_u are provided in [KB11]. Koren and Bell demonstrated that 25 and 10 are typical values for λ_2 and λ_3, respectively. We used the same values for the Yelp data set. Unknown ratings are predicted using similarity measures among items. Since our approach is based on similarity among users, we modified Equation 5.17 of [KB11] slightly to calculate unknown ratings as follows:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in S^t(u)} S_{uv}(r_{vi} - b_{vi})}{\sum_{v \in S^t(u)} S_{uv}}$$

Equation E.7. User-based unknown rating

v refers to any user with a similarity with user u higher than the threshold t (i.e., 0.7), and $S^t(u)$ denotes the set of all users similar to u. We call this approach *Adjusted Collaborative Filtering (ACF)*

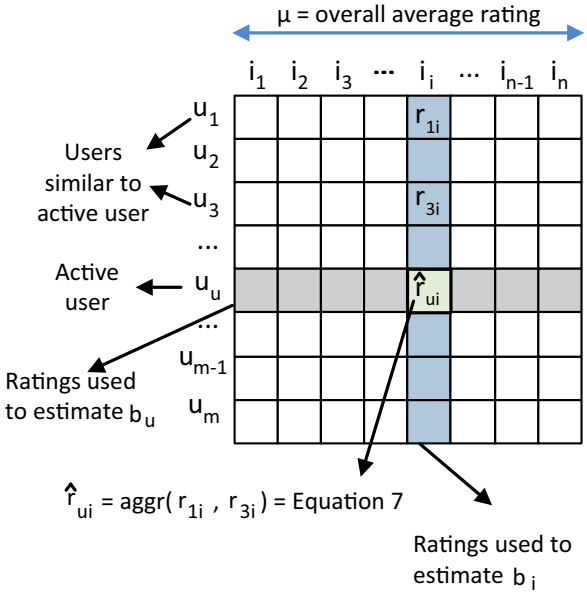


Figure E.2: Adjusted Collaborative Filtering (ACF) approach

Figure E.2 presents an abstraction of the input for the application of ACF with user-based similarities. The ratings used to calculate b_i (cf. Equation (E.5)) correspond to the ratings given to product i. Similarly, the ratings used to calculate b_u (cf. Equation (E.6)) correspond to the ratings given by user u. The unknown rating $\hat{r}_{ui}$ is calculated using Equation (E.7).