Application Infrastructure Evolution:
An Industrial Case Study

by

Marcus Csaky
B.Sc., University of Victoria, 2000

A ThesisSubmitted in Partial Fulfillment
of the Requirements for the Degree of

MASTERS OF SCIENCE

in the Department of Computer Science

Supervisory Committee

Application Infrastructure Evolution:
An Industrial Case Study

by

Marcus Csaky
B.Sc., University of Victoria, 2000

Supervisory Committee

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

Dr. Alex Thomo, (Department of Computer Science)
Departmental Member

Abstract

Supervisory Committee
Dr. Hausi A. Müller, (Department of Computer Science)
Supervisor
Dr. Alex Thomo, (Department of Computer Science)
Departmental Member

Modern computer infrastructure continues to advance at an accelerating pace. This advancement affects all aspects of computing systems such as hardware, device types, operating systems, and software. The constant change creates a problem for software vendors who must support not only the latest systems, but also previous versions. This infrastructure updating requirement limits software vendors when implementing new software features and in turn product innovation. There is a significant and increasing cost associated with adhering to application infrastructure requirement changes.

To accommodate infrastructure change requirements, immediate and measurable action must be taken. First, the current industrial software development practices and tooling need to be analyzed. Current industrial release patterns and product update tooling must be understood and their shortcomings investigated. Next, key use cases ought to be identified and new methodologies and extensions to existing frameworks ought to be proposed to circumvent these difficulties. Finally, systems must be architected and developed to test the solution hypotheses and then deployed in commercial applications. This thesis focuses on these critical steps in the call to action to change software infrastructure management.

In this thesis, we analyze an industrial problem including a real case study. We illustrate the currently available approaches and then describe how they are not sufficient

in solving the problem. Our approach is based on an extension of a previously developed framework (i.e., the SmartContext commerce framework). In particular, we design an architecture and implementation for our industrial problem domain.

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to thank my supervisor Dr. Hausi Müller for his constant encouragement. This thesis simply would not have been possible without his belief in me. As well, I would like to thank my colleagues in the Rigi lab at the University of Victoria. I really enjoyed discussing both my and their research initiatives as well as getting to know them all a little bit outside of research as well. I would like to specifically thank Norha Villegas for her assistance in helping me to understand her SmarterContext research work and for her very helpful review of my initial thesis drafts.

I would like to express my appreciation to my employer IBM Canada. They sponsored me financially and allowed me time away from work but more importantly they were very understanding of the personal significance of this work to me.

Finally, I would like to thank my friends and family for guiding me down this long path. It is difficult to find the motivation without them especially my mother, who passed away during the writing of this thesis.

Dedication

I would like to dedicate this thesis to my kids, Noah and Lacey.

Chapter 1 Introduction

1.1 Research Motivation and Context

Today's software industry faces a growing problem supporting changing system infrastructure requirements. Operating system developers (e.g., Google and their Android operating system) are distributing new versions frequently [34]. Hardware manufacturers are successfully attaining market share with new devices (e.g., Apple with iPads and iPhones). Modern web browsers (e.g., Firefox, Chrome, and Safari) are distributing new major releases every six weeks [35,36]. Finally, popular third-party libraries (e.g., Dojo) are keeping rapid release rhythms similar to the browsers. The problem involves intricate dependencies as new devices require new operating systems, new operating systems require new browsers, and new browsers in turn require new libraries. These dependencies increase the complexity in developing, testing, and maintaining software systems. Moreover, support provided by current infrastructures is insufficient to address this complexity.

These issues have not developed without reason as the market is demanding the changes. Software customers are interested in new technologies, mobile solutions for example, as they are desperately trying to stay in front of the curve and their competitors. Software users require both legacy system support and at the same time they demand the immediate support of new platforms.

The end result is that a large amount of software development time and effort is required to maintain platform support instead of using that time and effort to deliver innovation in the form of new features and capabilities.

1.2 Components of an Industrial Application

Software systems come in all varieties but there are some trends in terms of the components that comprise them. A software system runs on an operating system. A software system targets specific hardware platforms and increasingly important is the need for it to run on mobile platforms. One way to estimate the increasing rate of platform adoption is to look at the statistics for mobile and web browser usage.

Table 1: Mobile Platform Adoption Rate [21]

Date	Internet Explorer	Chrome	Firefox	Safari	Opera	Mobile
November 2012	31.23%	35.72%	22.37%	7.83%	1.39%	13.08%
October 2012	32.08%	34.77%	22.32%	7.81%	1.63%	12.30%
September 2012	32.70%	34.21%	22.40%	7.70%	1.61%	12.03%
August 2012	32.85%	33.59%	22.85%	7.39%	1.63%	11.78%
July 2012	32.04%	33.81%	23.73%	7.12%	1.72%	11.09%
June 2012	32.31%	32.76%	24.56%	7.00%	1.77%	10.40%
May 2012	32.12%	32.43%	25.55%	7.09%	1.77%	10.11%
April 2012	34.07%	31.23%	24.87%	7.13%	1.72%	9.58%
March 2012	34.81%	30.87%	24.98%	6.72%	1.78%	8.99%
February 2012	35.75%	29.84%	24.88%	6.77%	2.02%	8.53%
January 2012	37.45%	28.40%	24.78%	6.62%	1.95%	8.49%
December 2011	38.65%	27.27%	25.27%	6.08%	1.98%	8.04%
November 2011	40.63%	25.69%	25.23%	5.92%	1.82%	6.95%
October 2011	40.18%	25.00%	26.39%	5.93%	1.81%	6.55%
September 2011	41.66%	23.61%	26.79%	5.60%	1.72%	6.74%
August 2011	41.89%	23.16%	27.49%	5.19%	1.67%	7.12%
July 2011	42.45%	22.14%	27.95%	5.17%	1.66%	7.02%
June 2011	43.58%	20.65%	28.34%	5.07%	1.74%	6.53%
May 2011	43.87%	19.36%	29.29%	5.01%	1.84%	5.75%
April 2011	44.52%	18.29%	29.67%	5.04%	1.91%	5.21%
March 2011	45.11%	17.37%	29.98%	5.02%	1.97%	4.70%
February 2011	45.44%	16.54%	30.37%	5.08%	2.00%	4.45%
January 2011	46.00%	15.68%	30.68%	5.09%	2.00%	4.30%
December 2010	46.94%	14.85%	30.76%	4.79%	2.07%	4.10%
November 2010	48.16%	13.35%	31.17%	4.70%	2.01%	4.02%

Table 1 shows up-to-date statistics for web browser and mobile device usage collected from the company StatCounter [21]. These statistics are derived from over three million websites that total over 15 billion hits (not unique visitors) per month. The data shows that the mobile platform adoption rate has grown to 13.0% from approximately 4.0% over the last two years. Tablet and mobile phone support is almost always a mandatory requirement in software support these days. Table 1 also shows that providing a wide

range of browser support is important. Today, the three dominant web browsers (i.e., Internet Explorer, Firefox, and Chrome) have an almost equal user market share percentage whereas only Internet Explorer and Firefox dominated the market two years ago. Modern software systems either run exclusively or have some sub-components that run inside a web browser and so supporting popular web browsers is very important. Finally, supporting web browser rendering in a COTS (Commercial-Off-The-Shelf) way usually requires that a user interface library is used.

This thesis focuses on typical components of a commercial software product called Webform Server, a sub-component of the IBM Forms Server product. Webform Server is used as a reference but the overall thesis discussion is agnostic to a particular software product. The main reason that a particular product is mentioned is to illustrate that the problems mentioned in this thesis are real industrial problems.

IBM Forms Server delivers e-forms to web browsers and provides a platform to integrate e-forms into other applications. IBM Forms Webform Server processes XML e-forms and generates HTML, CSS, and Javascript. For example, the Webform Server Translator component allows users to fill in, digitally sign, and submit IBM Forms using only a web browser.

IBM Forms delivers many best-in-class features such as a dynamic user experience, data validation, the ability to simply integrate into other applications, extensive digital signature support, diverse language support, and standards-based accessibility support. A dynamic user experience delivers rich user experience (UX) and rules driven interactions that increase a form user's productivity. Examples of such UX are multi-step wizard forms, dynamic choice lists, and embedded HTML form components. Data validation is

composed of embedded constraints and customized business rules that eliminate data capture errors. Finally, the ability to simply integrate into other enterprise systems or web service extensions is achieved through a web-based architecture.

IBM Forms Server (version 8.0 eGA May 2012) supports a variety of operating systems, hardware platforms, and web browsers [11]. The operating systems supported on the server are Windows Server <2003, 2008: 32, 64 bit>, Linux (Red Hat Enterprise <4.0, 5.0, 6.0: 32, 64 bit> and Suse Enterprise <10.0, 11.0: 32, 64 bit>), AIX <5.3, 6.1: 64 bit>, IBM i <6.1, 7.1: 64 bit>, and Solaris SPARC <10: 64 bit>. The operating systems supported on the desktop, where a web browser is used to render forms, are Windows <XP, Vista, 7>, Mac OS X <10.5, 10.6>, Android <4.0>, and iOS <5.1>. The hardware platforms supported are the PC and tablet (iPad and Android powered). The supported web browsers are Internet Explorer <6, 7, 8>, Mozilla Firefox <10.0, 11.0, 12.0>, Chrome <19>, and Safari <5.1, 5.0>. Finally, the Dojo toolkit <1.7> is used to render UI components in the web browser.

The above platform support description does not include the required application server and versions, databases and versions, or possible integration platforms (e.g., Websphere Portal).

The purpose of describing the IBM Forms Webform Server product is to introduce an industrial software product so that we have a concrete example and case study to refer to throughout this thesis. Furthermore, it should be apparent that commercial software must support different environments to be a viable product.

1.3 Problem Description

Upgrading a software system to support new hardware devices, operating systems, web browsers and third-party libraries and multiple versions thereof requires a substantial investment and the cost associated is highly uncertain [18]. One can consider the software update process from either a client (i.e., updating a client software component) or a server (i.e., updating a server software component) perspective. The reality is that system infrastructure requirements are evolving at an increasingly rapid rate.

It is a fact that software consumers are intrigued by mobile devices and their proliferation across the general public and business. For example, Gartner forecasts that the tablet market will increase five-fold in the next four years from 64 million tablets purchased in 2011 to over 320 million tablets purchased in 2015 [9]. For example, the tablet adoption rate in Canada has doubled over the first eight months of 2011 [13]. Approximately 6% of online adults owned a tablet in August 2011 compared to 3% in January 2011. Finally, a mobile computing market analysis predicts that tablets will for the first time outsell laptops in 2013 [20]. According to all accounts, tablets and smart phones are becoming commonplace. Nowadays, tablets are thought of as being interchangeable with laptops and applications that run on a desktop are expected to work seamlessly on smaller devices such as tablets or mobile phones.

Operating systems are more frequently updated than in the past. For example, the release period between Microsoft Windows XP and Vista was 5.5 years and the release times between Windows Vista and Windows 7 was 2.5 years. This trend continues with the recent release of Windows 8. The operating systems that specifically target mobile

devices are released even more frequently. For example, Android and iOS minor version releases come out every one to two months and three to four months respectively.

Popular web browsers are also being updated more frequently than in the past. Internet Explorer (IE) 6 was the main supported Microsoft web browser for over six years until IE 7 was released. Subsequently, IE 8 and 9 have been released with approximately 1.5 to 2 years apart. The Firefox (cf. Table 2) and Chrome browsers have significantly more aggressive release schedules with major version releases created every six weeks. Mozilla states that their switch to accelerating the Firefox releases was undertaken to provide more frequent improvements to users and at the same time as preventing disruption of longer term work [7].

Table 2: Firefox Release Schedule [6]

merge date	release date	central	aurora	beta	release
2011-04-12		Firefox 6	Firefox 5		
2011-05-17				Firefox 5	
2011-05-24		Firefox 7	Firefox 6		
2011-06-21					Firefox 5
2011-07-05		Firefox 8	Firefox 7	Firefox 6	
2011-08-16		Firefox 9	Firefox 8	Firefox 7	Firefox 6
2011-09-27		Firefox 10	Firefox 9	Firefox 8	Firefox 7
2011-11-08		Firefox 11	Firefox 10	Firefox 9	Firefox 8
2011-12-20		Firefox 12	Firefox 11	Firefox 10	Firefox 9
2012-01-31		Firefox 13	Firefox 12	Firefox 11	Firefox 10
2012-03-13		Firefox 14	Firefox 13	Firefox 12	Firefox 11
2012-04-24		Firefox 15	Firefox 14	Firefox 13	Firefox 12
2012-06-05		Firefox 16	Firefox 15	Firefox 14	Firefox 13
2012-07-16	2012-07-17	Firefox 17	Firefox 16	Firefox 15	Firefox 14
2012-08-27	2012-08-28	Firefox 18	Firefox 17	Firefox 16	Firefox 15
2012-10-08	2012-10-09	Firefox 19	Firefox 18	Firefox 17	Firefox 16
2012-11-19	2012-11-20	Firefox 20	Firefox 19	Firefox 18	Firefox 17
2013-01-07*	2013-01-08*	Firefox 21	Firefox 20	Firefox 19	Firefox 18

Table 2 [6] illustrates the six week phases of the Firefox release schedule: i) central; ii) aurora; iii) beta; and iv) release. Note that the code merge date started to occur the day before the release date beginning after Firefox 13 but the reason for this is not clear. The Mozilla Foundation itself has acknowledged that their rapid release schedule has presented difficulties for organizations. They state that the schedule does not allow enough time for organizations to certify new versions and that the associated browser end-of-life policy (i.e., the undocumented policy roughly states that support for a previous major version ends when the next major version is ready for distribution) exposes corporations to significant risks if they choose to remain on non-current versions.

To that end, Firefox has proposed an extended support plan that will maintain releases for nine release cycles or roughly the equivalent of 52 weeks [8]. Organizations that deploy Firefox to their users in a managed environment could use this extended support release cycle to alleviate some deployment concerns.

Third-party libraries are also updated more frequently than ever before. For example, the Dojo toolkit is updated every three to six months depending on the type of release (i.e., major versus minor release number) and the amount of change between versions.

Thus the release schedules for hardware, operating systems, web browsers, and third-party libraries are accelerating. The effect is that an always-increasing amount of effort and cost is required for commercial software to keep up-to-date with the system infrastructure requirements that are both taken for granted and also required.

1.4 Contributions

This thesis discusses methodologies for alleviating the problem of rapidly evolving system infrastructure requirements within an industry environment. The contributions are as follows:

- Analysis of an industrial research problem including a real case study. Considering research problems from an industrial perspective is important to increase technology transfer. We illustrate the currently available approaches and then describe how they are not sufficient in solving the problems.
- Extension of a previously developed architectural framework (i.e., the SmarterContext commerce framework) to demonstrate its applicability to the

application infrastructure domain as a proof of concept. We illustrate the benefits of such a system.

- The design of an architecture for a specific problem domain instantiated from our architectural framework.

1.5 Thesis Outline

Chapter 1 provides an overview of and some context on the subject problem. Chapter 2 further builds on the background information by describing commercial software release patterns that are in existence today. These patterns show how the current industry standards need significant change to resolve rapidly changing system infrastructure requirement evolution issues. Chapter 3 provides a survey of some of the existing industrial and academic update mechanisms. These mechanisms can alleviate the fact that the software release patterns introduced in Chapter 2 do not allow for the rapid software evolution that is required in the marketplace. Shortcomings of each mechanism are highlighted. Chapter 4 describes previous research on software context, automated computing, the SmarterPlanet initiative specifically with the smart Internet, and the SmarterContext framework and in particular how it applies to the SmarterCommerce and SmartDeals systems. Chapter 5 describes how the SmarterContext system can be extended and tailored. Specific use cases illustrate how the linked data and ontologies are extended. In particular we describe a channel instance that exercises use cases. Chapter 6 describes the design and implementation of a new architecture that takes into account the unique aspect of this SmarterContext framework domain. Finally, in a case study we

demonstrate the functionality of our approach. Chapter 7 draws conclusions and outlines avenues for possible future research on this topic.

Chapter 2 Software Release Patterns

2.1 Introduction

There exist numerous software release patterns. These patterns range from those that distribute a major software upgrade every year (or longer) to those that provide updates at a very granular level. The patterns that are highlighted in this section are from the author's personal perspective gained over many years in the software industry. It is important to study these patterns in order to understand the current industrial software release processes. The reality is that none of these existing patterns satisfies the current rapidly changing software landscape. Furthermore, none of these techniques consider the context of the environment when deciding if and what product upgrade is necessary.

2.2 Major Version Release Pattern

A major version release is a typical software pattern whereby several (major and minor) product features, bug fixes, and support for all system infrastructure requirements are intended to be incorporated into one release of the entire software product. This pattern is such that usually a year or more passes between releases of major product versions. There are many examples of software products that use this type of pattern including Microsoft Windows and Internet Explorer to name just a few.

The long duration for this type of product release is necessary to sufficiently coordinate large and often distributed teams to perform tasks beyond writing code and functional testing. Table 3 describes selected tasks that are often required in major product releases.

Table 3: Additional Major Product Release Tasks

Task	Description
Product Architecture	Product architecture and design documentation is ideal to have before implementing product features. Functional and technical feature specifications are also necessary.
Translation	Products often need to be translated into many different languages. Usually teams around the globe provide these translations. There are often several rounds of translation necessary.
Third Party Library Business Relationships	Technically integrating third-party libraries is commonplace in commercial software products. There are often business relationships to establish and nurture at the same time.
Industry Certifications	Regulated industries (government, insurance, banking, health care) often require software to be industry certified. For example, there are industry certifications for engineering standards (ISO9001), accessibility (WCAG 2), and digital signature (JITC).
Legal Clearance	Legal teams need to analyze software integrity before approving a release. Lawyers are interested in filing patents to protect IP, certifying the legal cleanliness of shipped third-party libraries, and certifying the originality of product and feature names.
Usability Study	Product UX if often studied and certified by a usability study to ensure ease of use.
Test Cycles	There are several other testing cycles that are required beyond base-line functional verification testing of a software product. For example, some of these testing cycles are globalization verification testing, translation verification testing, and performance verification testing.
Product Documentation	Product user documentation must be written.
Product Marketing	Product marketing and sales materials must be written.
Product Release	Uploading the binary copies to a release server and creating the physical GA copy of the software

Supporting system infrastructure requirements with this type of product release pattern is problematic. As described above, infrastructure requirements are changing rapidly and third-party libraries and components are upgraded so often that there are sometimes

completely different versions of web browsers (for example) that need to be supported between the start and end dates of the release cycle. To this end, it is easy for the overall release development effort to be paralyzed by the simple mechanics of supporting the latest version of a web browser during the length of a release cycle.

2.3 Minor Version (or Fixpack) Release Pattern

Another typical software release pattern is one where fewer product features, bug fixes, and less support for select system infrastructure requirements are intended to be incorporated into the release. This pattern is such that a shorter time frame than the major version release pattern passes between releases. Most software products use this type of pattern regularly.

This pattern has the advantage of a release being created over a shorter time frame and therefore the support of system infrastructure requirements does not get stale. Shorter and more frequent release time frames are required and hence the software is able to keep up with system infrastructure requirement changes. However, there is a common problem with this pattern. A more frequent release cycle does not allow enough time to update some of the other tasks associated with releasing commercial software such as the ones described in Table 3 above. For example, completing industry certifications may require more time than is available in a minor version release cycle.

2.4 Update Release Pattern

Another typical software release pattern is one where select product features, critical bug fixes, and support for isolated system infrastructure requirements are intended to be

incorporated into a release of sub-components for a software product. This pattern differs from the previous release patterns in that only portions of the entire software package are distributed. An update release is usually distributed on-demand or only when it is required. Most software products use this type of pattern regularly and some examples of this are Windows Updates and Eclipse update sites.

This pattern can be used to distribute software product change sets efficiently. Unfortunately, it is required that the piece of software is architected so that sub-components can be updated in this way and this pattern has similar issues as the minor version release pattern. Namely, it is hard to satisfy the requirements of software development as detailed in Table 3. However, this pattern is ideal for distributing fixes to accommodate system infrastructure requirement changes. The fixes can be distributed in a timely manner such that the system infrastructure requirement can still be relevant by the time that the fix is created and then subsequently installed. It is not necessary to have a significant time lag in releasing the fix. In addition, the space requirements of the system can be optimized if only mandatory components are installed. Optimizing the system's memory footprint is especially important in small factor devices that have smaller memory capacity that desktop or server environments.

The update release pattern has ideal software update efficiency when taking into account the rapidly changing application infrastructure requirements that have already been discussed. However, none of these patterns, including the update release pattern, takes into account the context of software deployment. Software deployment context is discussed more thoroughly in chapter 5 by considering concrete use cases but it can take

on different forms and there are often many unanswered contextual questions related to any update scenario.

A few examples of such contextual information concern the operating system, the content of the fix, and the properties of the update fix. For example, what is the deployment environment for the fix? Which operating system type, version, patch level and configuration? Other considerations include what are the contents of the update fix? What are the properties of the update fix such as its physical size? These questions and many others can be answered by considering contextual details of the system environment.

2.5 Summary

This chapter provided an overview of the author's perspective, based on industrial experience of common software release patterns that range in duration and complexity. The major, minor, and update release types are examples of patterns that no longer work effectively in commercial software development in light of rapidly evolving system infrastructure requirements. We discussed the deficiencies of each pattern and described the one common construct that is missing, context.

Chapter 3 Software Update Mechanisms

3.1 Introduction

There are many software update mechanisms and well-established industrial update products available. These mechanisms and products include both static and dynamic (e.g., during the system runtime) methods. Also, there has been extensive academic research done to provide theoretical frameworks and proof-of-concept updater implementations. This chapter describes these industrial and academic update mechanisms.

3.2 Static Software Updater

IBM Installation Manager (IIM) is an example of a software installation and updater product. IIM uses script files that organize software packages to be statically installed on users' computers. The IIM Kit provides a directory with an instance of the IIM program executable as well as one or more software packages to install. Once the IIM program is installed, a user can choose the software packages to be installed on his or her system thereafter [12]. The specific package installations can be done in a silent (i.e., without user intervention) manner as well. The silent mode installations enable users to install packages without having to explicitly make the choices, however the user must initiate the installation process. Once a package is installed, IIM can be used to modify the software by installing or uninstalling certain features of the packages. The IIM static paradigm for installing and updating software packages is a common one amongst

software installation programs. Even though IIM uses a common and effective methodology, it does not fully allow the dynamic update of software packages or components.

3.3 Dynamic Operating System Based Browser Update

Some products rely on the underlying operating system to provide dynamic updating functionality. Internet Explorer and Safari rely on Windows and the OS X operating systems for this. Both of these operating systems have effective built-in update mechanisms that are run dynamically by default [22]. The OS update mechanisms are designed to help users keep their systems updated. In these cases, Internet Explorer and Safari software are considered to be part of the operating system itself and therefore the web browser software updates occur through the underlying operating system updating mechanism.

Windows Update and Mac OS update function in similar ways. With Windows Update, the user has the following updating options: i) automatically install updates; ii) download updates but optionally choose to install them; iii) check for updates but optionally choose to download and install them; and iv) do not check for updates. The Mac OS update functionality is the same except that there is no way to automatically install updates as the user is always first prompted to commence an update operation [17]. Figures 1, 2 and 3 show screenshots of how the Mac OS updater works. Figure 1 shows the user prompt indicating available updates.

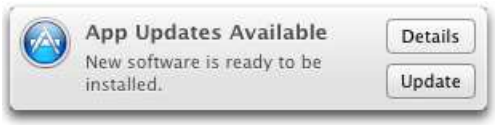

Figure 1: Mac OS X Software Update Available Screenshot [17]

The user has the option to look at details of the updates (cf. Figure 2) or to begin the updating process.

Figure 2: Mac OS X Software Update Details Screenshot [17]

Figure 3 shows the configurability of the updating options.

Figure 3: Mac OS X Software Update Configuration Screenshot [17]

3.4 Dynamic Third Party Browser Update

There are several common web browsers that are not considered to be part of the underlying operating system that have built-in mechanisms for dynamic software updating. Google Chrome and Mozilla Firefox browsers get automatically updated by default when a new version of the browser is made available [3,5]. The default updating process happens in the background and does not require any user intervention besides accepting a prompt for a browser restart (cf. Figure 4).

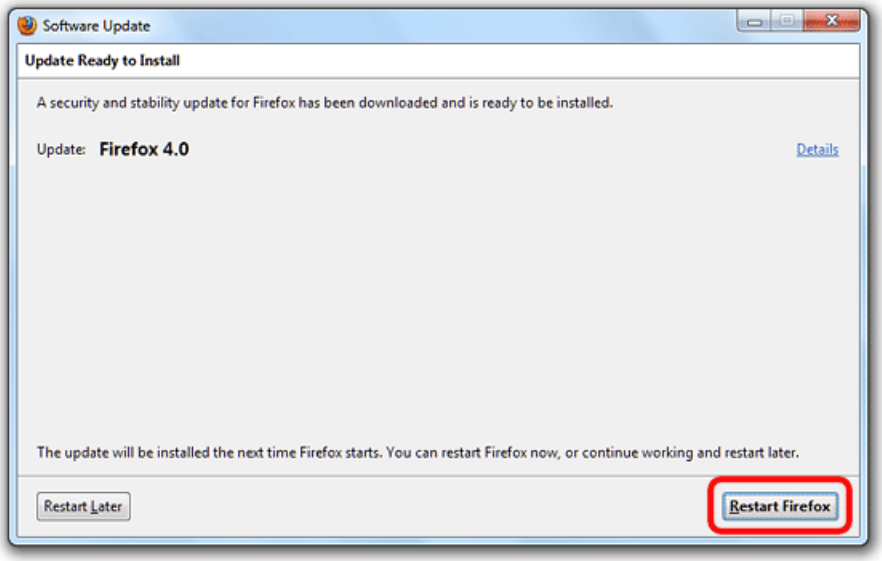

Figure 4: Firefox Software Update Available Screenshots [5]

Both Firefox and Chrome can be configured to disable automatic updating (cf. Figure 5) either completely or in such a way that the user will be first prompted for whether or not to install the updates.

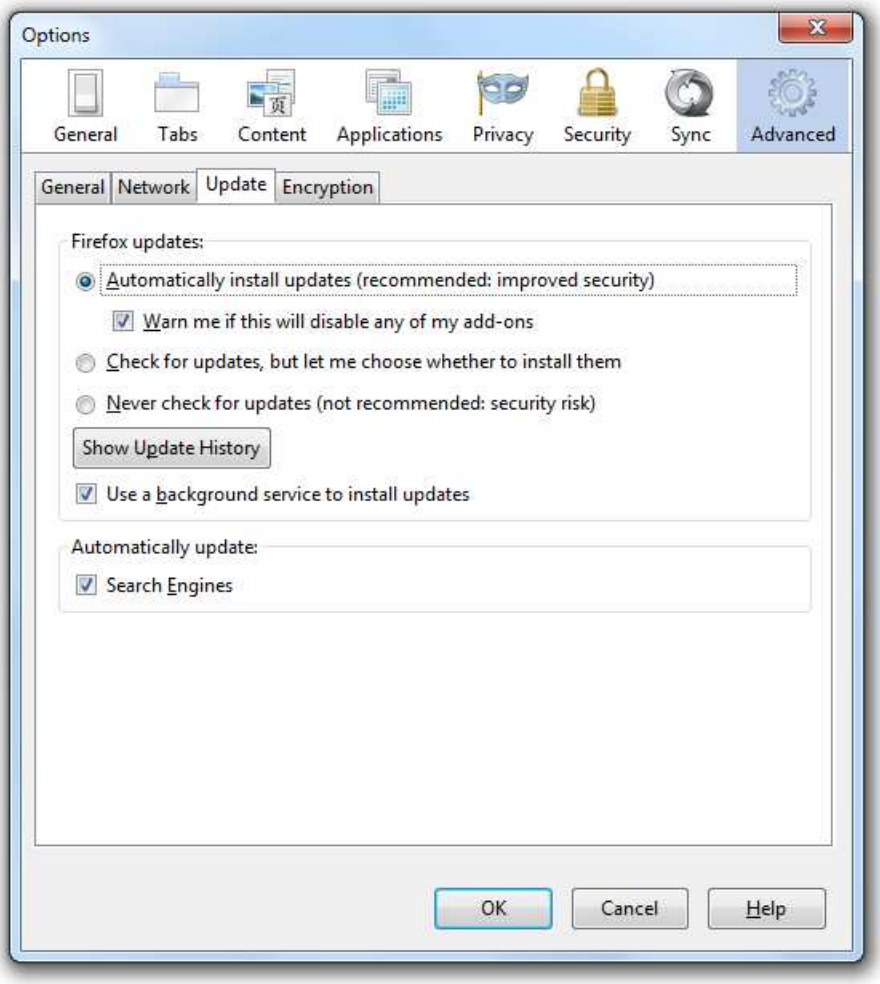

Figure 5: Firefox Software Update Configuration Screenshot [5]

3.5 Dynamic Third Party Application Update

Several common software applications (not web browsers) have built-in mechanisms to perform dynamic software updating. Adobe Reader is a well-known document viewing application that has a mechanism similar to the Chrome and Firefox update process as discussed. Adobe Reader is automatically updated by default when there is a new version available but this also can be configured so that the updates are downloaded but the user is first prompted to install them (cf. Figure 6).

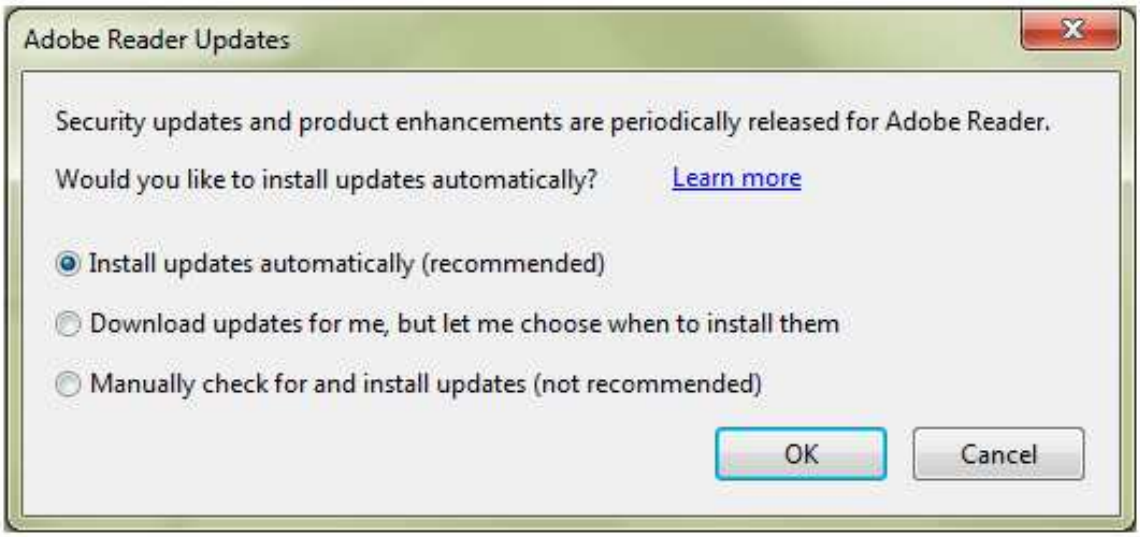

Figure 6: Adobe Reader Dynamic Update Configurability Screenshot [2]

3.6 Dynamic Services Oriented Architecture Update Mechanism

The academic community has also considered the problem of dynamically updating software components. One such representative research effort considers the problem from a services oriented architecture perspective [14]. Kaminski proposed a design technique called Chain of Adapters that makes the management of web service versions simpler.

There were several requirements identified that had to be satisfied with the Chain of Adapters approach. These requirements are: i) maintaining backwards compatibility; ii) using a common data store; iii) not duplicating code; iv) maintaining untangled versions; v) having unconstrained evolution; and vi) making the mechanism visible. Backwards compatibility requires that client software created to work with earlier versions of the web services must also continue to work with newer versions of the web services. Using a common data store refers to maintaining all service states for each version of the web

service supported. The lack of code duplication requirement speaks to adhering to common software engineering principles such that code should not be duplicated whenever possible to ease maintenance. In this particular case, code that handles the web service call should not be duplicated for each version of the web service. Untangled code is the requirement that each piece of code is assigned to one or more versions of the web service. Unconstrained evolution is the requirement that the developer should be allowed to seamlessly refactor, redesign, and re-architect the web service interfaces and implementation. Finally, the need for a visible mechanism requirement means that the manner in which the backwards compatibility is achieved should be readily discoverable.

The Chain of Adapters mechanism achieves all of the above requirements. This mechanism employs an additional delegation layer, or pass-through adapter, to the classic web service architecture. This means that for a given web service: i) the interface of the web service is duplicated into a different name space; ii) an implementation of the interface is created that forwards all service calls to the original endpoint and interface; and iii) this interface endpoint becomes the first version of the web service. When the web service is modified (e.g., by adding a new parameter or changing an interface operation) then a corresponding modification must be made in the adapter to maintain the original version of the web service contract.

Next, the adapter chain is created for the second version of the web service. The interface is duplicated with a new namespace, a new adapter is created for the new interface, the new adapter is set as the target for the new interface, and the new version interface endpoint is published as the second version of the web service. This adapter chaining technique is continued with each new version of the web service that is created.

The Chaining of Adapters method is one example of how a service oriented architecture can be modified with different web service version changes. This is one representative research result but of course there are many other academic software updating methodologies.

3.7 Summary

We provided an overview of common software update mechanisms and products that are available in the marketplace today. Industrial and academic examples were given for both static and dynamic methodologies. Products that employ update mechanisms that we considered include the underlying operating system, the web browser, and third party applications. Finally, we considered an academic Services Oriented Architecture update mechanism. Many update products and mechanisms exist but they all lack the ability to capture and utilize user context.

Chapter 4 Background and Related Work

4.1 Introduction

We live in a world where computer systems are becoming more and more pervasive. These systems transcend society and in doing so they are rapidly increasing in their complexity. They must be able to recognize state at any point in time, and as a result of their changing surroundings, they must be able to automatically adapt to a working environment. The fields of context-aware and self-adaptive software systems have been extensively researched within both industry and academia. Both of these topics often fall under the domain of autonomic computing which describes the ultimate computer system goal of being fully dynamic and automated. The Personal Web framework describes an infrastructure for a modern day and web-enabled autonomic system. The SmarterCommerce and SmarterDeals systems are two concrete examples that illustrate the ideals of the personal web. It is important to consider these related works in order to build up the terminology and conceptual knowledge to further describe the core contributions of this thesis.

4.2 Autonomic Computing

A system can be automated when there are sensors that collect environmental data and provide this data as input back into the system (i.e., a feedback loop). The aim is that the system can then become self-managing which is to say that it can be self-configuring, re-configurable, self-healing, self-protecting, and self-optimising [15].

Autonomic computing systems have a feedback control loop that functions by having the desired system output operate as a direct function of the reference system input (cf. Figure 7).

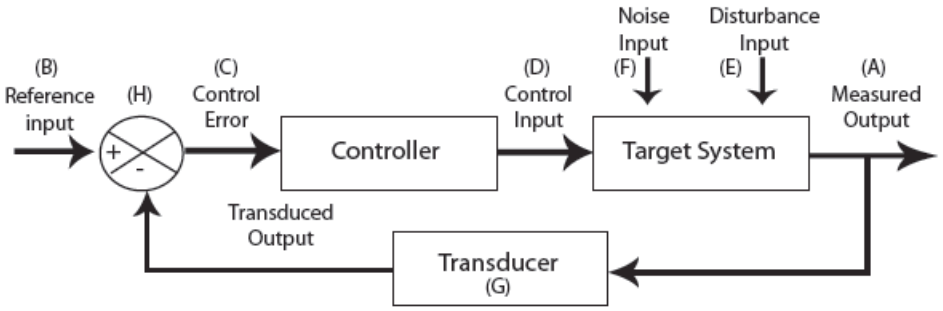

Figure 7: Feedback Control System [15]

A popularly cited autonomic feedback control loop model is MAPE-K as shown in Figure 8. The phases in this model are monitoring, analysis, planning, and execution operating over a knowledge base [1].

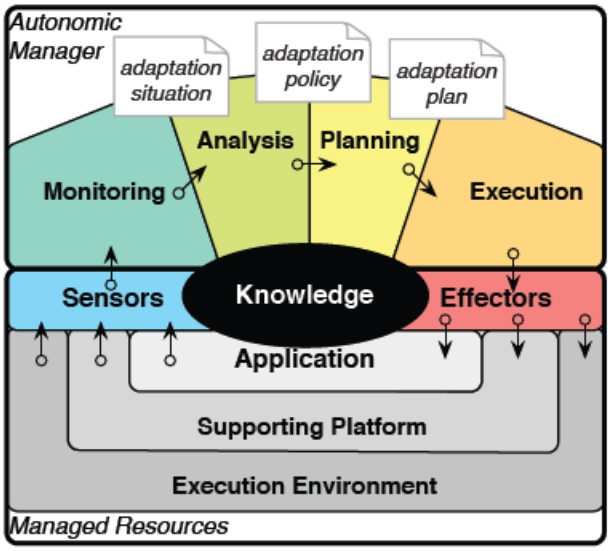

Figure 8: MAPE-K Autonomic Control Loop [1]

4.3 Context-Aware Systems

Context-aware systems use context information to adjust their operation according to changing environments and/or user behaviours [10]. Context information characterizes the state of separate entities and how relationships exist between them [23]. A context-aware system is modeled in such a way that it will allow: i) dynamic and automatic configuration of context information when it is needed; ii) better monitoring to support fault tolerance; iii) dynamic replacement of failed content sources; and iv) opportunistic use of sensors [10].

Context-aware systems can take the form of either: i) each application directly communicating with sensors, processing raw data, and then adapting based on the information that is collected; ii) applications using reusable libraries for processing

context information; or iii) applications having their own context model and using a shared context management architecture to populate the models dynamically [10]. It is the latter system example that we explore in the later *SmarterInfrastructure* system section of this thesis.

Villegas and Müller [23] survey context modeling and context management approaches within different problem domains. They describe how context information can be classified into five categories including individuality (i.e., anything that can be observed about a subject), time, location, activity, and relational context. This classification system is included in the operational definition of dynamic context information that is optimized for the *smarter Internet* use case. They also present a set of feature models and requirements for context representation and management.

There are many challenges related to creating context-aware systems. First, it is necessary to identify the context requirements for a specific set of user matters of concerns (referred to as MOCS). Second, the MOCS must be modeled in such a way that the context is able to be dynamically adapted. Third, a suitable architecture is required to support the dynamically changing context. These architectures must be able to support all aspects of the context management lifecycle as described in Figure 9.

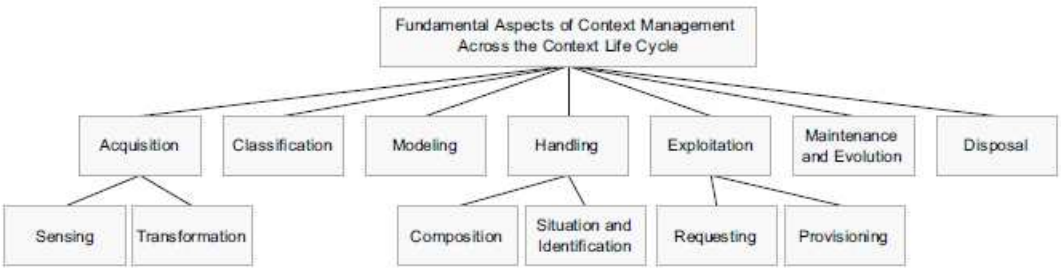

Figure 9: Context Management Lifecycle [23]

The provisioning mechanism of a context management system is an important aspect of its functionality. Provisioning endpoints refer to the communication mechanisms between context-aware services and the context management infrastructure. The ACoMS system [10] supports both querying (pulling) and publish/subscribe (pushing) mechanisms. The two different mechanisms are relevant because the SmarterInfrastructure use cases, as described below, have both a pull and a push mechanism to them.

Finally, the maintenance and evolution of context management infrastructures is also important to consider. This speaks to how context can change and how the infrastructure can evolve at the same time. It becomes important to have the ability to dispose of old and add new context information and this ought to be a feature of any evolutionary system [10].

Romero's thesis provides a thorough reference for considering context as a resource [1]. There are two component models for Service Oriented Architecture (SOA) applications. First, the OSGi framework specification provides an open and dynamic component model for service development, deployment, and management. In OSGi, the applications can be dynamically deployed and updated. Second, the Service Component Architecture (SCA) model is a set of specifications for building distributed applications based on SOA and component-based software engineering principles. The SCA model is evaluated below when considering the question of how to integrate the SmarterInfrastructure framework with the mechanics of updating a system. The SCA paradigm was also the chosen model for the *SmarterCommerce* system.

Context-aware systems have acquired significant traction in the commercial sector. Popular internet commerce systems such as eBay, Amazon, and Groupon [29,30,31] all capture and store information about the users of the system and their purchasing habits. This information is encapsulated in a user's system profile. Similarly, the Apple Genius system [32] stores information about what applications (apps) a user downloaded from the App Store. Related or interesting apps are identified for the user upon their subsequent app store login. The type of information that is being tracked in the commerce or app distribution examples is not explicitly visible to the user but these services provide add-on functionality based on the data. Using internet commerce systems for example, a user's previous purchases are stored and analyzed so their future buying decisions can be influenced. In the example of the Groupon system, targeted suggestions for purchases are made to the user (cf. Figure 10).

Figure 10: Groupon Suggested Items Based on Stored Historical User Data

Figure 10 illustrates the targeted marketing technique by illustrating how other outdoor activity Groupon deals are suggested based on one's previous outdoor deal purchase. In this case, I purchased a Groupon for a sailing adventure and then other outdoor boating activity deals are suggested.

The storage, analysis, and the later action on user data is an inherent pattern of context-aware systems. This data can be analyzed and provisioned in a dynamic manner but most commercial systems like the three mentioned above have static mechanisms. For example, my current Groupon purchase is not influenced in real-time. When I make a purchase, a future purchase suggestion is not provided for me until the next time I log into the system.

4.4 Self-Adaptive Systems

The complexity of current software systems has prompted the investigation of innovative ways of developing, deploying, managing, and evolving software intensive systems and services [16]. Systems must become more versatile, flexible, resilient, dependable, energy-efficient, recoverable, customizable, and self-optimizing by adapting to changes that may occur in their operational contexts, environments, and system requirements. Self-adaptation systems that can modify their behaviour and/or structure in response to their perception of the environment and the system itself have become an important area of research over many diverse application areas [16].

Villegas et al. [25] identified that even though self-adaptation mechanisms are becoming more important and that they are being investigated more, it is still the case that there are a lack of methods available for the verification and validation of such

adaptable applications and environments. For this reason, a framework for evaluating self-adaptive systems was proposed where the adaptation properties are specified explicitly and they are driven by quality attributes. The adaptation properties (cf. Table 4) are stability, accuracy, settling time or the time required for the adaptive system to achieve the desired state, small overshoot or the utilization of computational resources during the adaptation process to achieve the adaptation goal, robustness, termination or the ability to end the adaptation process and in turn avoid system deadlock as a result of the adaptation, consistency, scalability, and security.

Table 4: Quality Properties and Attributes of Self-Adaptive Systems [25]

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	Interact. Complex. Coupling Strength
Termination	Dependability	Reliability Integrity
Consistency	Dependability	Maintainability Integrity
Scalability	Performance	Latency Throughput Capacity
Security	Security	Confidentiality Integrity Availability

DYNAMICO is an effective reference model for control objectives and context relevance in self-adaptive software systems [27]. DYNAMICO helps guarantee the coherence of adaptation and monitoring mechanisms with respect to changes in adaptation goals. It improves self-adaptive systems by addressing: i) the management of adaptation properties and goals as control objectives; ii) the separation of concerns among feedback loops that are required to address control objectives over time; and iii)

the management of dynamic context as an independent control function that preserves context-awareness in the adaptation mechanism. The three control subsystems are a control objectives manager, an adaptation controller mechanism, and a context manager or infrastructure monitoring controller mechanism. These three subsystems form independent feedback control loops (i.e., control, adaptation, and monitoring loops as shown in Figure 11) and hence they greatly increase the system's separation of concerns.

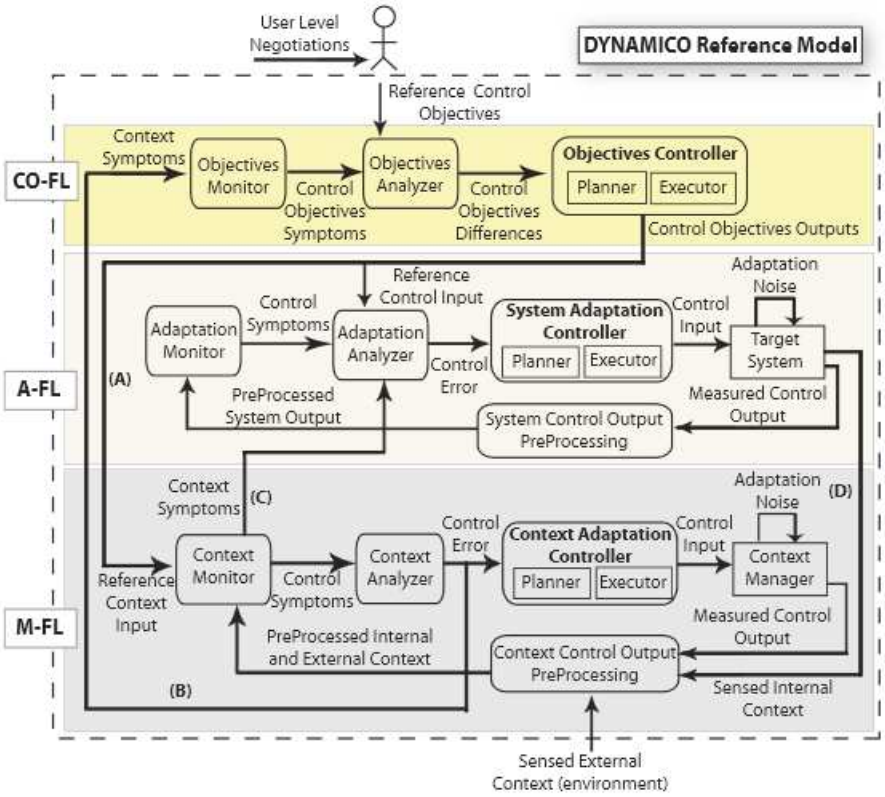

Figure 11: DYNAMICO Reference Model [27]

The control objectives feedback loop (CO-FL) manages the two other loops in collaboration. The adaptation feedback loop (A-FL) serves as a guarantor for regulating the target system's requirements satisfaction and preserving the adaptation properties. The

monitoring feedback loop (M-FL) is crucial for addressing the dynamic nature of the context information. This control loop separation of concerns makes it possible to address three different types of adaptation which are preventive, corrective, and predictive. Preventive adaptation describes how the monitoring feedback loop notifies the adaptation feedback loop about context events. Corrective adaptation takes place when monitoring mechanisms that support the adaptation feedback loop detect that adaptation goals are no longer valid. Finally, predictive adaptation takes advantage of both the historical information to anticipate risks of goal violation and the identification of plausible systems that provide evidence for eventual adaptation.

4.5 Personal Web Framework

The World Wide Web is a good example of a complex environment where dynamic groups of users, stakeholders, businesses, and software and hardware infrastructures must cooperate in complex and changing environments [23]. The smarter Internet initiative was created to architect dynamic applications that are centered on user's goals.

The smarter Internet includes two specific areas of research: *Smarter Interactions* and *Smart Services* [19]. Smarter Interactions deal with aspects as related to the discovery, aggregation, and delivery of resources from the Internet. For example, this could include the deployment of system components to provide a user with personalized services. Smart Services focus on providing the suitable infrastructure to support Smart Interactions. For example, this could include the required functionality to support the dynamic adaptation of the system.

The *Personal Web* is a concrete realization of the smarter Internet that focuses on the user as the center of the web interaction [24]. Smart Interactions are required to support, from a user centric perspective, the discovery, aggregation, and delivery of resources from the Internet. Smart Services must provide the infrastructure that is required by these interactions to assist users in their web interactions. The semantic information that categorizes the relevant web resources, the relationships between them, and the way that the user interacts with them is defined as the Personal Web Sphere. The personal web sphere defines the environmental context that affects the integration as performed by the user on the web. The personal web framework exploits feedback loops and context-awareness techniques to define a context management framework that supports and enables the user to integrate web resources according to their personal matters of concern.

The personal web framework enables the user to readily compose entities into their personal web sphere. Also, the framework is designed with the ideals of open integration and simplicity of operations within web integrations. Finally, the approach supports the instantiation of interactions between the user and instances of context entities, as well as the relationships among these entities that are relevant for the user's interaction. Figure 12 shows the four layers defined in the personal web framework [24].

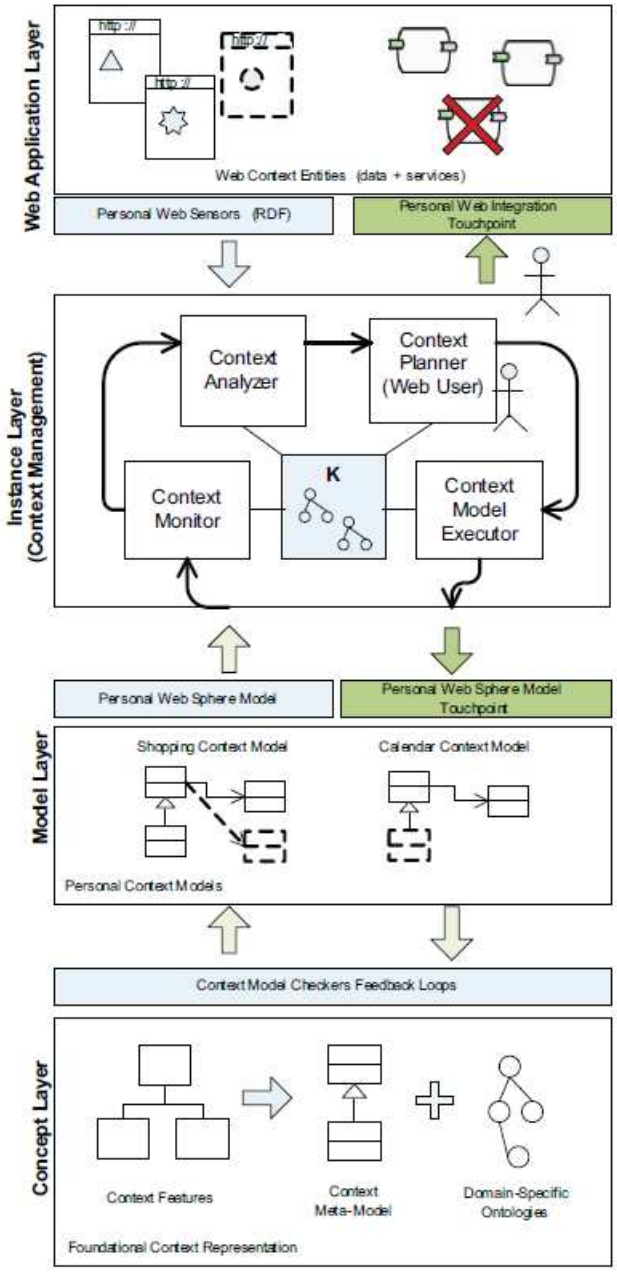

Figure 12: Personal Web Framework Layers of Abstraction [24]

The four layers of system abstraction are modeled after the layered meta-model as defined in the Personal Web. The Concept Layer is composed of a foundational feature-based meta-model that defines the types of context entities and the relationships amongst

them according to the general taxonomy proposed for the smart Internet. The Model Layer contains domain-specific ontologies that instantiate the domain specific personal context models. The Instance Layer describes the instances of the personal context models which are derived from the interaction between the user and the personal web enabled context entities. Personal context instances represent entities of the real world and the manner by which users interact with them. Lastly, the Web Application layer defines a customized context-aware environment to interface between the web and the user.

The personal web framework also takes advantage of user-driven monitoring feedback loops. The personal models that support the instantiation of context entities and a user's interactions can be dynamically updated. By enabling this, the user can control their personal web sphere to integrate new personal data and services [24].

The context information is integrated into or eliminated from the user's context sphere according to the user's web interactions and a life cycle description must be created to do this. This life cycle includes: i) the secure acquisition of context information from personal web-enabled web sites according to user interactions; ii) the integration of contextual information into the user's context sphere; iii) context reasoning to infer new facts from the information in the user's context sphere; iv) the secure provisioning of context information to web sites; and v) the disposal of context information whenever it is no longer relevant to the user [26].

4.6 *SmarterCommerce* Case Study

Villegas developed a SmarterCommerce case study that took advantage of the personal web framework [26]. This case study illustrates how user-driven web integrations can be realized by allowing the user to manage their personal context sphere. In this case, the context domain is a personal shopping web experience.

There are several phases during which the user interacts with the shopping web sphere. First, the user must configure their personal web sphere by completing their web profile. This web profile includes personal information (e.g., city and preferred payment method), the registration of their personal web-enabled web sites, and the user's desired privacy settings. Registering with a site is a user agreement that provides the site with context information and acquires context information from the user’s interactions with web resources that are exposed by the website. Next, the user interacts with a registered web site and the site is provided with relevant context information about the user's situation and preferences. The web site exploits this context to offer products and services accordingly. Finally, the acquired context information is stored into the user's context sphere for future website interaction when the user is finished interacting with the website [26].

The SmarterCommerce system uses the *SmarterContext* framework that defines an extensible RDF-based taxonomy for context representation. RDF is utilized as a machine-readable context information specification that represents the way that the user interacts with web entities, or in other words the user's personal context sphere [24]. SmarterContext is composed of an RDF representation that is useful in characterizing the situation of web entities in the smarter Internet domain. Also, SmarterContext provides a

set of semantic RDF and OWL-lite rules and an extensible taxonomy for context representation and reasoning. Finally, SmarterContext defines the mechanism by which context information is exchanged between different sources such as context providers and consumers [26].

The idea of ontological representation for data and systems is not a new one to SmarterContext. In fact, ontology representation of the software maintenance domain has been considered by the academic community before [28]. However, this particular study was academic in nature and its main purpose was to answer the question of how the methods, tools, and skills of software maintenance work differ from those of development. Furthermore, the study did not address the dynamic nature of software updates. For these reasons, there is not much synergy between this academic study and the work for SmarterInfrastructure.

SmarterContext was designed as a modular and extensible architecture [24]. The modularity of the design adds to its ease of processing, maintenance, and evolution. Further, the ontology modularity adds a level of security as it facilitates the encapsulation of sensitive data [24]. The SmarterContext ontology also supports extensibility both vertically and horizontally. Vertical extensibility makes different problem domains tenable and horizontal extensibility enables the import of existing ontologies into the modules defined in the SmarterContext ontology [24].

The SmarterContext taxonomy has three different layers (cf. Figure 13) depending on whether it describes the smart Internet, the personal web, or a specific domain in the personal web. In the case of SmarterCommerce, the specific domain is shopping.

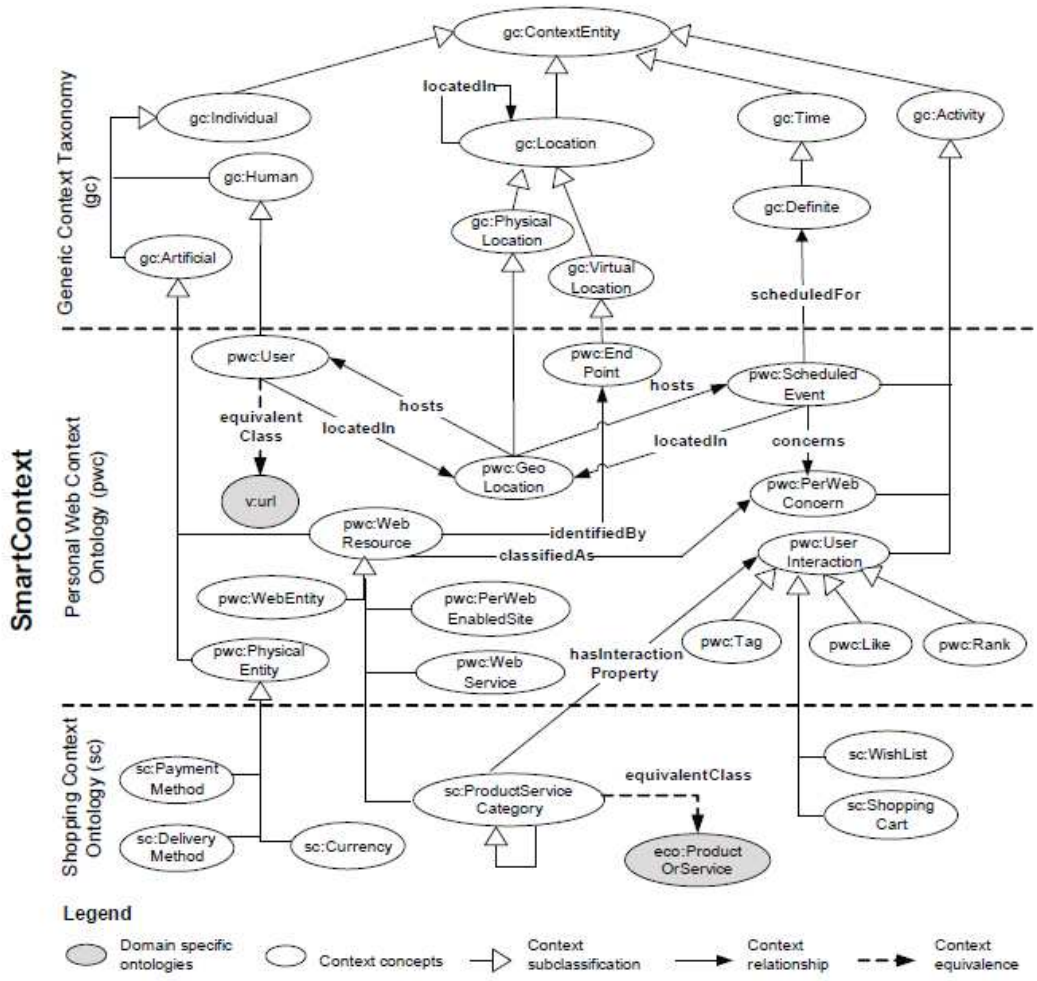

Figure 13: SmarterContext Taxonomy [26]

The general context (gc) taxonomy defines the foundational elements that represent context information in any problem domain of the smart Internet. The personal web context (pwc) ontology, the middle layer, describes the personal web vocabulary. Finally, the lowest layer, the shopping context (sc) ontology, describes the context types in the shopping problem of the personal web. The modular structure of the SmarterContext

framework enables the pwc and sc ontologies to vertically extend the gc and pwc ontologies, respectively.

SmarterContext also enables a methodology by which context reasoning can be used to characterize a user's situation and then provide web sites with applicable context information. Inferring relevant context facts is based on the vocabulary defined and the user-defined rules that exist at different levels of the ontology. For example, some context reasoning rules defined were *isNearTo* (physical proximity to), *locatedIn*, and *subClassOf* [26].

4.7 SmarterDeals System

SmarterDeals is a deal recommendation system that exploits a user's changing personal context information to deliver highly relevant shopping offers to users. The SmarterDeals system successfully extends the SmarterContext language by describing an expanded geographical location list linked data class and by defining a different deal recommendation system domain specific ontology [4, 37, 38]. The SmarterDeals system is an example of how the personal web framework can be extended to support a specific domain. This thesis illustrates how the personal web framework can be extended to support a different domain.

4.8 Summary

We provided an overview of background research leading up to the SmarterInfrastructure system described in the next chapter. We discussed seminal work in autonomic computing, context-aware systems and self-adaptive systems. We also

introduced the Personal Web Framework and the SmarterCommerce case study. Extending the SmarterContext framework is the key idea behind the SmarterInfrastructure framework. Finally, we presented a similar SmarterContext framework extension called the SmarterDeals system.

Chapter 5 *SmarterInfrastructure*

5.1 Introduction

In this chapter we extend the SmarterContext framework vertically to describe aspects of select system infrastructure evolution use cases. To do this, our *SmarterInfrastructure* framework takes advantage of the extensible nature of the SmarterContext linked data, ontologies, and rules to satisfy the evolving system infrastructure problem domain.

5.2 SmarterInfrastructure Use Cases

There are many different use cases that could be targeted within the evolution of system infrastructure domain. User context by definition can be one of three types: explicit, implicit, and inferred [4]. Explicit context refers to information that is manually gathered from a user. For example, users may be queried to enter their personal information or system settings into a configuration screen. Implicit context refers to information that is automatically gathered from or for a user. In this case, the system is able to query settings and observe information without explicitly interacting with the user. Inferred context refers to information that the system automatically generates based on input data and various algorithms. For example, inferred context can be information about the system load as it is determined based on the number of server requests by clients made over a specific time period.

Here we concentrate on one type of implicit context and one type of inferred context to illustrate how our SmarterInfrastructure framework works. The two key use cases are

detecting a specific user environment and notifying users that a particular distribution of an available library or patch is available. A variety of analytics can be run on the user environment once its information is captured.

5.2.1 Use Case 1: User Environment Awareness

A system can be architected to identify a user's environment. For example, the environmental information of interest includes details such as the operating system type, the web browser used, the currently installed software, the device (e.g., phone, tablet, desktop) type, and the preferred language of the user. The system can also track each time the user accesses system-specific features. The system can query the environmental information for a variety of reasons or simply generate user analytical data. This analytical information can identify the base user demographics for a system. For example, the system can determine how many users are using a particular operating system or browser. These usage statistics can guide further product development decisions. If only a small percentage of users access the system using the Internet Explorer browser, then less development effort can be placed into supporting IE in future product versions. The collation of these statistics is an example of inferred context.

To satisfy the user environment awareness use case within the SmarterInfrastructure framework, we specifically consider the parameters *OSType*, *deviceType*, *browserType*, *preferredLanguage*, and *userSystemAccessTime (*cf. Table 5 for a description of these context elements).

Note that the queried information adds context to the user's context sphere. The system pulls information from the user environment to populate their context sphere (as described earlier in this thesis in the description of the personal web).

The aforementioned user environment information exists in the HTTP_USER_AGENT header setting of all modern browsers. Of course, the HTTP_USER_AGENT is queried in this implementation to access this information but it is worth asking the question as to what benefit there is in storing this data in a user's personal context sphere. There are several reasons why this is the case and hence why this approach is valuable. The first reason is that the data will remain with the user as they move on to different systems or into different locations. This data persistence is more useful in some cases than others. For example, a user's language of choice will usually not change as they move between environments and a user's browser may not change as people tend to try to use their favourite browser even when they move between systems. Conversely, a user's operating system will likely change as they move from device to device. If the data already exists and is unchanged then there is no point in re-querying for it and hence this is a key area for system optimization. The second reason that this approach is advantageous is that there is opportunity for implementation simplification. In some cases, there is no reason to query for browser header information continuously (as mentioned above). Imagine the possibility of significant code simplification whereby conditional logic that changes based on the particular operating environment types is no longer required. Instead specific code can be delivered (e.g., to a client) that is tailored for a specific environment. The third reason this approach is useful is that there is no reason to statically perform analyses if the data already exists. For example, peak load times for a system can be

dynamically calculated by querying a user's access numbers and times that are stored in their personal context spheres. This information can be used to determine the times that are not good candidates for a system update. If the system is mostly accessed during weekdays then a weekend update would likely be more risk adverse.

5.2.2 Use Case 2: Library or Patch Updates

A system should be able to notify its user base when new libraries or patches are available. A notification can be followed by an updating process where the update can be performed in either a static or dynamic way. For example, a dynamic update would not require that the system be taken down or offline.

To satisfy the library or patch update use case within our SmarterInfrastructure framework, we specifically consider the parameters *userSystemAccessTime* which is also important for use case 1 above, *installedLibraryType*, *installedLibraryVersion*, *availableUpdatedLibraryType*, and *availableUpdatedLibraryVersion* (cf. Table 5 for a description for each of these context elements).

The notification information from the system pushes data into the user's context sphere. The user's context sphere is populated from the system. This *push* methodology of user context population is different than the aforementioned *pull* user environment use case and it is also a fundamental extension mechanism to the personal web system.

There is value in storing the notification information in the user's context sphere. Of course, once an update is available, a system can be updated right away, but often it is not convenient. As previously described, it could be a peak load time (e.g., during tax filing season) in which case any system updates would be deferred to a non-peak load time.

5.3 SmarterInfrastructure Channel Instance

It is possible to explore the SmarterInfrastructure possibilities closer now that key use cases and the underlying framework are developed. The idea is that specific channels or instances of the given use cases can be identified that would benefit from the system. Consider the following example of a technical sales professional who works for a tax software company located in Vancouver, British Columbia, Canada.

The sales professional normally accesses a production tax application from his office desktop Windows computer. He prefers to use the Firefox web browser and his native language is English. The tax application has been quickly spot tested to support different device types, operating systems, web browsers, and languages but the production system does not have all of the configurations possible running at the current time. This tax application has employed the SmarterInfrastructure architecture.

The sales representative learns of a sales opportunity in Hong Kong and finds himself on a plane to meet with the clients without much time to prepare for the meeting. Upon arrival in Hong Kong, the vendor becomes aware that the potential client requires Android tablet device support in a future tax application. The Android tablet is not a currently supported software configuration.

The sales representative has brought his own personal Android tablet with him on the trip and he or she tries to log into the tax application hosted back in Canada. The tax application is displayed on his device but he notices a few things that he does not feel comfortable showing to the client. His web browser language is set to English which should ideally be Cantonese when demonstrating the application in this part of the world and hence changes his language settings in his tablet web browser. Additionally, some of

the layouts on the screen are sized incorrectly and selected buttons are not active. First, the SmarterInfrastructure analyzes the sales representative dynamically changing environment. His personal context sphere is updated with the new device type, operating system, web browser, and language selection information. SmarterInfrastructure also records the system access time which is during the working day in Hong Kong and hence outside of normal working hours in Vancouver due to the different time zones. Next, the SmarterInfrastructure system proposes possible system updates. The server side of the tax application is updated with a new localization bundle for Cantonese. This update only proceeds after a risk analysis is performed where it is determined that the non-peak system load time is within the boundaries of the sales representative's system access time. Further, the next time he logs into the system he is presented with the information that a new Dojo Toolkit [33] library is available to support the Android tablet device better. After the sales representative chooses to update the client library then the rendering and other UI deficiencies are alleviated. Also, the tax application appears now in Cantonese. The sales representative now has a system that he is confident in demonstrating to the client. Moreover, he did not have to perform any manual system configuration personally.

The possible benefits for the user in this example are obvious. There are several benefits from the tax application system point of view as well. The application was updated during a non-peak hour so as to not jeopardize the system uptime service level agreement. Also, there was no manual maintenance required on the system and hence the system maintainers did not have extra work to do with this new system requirement requests.

Obviously this is a contrived example but it illustrates the power of the SmarterInfrastructure system from two key user groups: clients and system maintainers.

5.4 Summary

This chapter provided an overview of the SmarterInfrastructure system from the perspective of two specific use cases. These use cases are the pulling of information for user environment awareness and the pushing of information for patch or library updating. The motivation behind each use case was explained in detail including the description of a channel instance example that exercises the use cases.

Chapter 6 *SmarterInfrastructure* System

6.1 Introduction

In this chapter we extend the SmarterContext framework vertically to describe aspects of select system infrastructure evolution use cases. To do this, our *SmarterInfrastructure* framework takes advantage of the extensible nature of the SmarterContext linked data, ontologies, and rules to satisfy the evolving system infrastructure problem domain.

6.2 SmarterInfrastructure Framework

The SmarterInfrastructure framework builds on top of the ontologies that are defined in the SmarterContext framework. Further, the SmarterInfrastructure framework uses the same RDF schema as was used in the SmarterContext system. The SmarterContext framework ontologies (i.e., Generic Context: gc.owl and Personal Web Context: pwc.owl) are included in Figure 13. Note that the SmarterContext shopping ontology (i.e., Shopping Context: shopping.owl) is not applicable to the SmarterInfrastructure use case and thus is not included.

Table 5 describes the SmarterInfrastructure ontology where the application infrastructure ontology (i.e., infrastructure.owl) vertically extends the personal web content ontology. The application infrastructure ontology definition is fully elaborated in Appendix A.

Table 5: SmarterInfrastructure Framework Details

Context Type (Class)	Description	Supertype
gc:ContextEntity	Entity. The superclass of any context type.	owl:Thing
pwc:PWESite	Entity. Any web site compliant with SMARTERCONTEXT – eg, an b2c website	pwc:WebResource
pwc:User	Entity. Any person registered into SMARTERCONTEXT.	gc:HumanEntity
infrastructure:OSType	Entity. The client OS type – eg, Windows 7.	pwc:WebEntity
infrastructure:deviceType	Entity. The client device type – eg, iPad.	pwc:WebEntity
infrastructure:browserType	Entity. The client browser type – eg, Google Chrome 21.	pwc:WebEntity
infrastructure: userSystemAccessTime	Entity. The (server local) time that the user accessed the system – eg, Thursday, 03-May-2001 21:18:54 GMT.	pwc:WebEntity
infrastructure: installedLibraryType	Entity. A library type that is currently installed in the user's environment – eg, Firefox signature plugin.	pwc:WebEntity
infrastructure: installedLibraryVersion	Entity. A library type version that is currently installed in the user's environment – eg, 4.0.0.2.120.	pwc:WebEntity
infrastructure: availableUpdatedLibraryType	Entitty. A library type that is not currently installed but is available to use. - eg, Firefox signature plugin.	pwc:WebEntity
infrastructure: availableUpdatedLibraryVersion	Entity. A library type version that is available to use/install. - eg, 5.0.1.	pwc:WebEntity
pwc:hasIntegrated	Object property. Its value represents a context entity that has been integrated into a personal context sphere.	gc:locationRelationship
pwc:preferredLanguage	Data property. Its value defines the preferred language of a user.	gc:associationRelatinship
infrastructure:relatedLibraries	Object property. Denotes that two libraries are related to each other.	pwc:userInteraction
infrastructure:libraryTypeBindings	Object property. Denotes that a library type is bound to some other entity – eg, library type version.	gc:associationRelatinship

6.3 SmarterInfrastructure System Components

The SmarterInfrastructure system is an instantiation of the SmarterInfrastructure framework and this system can be used to satisfy the changing infrastructure requirements of a computer program. There are three main components (cf. see Figure 14) of the SmarterInfrastructure system: i) a Personal Context Sphere for the encapsulated contextual system information of a user; ii) an automated and dynamic *Analytic Engine* that collects and processes the system contextual information for a user; and iii) an automated and dynamic *Recommender Engine* that suggests contextual information to the user.

Similar to the SmarterContext system, a prerequisite to managing a user's contextual information in the SmarterInfrastructure system is the creation of the user's personal context sphere. First, a user must register the system as being compliant with the SmarterInfrastructure system. This action states that the application is instrumented to exchange context information with the SmarterInfrastructure infrastructure. The gathering of context information from context sources and its provisioning to web applications is controlled by the user. SmarterInfrastructure exchanges user and system context into their personal context sphere only with applications that are registered by the user. Users register their environment context by providing the URL of the web site. Moreover, a user can optionally initialize his or her context sphere with selected information. The values of the SmarterInfrastructure framework that the user can initialize are browserType, OSType, deviceType, and preferredLanguage. Note that the user is not required to manually enter these values as the system dynamically queries for

these using the Analytic Engine. The user may not be aware or technically knowledgeable enough to know his or her browser or OS type and version.

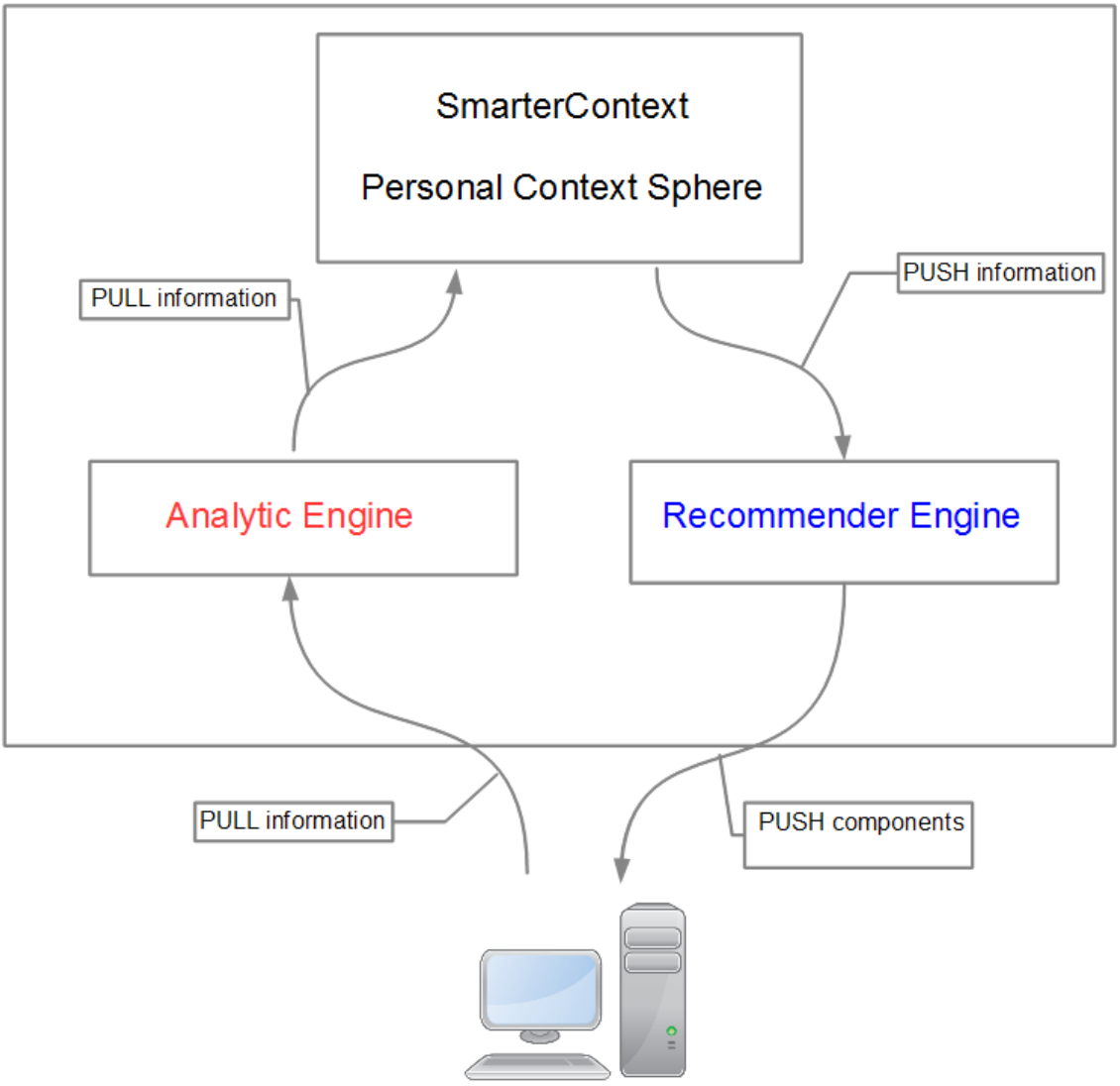

Figure 14: The SmarterInfrastructure System

6.3.1 SmarterInfrastructure Analytic Engine

The Analytic Engine of the SmarterInfrastructure system performs several functions such as querying for a user's contextual information. Note that the Analytic Engine executes the 'pull' phase of the SmarterInfrastructure information system. The pull phase collects user client information and it analyzes the HTTP_ACCEPT_HEADER for the *browserType*, *OSType*, *deviceType*, *preferredLanguage*, and the *userSystemAccessTime* settings. The first four values can also be optionally entered by the user statically during system initialization. The SmarterInfrastructure system can be statically configured by the administrator to query for these values dynamically at varying rates such as periodically, never, and always. For example, the system can query periodically to save computational time (and hence optimize performance) that would be required in querying for the values on each web request. Also, these values will not change at every web request. Similarly, it is optimal to not query at all for the values but instead simply rely on the static values that the user originally entered when the system was initiated. The userSystemAccessTime can also be configured to not be queried (for possible performance reasons) but this setting would change for every web request and so turning the query on or off for this system value would be the only option applicable. All of these values are entered into the user's personal context sphere.

Once these values are in the user's context sphere then there are two processes that query for the specific information. First, the browser, OS, device type, and preferredLanguage settings can be accessed to determine the configuration of the current user's environment. Based on this configuration, software updates that are (or are more) relevant to the specific environment can be distributed to the user's environment. Second,

an appropriate time window for such a software update can be deduced from analyzing the system access times. As previously described, if the system is being accessed very frequently in a peak load scenario then likely it would not be a good time to suggest a software update from a risk management and avoidance perspective.

6.3.2 SmarterInfrastructure Recommender Engine

The Recommender Engine of the SmarterInfrastructure system populates the user's personal context sphere with software update recommendations. The Recommender Engine executes the 'push' phase of the SmarterInfrastructure system. The push phase stores the currently installed system libraries and their associated versions. Also, the SmarterInfrastructure server updates the user's personal context sphere with the latest system updates that are available. It is the responsibility of the Recommender Engine to analyze the current software, identify the latest software that is available, and to recommend a time that the current software should be updated. Again, the system update time should likely be during an identified lower utilization period.

6.4 SmarterTax: A SmarterInfrastructure Proof of Concept

We designed and implemented the SmarterTax application to demonstrate the potential of the SmarterInfrastructure system. The SmarterInfrastructure system leverages the architecture of SmarterContext [26]. Figure 15 illustrates the SmarterInfrastructure components and their interdependencies in red. This proof of concept implementation does not facilitate the execution of any software updates. The mechanism of the software

system update is outside of the scope of this thesis although some common system update packages are described in detail in Chapter 3.

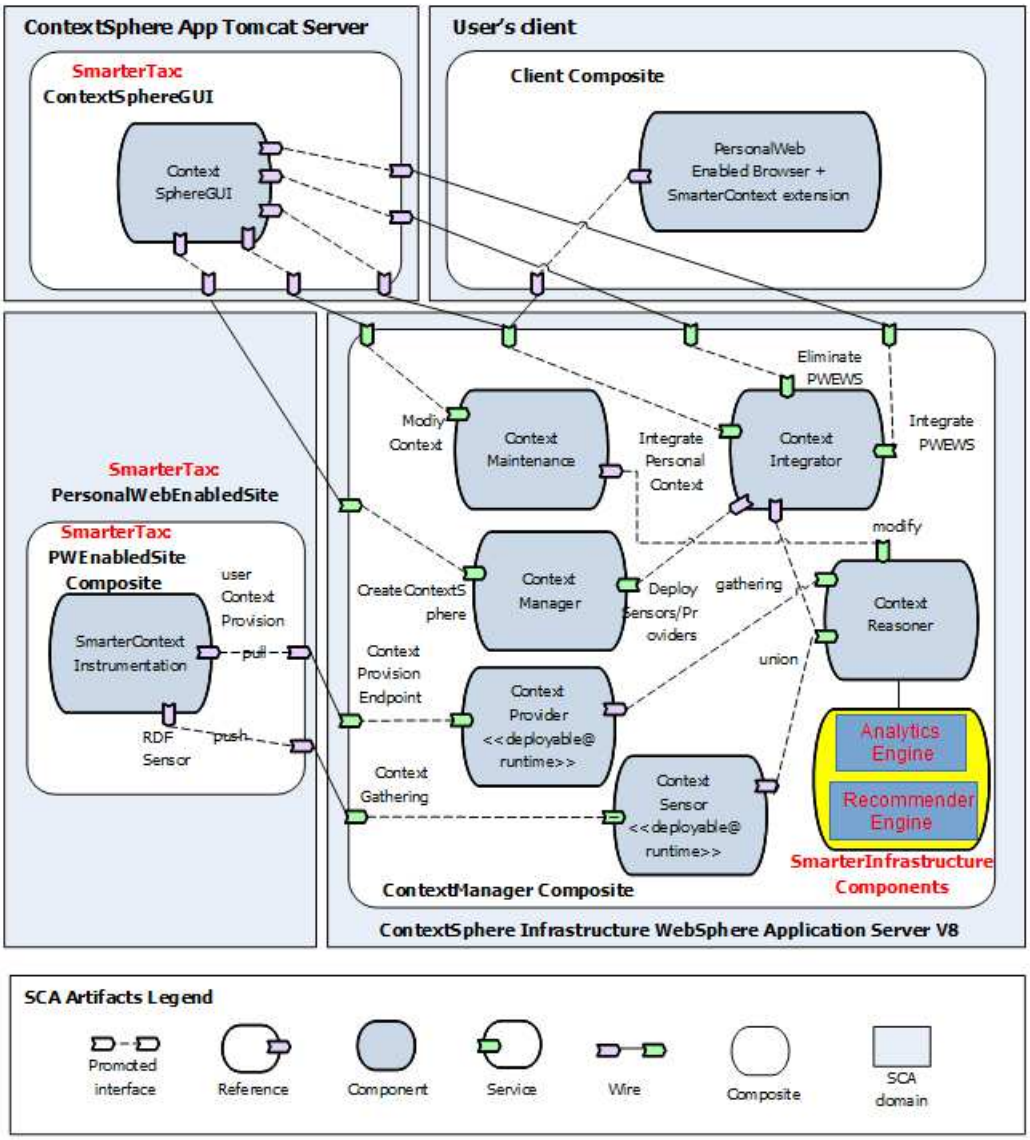

Figure 15: SmarterInfrastructure Components in SmarterContext System [26]

Context reasoning is defined by RDF markup, OWL-lite ontologies and user-defined rules. The intention is that both the grammar and the defined rules are extensible. In this

case, the rules are derived directly from SmarterContext but SmarterInfrastructure can be extended by adding custom rules applicable to a company's specific infrastructure update scenario.

The context inference engine is provided by Jena as in the SmarterContext framework. An example of an inference in SmarterInfrastructure is the link between installedLibraryType, availableUpdatedLibraryType, and relatedLibraries.

SmarterTax is the proof of concept implementation describing the SmarterInfrastructure channel instance in Section 5.5. Appendix B contains a sample RDF description (i.e., joetax.rdf) of the markup for the same channel instance. The system has three entry points: system configuration, the Analytic Engine, and the Recommender Engine. The following sections describe these three entry points from the point of view of the SmarterInfrastructure system (cf. Figure 15) and the SmarterTax application.

6.4.1 SmarterTax: System Configuration

Assuming that a user is properly registered with the SmarterTax system, he is asked to enter some optional system infrastructure values (cf. Figure 16). These values are used to create the user's personal context sphere in the SmarterInfrastructure system. The application does not become enabled with the SmarterInfrastructure system unless the user explicitly turns on the checkbox for running the application in SMART mode.

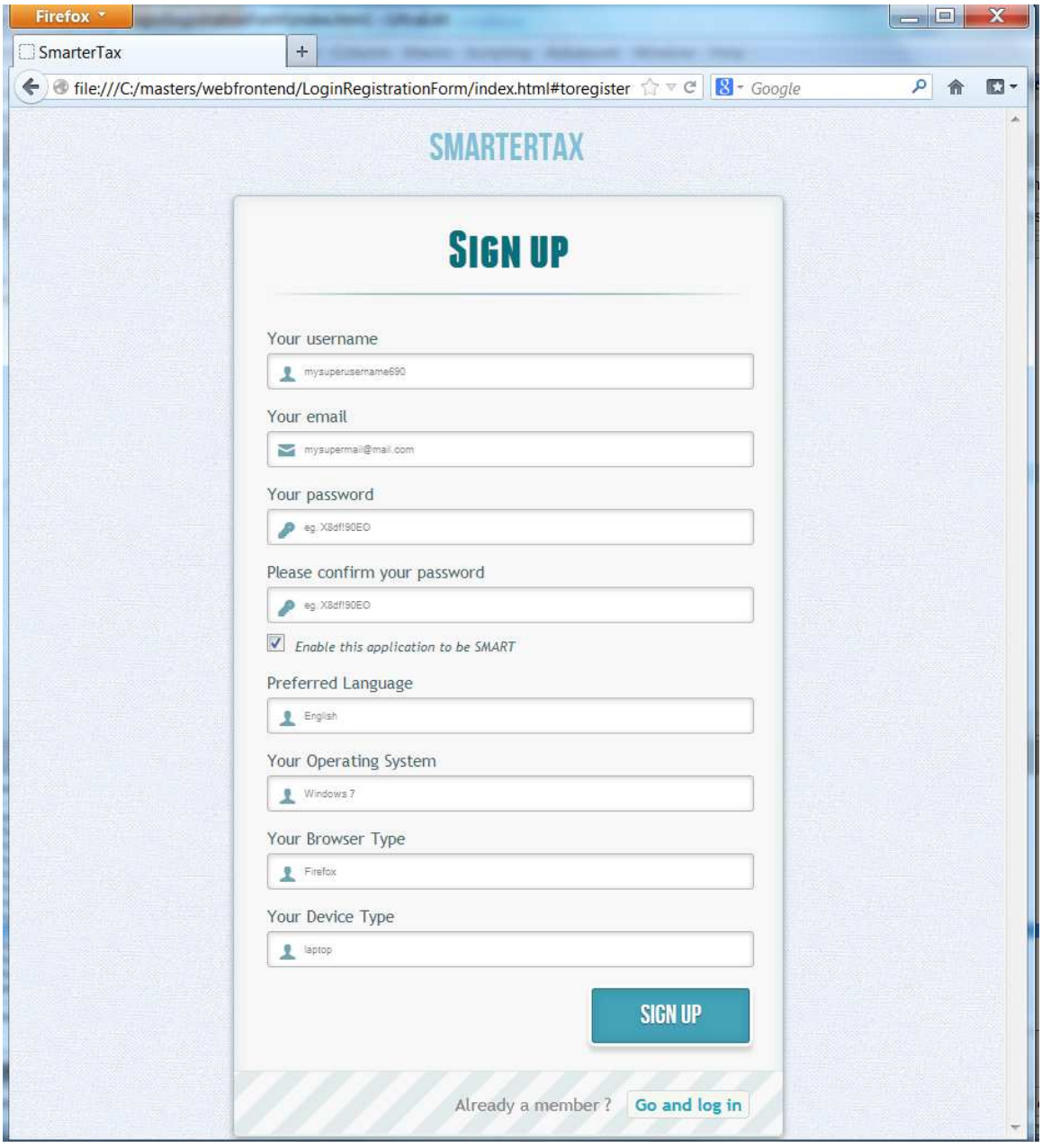

Figure 16: SmarterTax User Registration Screen

The SmarterTax administration component, represented as the ContextSphereGUI in Figure 15, submits web service calls to the SmarterInfrastructure system when the user clicks the button on the web page to sign up. First, the application is registered to the specific user with the SmarterInfrastructure system by invoking the *IntegratePwews* web

service call. Note that the user can later unregister the application by unclicking the same checkbox which will invoke the *EliminatePwews* web service. Next, the *CreateContextSphere* web service call is invoked with the user's configuration settings.

6.4.2 SmarterTax: Analytic Engine

SmarterTax uses the Analytic Engine to directly communicate with the ContextReasoner subsystem. HTTP requests come from the application into the server and the Analytic Engine obtains context sphere values from the HTTP_ACCEPT_HEADER. Note that the SmarterInfrastructure system also provides a different mechanism for gathering client information with the *ContextGathering* web service call. The Analytic Engine formats the HTTP request data into an RDF string and populates the users context sphere by invoking a *ModifyContext* web service call.

6.4.3 SmarterTax: Recommender Engine

SmarterTax uses the Recommender Engine to directly communicate with the ContextReasoner subsystem. The Recommender Engine is notified of a system update that is available and each registered user in the system has his or her context sphere updated with the information of a newly available library. The Recommender Engine uses the *ModifyContext* web service call to update the user's context sphere with this information. Next, the *ContextProvisioning* web service call is used to notify the SmarterTax user of the availability of a new library. SmarterTax prompts the user to perform a system update (cf. Figure 17).

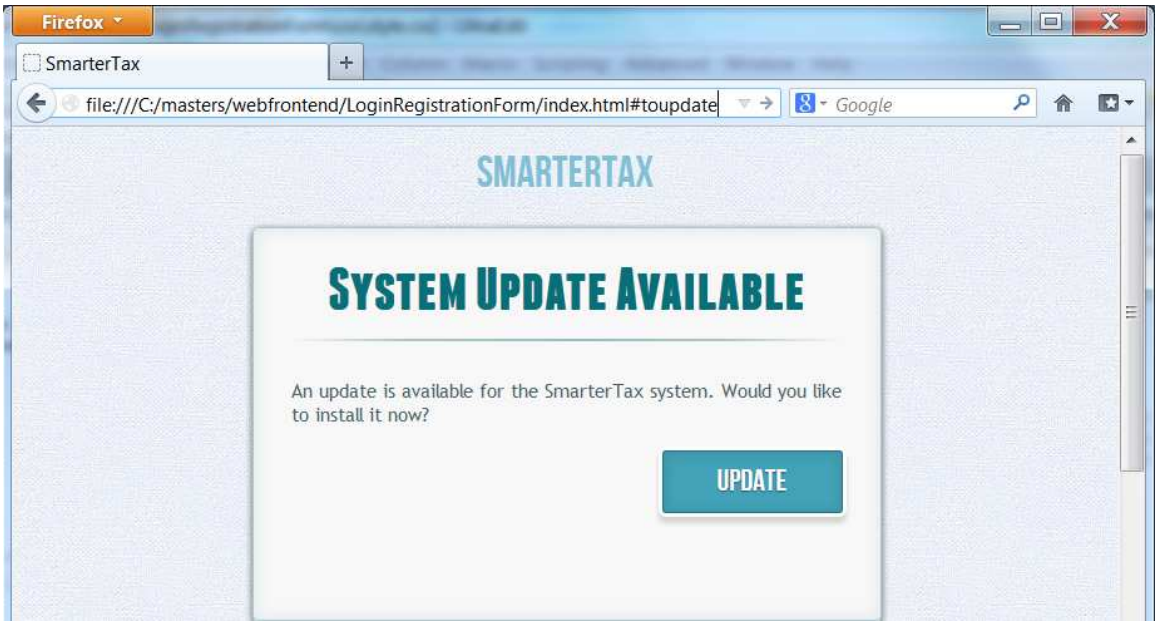

Figure 17: SmarterTax System Update Screen

6.5 Summary

This chapter provided an overview of the SmarterInfrastructure system. The ontological framework of the SmarterInfrastructure system is formalized in a way that extends the framework of the SmarterContext reference system. A theoretical system architecture for this framework is described with a Services Oriented Architecture paradigm. Finally, we detail a proof of concept implementation of the SmarterInfrastructure system.

Chapter 7 Conclusions and Future Work

The software industry is consistently faced with a difficult problem satisfying ever evolving system infrastructure requirements. Hardware devices, operating systems, web browsers, and third party software libraries are being updated ever more frequently. This problem involves intricate dependencies as new devices require new operating systems, new operating systems require new browsers, and in turn new browsers require new libraries. This continuous updating of related computer system components challenges current maintenance infrastructure.

7.1 Summary

This thesis develops the SmarterInfrastructure system that can be used to update system software dynamically. We motivate the development of such a system in the context of the current industrial system software development landscape. We designed this smart system using well-defined use cases from the infrastructure maintenance domain, an extensible ontology to support the use cases, and a reference architecture to implement the system. Key benefits of this approach are discussed. Our SmarterInfrastructure approach provides a framework for realizing the specific use cases of context aware automatic software component updates. Thus our framework is also extensible and hence new update requirements can readily be realized. SmarterInfrastructure is built on top of the extensible SmarterContext system.

7.2 Contributions

The main contributions of this thesis are:

- A research problem described from an industry-specific point of view. This required an analysis of the current development processes and issues with a real-world example. We described the currently available approaches and show how they do not sufficiently create viable solutions.
- Specific use cases identified within the system infrastructure requirement domain illustrate that there are real problems and there is significant value in solving them.
- The development of the SmarterInfrastructure framework based on vertically extending an existing research context framework (i.e., the SmarterContext commerce framework). The extensibility further proved the applicability of the SmarterContext framework to a different problem domain than commerce. We present a concrete proof point for the success of this framework concept in an industrial setting.
- Finally, we created a theoretical architecture to describe how the SmarterInfrastructure architecture satisfied a specific domain problem. We also provide proof of concept to illustrate the concepts working.

7.3 Future Work

The SmarterInfrastructure problem domain and system shows promise and there are several possible avenues for future work. The top candidates that were identified are:

- Incorporate a software updating system with the SmarterInfrastructure framework. The framework would serve to identify software updates that are necessary and then

the update system would perform the update operation. Such a system is possible as demonstrated in the proof of concept implemented but incorporating these two systems together was outside of the scope of this thesis work.

- Incorporate the SmarterInfrastructure framework into a commercially available software product. Measuring the frameworks real-world usability would be a valuable future case study. Implementing the proof of concept implementation into a production ready software product was also beyond the scope of this thesis work.

Bibliography

[1] D. Romero. Context as a Resource: A Service-Oriented Approach or Context-Awareness. PhD Thesis, Labratoire d'Informatique Fondamentale de Lille, July 2011.

[2] Adobe Update. *http://krebsonsecurity.com/2011/06/adobe-ships-security-patches-auto-update-feature/*. Retrieved Oct. 2012.

[3] Update Google Chrome. *http://support.google.com/chrome/bin/answer.py hl=en&answer=95414*. Retrieved Oct. 2012.

[4] S. Ebrahimi. SmarterDeals: A User-Centric Context-Aware Deal Recommendation System. M.Sc. Thesis, University of Victoria, Nov. 2012.

[5] Update Firefox to the latest version. *http://support.mozilla.org/en-US/kb/update-firefox-latest-version.* Retrieved Oct. 2013.

[6] RapidRelease/Calendar. *https://wiki.mozilla.org/RapidRelease/Calendar.* Retrieved Oct. 2012.

[7] RapidRelease Overview. *https://wiki.mozilla.org/RapidRelease#Overview.* Retrieved Oct. 2012.

[8] Extended Support Proposal. *https://wiki.mozilla.org/Enterprise/Firefox/ExtendedSupport:Proposal*. Retrieved Oct. 2012.

[9] Gartner Research. iPad and Beyond: The Future of the Tablet Market. *http://www.gartner.com/technology/research/ipad-media-tablet/future-of-tablet-market.jsp.* Retrieved Dec. 2012.

[10] P. Hu, J. Indulska and R. Robinson. An Automatic Context Management System for Pervasive Computing. *Proceedings of the Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM 2008).* pages 213-223, IEEE Press, 2008.

[11] Detailed system requirements for IBM Forms Server 8.0. *IBM, https://www-304.ibm.com/support/docview.wss?uid=swg27018408, June*, 2011.

[12] Installing, updating, and scripting installations for IBM Installation Manager. *http://http://pic.dhe.ibm.com/infocenter/install/v1r6/index.jsp? topic=/com.ibm.im.articles.doc/topics/SCRIPTING_IM.htm.* Retrieved Dec. 2012.

[13] The Tablet Begins Take Off As Sales Double in Eight Months. *http://www.ipsos-na.com/download/pr.aspx?id=11090.* Retrieved Dec. 2012.

[14] P. Kaminski, H. Müller and M. Litoiu. A Design for Adaptive Web Service Evolution. *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems (SEAMS 2006).* pages 86-92. ACM Press, 2006.

[15] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, pages 41-50. IEEE Press, 2003.

[16] R. Lemos, H. Giese, H.A. Müller and M. Shaw. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. *Software Engineering for Self-Adaptive Systems*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[17] Updating your software in OS X Mountain Lion. *http://support.apple.com/kb/HT1338.* Retrieved Nov. 2013.

[18] M.R. Vigder and J.C. Dean. Building Maintainable COTS Based Systems. *Proceedings of the International Conference on Software Maintenance.* pages 132-183. IEEE Press, 1998.

[19] J. Ng, M.H. Chignell and J.R. Cordy. The Smart Internet: Transforming the Web for the User. *Technical report, IBM Canada Center for Advanced Studies, Technical Report*, 2009.

[20] Tablet PC Market Forecast to Surpass Notebooks in 2013. *http://www.displaysearch.com/cps/rde/xchg/displaysearch/hs.xsl/130107_tablet pc_market_forecast_to_surpass_notebooks_in_2013.asp*. Retrieved *Jan. 2013.*

[21] Global desktop stats. *http://en.wikipedia.org/wiki/Usage_share_of_web_browsers.* Retrieved Jan. 2013.

[22] How Windows Update Keeps Itself Up-toDate. *http://blogs.technet.com/b/mu/archive/2007/09/13/how-windows-update-keeps-itself-up-to-date.aspx.* Retrieved Dec. 2012.

[23] N.M. Villegas and H.A. Müller. Managing Dynamic Context to Optimize Smart Interactions and Services. *The Smart Internet.* pages 289-318, Springer-Vertag, 2010.

[24] N.M. Villegas and H.A. Müller. Managing Dynamic Context to Enable User-Driven Web Integration in the Personal Web. *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research* (*CASCON 2010)*. pages 1-9. ACM Press, 2010.

[25] N.M. Villegas, H.A. Müller, G. Tamura, L. Duchien and R. Casallas. A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems. *Proceeding of the 6th International Symposium on Software Engineering for Self-Adaptive and Self-Managing Systems* (*SEAMS 2011)*. pages 80-89. ACM Press, 2011.

[26] N.M. Villegas, H.A. Müller, J.C. Munoz, A. Lau, J. Ng, C. Brealey. A Dynamic Context Management Infrastructure for Supporting User-driven Web Integration in the Personal Web. *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research* (*CASCON 2011)*. pages 200-214. ACM Press, 2011.

[27] 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. *Self-adaptive Systems.* pages 265-293. Springer-Vertag, 2012.

[28] B. Kitchenham, G. Travassos, A. von Mayrhauser, F. Niessink, N. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, H. Yang. Towards an Ontology of Software Maintenance. *Journal of Software Maintenance: Research and Practice*, pages 365-389, John Wiley & Sons, 1999.

[29] eBay. *http://www.ebay.com.* Retrieved Jan. 2013.

[30] Amazon. *http:www.amazon.com.* Retrieved Jan. 2013.

[31] Groupon. *http://www.groupon.com.* Retrieved Jan. 2013.

[32] Apple Genius. *http://appleinsider.com/articles/12/09/03/apple_turns_on_app_store_genius_recommendations_for_developers*. Retrieved Jan. 2013.

[33] Dojo Toolkit. *http://dojotoolkit.org/*. Retrieved Oct. 2012.

[34] Microsoft prepares Window Blue, to accelerate Windows release cycles? *http://www.hitechreview.com/it-products/pc/microsoft-prepares-windows-blue-to-accelerate-windows-release-cycles/40699/*. Retrieved Dec. 2012.

[35] Google Details Successes of its Chrome Release Process. *http://www.tomshardware.com/news/google-chrome-update-download-browser,14249.html.* Retrieved Oct. 2012.

[36] Rapid-release Firefox meets corporate backlash. *http://news.cnet.com/8301-30685_3-20074590-264/rapid-release-firefox-meets-corporate-backlash/*. Retrieved Oct. 2012.

[37] N. Villegas. Context Management and Self-Adaptivity for Sitution-Aware Smart Software Systems. PhD Thesis, University of Victoria, Feb. 2013.

[38] S. Ebrahimi, N.M. Villegas, H.A. Müller. SmarterDeals: A Context-aware Deal Recommendation System based on the SmarterContext Engine. *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research* (*CASCON 2012)*. pages 113-127. ACM Press, 2012.

Appendix A: infrastructure.owl

Additional Information: infrastructure.owl is the ontology that describes the vertical extension to the SmartContext commerce framework that was used in this thesis. This ontology provides the markup description used to satisfy the changing infrastructure requirement use cases desribed in Chapter 6.

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
   <!ENTITY gc "http://smartercontext.org/vocabularies/gc/v5.0/gc.owl#" >
   <!ENTITY pwc "http://smartercontext.org/vocabularies/pwc/v5.0/pwc.owl#" >
]>

<rdf:RDF xmlns="http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#"
    xml:base="http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:pwc="http://smartercontext.org/vocabularies/pwc/v5.0/pwc.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:gc="http://smartercontext.org/vocabularies/gc/v5.0/gc.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <owl:Ontology
rdf:about="http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl">
      <gc:creationdate>January 2013</gc:creationdate>
      <gc:author>Marcus Csaky</gc:author>
      <gc:copyright>2013, Marcus Csaky</gc:copyright>
      <owl:versionInfo>1.0 - January 2013</owl:versionInfo>
      <rdfs:comment>The Infrastructure ontology: defines the vocabulary for application infrastructure
                        requirements in the Personal Web</rdfs:comment>
      <owl:imports rdf:resource="http://smartercontext.org/vocabularies/pwc/v5.0/pwc.owl"/>
   </owl:Ontology>

   <!--
   ///////////////////////////////////////////////////////////////////////////////////
   //
   // Annotation properties
   //
   ///////////////////////////////////////////////////////////////////////////////////
    -->
```

```
    <!--
    ///////////////////////////////////////////////////////////////////////////////
    //
    // Datatypes
    //
    ///////////////////////////////////////////////////////////////////////////////
     -->

    <!--
    ///////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://smartercontext.org/vocabularies/pwc/v5.0/pwc.owl#userInteraction -->
    <owl:ObjectProperty rdf:about="&pwc;userInteraction"/>

            <!--
http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#libraryTypeBindings -->
            <owl:ObjectProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#libraryTypeBindings">
                    <rdf:type rdf:resource="&owl;SymmetricProperty"/>
                    <rdfs:comment rdf:datatype="&xsd;string">Denotes that a library type is bound to some
                                    other entity - eg., library type version</rdfs:comment>
        <rdfs:subPropertyOf rdf:resource="&gc;associationRelationship"/>
        <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#installedLibraryType"/>
        <rdfs:range rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#installedLibraryVersion"/>
            </owl:ObjectProperty>

            <!-- http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#relatedLibraries
-->
            <owl:ObjectProperty rdf:about="&pwc;relatedLibraries">
                    <rdfs:comment rdf:datatype="&xsd;string">Denotes that two libraries are related to each
                                    other as defined by a system user.</rdfs:comment>
                    <rdfs:range rdf:resource="&gc;IndividualContext"/>
                    <rdfs:domain rdf:resource="&pwc;User"/>
                    <rdfs:subPropertyOf rdf:resource="&pwc;userInteraction"/>
            </owl:ObjectProperty>

    <!-- http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#pullInfrastructureInfo
-->
    <owl:ObjectProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#pullInfrastructureInfo">
        <rdfs:comment rdf:datatype="&xsd;string">Its value represents the pull phase of the infrastructure
                                    transaction of the user.</rdfs:comment>
        <rdfs:subPropertyOf rdf:resource="&gc;functionalRelationship"/>
        <rdfs:domain rdf:resource="&pwc;User"/>
        <rdfs:range rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#InfrastructureInfo"/>
    </owl:ObjectProperty>
```

```
    <!-- http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#pushInfrastructureInfo
-->
    <owl:ObjectProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#pushInfrastructureInfo">
        <rdfs:comment rdf:datatype="&xsd;string">Its value represents the push phase of the infrastructure
                                                transaction of the user.</rdfs:comment>
        <rdfs:subPropertyOf rdf:resource="&gc;functionalRelationship"/>
        <rdfs:domain rdf:resource="&pwc;User"/>
        <rdfs:range rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#InfrastructureInfo"/>
    </owl:ObjectProperty>

    <!--
    ///////////////////////////////////////////////////////////////////////////////
    //
    // Data properties
    //
    ///////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#OSType -->
    <owl:DatatypeProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#OSType">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:comment rdf:datatype="&xsd;string">Denotes the client OS type - eg., Windows
                                                7.</rdfs:comment>
        <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v5.0/infrastructure.owl#InfrastructureInfo"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>

    <!-- http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#deviceType -->
    <owl:DatatypeProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#deviceType">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:comment rdf:datatype="&xsd;string">Denotes the client device type – eg.,
                                                iPad.</rdfs:comment>
        <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v5.0/infrastructure.owl#InfrastructureInfo"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>

    <!-- http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#browserType -->
    <owl:DatatypeProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#browserType">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:comment rdf:datatype="&xsd;string">Denotes the client browser type – eg., Google Chrome
                                                21.</rdfs:comment>
        <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v5.0/infrastructure.owl#InfrastructureInfo"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>

    <!--
```

```
http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#userSystemAccessTime -->
   <owl:DatatypeProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#userSystemAccessTime">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
      <rdfs:comment rdf:datatype="&xsd;string">Denotes the (server local) time that the user accessed the
                                    system – eg., Thursday, 03-May-2001 21:18:54 GMT.</rdfs:comment>
      <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v5.0/infrastructure.owl#InfrastructureInfo"/>
      <rdfs:range rdf:resource="&xsd;string"/>
   </owl:DatatypeProperty>

   <!-- http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#installedLibraryType
-->
   <owl:DatatypeProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#installedLibraryType">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
      <rdfs:comment rdf:datatype="&xsd;string">Denotes a library type that is currently installed in the
                                    user's environment – eg., Firefox signature plugin.</rdfs:comment>
      <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v5.0/infrastructure.owl#InfrastructureInfo"/>
      <rdfs:range rdf:resource="&xsd;string"/>
   </owl:DatatypeProperty>

   <!--
http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#installedLibraryVersion -->
   <owl:DatatypeProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#installedLibraryVersion">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
      <rdfs:comment rdf:datatype="&xsd;string">Denotes a library type version that is currently installed in
                                    the user's environment – eg., 4.0.0.2.120.</rdfs:comment>
      <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v5.0/infrastructure.owl#InfrastructureInfo"/>
      <rdfs:range rdf:resource="&xsd;string"/>
   </owl:DatatypeProperty>

   <!--
http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#availableUpdatedLibraryType
-->
   <owl:DatatypeProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#availableUpdatedLibraryTyp
e">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
      <rdfs:comment rdf:datatype="&xsd;string">Denotes a library type that is not currently installed but is
                                    available to use. - eg., Firefox signature plugin.</rdfs:comment>
      <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v5.0/infrastructure.owl#InfrastructureInfo"/>

      <rdfs:range rdf:resource="&xsd;string"/>
   </owl:DatatypeProperty>

   <!--
http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#availableUpdatedLibraryVersi
on -->
   <owl:DatatypeProperty rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#availableUpdatedLibraryVers
```

```
ion">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:comment rdf:datatype="&xsd;string">Denotes a library type version that is available to
                                        use/install. - eg., 5.0.1.</rdfs:comment>
        <rdfs:domain rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v5.0/infrastructure.owl#InfrastructureInfo"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://smartercontext.org/vocabularies/pwc/v5.0/pwc.owl#User -->
    <owl:Class rdf:about="&pwc;User"/>

    <!-- http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#InfrastructureInfo -->
    <owl:Class rdf:about=
"http://smartercontext.org/vocabularies/infrastructure/v1.0/infrastructure.owl#InfrastructureInfo">
        <rdfs:subClassOf rdf:resource="&pwc;PhysicalEntity"/>
        <rdfs:comment rdf:datatype="&xsd;string">Represents the infrastructure information of the
                                        user.</rdfs:comment>
    </owl:Class>

</rdf:RDF>
```

Appendix B: joetax.rdf

Additional Information: joetax.rdf is a sample RDF used for the channel instance use case that is described above in section 5.5.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:gc="http://smartercontext.org/vocabularies/gc/v1/gc.owl#"
   xmlns:pwc="http://smartercontext.org/vocabularies/pwc/v1/pwc.owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:shopping="http://smartercontext.org/vocabularies/infrastructure/v1/infrastructure.owl#"
   xmlns:owl="http://www.w3.org/2002/07/owl#" >

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/geo.rdf#Vancouver">
    <gc:geoLocationClassification rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">City</gc:geoLocationClassification>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/geo.rdf#Kowloon">
    <gc:geoLocationClassification rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">City</gc:geoLocationClassification>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/joetax.rdf#BusinessTripTime">
    <gc:endDateTime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2013-01-
                    12T06:09:10.616Z</gc:endDateTime>
    <gc:startDateTime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2013-01-
          17T06:09:10.616Z</gc:startDateTime>
    <rdf:type rdf:resource="http://smartercontext.org/vocabularies/gc/v1/gc.owl#DefiniteTime"/>
  </rdf:Description>

  <rdf:Description rdf:about=
"http://smartercontext.org/vocabularies/gc/v1/gc.owl#geoLocationClassification">
    <rdfs:subPropertyOf rdf:resource=
"http://smartercontext.org/vocabularies/gc/v1/gc.owl#geoLocationClassification"/>
    <owl:equivalentProperty rdf:resource=
"http://smartercontext.org/vocabularies/gc/v1/gc.owl#geoLocationClassification"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/joetax.rdf#BusinessTrip">
    <pwc:scheduledFor rdf:resource=
"http://smartercontext.org/vocabularies/rdf/joetax.rdf#BusinessTripTime"/>
    <gc:hostedBy rdf:resource="http://smartercontext.org/vocabularies/rdf/geo.rdf#Kowloon"/>
    <rdf:type rdf:resource="http://smartercontext.org/vocabularies/pwc/v1/pwc.owl#ScheduledEvent"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/geo.rdf#Canada">
    <gc:geoLocationClassification rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">Country</gc:geoLocationClassification>
```

```
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/geo.rdf#HongKong">
    <gc:geoLocationClassification rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">Country</gc:geoLocationClassification>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/taxapplicationsite/">
    <rdf:type rdf:resource="http://smartercontext.org/vocabularies/pwc/v1/pwc.owl#PWESite"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/jsmith.rdf#Home">
    <gc:locatedIn rdf:resource="http://smartercontext.org/vocabularies/rdf/geo.rdf#Vancouver"/>
    <gc:zipCode rdf:datatype="http://www.w3.org/2001/XMLSchema#string">V4M3G6</gc:zipCode>
    <gc:address rdf:datatype="http://www.w3.org/2001/XMLSchema#string">305-54th Street</gc:address>
    <gc:geoLocationClassification rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">Place</gc:geoLocationClassification>
    <rdf:type rdf:resource="http://smartercontext.org/vocabularies/gc/v1/gc.owl#GeoLocation"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/geo.rdf#BritishColumbia">
    <gc:locatedIn rdf:resource="http://smartercontext.org/vocabularies/rdf/geo.rdf#Canada"/>
    <gc:geoLocationClassification rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">Region</gc:geoLocationClassification>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/pwc/v1/pwc.owl#hasIntegrated">
    <rdfs:subPropertyOf rdf:resource=
"http://smartercontext.org/vocabularies/pwc/v1/pwc.owl#hasIntegrated"/>
    <owl:equivalentProperty rdf:resource=
"http://smartercontext.org/vocabularies/pwc/v1/pwc.owl#hasIntegrated"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/jsmith.rdf#pullInfrastructureInfo">
    <infrastructure:OSType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Windows
7</infrastructure:OSType> data
    <infrastructure:deviceType rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">laptop</infrastructure:deviceType>
    <infrastructure:browserType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">John A.
                                        Smith</infrastructure:browserType>
    <infrastructure:userSystemAccessTime
rdf:string="http://www.w3.org/2001/XMLSchema#string">03-
          March-2013 21:18:54</infrastructure:userSystemAccessTime>
    <infrastructure:installedLibraryType rdf:datatype=
"http://www.w3.org/2001/XMLSchema#int">Dojo</infrastructure:installedLibraryType>
    <infrastructure:installedLibraryVersion rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">4.7</infrastructure:installedLibraryVersion>
    <infrastructure:userSystemAccessTime rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">03-March-2013
                                        21:18:54</infrastructure:userSystemAccessTime>
    <pwc:preferredLanguage rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">English</pwc:preferredLanguage>
    <rdf:type rdf:resource=
"http://smartercontext.org/vocabularies/infrastructure/v1/shopping.owl#InfrastructureInfo"/>
  </rdf:Description>
```

```
  <rdf:Description rdf:about=
"http://smartercontext.org/vocabularies/rdf/joetax.rdf#pushInfrastructureInfo"> object-
>pullInfrastructureInfo/pushInfrastructureInfo
    <infrastructure:userSystemAccessTime rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">03-March-2013
                                21:18:54</infrastructure:userSystemAccessTime>
    <infrastructure:availableUpdatedLibraryType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dojo</infrastructure:availableUpdatedLibrar
yType>
    <infrastructure:availableUpdatedLibraryVersion rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">4.8</infrastructure:availableUpdatedLibraryVersion>
    <rdf:type rdf:resource=
"http://smartercontext.org/vocabularies/shopping/v1/shopping.owl#InfrastructureInfo"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/rdf/joetax.rdf#joetax">
    <pwc:emailAccount rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string">joetax@mail.com</pwc:emailAccount>
    <rdf:type rdf:resource="http://smartercontext.org/vocabularies/pwc/v1/pwc.owl#User"/>
    <pwc:hasIntegrated rdf:resource="http://smartercontext.org/vocabularies/rdf/joetax.rdf#BusinessTrip"/>
    <pwc:lastName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Tax</pwc:lastName>
    <infrastructure:pullInfrastructureInfo rdf:resource=
"http://smartercontext.org/vocabularies/rdf/joetax.rdf#pullInfrastructureInfo"/>
    <infrastructure:pushInfrastructureInfo rdf:resource=
"http://smartercontext.org/vocabularies/rdf/joetax.rdf#pushInfrastructureInfo"/>
    <pwc:hasIntegrated rdf:resource="http://smartercontext.org/taxapplicationsite/"/>
    <pwc:hasIntegrated rdf:resource="http://smartercontext.org/vocabularies/rdf/jsmith.rdf#BusinessTrip"/>
    <pwc:givenName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Joe</pwc:givenName>
    <pwc:hasGender rdf:datatype="http://www.w3.org/2001/XMLSchema#string">male</pwc:hasGender>
    <gc:locatedIn rdf:resource="http://smartercontext.org/vocabularies/rdf/geo.rdf#Vancouver"/>
    <pwc:preferredLocation rdf:resource="http://smartercontext.org/vocabularies/rdf/joetax.rdf#Home"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://smartercontext.org/vocabularies/gc/v1/gc.owl#locatedIn">
    <rdfs:subPropertyOf rdf:resource="http://smartercontext.org/vocabularies/gc/v1/gc.owl#locatedIn"/>
    <owl:equivalentProperty rdf:resource="http://smartercontext.org/vocabularies/gc/v1/gc.owl#locatedIn"/>
  </rdf:Description>
</rdf:RDF>
```