Embodied Context Models and an Approach to Re-using Context-Aware Middleware

by

David Dahlem
B.A., University of Victoria, 1995

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

MASTER OF SCIENCE

in the Department of Computer Science

University of Victoria

Supervisory Committee

Embodied Context Models and an Approach to Re-using Context-Aware Middleware

by

David Dahlem
B.A., University of Victoria, 1995

Supervisory Committee

Dr. Jens H. Weber, (Department of Computer Science)
Supervisor

Dr. Yvonne Coady, (Department of Computer Science)
Departmental Member

Dr. Issa Traore, (Department of Electrical Engineering)
Outside Member

Abstract

Supervisory Committee

Dr. Jens H. Weber, (Department of Computer Science)
Supervisor

Dr. Yvonne Coady, (Department of Computer Science)
Departmental Member

Dr. Issa Traore, (Department of Electrical Engineering)
Outside Member

This thesis develops a generalized approach for decoupling how a context model is defined and executed from how context-aware data is acquired and managed within a given middleware system. Decoupling the model specification from the data will likely provide more avenues of context-aware investigations due to an increased flexibility in the choice of a middleware system for handling context data. We provide a detailed description of the approach developed for this decoupling task, called Inspect, Adapt, Model, and Integrate (IAMI). By engaging the steps we show that a context model need not be specifically tied to a given context-aware middleware. This successful decoupling will likely add to the future development of context-aware systems by allowing for researchers to build upon existing frameworks as opposed to repeatedly engaging in ground-up development. Moreover, we submit that this decoupling is important in that the number of possible ways of representing and expressing a context model is potentially infinite, but the choice of context-aware middleware systems is limited.

Table of Contents

List of Tables

List of Figures

Acknowledgements

To my supervisor, Dr. Jens Weber: thank you for your support, encouragement, intelligence, and patience (beyond the call of duty).

To my friend and life partner, Megan O'Connell: thank you for your love and support through this whole process. You helped me to keep things in perspective.

To my colleagues in the PPCI Lab: thank you for your friendship and intellectual contributions to this research, especially those affiliated with the original COWSPOTS project: Luay Kawasme, Yury Bychkov, and Paul Crawford.

To Tim Draude of Doepker Industries Ltd.: thank you for giving me flexible time hours to help me finish this thesis work.

To Mom, Dad, Flo, and George: thank you for all your encouragement and babysitting help. A special thanks to Mom who had the intestinal fortitude to read a draft of this thesis from front to back, in spite of the fact she dislikes almost everything to do with computers.

To my sons, Mattheus O'Connell Dahlem and Alexander O'Connell Dahlem: thank you for giving me immeasurable joy.

Dedication

For Megan

1 Introduction

With each passing year, computing devices are becoming more numerous within our living and working environments. It is commonplace for televisions, microwaves, thermostats, phones, elevators, radios, and many other everyday objects to possess embedded computer chips. Given the rate of advances in computing technology, it seems inevitable computing technology will become completely enmeshed or integrated into the human world, to the point perhaps where all objects within a given human environment will be subject to some degree of computation. Mark Weiser believed the 'inevitable' infiltration of computing technology into the human world will lead us to the era of *ubiquitous computing* (ubicomp) [3] An important characteristic of ubicomp, according to Weiser, is how computing technology will seem to "disappear" or "vanish into the background" of human consciousness, the consequence of computers becoming more aware of and adaptive to the "natural human environment." [3].

The road leading to Weiser's vision for ubicomp has proven to be slow and uneven [4]. Although, according to [1, 4, 5], most barriers to reaching ubicomp as identified by Weiser have been overcome, many new and perhaps unexpected challenges have been encountered during the journey. Many of these challenges are related to objectives of context-aware computing; specifically, in the context-aware research goal of making computers more aware and reactive to human users as they engage in every day activities. The relative perceptiveness of context-aware computing systems is largely dependent on the context model utilized within a given context-aware system.

In this thesis we propose to mitigate one of the many identified difficulties with implementing context-aware applications: the lack of sharing software components across the many context-aware research groups. Based on our review of the context-aware research literature, most context-aware research groups opt to build entirely new context frameworks rather than build on frameworks developed by others. We introduce a generalized approach for decoupling how a context model is specified from how context-aware data is acquired and managed within a given middleware system. We hypothesize that this decoupling will provide more avenues of context-aware

investigations due to an increased flexibility in the choice of a middleware system for handling context data. We provide a detailed description of the approach developed for this decoupling task, called Inspect, Adapt, Model, and Integrate (IAMI). In pursuit of demonstrating how the IAMI approach is utilized, we provide an accounting of the following steps undertaken in this thesis:

1. We created a software infrastructure for detecting human activity processes, a model we refer to as *embodied processes* (EmPro).
2. We implemented the IAMI approach targeting two separate context-aware middleware systems, the Context Toolkit [6] and the Java Context-Aware Framework (JCAF) [7].

By engaging the steps outlined above we show that a context model need not be specifically tied to a given context-aware middleware. We submit that this is important in that the number of possible ways of representing and expressing a context model is potentially infinite, but the choice of context-aware middleware systems is limited. The ability to infuse a context-aware middleware with new context perceptions provides much more flexibility than what currently exists, and hopefully will promote the exploration of context-aware research and the creation of new context-aware system.

1.1 Thesis Contributions

The following is a list of thesis contributions:

- Definition of context-aware computing: we provide a reworking of Dey's definition of context [6]. Dey's definition of context is currently the most widely cited and accepted definition within the context-aware research community. We rework the Dey definition to include the concept of an *embodied context model*, which is a module of contextual knowledge. We argue that the inclusion of the embodied context model concept provides a more comprehensive and flexible definition of context.
- An introduction to the concept of an Embodied Process (EmPro): a concept that is currently lacking representation in existing context models.
- A UML 2.0 profile for modeling Embodied Processes within UML 2.0/XMI 2.1 standards compliant UML modeling tools.

- A software library, collectively called Context Aware Embodied Processes (ContEmPro), which can be used for executing EmPro models.
- An approach to integrating context models into existing context-aware systems, called Inspection, Adaptation, Model, and Integration (IAMI).

1.2 Thesis Structure

In Chapter 2 we seek a foundation for the research field of context-aware computing. We begin with a description of Mark Weiser vision for ubicomp and how the research realm of context-aware computing fits into Weiser's vision. We follow with an in-depth discussion of the barriers and challenges inherent in the Weiser's ubicomp vision; since context-aware computing is a sub-research field of ubicomp, context-aware computing is subject to the same barriers and challenges. Then, we proceed to discuss context-aware computing, a research area without a clearly defined shape. We provide a discussion of the characteristics of a typical context-aware system, present some of the offerings for defining the word "context," and examine some of the challenges specific to deploying context-aware computing systems. To explore the level of activity/interest in the research field of context-aware computing, we present statistics on the number of publications on the subject on context-aware computing. In the final part of Chapter 2, we provide a definition for the word "context" as it relates to context-aware computing. This new definition builds on a widely accepted, previously offered definition for "context" and, we assert, provides added meaning to the phrase "context entity" by including the concept of an *embodied context model* (ECM). An ECM is a module of context knowledge and behaviour corresponding with, using software engineering terminology, a *concern* of the context-aware system.

In Chapter 3 we introduce the concept of an *embodied process* (EmPro), a representation of process/activity flows within ubiquitous context-aware systems. An EmPro is a particular type of ECM. We discuss how an EmPro is a representation of information within a ubiquitous context-aware system such as social convention, a habit, a ritual, or a protocol. We describe the characteristics of an EmPro and discuss our approach to modeling this concept for context-aware system applications. Further, we discuss our implementation of the Context-Aware Embodied Process software, which is designed to instantiate EmPro models.

In Chapter 4 we discuss and demonstrate our approach for integrating a foreign ECM into a specific context-aware system. We refer to this approach as the Inspect, Adapt, Model, and Integrate process, (IAMI). To demonstrate the IAMI approach we discuss how we integrate the ContEmPro software discussed in the previous chapter into two different context-aware systems, the Context Toolkit [6] and the Java Context Aware Framework (JCAF) [7]. Neither the Context Framework nor the JCAF provide a context model specification medium that can be easily leveraged to produce a model similar to EmPro.

In Chapter 5 we evaluate the contributions made in this thesis and we follow with a discussion of the contributions and results of the thesis in Chapter 6.

2 The Foundations of Context-aware Computing

2.1 Introduction

The context-aware computing research field began to emerge in the early 1990's. Since that time we have seen significant progress and academic activity in the context-aware research field, although some may argue this progress has been slow. It is not always clear what constitutes a context aware system, what differentiates it from a non-context-aware computing system, and what issues context-aware computing is supposed to address. The primary objective of this chapter is to clarify some of these issues and to establish a foundation for the context-aware computing research field.

We begin this discussion in Section 2.2, with a description of Mark Weiser's vision for ubiquitous computing (ubicomp), a research field which encompasses the context-aware computing research field. Inclusive in this description is a discussion about the relationship between ubicomp and context-aware computing. In Section 2.3 we identify some of the challenges which have thus far impeded the realization of Weiser's vision for ubiquitous computing, and how these challenges have affected context-aware computing. In Section 2.4 we provide an in-depth examination of the characteristics of context-aware computing, including a discussion on how the word "context" has been defined. We identify in Section 2.5 some of the unique challenges to context-aware computing. In Section 2.6 we attempt to quantify the level of research interest in context-aware computing. In Section 2.7 provide a reworked definition for the word "context" with respect to context-aware computing, a definition that builds on the widely accepted definition of context offered by Dey [6]. Finally, we summarize the chapter.

2.2 Ubiquitous Computing and the Need for Context-Awareness

2.2.1 Mark Weiser: The Father of Ubiquitous Computing

Mark Weiser famously wrote, "the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they're indistinguishable from it." [3] Weiser articulated a vision for a new era of computing called ubiquitous computing (ubicomp). Weiser predicted that the era of ubicomp would be upon us within twenty years of his writing in 1991 and would take hold as a result of

two technical factors. First, computing devices were getting smaller, more powerful, and more economical, at a rate on par with Moore's Law. Second, he expected advances in networking and wireless technologies would enable integrated communication between various computational devices. These two factors together, he argued, would contribute to a new era of computing, where our natural human environments – our workspaces, living rooms, kitchens, etc. – will become invisibly enhanced by hundreds of interconnected computing devices. As of 2008, it is still too early to evaluate the predictive value of Weiser's vision. Nevertheless, Weiser's vision has been influential in deciding the directions of various computer science and engineering research initiatives.

The idea that computers in the future will become smaller, cheaper, more numerous, and better connected is not what distinguishes Weiser's vision as revolutionary. Rather, it was Weiser's assertion that a new relationship between human and computer is on the horizon, where computers become more human-centric, and humans become less computer-centric. The prevailing human-computer interaction will cease to be defined by the desktop computer, which demands the focus of attention of the user, but by computers embedded in everyday objects. An important characteristic of the ubicomp era, according to Weiser, will be the "disappearance" of computers from the consciousness of the human user. In a literal sense, computers will seem to disappear because they will be small and embedded in everyday objects, and thus not necessarily visible to users. Weiser suggested that computers will also disappear in a metaphorical sense, by "invisibly enhancing the world that already exists," by adapting to changes in the human environment to facilitate the activities of users.

Weiser offered an illustrative example of how a typical user would interact with a system of diverse interconnected network of ubiquitous devices. Although he did not offer an explicit description of what would be required to produce a system that is capable of orchestrating the collective behaviours of these devices, he did imply that location-awareness is part of the solution: "if a computer merely knows what room it is in, it can adapt its behaviour in significant ways." Weiser continued: "No revolution in artificial intelligence is needed—just the proper imbedding of computers into the everyday world…knowing where people are can yield complex dividends…." He and his colleagues sought to leverage the location-specific nature of many human activities. For

example, a meeting room is a place where people convene for a discussion; a hallway is a place that allows people to move from room to room; an office is a place where people engage in work activities. Although Weiser vaguely outlines the significance and advantages of location sensitivity in ubiquitous systems, it is clear that he was describing the necessity of having some sort of awareness of context (in the form of location information) as a means for promoting "invisibility" in ubiquitous systems. Invisibility is possible when a system of interconnected devices is "aware" of the task or activity of the user, and cohesively adjusts the system behaviour based on this awareness to help facilitate the user's activity. Awareness of context provides a measure of collective purpose among the distributed devices upon which system behaviour modification is based.

Although Weiser did not define this approach as "context-aware", it is easy to see the roots of context-aware computing are firmly implanted in Weiser's seminal article on ubicomp. It was not until three years later that the research area of context-aware was semi-formally defined by Schilit and Theimer [8]; but context-aware research has been profoundly influenced by Weiser's work as evident in the fact that most context-aware research projects rely on location as the main source of user context [6].

2.2.2 Related Research Fields

A significant number of ubicomp-related research fields have appeared since Weiser's famous ubicomp article. The most prominent of these related research fields are: Pervasive Computing [1], Invisible Computing [9], Sentient Computing [10], Ambient Intelligence [11], Smart Spaces (e.g.,[12]), Disappearing Computer (see [13]), Wearable Computing [14], and Augmented Reality (see [15]), Everyware [16], Human-centric Computing [17], and others. It is not always clear what differentiates one ubicomp-related research topic from another. For example, pervasive computing is often used as a synonym for ubicomp, and sentient computing is similar to context-aware computing in objective and approach.

We find it useful to categorize the ubicomp-related research fields from the perspective of two themes discussed in Weiser's famous article. The first theme focuses on the technical challenges required to engender or stimulate the conditions necessary for ubicomp from the perspective of hardware, networking, and software technologies. We

assign this theme the generic name “Technology Focused”. On the other hand, Weiser also emphasized the human-centric nature of ubicomp computing, that is “machines that fit the human environment, instead of forcing humans to enter theirs.” We refer to this second theme as “HCI Focused” (or “Human Computer Interaction Focused”) because it examines how humans are affected by, and interact with, computer technology from a human-centric or usability perspective. The two themes presented clearly do not represent polar opposite extremes nor are mutually exclusive since there is a great deal of overlap between HCI and technology. Weiser might have argued that ubicomp is the realization of a complete overlap of computing technology and human concerns.

In Figure 1 we list a number of ubicomp-related research fields and establish a Cartesian coordinate for each in accordance with our subjective estimate of how well that particular research field fits within one of the two themes described above. The demarcations between the encircled representations of the various ubicomp research groups are fuzzy and the actual position of that group on the chart is an approximation. An HCI-focused ubicomp research field (e.g., Invisible Computing) will appear near the top of the Y-axis whereas a ubicomp research field that focuses primarily on efficiency or feasibility (e.g., mobile computing) will appear on the right of the X-axis. We included the mobile computing and distributed computing research fields in the chart although they are not ubicomp-related *per se*. But both mobile computing and distributed computing are generally acknowledged to be foundational technologies for ubicomp [1], and show the technology focus of these research fields relative to other ubicomp research fields.

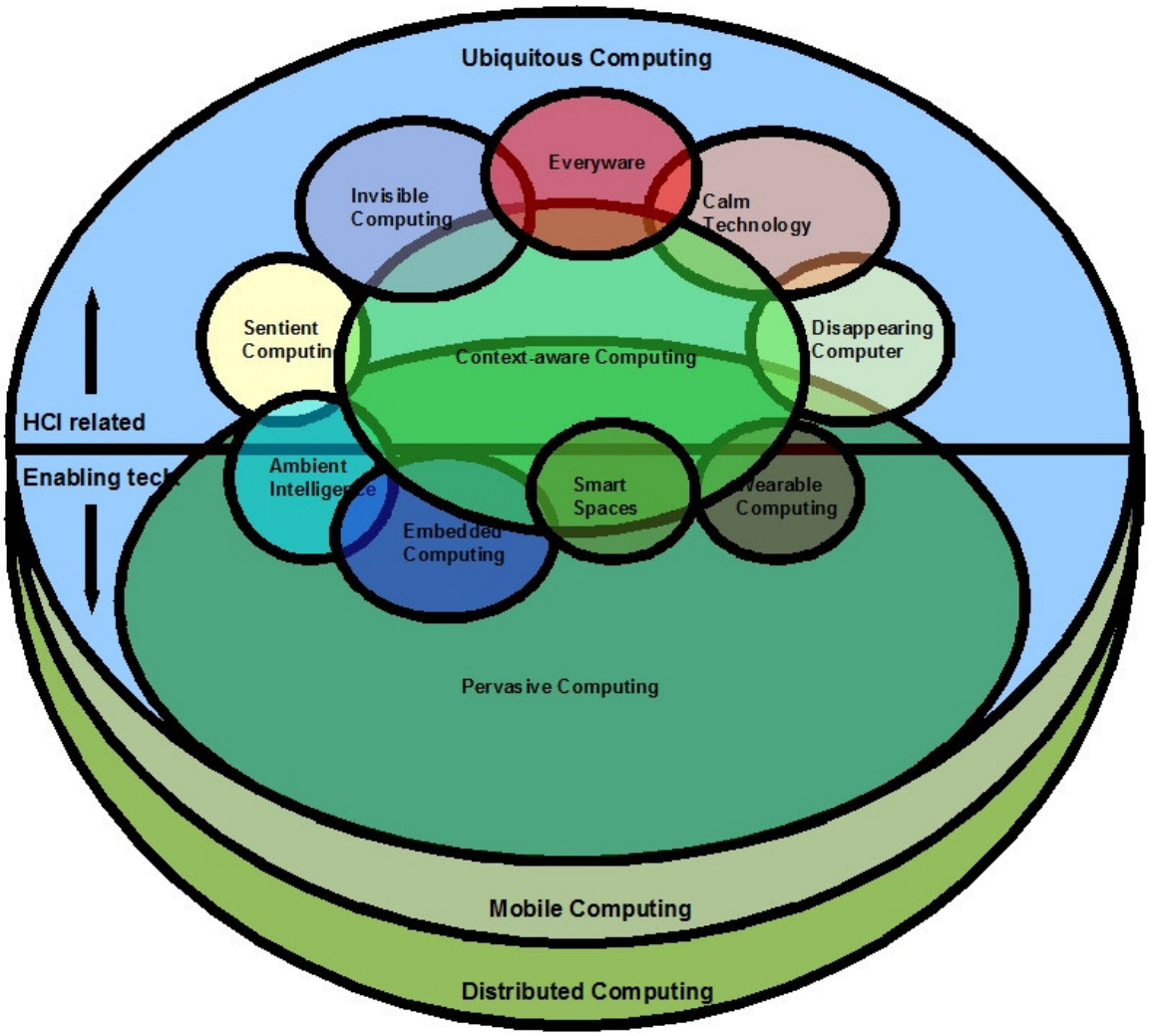

Figure 1: Ubiquitous Computing and Related Research Fields

Of particular interest to this thesis, we place the context-aware computing sphere in equal parts "HCI Focused" and "Technology Focused". Although most context-aware computing research conducted to date has focused on the technical foundation of establishing a context-aware system (e.g., creating electronic sensors, location determinations, context middleware, etc.), in general the ultimate goal of context-aware computing is to make computing more aware and responsive to human activities. There are exceptions to this assertion (e.g., context-aware, self-adapting networks [18]), but these exceptions refer to research that, we argue, fits more comfortably in networking and mobile computer research fields.

2.3 The Challenges of Implementing Ubicomp Systems

2.3.1 Challenges Identified by Weiser

Weiser identified a number of barriers which he believed would impede progress towards ubicomp. In the time that has passed since the article was published, many of these barriers have been overcome, but others have not. The following is a listing of Weiser's concerns as presented in his article, followed by a brief discussion regarding whether or not the barrier exists today (as of early 2008).

- "*Software systems today barely take any advantage of the computer network*." It is unlikely Weiser would see this as a barrier today given the universal acceptance of the TCP/IP protocol and the exponential growth of the internet since the mid 1990's.
- "*Trends in 'distributed computing' are to make networks appear like disks, memory, or other non-networked devices, rather than to exploit the unique capabilities of physical dispersion*." Distributed computing has become more than just about "disks [and] memory." For example, the World Wide Web (WWW) is a distributed system that specialises in disseminating human-readable documents.
- "*Today's operating systems, like DOS and Unix, assume a relatively fixed configuration of hardware and software at their core… but in an embodied virtuality, local devices come and go, and depend upon the room and the people in it*." Today's operating systems are far more flexible in terms allowing on-the-fly hardware (e.g., plug-and-play) and software configurations (e.g., the Open Services Gateway initiative (OSGi) Alliance [19]). Further, the foundations for mobile ad-hoc networking are now well established to handle unstable, highly fluid network communication configurations.
- "*Today's window systems, like Windows 3.0 and the X Window System, assume a fixed base computing on which information will be displayed…they do not do well with applications that start out in one place (screen, computer, or room) and then move to another…there are no systems that do well with the diversity of inputs to be found in an embodied virtuality*." In the time since Weiser wrote these words, computer displays have become far more sophisticated and so too

have information visualisation techniques. Strides have been made to improve the location sensitivity of some software system types, Global Positioning Systems software for automobiles, for example. This is in some respects a 'clarion call' and a foundational use case for context-aware computing whereby the software changes its behaviour based on the location of the user.

- *"…the transparent linking of wired and wireless networks is an unsolved problem*." This is no longer a problem with the advent of 802.11x and other modern wireless protocols such as Bluetooth, which integrate relatively well with existing wired networking protocols because they support TCP/IP.
- "*…the number of channels envisioned in most wireless network schemes is still very small, and the range large (50-100 meters), so that the total number of mobile devices is severely limited. The ability of such a system to support hundreds of machines in every room is out of the question*." This is still an issue with most wireless technologies as they are not capable of supporting hundreds of machines in every room.
- "*Present technologies would require a mobile device to have three different network connections: tiny range wireless, long range wireless, and very high speed wired. A single kind of network connection that can somehow serve all three functions has yet to be invented*." This is less of an issue since most mainstream wireless technologies support TCP/IP, including Bluetooth (tiny range) and 802.11 (medium range) and General Packet Radio Services (GPRS) (long range). A number of portable devices currently in the marketplace allow TCP/IP communications to take place over all three wireless ranges, although these devices currently do not transparently hand-off from one network to another. Rather, in the devices we tested, the networks run in parallel and must be turned on and off manually. In any respect, these multi-networked devices are currently quite expensive (i.e., not ubiquitous because of the high cost) and thus the technology must progress much farther to facilitate ubicomp.
- *"…although active badges and self-writing appointment diaries [i.e., all potential ubicomp technologies] offer all kinds of convenience, in the wrong hands their information could be stifling*." The issue of security and privacy

> will always be a consideration in the deployment of ubicomp technologies. In the time since the Weiser article was written, the technical foundation for ubicomp security and privacy, e.g., public/private key encryptions, x509 security certificates, identity management, etc., has been widely implemented. It is difficult, however, to predict whether these technologies will be sufficient to give users the belief that their data is safe and will not be abused. This is of particular concern because, as Weiser points out, the potential for privacy breach is much greater when ubicomp technologies become capable of capturing many forms of human activity.

Overall, many of the barriers foreseen by Weiser in the early 1990's have been overcome by advances in software in terms of distributed computing and operating systems. Wireless network technologies continue to be an impediment to ubicomp, although tremendous advances have taken place in the wireless technology realm in recent years. Given the speed with which network technology has advanced, there is reason to be optimistic that Weiser's concerns will be addressed and overcome within a few years. It remains to be seen how willing users will be to accept and adopt ubicomp technologies as these technologies attempt to enter the mainstream.

2.3.2 Ubicomp Challenges Identified by Satyanarayanan

In his 2001 paper, *Pervasive Computing: Vision and Challenges* [1], Satyanarayanan discusses the rise of ubicomp (a term he declares is synonymous with pervasive computing) in relation to the established research fields distributed computing and mobile computing. Both distributed computing and mobile computing, he asserts, are foundational components for ubicomp. Based on the state of technology in these two research fields, he asserts "many critical elements of [ubicomp]…are now viable commercial products…we are now better positioned to begin the quest for Weiser's vision." Satyanarayanan describes distributed computing and mobile computing as "foundational" research areas to ubicomp. Furthermore, he identifies a number of research areas within ubicomp will need to be developed in order to address the unique challenges posed by ubicomp.

The relationships between the various research fields described by Satyanarayanan are illustrated in the Figure 2 taken from his paper. As implied by Figure 2, pervasive

computing (i.e., ubicomp) is dependant on the research field of mobile computing; similarly, mobile computing is founded on, and is largely preceded by in a temporal sense, distributed computing research. Satyanarayanan points out that new challenges and/or barriers are confronted as one moves from left to right in the diagram. Furthermore, because "many previously encountered problems becomes more complex" as one moves to the right, the level of complexity increases multiplicatively (as opposed to additively) as solutions approach ubicomp dimensions.

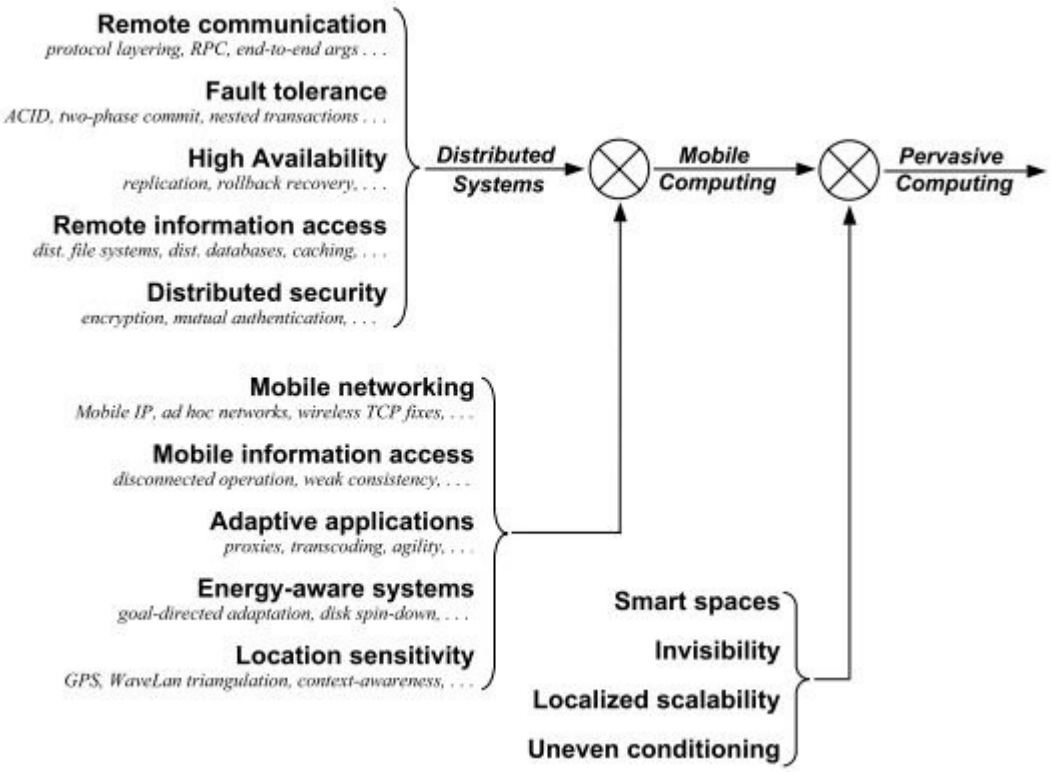

Figure 2: Taxonomy of Computer Systems Research Problems in Pervasive Computing, from [1]

Satyanarayanan lists a number of research sub-fields within both the distributed computing realm and mobile computing research realm that are germane to ubicomp. Most of the sub-fields of distributive computing listed by Satyanarayanan were in existence when Weiser wrote his well-known Scientific American article, and thus were not considered by Weiser to be ubicomp research challenges. Mobile computing, however, is a more recent incarnation, the result of the "appearance of full-function laptop computers and wireless LANs in the early 1990's." Many of the research sub-

fields of mobile computing, as listed by Satyanarayanan, address barriers identified by Weiser relating to wireless networking and wireless scalability.

2.3.3 Challenges Identified by Davies and Gellerson

Davies and Gellerson [5] continue the discussion of ubicomp challenges in their 2002 paper, *Beyond Prototypes: Challenges in Deploying Ubiquitous Systems*. From their perspective, many of the technical challenges described by Weiser in his article – inadequate operating systems, underdeveloped wireless networking technologies, and few location detection technologies – have been overcome. In reference to an ubicomp scenario presented in the Weiser article, Davies and Gellerson argue "the technologies required [to deploy the scenario] are either already deployed or could be deployed relatively trivially." Furthermore, the advent of the World Wide Web (WWW) and the Internet was also helping to create "a culture that is substantially more amenable to the deployment of ubiquitous environments," as compared to the early 1990's. Yet, they argue Weiser's vision for ubicomp "appear as futuristic today [i.e., 2002] as they did in 1991," and "we are still many years from creating such systems." The implication of this statement is: in spite of all the research activity dedicated to ubicomp over the past ten years, progress has been slow. Davies and Gellerson proceed to discuss the problems which have impeded, and will continue to impede, progress towards Weiser's vision.

Davies and Gellerson's differentiate between sociological and technical challenges to ubicomp. In terms of the sociological issue, they echo many of the issues discussed in Weiser's article, including the privacy and legal implications of ubicomp technologies. They also argue that ubicomp lacks an effective and sustainable revenue-generating business model (e.g., the lack of a "killer" application) and thus the business world is less willing to dedicate resources to designing and incorporating ubicomp technologies.

From a technical perspective, Davies and Gellerson focus on issues that are target levels of abstraction much higher than Weiser did in his article. Whereas Weiser discusses the inadequacy of wireless, operating systems, distributed systems, and (to a lesser degree) location-detection technologies, Davies and Gellerson assert that the main problem is a "lack of integration in existing systems." They decompose the software integration problem in two ways. First, they suggest ubicomp lacks an open, extensible, and widely accepted middleware to allow "us [to] combine components to form

applications unforeseen at the time of their deployment," and will provide "assurances from their components in terms of metrics such as performance, security, and reliability." In the last few years, many ubicomp middleware frameworks/systems have been developed to address this need (see [6, 7, 18]); to our knowledge none, however, have achieved widespread adoption within the ubicomp community.

The second high-level software problem identified by Davies and Gellerson is that existing systems "are typically conceived and operated independently, in the context of their own restricted view of the world." [emphasis added] Davies and Gellerson suggest that in order to progress towards ubicomp, software systems must be able to understand and communicate to other systems what a user or a group of users are doing at a particular time. Furthermore, the software must have some "form of intelligence working on the user's behalf to coordinate the actions of components in the infrastructure." To achieve this, Davies and Gellerson argue the system must have the ability to (1) "accurately determine a user's task and intention," and (2) "develop associations between [software systems] to assist the user in these activities." In other words, Davies and Gellerson are suggesting that ubicomp systems will need to have an awareness of user context, although they never explicitly use the term "context-aware."

2.3.4 Ubicomp Challenges Identified by Want and Pering

In a 2005 article entitled, *System Challenges for Ubiquitous & Pervasive Computing*, Want and Pering [4] provide further insight into the challenges confronting ubicomp. Like Davies and Gellerson, Want and Pering suggest most of the hardware and low-level technical challenges described by Weiser in the early 1990's have been overcome. They describe how modern mobile and portable devices are now, as compared to mobile devices in the early 1990's, infused with substantially faster processors and far greater storage capacities. Furthermore, wireless technologies have become commonplace and better integrated into wired networking standards since the early 1990's, whereas in the early 1990's there were "no widely adopted wireless standards." With these technological advances, Want and Pering suggest "each of the basic components required for Ubiquitous computing have fallen into place." The limitations on ubicomp progress, however, have not disappeared, but rather "have moved to the higher levels of abstraction," primarily into the software area of ubicomp systems. Specifically, they

argue that the high-level "system software capabilities have not advanced at a pace that can take full advantage of this infrastructure".

2.3.5 Ubicomp Challenges Summarized

In the papers discussed above, those published after Weiser's 1991 Scientific American article that is, all suggest that most of the basic technical infrastructure for ubicomp is currently in place. This includes the advent of fast wired and wireless networking technologies, smaller and more powerful computer chips, more efficient power sources, etc. Significant challenges, however, remain to be addressed as we approach Mark Weiser's vision for ubicomp. As described in the four works cited above, the remaining challenges reside mostly at higher levels of abstraction: middleware, social/legal/privacy concerns, and human activity awareness. The main exception is with regard to power supply technology, which has lagged behind advances in computing power [4].

In the next section, we discuss the parameters of context-aware computing, which is dedicated to addressing the challenges of making computers more human-centric.

2.4 What is Context-Aware Computing?

2.4.1 Defining "Context"

Humans have an innate capacity for understanding context. We use context as a means for categorising information we receive from our senses as we interact with the world. We tend to process contextual data without much thought. Examples of such information include where we are, the role we are playing, who is in the same room, the room temperature, lighting conditions, etc. It is just something human beings do, and are (mostly) proficient at. Computers, on the other hand, have no innate capacity for sensing and processing contextual information, in the same way that humans do not have an intrinsic capacity for understanding binary machine code.

Context-aware computing research endeavours to enrich computing technology with the ability to sense and reason about human context. But what is "context"? Within computer science, the word "context" can mean different things depending on whether it is in reference to ubicomp, artificial intelligence, language processing, or graphical user interface research. Even within the ubicomp research area, context does not exclusively refer to human context concerns (e.g., dynamic network reconfigurations). Further,

within the context-aware research community there is no consensus on a universal definition of context. Schilit and Theimer [8] are acknowledged to be the first coin the phrase "context-aware" in the ubicomp sense. Their definition of context – location, nearby people, and objects, and changes to those objects – largely reflects Weiser's notion [3] of location as the main contextual premise for automatic adaptation. As Schmidt [20] points out, the Schilit and Theimer definition of context is indicative of the prevailing notion of context in the early phase of context-aware computing, where context was defined primarily in terms of "measurable information", especially spatial information.

Dey proposed a well-received definition for context, one that is more abstract and not tied to any specific technology or application domain:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the applications themselves. [6]

This definition makes it easier for an application developer to enumerate context for a given application scenario, especially in consideration of the many research projects which utilize non-traditional contextual factors such as motion, gestures, i.e., context types not represented in the Schilit and Theimer definition. In Section 2.7 we offer a more thorough critique of Dey's definition and extend it to include references to context modeling concepts.

2.4.1.1 Our Use of the Word "Context" and the Phrase "Context Aware Computing"

Given the diversity of opinion on how 'context' and 'context-aware computing' should be defined, it is important to clarify what we mean when we use the word 'context' and the phrase 'context-aware computing' in this thesis. As a basic rule, we emphasise the characteristics of context-aware computing which are in keeping with the human-centric aspects of Mark Weiser's vision for ubiquitous computing. That is, in this thesis we will not consider the subject of computing context (e.g., network connectivity, communication costs, communication bandwidth, nearby resources, etc.). This is not to suggest that computing context issues are not important to the eventual realisation of ubiquitous

computing, but we exclude these context issues to focus on the human-centric ramifications of context-aware computing.

2.4.2 The Characteristics of a Context-Aware System

There is an abundance of literature documenting the numerous implementations of context-aware systems. Based on our review of existing context-aware literature, it is typical for a context-aware system to have been built completely independent of systems developed by other research groups, from the sensing technology upwards. In spite of the ad hoc evolution of the context-aware research community, we can make robust generalisations about the characteristics of a typical context-aware system. At a high level, a typical context-aware system will possess the following attributes or capabilities:

- The capacity to sense implicit human-computer interactions
- The capacity to perceive context
- The capacity to change or adapt system behaviour depending on perceived context

In the following sections we deal with each point in turn in the following sections.

2.4.2.1 Sensing Implicit Human-Computer Interactions

The dominant way that humans currently interact with computers is through a keyboard, mouse, and computer screen. Much of the research in context-aware computing has been dedicated to expanding the number communication channels between computer and human. Although a great deal of research has been conducted in terms of making computer interfaces more transparent (i.e., make them more natural to use), implicit interfaces focus more on making computers better able to sense natural human activities (i.e., to make computers more human-centric). In the overwhelming majority of context-aware research initiatives conducted to date, these new human-computer interfaces have been implicit rather than explicit in nature. An explicit human computer interface is one where the human interacts directly and deliberately with the computer. Examples of explicit HCI are, therefore, when a user negotiates a keyboard, mouse, pen, or speech interface to communicate with a computer. On the other hand, implicit HCI occurs when a computer system is able to obtain information from the user

based on indirect uses of the computing or context-aware system. Albrecht Schmidt defines implicit human-computer interaction as follows:

> Implicit human computer interaction is an action performed by the user that is not primarily aimed to interact with a computerized system but which such a system understands as input. [21]

Most research conducted in context-aware computing embraces some form of implicit human-computer interaction. The most prevalent and simple usage of an implicit interface within context-aware research is location awareness. In moving from location to location – from one office to another, or from one area of the room to another area, for example – a user indirectly or passively communicates to a location-sensitive system that his or her immediate needs or goals may have changed.

Perhaps the earliest research initiative to utilise location implicit input was the Active Badge project [22], developed in the early 1990's at Olivetti Cambridge Research Labs. The Active Badge system was initially used as a system for enhancing communications among employees in large organisations. Each participant was given an 'active badge', which tacitly emitted an infrared (IR) signal to the IR sensors embedded in the office environment. A centralised station aggregated and processed the sensor information to allow, for example, telephone receptionists to forward calls to the proper room. Assuming that it was the goal for employees to be found (which may not always be the case), the employee (through the active badge) implicitly communicated to the system how to most effectively contact the employee, regardless of whether she was in her office or not.

Surveys of early context-aware research conducted by Chen and Kotz [23] in 2000, and by Dey and Abowd [24] in 1999, reveal that "few contexts other than location have been widely studied." Of the projects selected by Chen and Kotz in their survey, ten of twelve exploited user location for determining context; similarly, thirteen of fourteen selected by Dey and Abowd focused on location-based context. What is the reason for the emphasis on location-based context in these early projects? Chen and Kotz suggest that, perhaps "other contexts are difficult to sense", whereas technologies existed to provide relatively accurate location sensing in both indoor and outdoor environments. For example, Global Positioning Systems (GPS) were being used for determining location with an accuracy of

about 3 metres. Further, for indoor environments, infrared (IR) and radio frequency (RF) technology had provided early projects with acceptable location sensing capabilities. Chen and Kotz also hypothesised that other forms of contextual information – context history, for example – "are not as useful as we have thought", and therefore it may be sufficient to only consider location-awareness.

In a 1999 paper written by Schmidt et al. [20], the location-centric notion of context is reconsidered. They argued that with advances in sensor technology, the time was ripe for a consideration of features other than location that contribute to context. Schmidt cites a number of examples of context-aware research initiatives that consider non-location based contextual factors, including:

- Gesture awareness [25]
- User attention level [26]
- User emotional state [27, 28]

Of the research cited by Schmidt above, none appeared in the Dey or Chen surveys. Schmidt also describes advances in various sensor technologies (e.g., optical/vision, audio, motion, location, biosensors) will enable more advanced context information acquisition in the future. Presently, much of the cutting edge research conducted in context-aware computing today focuses on non-location based factors, such as human physiological and physical activity [25] and visual awareness [29]. Furthermore, physical or environmental sensors such as accelerometers and infrared readers are no longer the only contextual information sources. Factors such as user history [30] and information gathered from distributed sources, such as information gathered from an external Web service source (e.g., a weather or traffic Web service) [31], are now considered first order context information sources.

2.4.2.2 Context Perception

Human beings instinctively use context information to help categorise and understand the things we see, hear, smell, touch, and taste. According to cognitive psychologists, as we interact with the world around us, we evaluate sensory information against a pre-existing mental model [32]. When our sensory information gives us conflicting information (e.g., when we see snow in Florida) we tend to divert more resources to the task of determining how the information fits into a mental model. Sometimes the mental

model must change to incorporate new combinations of information derived from the senses. Context-aware systems more or less perceive context in a similar fashion: a context reasoning engine or inference engine evaluates sensor information against a semi-static context model to determine the current context state(s).

Strang et. al. compiled a survey of existing approaches for modeling and reasoning about context [33]. A context model is a repository of knowledge about the entities, scenarios, and interaction patterns within a given context-aware application or applications. Context models are represented or expressed in a context model specification. Strang et. al. arrived at six different categories of context model specification types, including:

- Key-value models: The simplest data structure for modeling context whereby the services are described as a list of key-value attributes. The advantage to using this approach, assert Strang et. al., is that they are easy to manage, but at the expense of model expressiveness and sophistication.
- Markup scheme models: The data is organized hierarchically in SGML/XML. The advantage to using this approach is that it can be used to interoperate with non-context markup languages like XML Web services. Strang et. al. reports, however, that this approach is susceptible to proprietary methods for handling incomplete and/or ambiguous context information.
- Graphical models: The data is organized visually in a structured way in a way that is understandable to a human. Is mainly used for structuring contextual knowledge visually; thereafter the model is transformed into code or another type of visual model to capture applicability requirements. According to Strang et. al., the main disadvantage is that the model is not directly suitable for computer evaluation
- Object oriented models. Leverages the advantages of object orientation (specifically encapsulation and reusability) to manage a wide variety of sensor information types and to promote scalability. Further, object oriented models translate well regarding distributed composition requirements of ubiquitous systems. The main disadvantage currently, according to Strang et. al., is that

object oriented models require additional resources and perhaps are not feasible for current ubiquitous environments.

- Logic based models. Logic based models provide a high degree of formality. Context is defined as facts, expressions, and rules. According to Strang et. al., logic based models are not good for specifying contextual knowledge, are difficult to maintain, and cannot be partially validated due to the distributed nature of context-aware computing. Also, full logic inference engines are usually not available for ubiquitous computing devices.
- Ontology based models. An ontology is an instrument for specifying concepts and relationships between concepts. Ontologies are especially good for translating real-world concepts into something that a computer understands. According to Strang et. al. the ontology based approach is most appropriate for ubiquitous computing (and thus context-aware computing)

2.4.2.3 Context Adaption

In addition to implicit information sensing and context perception, we assert that another characteristic of context-aware systems is self-adaptation. Adaptation is an important characteristic of context-aware systems as a means to reducing the intrusiveness of the system. As mentioned earlier, context-aware systems typically process implicit information (e.g., location information) about users and environmental conditions – i.e., without direct or conscious intervention from users. Context-aware systems, based on perceived context, must predict and adapt to the future needs of users to reduce the intrusiveness of the system in the daily activities of users.

Dey et. al. assert that adaptation is not a critical component of context-aware computing, “that an application that displays the context of the user's environment to the user is not modifying its behaviour, but it is context-aware.” [24]. We do not disagree with this point: the raw definition of “context-aware” does not imply adaptation or behaviour modification; however, the overwhelming majority of research conducted in the name of context-aware computing describes the adaptive features of their respective systems (see [34-36]). Given the magnitude of sensors and computing devices envisioned by Weiser and others for ubicomp environments, we assert that proactive and adaptive systems will be necessary to keep context-aware systems human-centric.

Furthermore, adaptation is what helps to distinguish a context-aware system from a non-context-aware system.

2.5 The Challenge of Context-Aware Computing

As a sub-research field of ubicomp, context-aware computing inherits the complications or challenges inherent in implementing ubiquitous systems, and then adds a greater magnitude of problems. Referring back to Figure 4, Satyanarayanan argues that as we overcome the problems of distributed computing and mobile computing, "previously-encountered problems become more complex" as we begin to tackle ubicomp problems. In an adaptation of Satyanarayanan's taxonomy as provided in Figure 2, we present Figure 3, in which we drill a little deeper into the complexities of ubicomp. We differentiate between the three ubicomp concerns we introduced in the previous chapter and the complexity of challenge as it relates to these concerns. As in Satyanarayanan's illustration, we concur with his assessment that the challenges of ubicomp become progressively more complex as we move left to right; that is, the technological concerns of ubicomp will be subsumed by human-centric concerns, which in turn will be subsumed by social concerns, and with each progression the problems get more complex. We further submit that the progression from technological to human-centric to social concerns is not a strict temporal relationship, but as suggested by Satyanarayanan, represents a logical relationship. But this logical relationship does loosely imply a temporal relationship; for example, the technological foundations of ubicomp will enable researchers to focus more on human-centric (micro-HCI) endeavours, and thereafter social (macro-HCI) concerns become more manifest once the human-centric concerns have been dealt with.

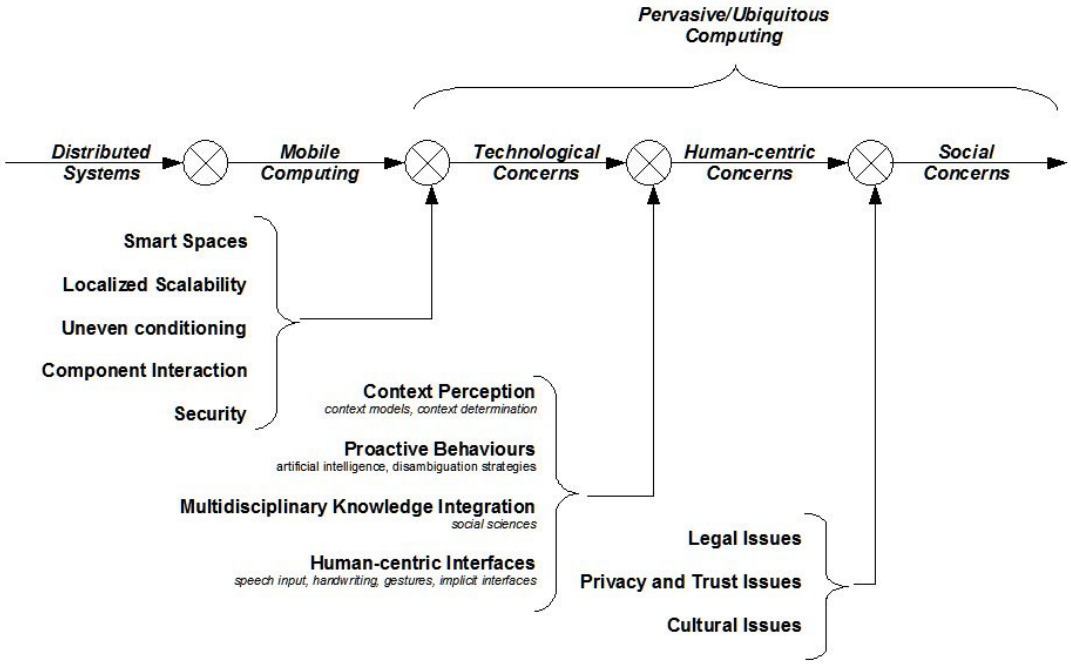

Figure 3: Taxonomy of ubicomp research problems, adapted from Satyanarayanan [1]**, with intra-ubicomp categorizations**

As we argued in the previous section on ubicomp challenges, most of the low-level technical foundation of ubicomp is in place. Given this, we submit that the main challenges to the realization of ubicomp have become, and will become increasingly so, more focused on human-centric issues such as Human-Computer Interactions (HCI), multidisciplinary software collaborations, and context-awareness. But as computer technology attempts to become more human-centric, the challenges become more complex and difficult to overcome. In this section we examine the challenges of implementing context-aware computing systems – a human-centric technology – in light of these observations.

2.5.1 Traditional Challenges of Context-Aware Computing

In the previous chapter we described the fundamental characteristics of a context-aware computing system, from the perspective of multiple authors and from our own inferences. Briefly, we asserted the main characteristics of a context-aware computing system are:

1. A system that can sense implicit human input
2. A system that perceives human context
3. A system that behaves differently based on the context perceived

Most research in context-aware computing falls generally somewhere into one of the three categories listed above. Based on our review of the context-aware research literature, most of the activity has focused on characteristic (1) listed above; it is not coincidental that characteristic (1) is enmeshed with the larger research realm of ubicomp in terms of developing the means for sensing and has been fundamentally focused on the technological aspects of ubicomp. It is in characteristics (2) and (3) where context-aware computing represents an area of ubicomp that is clearly distinguishable from other ubicomp-related research fields, especially in terms of its human-centeredness.

2.5.2 Challenges in Sensing Implicit Human Input

Much progress has been made in the technical foundations of ubicomp in terms of computer networks (especially wireless networks) and the miniaturization of computing technology since the publication of Weiser's Scientific American article. As was reported in Section 2.3.5, it is generally concluded that ubicomp has become feasible from a technology standpoint. Gellerson and Davies suggest Weiser's 'Sal' scenario was technically possible already by 2002. However, there is a great deal of room for improvement in the technology used for sensing implicit and explicit human input. In the following, we emphasise three areas where from our perspective ubiquitous sensing is most challenged.

First, context sensing technology is still relatively expensive, in most cases too expensive for everyday/every person/everywhere use. Furthermore, as Davies and Gellerson point out in [5] in 2002, and according to our experience is still the case, a viable business model for ubiquitous (i.e., everyday/every person/everywhere) usage of environmental sensors remains elusive. Still, there is reason to be optimistic that such deficiencies will become negligible with time as long as Moore's law continues to accurately predict the rate of advancement – and relative expense – of computing technology. The amount of time that is required is unclear given the physical limits of Moore's Law related progress.

Second, according to Michael S. Malone, "the biggest impediment to our technological future isn't extending Moore's Law…It's battery life." [37] Battery capacity, or the lack thereof, has emerged as perhaps the most significant technical barrier to context-aware sensing, mobile computing, ubicomp, and to computing technology as a whole. Over the last forty years or so, semiconductor technology has advanced roughly at the rate predicted by Moore's Law, but increases in battery capacity have not progressed at nearly the same rate. As Want and Pering point out, "[inadequate battery] power has always been a thorn in the side of the Ubiquitous computing vision," [4] and will continue to be so in the foreseeable future. This has important implications for the feasibility of context-aware computing regarding the types of implicit information that can be sensed in the human environment and how services can be delivered to mobile (i.e., battery empowered) users.

A third challenge area for implicit context sensing is can be broadly described as device management, i.e., how do we manage the attributes of the hundreds, thousands, millions, and possibly billions of computing sensors and devices? This is a huge issue, covering the problem of how to come up with standards for ubicomp in terms of component interactions, how to model these dynamic networks of devices, and how to deploy system changes. Furthermore, as Davies and Gellerson point out, the management of ubicomp systems is unlikely to be administered in the context of a single domain, and that the administrative domains likely will change dynamically [5].

2.5.3 Challenges in Perceiving Human Context

In Section 2.4.1 we discussed the debate over how to define 'context' with regard to human-centric concerns. Although most software engineers involved in context-aware computing have settled on the Dey definition of 'context' (i.e., any information that can be used to characterize the situation of persons, places, or things, that are considered relevant to the interaction between a user and an application [38]) this debate has by no means has been resolved. For example, Greenberg argues that the simplicity of the Dey definition for 'context' belies the actual complexity of arriving at a machine-actionable determination of human context, as it fails to recognize the fluid and dynamic nature of human context [39]. Not only is it difficult to arrive at a correct set of rules for capturing and determining the context of a user, but the information derived from various context

information sources may not be accurate and/or has ambiguous implications. Many believe the goals of context-aware computing are foolhardy and not worth pursuing because it is unlikely that any computing system will possess in the foreseeable future the requisite intelligence to sift through these issues to properly interpret human context (e.g., [36]). Others believe that context inference problems can be mitigated with human mediation strategies [40].

A central factor in promoting context perceptiveness is the context model. The effectiveness of a given context model for a given context-aware system is predicated on how well the context model specification addresses the requirements of the application. We identify two challenges regarding the selection of context modeling specification types for a given context-aware application.

First, if a given context model specification is too simplistic and not sufficiently expressive to represent the context model required for a given application it will (obviously) not serve the objectives of the context-aware system. For example, the key-value pair context model specification type used within the Context Toolkit [6] would not suffice for representing the attention model described by [26]. This has become less of an issue in recent years as XML-based ontology formats (e.g., [2]) are increasingly being used as context model specification types. Still, it is a challenge to create context model specifications that are sufficiently expressive to represent human-based context.

Second, the construction of context models using more sophisticated context model specification types requires sophisticated tool support. For more advanced context model specification types like those based on XML-based ontology formats, it is important that these specification types are accompanied by visual tools for helping construct context model instances. For example, the COBRA-ONT ontology for the Context Broker Architecture (CoBrA) [2] is supplemented with a visual editor called the CoBrA Eclipse Viewer (CEV) (see Figure 4 below). Without visual tool such as the CEV it would be difficult to construct and adapt context models, thereby reducing the effectiveness of the model.

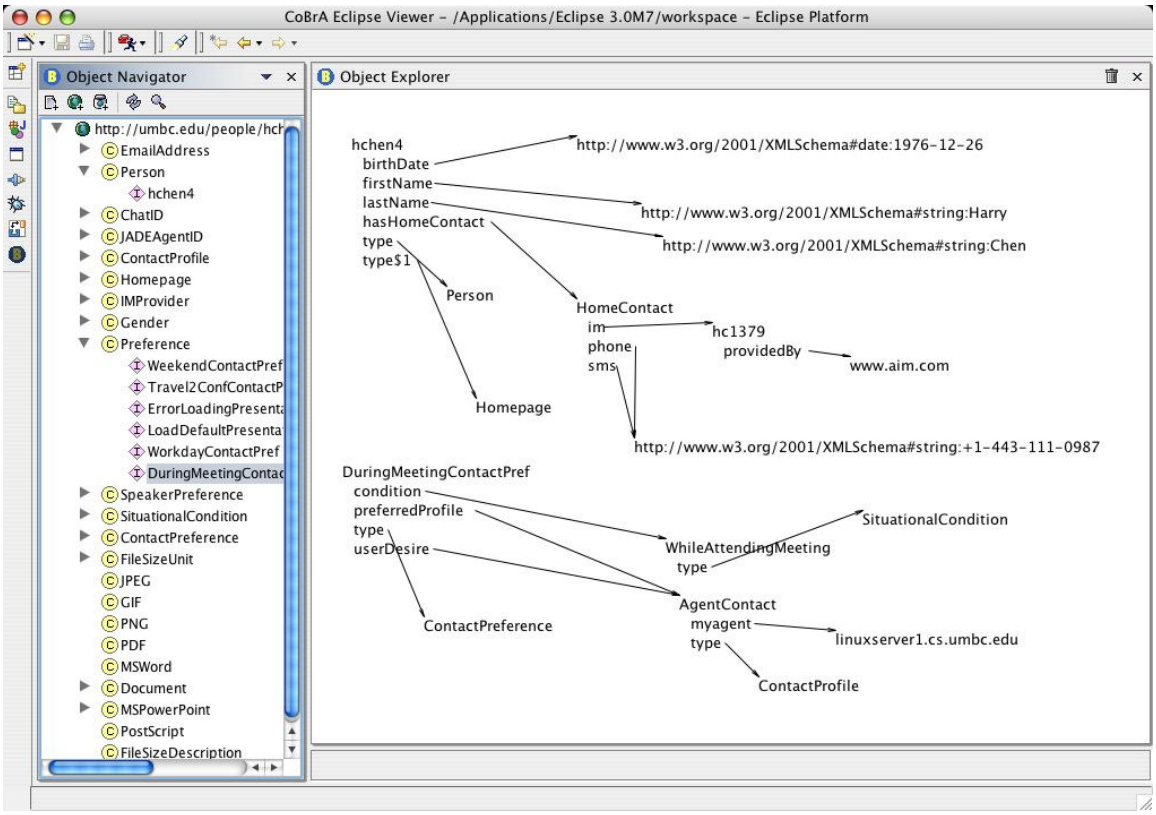

Figure 4: The CoBrA Eclipse Viewer (from [2])

2.5.4 Challenges in Context-Aware Systems Behaviour

One of the main goals of context-aware computing is to promote the invisibility of ubiquitous computing systems by making computers more sensitive to human activities and more proactive in helping users conduct their everyday tasks. Promoting invisibility of computing is often viewed as synonymous with reducing intrusiveness of computing, but this is not always the case. As Dey et. al. suggest in[40], that it is appropriate for a context-aware system to request explicit user instructions when ambiguous context data is received. Furthermore, different tasks require different levels of explicit human-computer interactions. For example, the act of writing is almost completely a non-implicit human-computer interaction activity and this is unlikely to change with ubicomp, although the modes of data input may change as speech recognition and other so-called "natural" interfacing technologies are developed. Contrast this to a physician who must visit the beds of twenty or more patients within a hospital. Many of interactions between physician and computer can be made "invisible" to the physician using implicit information, such as the physician's proximity to a patient and the time of day. For

example, when the physician approaches the bed of one of his patients, a context-aware system could automatically deliver the latest information for that patient, based on his nearness to the patient. The same scenario could also produce an adverse experience for the physician if the system was calculating inaccurate location information, or the system had not been updated with the latest patient location information, or if the physician was visiting a patient for non-professional reasons. In fact, there are many conceivable ways that a computing system may infer the wrong context of the physician thereby making the system more of a hindrance than a help. Thus, determining the appropriateness of the action taken by context-aware systems is much an art as it is science, and highly dependent on human psychological state, the physical surroundings, the social situations, and manifold more factors.

2.5.5 Other Challenges in Context-Aware Computing

2.5.5.1 Multidisciplinary Participation in Context-Aware Computing

The research area of human-computer interactions (HCI) has traditionally focused on the personal computing interaction paradigm, with its focus on keyboard-video-mouse (KVM) modes of human-computer interactions. As time moves forward and as the technological foundation for ubicomp becomes more entrenched, new patterns of HCI will emerge. Already significant inroads have been made in terms of building natural user interfaces, such as speech, gesture, and pen input interfaces, interfaces which will potentially allow users to interact with computers in more human-friendly ways. The construction of these new modes of HCI has required the expertise of multiple research disciplines. For example, the speech recognition systems of today leverage expertise from the disciplines of physics, linguistics, mathematics, statistics, engineering, software engineering, and others.

We argue that the context-aware computing systems of tomorrow will similarly require help from multidisciplinary sources. As we described in the previous chapter, an implicit interface – a core fundament of context-aware computing – is sensitive to human actions that, from the perspective of the user, is not an explicit or intentional directive to a computerized system. The implicit user interfaces of tomorrow will need to be tightly integrated into the human environment with the implication that such interfaces will need to be knowledgeable of the human world in order to be effective. Thus, such interfaces

will need to incorporate knowledge from multiple disciplines and domains of knowledge, particularly from the social sciences like psychology and sociology. Although such multidisciplinary collaborations are common in HCI currently, there will be an even greater need for humanistic influences on the design of context-aware, ubicomp human-computer user interfaces. The challenge is in finding a way to harness the expertise of such a diverse group of experts, some of whom will not be technically savvy. This problem is related to the recurring software engineering problem of capturing requirements from end-users.

2.5.5.2 The Nature of the Context-Aware Computing Research Community.

Given the age of the context-aware computing research field, relative to other computing research fields, one might expect it to be mature and well-established. By most measures, however, context-aware computing remains an immature research area. We make the following axiomatic assertions:

- There are no widely accepted context-aware standards or protocols to govern the research community.
- Based on our research, to date, there has been virtually no collaboration between the many context-aware research groups in terms of sharing software components. In other words, most context-aware research groups opt to build entirely new context frameworks rather than build on frameworks developed by others.
- Little context-aware computing technology has been commercialized, has been backed in a significant way by a large corporation, or has been utilized outside of a research laboratory setting

Contrast the above assertions to the state of GRID Computing, a research field of similar age. GRID technologies are backed and commercialized by large corporations (e.g., IBM, Oracle), are governed by established and well-articulated open standards (e.g., OGSA), and are cultivated by a large collaborative software development community (e.g., the Globus Toolkit).

There are many potential reasons why the context-aware research community has not congealed like other computer science research areas of similar age and research activity magnitude (e.g., GRID computing). Probably the biggest reason is due to the nature of

the technology of ubicomp, with the emphasis on embedded, mobile, and sensor networks of computing devices. Ubicomp technology is extremely disparate in terms of hardware capabilities and in operating systems platforms. Compare the diversity of ubicomp technologies to the technologies used for driving the personal computing revolution, where over ninety percent of the devices were using the same hardware platform (i.e., IBM-compatible platform) and the Windows platform. Such uniformity in hardware and software is not a reality in the ubicomp devices of today and thus there are fewer opportunities for sharing software strategy.

2.6 Analyzing Trends in Context-Aware Research

Given what we see as a lack of cohesiveness in the context-aware computing research community, we sought to quantify the level of research activity in this field. Does the context-aware computing research field receive attention mostly from a small but highly vocal research group, or is it a research field that has many contributors? One way to gauge the popularity (and arguably, relevancy) of a particular research topic is to investigate how much that topic is discussed within various peer-reviewed academic publications. To this end, we sought to compile statistics on the occurrence of the topic of context-aware computing within two of the leading publishers of computer science research: the Association for Computing Machinery (ACM), and the Institute of Electrical and Electronics Engineers (IEEE). We describe the method with which we compile the topic-occurrence statistics in the following section.

2.6.1 Method for obtaining Topic-Occurrence Statistics

Both the IEEE and the ACM offer digital versions of their publications and allow users to perform full-text searches on the contents of these publications from a Web browser. The IEEE Xplore Web site provides access to over 1.4 million IEEE documents (journal articles, transactions, letters, magazines, conference proceedings, and standards) published since 1988. The ACM Digital Library provides access to over 192,000 ACM documents (journal articles, magazines, transactions, conference proceedings, newsletters, master's theses, and doctoral dissertations) published since 1947. Both the IEEE Xplore and ACM digital library search engines allow queries to be defined in terms of words and/or phrases, by author, and basic boolean operators. Furthermore, both

allow the results to be filtered based on topic/publication type and publication date. It is not clear, however, how similar the search engines are in producing the query results, e.g., given the same inputs, would they produce the same results? To normalize the query results obtained from the IEEE and ACM digital libraries we turned to Google Scholar (TM). Both the IEEE and the ACM have given Google Scholar (TM) full-text document access to their digital libraries, thereby allowing us to use the Google search engine on both sets of documents.

Google Scholar (TM) allows articles to be queried using similar query criteria parameters as by the IEEE and ACM digital libraries. The interface for conducting advanced searches on Google Scholar (TM) is shown in Figure 5.

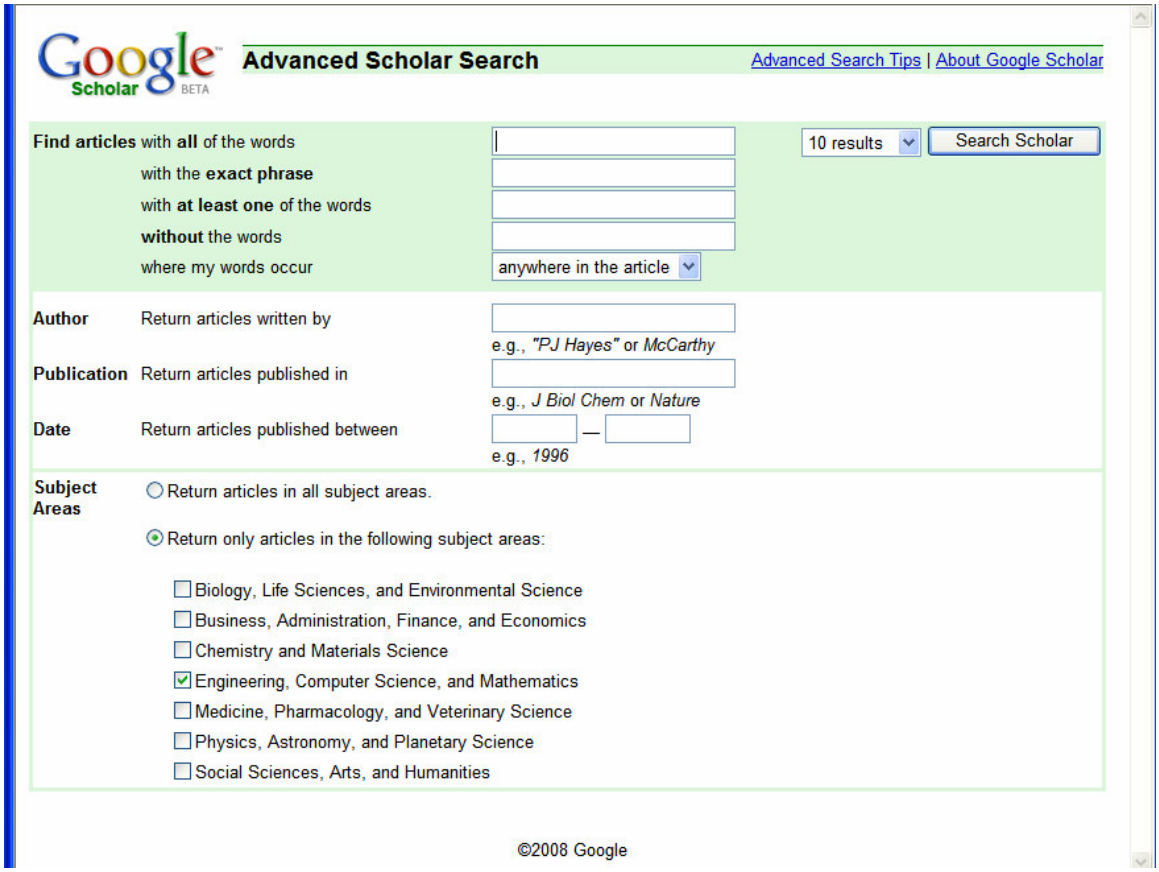

Figure 5: Google ™ Scholar advanced search interface

2.6.1.1 Search Parameters

To reiterate the objective of this exercise, we wish to quantify the level of research activity dedicated to context-aware computing since the early 1990's to determine if context-aware research is circumscribed to few researchers or consists of many as a way

of quantifying the level of research, which might explain the lack of cohesiveness in the context-aware computing research community. We assume that a reasonably accurate measure of this activity level can be obtained by submitting carefully selected query parameters to the Google Scholar search engine and recording the number of documents produced by the query. In carefully selecting the query parameters, the goal is to create a query that produces results from a wide range of context-aware related documents, but does not produce a high number of false positives. Using Figure 5 above as a reference, we describe our decisions regarding entering query parameters for searching context-aware computing related documents.

The most important query parameters on a full-text search are word/phrase inputs. These are captured in the green area of the Google Scholar advanced search interface in Figure 5. It took some trial and error to determine the appropriate word/phrase search criteria for returning optimal (i.e., high sensitivity, few false-positives) results. Entering the words "context aware" without quotes into the topmost input box ("all the words") produced almost 9,000 results over all Google Scholar publications, but upon examination it became clear that most of those results were false positives, e.g., "Phase-Aware Remote Profiling", "Power Aware", etc. The words "context aware" proved to be not sufficiently sensitive, as many documents were indexed with the phrase "context-awareness", or "location-aware", or "context-enabled", but not explicitly "context-aware". An example where this happens is in [13], where the phrase "context aware" does not explicitly appear, but "context-awareness", "situational context", "context-enabled" do.

We compiled a list of synonyms for the phrase "context aware" and chained the synonyms together with OR operators to produce the following Google Scholar word/phrase query input:

context-aware | context-awareness | situation-aware |
location-aware | location-awareness | context-enabled

To further reduce the number of false-positives produced by the query we selected the "Engineering, Computer Science, and Mathematics" check-box under the "Subject Areas" heading.

2.6.2 Context-Aware Trend Results

The results of our tests performed on November 20th, 2006 are presented in Table 1. The "ACM+IEEE" plot lines correspond to the total number of articles published by the ACM and IEEE that possess the word/phrase query parameters whereas the "All" plot lines represent the number of articles indexed by Google Scholar from all publication sources. The "All" plot lines likely include some redundancies since some articles appearing in the ACM and IEEE may appear elsewhere. We do not consider this as a severe problem because it is the comparative number between context-aware computing and Grid computing that is important, and the redundancies likely apply to articles in both topics.

	ACM	IEEE	ACM+IEEE	All
1990	1	0	1	8
1991	0	0	0	7
1992	0	0	0	2
1993	4	2	6	15
1994	6	8	14	36
1995	3	3	6	33
1996	8	5	13	64
1997	15	20	35	114
1998	11	20	31	146
1999	27	43	70	261
2000	72	44	116	461
2001	87	99	186	861
2002	130	174	304	1350
2003	144	228	372	2030
2004	307	375	682	2810
2005	404	498	902	3450
2006	453	636	1089	3500
2007	318	639	957	3520
2008	3	36	39	164
Sum	1993	2155	4148	18832

Table 1 : Google Scholar estimate of the number of publications for context-aware computing. Statistics compiled on February 14, 2008.

These results show that publications on context-aware computing continue to increase, suggesting an increase it the popularity, and potentially the relevancy, of context-aware computing. Despite the relative infancy of context aware computing as shown by the lack of consensus on protocols and standards, never mind definition of context, these data show that publications focusing on context-aware computing has grown steadily.

2.7 A Reworked Definition of "Context" and Embodied Context Model (ECM)

In Section 2.4.1 we discussed how others have defined the word "context" from the perspective of context-aware computing. In this section we argue for a reworked definition of the word "context", one that builds on Dey's definition [6].

The main area where we suggest the definition can be improved is with respect to Dey's definition of "entity", i.e., "a person, place, or object." Dey's definition of "entity" implies an object in the physical dimension. We submit that a context entity need not have a physical presence. Rather, we argue it can also be an abstraction, a concept, an initiative, or a model of the human world as it contributes to an understanding of context from the perspective of the computing system. Borrowing terminology from Weiser ("embodied virtuality") [3] and Dourish ("embodied interaction") [41], we refer to such entity abstractions as "embodied" because they represent a certain recognizable pattern, an aggregate of physical and virtual conditions that suggest a state or collection of states of context. The entity abstractions we are describing are in fact embodied or instantiated *models* of context, which in turn are approximations of human mental models or perceptions. Thus, we refer to these "embodied" entity abstractions as *embodied context models*, or ECMs.

According to our notion of the ECM concept, ECMs are already present in existing context-aware systems, embedded within explicitly and implicitly (i.e., inferred by a context reasoning engine) defined context models. Using software engineering terminology, an ECM represents a context *concern* or an *aspect* of context within a context application that encapsulates a module of context knowledge and behaviour. For example, a context-aware system deployed in a college or university might posses a generic "undergraduate lecture" ECM, which is a model of the types of activities – desired and not – that take place when a professor delivers a lecture to undergraduate students. Such an ECM might be defined to jam all cell phone signals, to restrict the Web pages that can be accessed by students while the lecture is in session, and provide dynamic content to students based on the progress of the lecture. A "graduate lecture" ECM, on the other hand, might allow certain cell phone signals to enter the room and define attributes to promote inter-student communications.

The most common ECM type in context-aware computing is based on a spatial model, where the activity or task of a given user is inferred by a context-aware system based on the user's location and proximity to certain objects [6]. For example, a user who is physically located one or two feet in front of a computer workstation is likely (but not necessarily) engaged with that computer. A more complex ECM example is described in [26] whereby a user's "focus of attention" is modeled using "contextual elements such as position, speed and saliency of objects in the scene to estimate shared attention."

An ECM can be defined in very generic terms, such as the "Location" and "Device" entity types within the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) ontology [2], or the "Person" and "Location" entities in the Java Context-Aware Framework (JCAF) [7], or they can define very specific requirements, as in the attentional model described in [26] which requires special sensors to very specific types of information. There are an unlimited number of possible ways to define any given ECM. For example, a "meeting" ECM in one context-aware system may be represented by the information in a user's agenda organizer (e.g., Microsoft ™ Outlook or Google ™ Calendar). In another context-aware system, a conceptually similar ECM for "meeting" might go so far as to ensure that all participants are present, all participants are sitting at the table, and other very specific conditions have been met. The level of sophistication of a given ECM is subject to many factors, including:

- The types, variety, and the amount of data being used for input
- The granularity of the data being used, e.g., GPS location versus locations calculated to the nanometre
- The medium used for expressing or representing the ECM, e.g., key-value pairs in the Java programming language versus an XML-based ontology specification
- The types of users, their cultural backgrounds, social interactions
- The tasks or activities performed in a given environment

In general, the sophistication level of ECMs in context aware computing rises as the supporting technology advances, i.e., context sensors, context modeling techniques, and context reasoning capacities.

We argue that the "undergraduate lecture" and "graduate lecture" ECMs should be considered first-order context entities, something which possess encapsulated concerns

and behaviours. ECM's are composible in that they can represent an aggregate or collection of properties from other entities including other ECMs. Returning to the objective of finding a definition for the word "context" from the perspective of context-aware computing, there is one more area where we would like to improve on the Dey definition of context. Dey's defintion refers to the relevancy of information concerning "the interaction between a user and an application, including the user and the applications themselves." We submit that the "user and an application" interaction pattern does not adequately capture higher-level concerns, e.g., social, cultural, organizational, etc. Given our previous re-definition of the word "entity", we thus offer the following definition for the word "context":

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, object, or an *embodied context model* that is considered relevant to the interaction between an entity and an application, including the entity and application themselves.

2.8 Chapter Summary

In this chapter we sought to establish a foundation for context-aware computing – a research field that is popular in ubiquitous computing studies, but not clearly understood. In Section 2.2 we discussed Mark Weiser's vision for coming age of ubiquitous computing, and the role that context-aware computing may play within this vision. We followed in Section 2.3 with a description of the challenges of deploying ubicomp applications. In Section 2.4, we examined the characteristics of a context-aware computing system, including a discussion of how to definition of the word "context." In Section 2.6 we sought to quantify the level of popularity of the context-aware computing research field. We did this by searching for context-aware computing related, peer-reviewed articles published by the IEEE and the ACM, and compared the results to the established research field of GRID computing. According to our results, context-aware computing has grown in popularity steadily since the early 1990's and, furthermore, is nearly as active in terms of publications as the GRID computing research field.

Finally, in Section 2.6, we introduced the concept of an *embodied context model* (ECM) and offered a reworked definition of the word "context," which takes into account

Dey's widely-accepted definition [6]. The ECM concept is a central element to this thesis work. In the next chapter, Chapter 3, we describe an implementation of an ECM called ContEmPro which has been designed to detect human activity processes. In the following chapter, Chapter 4, we describe how the ContEmPro ECM can be integrated into existing context-aware systems, applications, and middleware. Our objective is to illustrate our general approach to integrating any ECM into any context-aware system (middleware or application).

3 The ContEmPro System

3.1 Introduction

In the previous chapter we discussed many of the challenges confronting the context-aware computing research community. In this chapter we present the Context-aware Embodied Process system (ContEmPro), one of the central contributions of this thesis. Based on our review of context-aware modeling research literature, we assert that little previous context-aware computing research addresses how to explicitly model high-level context-aware interactions. An important concept we derived for representing process/activity flows within ubiquitous context-aware systems is something we call an *embodied processes* (EmPro). Using the terminology presented in the previous chapter regarding *embodied context models* (ECMs), an EmPro is a particular kind of ECM. An EmPro is the manifestation of social convention, a habit, a ritual, or a protocol that can be used as information within a ubiquitous context-aware system. In Section 3.2 we describe in greater detail the characteristics of an EmPro and our approach to modeling this concept for context-aware system application.

The ContEmPro system will serve a secondary purpose which will be discussed in-depth in the next chapter. We stated that an EmPro is a type of ECM and the ContEmPro system executes instances of EmPro activity graph models. In the next chapter we will use ContEmPro as a case study implementation of how an ECM executable can be integrated into one or more context-aware middleware systems using an approach we developed for this purpose.

In this chapter we describe the two main components of the ContEmPro system: EmPro-Model and EmPro-Execute. We begin with a discussion of the concept of embodied processes, including a definition of an EmPro.

3.2 Embodied Processes

3.2.1 Defining Embodied Process

Many things can be described in terms of a process. There are naturally occurring processes (e.g., photosynthesis), artificially constructed processes (e.g., the democratic

process, a computer process), process that take place over millions of years (e.g., the process of "continental drift"), and processes that broken down into sub-processes of the smallest scale (e.g., nuclear fission). A process may consist of many variable or conditional steps (e.g., driving from point A to point B), or may be proscribed to consist of a single flow path. The word "process" is defined by Merriam Webster as follows [42]:

```
1. A natural phenomenon marked by gradual changes that lead toward a
particular result <the process of growth>
2. A continuing natural or biological activity or function <such life
processes as breathing>
3. A series of actions or operations conducing to an end
```

In general, all processes have three things in common: a beginning, a series of intermediary steps, and (usually) an end point.

Much of everyday human activity can be described by, and are governed by, processes. The ContEmPro system allows context designers to construct models of processes found in a given environment. An EmPro model, a kind of *embodied context model* (ECM), is constructed to represent how context entity interactions are undertaken from the perspective of a larger, relatively cohesive, multi-step process as time unfolds. We declare a process as embodied when the process is perceivable by users and indicative of the conditions which collectively fall within an established interaction pattern – a social convention, a personal habit, a ritual, a routine, a protocol, or a procedure. For example, people tend to engage the same ritual, with minor variations, every morning in preparation for a day at work and usually the same ritual every night in preparation for sleep. In Figure 6 we depict the types of activities and/or actions which might constitute the process of preparing oneself for going to work, in the format of a Unified Modeling Language (UML) Activity diagram.

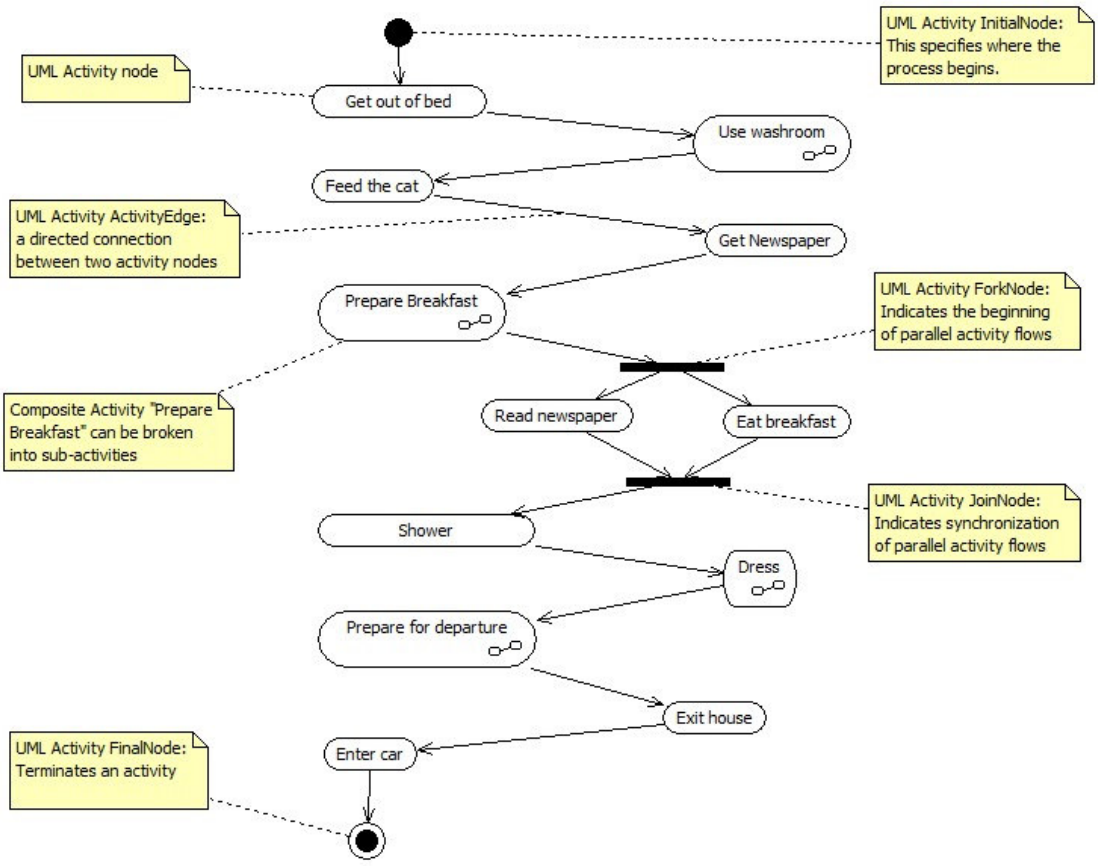

Figure 6: The "Prepare for Work" activity

Upon recognizing that a user is engaged in a particular EmPro, a context-aware system could generate conjectures about what information and services the user will likely need in the near future. An EmPro need not be explicitly associated with an overarching goal but often an EmPro can correspond directly to a detectable human-user goal, e.g., “feed the cat”; however, an EmPro can embody a multitude of user goals as they pertain to individual tasks within the process. Importantly, an EmPro adheres first and foremost to an established structure of implicit and explicit human-computer interactions that may correspond to no detectable user goals, or a chain of multiple user goals and related tasks.

3.2.2 The Characteristics of an EmPro Model

In order to properly represent an EmPro within a context-aware modeling system, we assert that the context model must exhibit the four following characteristics. First, it must allow for the specification of ordered events, as order often matters when it comes to understanding context. Referring back to the “Prepare for work” activity depicted in

Figure 6, it would make little sense to engage in the “Dress” activity before engaging in the “Shower” activity. In fact, such a reordering of activities might be indicative of a different EmPro, e.g., “Prepare for exercise”, or “Prepare for walking the dog”.

Second, there are many situations when order does not matter, but the timing, or what we refer to as the temporal proximity of events, does matter. In other words, again referring back to the “Prepare for work” activity depicted in Figure 6, if the model reflects a user preference for the concurrent activities of reading the newspaper and eating breakfast (“i.e., “Read newspaper” and “Eat breakfast”, respectively), it is not necessarily important to know which activity was engaged in or completes first, but it may be important to know that the activities have been engaged concurrently or within a shared time interval. Thus, the EmPro model must be capable of representing relationships that exist between two or more parallel activities occurring within a particular temporal proximity window. This temporal proximity is defined in terms of the completion of the common predecessor to the parallel activities, and the start of the common successive synchronization node.

The third characteristic of an EmPro model is that it must provide for the execution of what we refer to as a “negation event” for parallel activities. A “negation event” acts like a poison pill for the whole EmPro activity model whereby if a certain event is raised during a parallel execution path it effectively nullifies the execution of the EmPro model graph. In the “Prepare for work” example shown in Figure 6, a “negation event” may take place if the user were to go back to bed after preparing breakfast, a behaviour that would indicate that the user is not participating in the “Prepare for work” scenario. The requirement for a “negation event” is needed to restrict the amount of time the EmPro execution graph waits for successive events, events that will never arrive or be sensed by the context-aware system.

Fourth, the EmPro model must allow for embedding of outbound signals from the EmPro graph, to provide updates on the status of the EmPro graph. There are two types of outbound signal events required: “termination events” and “incremental events.” Termination events signify the completion of the EmPro execution graph, e.g., when the all transitions and nodes have been traversed the “Prepared for work” EmPro activity is complete. The termination signal from an EmPro graph may provide important contextual

information for other nodes within the context-aware system network, including other EmPro graphs. For example, the termination signal for the “Prepare for work” scenario might serve as an initiator for the “Drive to work” EmPro activity. An “incremental event” outbound signal, on the other hand, may be used by other context nodes as evidence of the progression towards a termination event. It represents a less accurate indication of the progression of the EmPro activity, but is required so that the context-aware system can make predictions on what will take place in the future, and thus adapt itself accordingly.

The fifth and last requisite characteristic of an EmPro model is in reference to the ability of the EmPro model to be composed within other EmPro models. That is, it is important that an EmPro model can be used for constructing other EmPro models so that a library of EmPro activity graphs can be reused many times. This will potentially allow designers of various levels of technical and domain-related expertise to participate in the modeling of EmPro activity graphs by allowing high level EmPro models to be built on top of lower-level EmPro abstractions.

3.3 EmPro-Model

There are currently many standards for expressing and executing process/workflow information, including Business Process Modeling Notation (BPMN) [43] and Business Process Execution Language (BPEL) [44], for example. The advantage of using an existing process/workflow standard for designing and executing models is that we could leverage software produced by the community supporting the standard. Furthermore, pursuing the standards-based route for EmPro models would promote interoperability with other standards compliant systems. However, we decided against using a specific process/workflow standard primarily because we wanted maximum flexibility for realizing the EmPro concepts we wanted to explore. For example, we could not easily ascertain whether the BPEL supported our concept of *time proximity*. Another reason we opted against utilizing a process/workflow standard is that we wished EmPro models to possess only the essential aspects of an EmPro as outlined in the previous section, to keep the EmPro-Execution engine relatively light-weight. The process/workflow standards we examined were general-purpose and thus they possessed a number of extraneous

attributes and/or were tied to a specific middleware technology (e.g., BPEL is closely tied to XML Web services).

For the purposes of this thesis work, we decided that the Unified Modeling Language (UML) activity diagram type would provide the syntactical foundation for EmPro model designs. UML activity syntax provides many advantages for EmPro designs, including:

- UML activity diagrams are easily to create and to understand
- UML activity diagram syntax is widely accepted as a standard
- UML activity diagrams are constructed graphically; yet, these diagrams can be represented in XML Metadata Interchange (XMI 2.1) and thus (in theory) can be used within any software that is XMI 2.1 compliant
- UML design tools are numerous and ubiquitous

As compared to a specific process/workflow standardized language, UML activity diagram notation is ubiquitously understood throughout the software engineering community.

3.3.1 EmPro UML Profile

Although the EmPro modeling requirements specified above cannot all be readily expressed in UML activity notation, the UML 2 standard provides extensibility options to customize various aspects of the UML 2 standard. These come in the form of UML profiles, which is an extension mechanism for tailoring the UML for different purposes including domain-specific uses. UML profiles are a collection of tagged values, constraints, and stereotypes. Figure 7 illustrates the UML profile created to address the requirements of EmPro models discussed in Section 3.2.2.

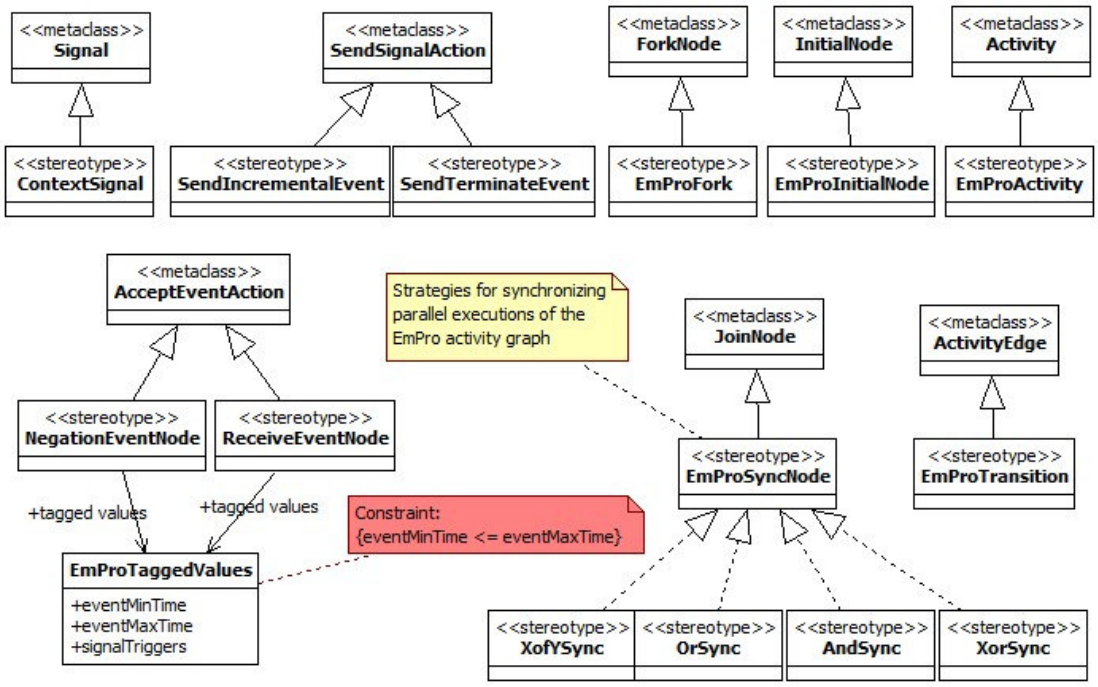

Figure 7: The EmPro UML Profile

The EmPro UML profile, as shown in Figure 7 specifies new stereotypes as well as a set of tagged values, primarily to address the need for modeling timing constraints on EmPro activity nodes. This is in recognition that a context event may only be considered significant if it occurs within a certain time range of another context event. Returning to the "Preparing for work" activity diagram in Figure 6, the "Feed the cat" activity is significant in relation to other context events like "Get out of bed" for detecting whether the person is preparing to go to work, or preparing to go to bed. Thus, a time range needs to be specified on the EmPro model so that the night feeding is not confused with a morning feeding!

3.3.2 Adoption Centric Approach: EmPro Model Design within Existing UML Tools

UML profiles provide a means for customizing existing UML elements. Although this is a powerful feature, it does not go far enough to restrict how the various EmPro-stereotyped UML elements are drawn in all UML drawing tools. It is imperative that the EmPro models constructed using UML syntax has a somewhat predictable serialization structure. Thus, we define a number of EmPro activity syntactical rules that govern how EmPro models are to be constructed.

- Non-EmPro elements cannot be used within an EmPro model. Only elements specified within the EmPro UML profile are allowable.
- UML activity diagrams typically consist of one InitialNode, multiple ActivityNodes and ActivityEdges, and one or more ActivityFinalNode elements. Given the event-based nature of typical ubicomp and context-aware systems, the most semantically appropriate notation for representing the realization of an activity or action within the EmPro model is the UML AcceptEventAction. According to the UML 2 specification [45], an AcceptEventAction is an action that waits for the occurrence of an event, and does not terminate after accepting an event. The EmPro stereotype <<ReceiveEventNode>> AcceptEventAction node, in EmPro semantics, represents the reception of a signal generated by the context-aware system.
- An EmPro model can posses only one <<SendTerminateEvent>> node. The UML does not have the same restriction. Further, the <<SendTerminateEvent>> node replaces the need for an ActivityFinalNode UML element and thus this type of node is not included in EmPro models (an ActivityFinalNode is not required for UML 2.0 diagrams). We chose to utilize the <<SendTerminateEvent>> symbol over the ActivityFinalNode symbol to show that the completion of the EmPro execution will generate a signal.
- All UML JoinNode elements must be stereotyped by a subtype of <<EmProSyncNode>>. Currently, the list of subtypes include <<XofYSync>>, <<OrSync>>, <<AndSync>>, and <<XorSync>>.

The rules outlined above cannot be enforced universally in UML tools used for constructing EmPro models. It is possible within specific UML tools to enforce some of these rules. For example, the open source UML tool StarUML offers many ways in which to customize the tool's user interface based on a UML profile, including new diagram types (e.g., EmPro diagram, extends a UML Activity diagram), new drawing elements (e.g., <<XofYSync>> synchronization node extends the UML JoinNode), and drawing palettes (i.e., collections of new or existing drawing elements). For the purposes of EmPro model design, we customized the StarUML environment with a specialized StarUML drawing palette. Although this prevents users from adding non-EmPro

elements to a particular diagram, thus enforcing rule one, a great deal of further customizations to the StarUML environment would be needed to enforce the other rules (e.g., writing plug-ins or changing the source code for StarUML).

Figure 8 depicts a EmPro model diagram of the “Prepare for work” scenario presented above, as created within StarUML. In accordance with the semantic rules outlined previously, each “Activity” is represented with a <<EmProReceiveEvent>> stereotyped UML AcceptEventAction node. Each <<EmProReceiveEvent>> node consists of a short description of the activity being sensed (e.g., “Exit Bed”) and UML profile tagged values for eventMinTime, eventMaxTime, and triggers. Unfortunately StarUML does not readily display UML profile tagged values; these values can be only accessed by invoking a menu-context item on a selected element.

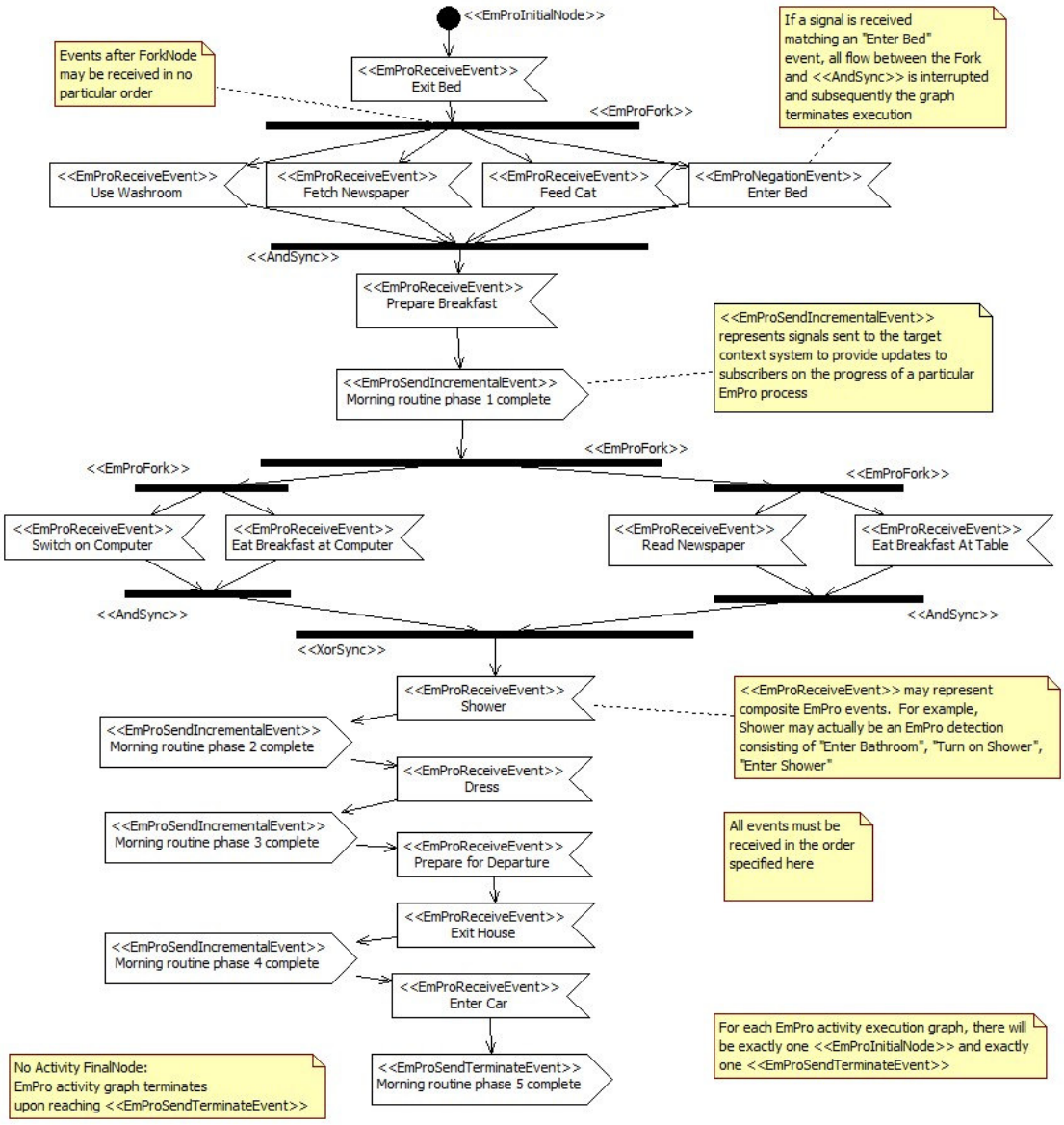

Figure 8: An EmPro representation of "Prepare for work"

3.4 EmPro-Execute

The purpose of the EmPro-Execute module is to provide the software required to execute EmPro activity models (ECM executables) within a context-aware system. The EmPro-Execute module has four basic responsibilities. First, it must be able to handle signals or events generated by the context-aware system, which are used as inputs into the EmPro activity execution engine. Second, it must be capable of maintaining and executing an EmPro activity model using the input signals generated from the context-aware system. Third, it must be capable of generating signals or events that can be

interpreted by the context-aware system so that they can be used by context sensors, by the reasoning/inference engine, or by another EmPro activity execution instance. Finally, the EmPro-Execute module must be capable of being integrated into existing context-aware system, perhaps as part of the reasoning/inference engine, or as the integral component of a *logical sensor*, which is, according to Baldauf et. al., a sensor that combines physical and virtual information to solve higher tasks [46]. In this section we focus on the second requirement – the EmPro activity execution engine; requirements one, three, and four have mostly to do with integration within various context-aware systems, and will be discussed at length in the next chapter.

3.4.1 The Behaviour of EmPro Activity Graphs

EmPro activity graphs are modeled in the UML as activity diagrams with additional properties, constraints, and semantics described in earlier sections. At a fundamental level, an EmPro model is a graph – a collection of nodes, and a collection of transitions which connect pairs of nodes. Each node and transition in an EmPro activity graph have the ability to handle – either consume or transmit – signals received from the context-aware system. Each node maintains an internal state which corresponds with the node's readiness to handle these signals:

- inactive – not ready to consume or transmit the signal. All nodes default to this state when initialized
- active – occurs when the node has been activated by its predecessor node(s) (i.e., the source node(s) on its incoming transition(s)). This indicates that the predecessor node is ready to transmit context signal.
- satisfied – the node will immediately transmit signals received to successor nodes without changing state
- interrupted – the execution path on which this node resides has been interrupted and is waiting for the entire EmPro activity to terminate.

The state of an EmPro activity model is defined by the collective states of its constituent nodes. How each node changes state depends on the type of node. We categorize EmPro nodes into three different groups: *consumer nodes, controller nodes,* and *signal nodes*.

3.4.1.1 Consumer Nodes

Consumer nodes, or nodes that receive and *consume* context-aware signals, include UML AcceptEventAction nodes with the stereotypes <<EmProReceivEvent>> and <<EmProNegationEvent>>. These nodes possess extended values for defining time durations and for defining which context-aware signals are of interest. These nodes enter the *active* state when its predecessor node has entered into a *satisfied* state and explicitly activates the successive consumer node. Thereafter, a consumer node waits for a signal with a specific signature, as defined within the list of triggers specified for the consumer node. When the consumer node receives one of those signals, it *consumes* the signal and enters the *satisfied* state. Thus, For example, referring back to Figure 8 sample EmPro model for "Prepare for work", the "Exit Bed" <<EmProReceiveEvent>> node transitions into the successor ForkNode upon receiving a signal from the context-aware system that corresponds with a sensor responsible for signaling when a particular person leaves his or her bed. Furthermore, the "Exit Bed" node has *consumed* the signal generated by the context-aware system in that it will not pass that signal on to the ForkNode successor at the end of its outgoing transition.

3.4.1.2 Controller Nodes

Controller nodes control the path of execution of the defined EmPro activity model. This group includes UML InitialNode <<EmProInitialNode>>, JoinNode <<EmProSyncNode>> (and all generalizations of this stereotype), and ForkNode <<EmProFork>>. These nodes do not possess any timing attributes nor triggers and thus are dependent on the activation and state of predecessor nodes (i.e., nodes at the source of each incoming transition) for state changes. In the cases of <<EmProInitialNode>> and <<EmProFork>>, the node will never enter the state of *active* because upon activation they change state from *inactive* to *satisfied*. <<EmProSyncNode>> will transition from *inactive* to *active* to *satisfied*, and will enter the *satisfied* state only when the predecessor nodes meet certain satisfaction criteria specific to a synchronization strategy represented by the node. For example, in the case of an <<AndSync>> synchronization node, the node will not become *satisfied* until all predecessor nodes have reached the state of *satisfied*.

3.4.1.3 Signal Nodes

Signal nodes are nodes that send signals to the context-aware system. This group includes UML SendSignalAction EmPro stereotypes <<SendIncrementalEvent>> and <<SendTerminateEvent>>. Like controller nodes and unlike consumer nodes, signal nodes do not possess timing or trigger information. Upon activation, they immediately skip from the *inactive* state into the *satisfied* state.

3.4.1.4 Basic EmPro Activity Flows

To illustrate the basic behaviours of an EmPro activity graph we present Table 2. Table 2 shows how the state of the "Prepared for work" EmPro activity graph changes as it receives various signals from a context-aware system. To simplify the basic flow we omitted timing attributes. Each signal received by an EmPro activity graph as a whole visits all nodes in the *satisfied* state, searching for a node to consume the signal. Upon reaching a node in the *active* state, the signal is either consumed by a consumer node with a matching trigger signature, or it is disposed of.

Signal	Notes
Exit Bed	1. Assumes EmPro activity graph has been previously initialized (i.e., the InitialNode is in a *satisfied* state). 2. "Exit Bed" <<EmProReceiveEvent>> node consumes signal, becomes *satisfied*, and activates successor <<EmProFork>> 3. <<EmProForkNode>> immediately moves from an *inactive* state into the *satisfied* state 4. All successor <<EmProReceiveEvent>> nodes from <<EmProFork>> are activated

Feed cat	1. The “Feed cat” signal visits nodes <<EmProInitialNode>>, “Exit Bed” <<EmProReceiveEvent>>, and <<EmProFork>>. <<EmProFork>> transmits the signal to “Feed Cat” <<EmProReceiveEvent>> node 2. The “Feed Cat” <<EmProReceiveEvent>> node is previously in the *active* state 3. “Feed Cat” <<EmProReceiveEvent>> node consumes the “Feed cat” signal *4.* “Feed Cat” <<EmProReceiveEvent>> node enters the state of *satisfied* *5.* “Feed Cat” <<EmProReceiveEvent>> node activates the <<AndSync>>; the <<AndSync>> node moves from the *inactive* state to *active* 6. The <<AndSync>> node checks predecessor nodes to determine whether it can enter the *satisfied* state. Since only one predecessor is *satisfied*, it remains in the *active* state
Fetch newspaper	1. The “Fetch newspaper” signal visits all predecessor nodes as well as the “Feed Cat” <<EmProReceiveEvent>> node 2. The “Fetch Newspaper” <<EmProReceiveEvent>> node is previously in the *active* state 3. “Fetch Newspaper” <<EmProReceiveEvent>> node consumes the “Fetch newspaper” signal *4.* “Fetch Newspaper” <<EmProReceiveEvent>> node enters the state of *satisfied* 5. “Fetch Newspaper” <<EmProReceiveEvent>> node activates the <<AndSync>> 6. Since the <<AndSync>> is already in the *active* state and not all predecessor nodes are in an *satisfied* state, <<AndSync>> remains in *active* state
Fetch newspaper	1. A second “Fetch newspaper” signal is received by the EmPro activity engine. This may be the result of the sensor sending multiple signals to the context-aware system because, for example, the user stands in one spot too long because he or she has become distracted by the newspaper headline 2. The “Fetch newspaper” signal visits all predecessor nodes as well as the “Feed Cat” and “Fetch Newspaper” <<EmProReceiveEvent>> nodes ; no change of state is enacted

Use washroom	1. The "Use washroom" signal visits all predecessor nodes as well as the "Feed Cat" and "Fetch Newspaper" <<EmProReceiveEvent>> sibling nodes 2. Previously in the *active* state, the "Use Washroom" <<EmProReceiveEvent>> node becomes *satisfied* 3. "Use washroom" <<EmProReceiveEvent>> node activates the <<AndSync>> 4. Since the <<AndSync>> is already in the *active* state and not all predecessor nodes are in an *satisfied* state, <<AndSync>> remains in *active* state
No Signal for Enter Bed	1. The "Enter bed" signal is not received from the context-aware system. The <<EmProNegationEvent>> is in an *active* state until the eventMaxTime (a timing *attribute*) duration expires 2. Once the eventMaxTime expires, the <<EmProNegationEvent>> node becomes *satisfied* and sends an activation notice to the <<AndSync>> node 3. The <<AndSync>> node, upon being activated by the "Enter Bed" <<EmProNegationEvent>> will check whether all predecessor nodes are in a *satisfied* state. If so, the <<AndSync>> node becomes *satisfied* and sends an activate notice to the "Prepare Breakfast" <<EmProReceiveEvent>> node
Prepare breakfast	1. The "Prepare breakfast" signal visits all predecessor nodes 2. Previously in the *active* state, the "Prepare breakfast" <<EmProReceiveEvent>> node becomes *satisfied* 3. "Prepare Breakfast" <<EmProReceiveEvent>> node activates the "Morning routine phase 1 complete" <<EmProSendIncrementalEvent>> node 4. Upon the receiving the activation notice, the "Morning routine phase 1 complete" <<EmProSendIncrementalEvent>> node sends a "morning routine phase 1 complete" signal to the context-aware system, and enters the *satisfied* state

Table 2 : EmPro activity graph behaviour with given sample inputs

The basic flow of an EmPro activity graph shown in Table 2 focused on the sequential or ordered characteristics of an EmPro model. As stated earlier, an important functional aspect of EmPro models is that it allows users to model *temporal proximity* of related activities. We model temporal proximity of activities using time period attributes on the

consumer nodes. For each consumer node, an *eventMinTime* and *eventMaxTime* long integer must be specified. The eventMinTime attribute corresponds with the time when a consumer node becomes ready to accept and thus consume signals matching one of the signatures in its constituent trigger set. Each consumer node possesses a timer which is initialized when the node is activated (i.e., when its predecessor nodes sends an activation notice). If a signal is received before the specified eventMinTime, the signal is ignored. The eventMaxTime attribute specifies the time after which the node either becomes invalid/*interrupted* (in the case of an <<EmProReceiveEvent>>) or *satisfied* (in the case of an <<EmProNegationEvent>>).

An <<EmProReceiveEvent>> node enters the state of *interrupted* when time passes beyond the eventMaxTime and no matching signal signature is received. This allows temporally related activities such as “Feed cat” and “Fetch Newspaper” to be linked by time by setting an overlapping time period and duration. Although the “Feed the cat” and “Fetch newspaper” activities are completely independent activities, a relationship can be drawn between the two based on a related time interval set by a common predecessor event (e.g., “Exit bed”). If during the execution of the context-aware system, the “Feed the cat” and “Fetch newspaper” activities are conducted outside of the common time period specified in the EmPro activity model, or one or both activities are not undertaken (as sensed by the context-aware system), this indicates that the relationship may not be exist at a particular time.

3.4.1.5 Interrupts

An EmPro activity graph is *interrupted* whenever an exceptional event takes place during the execution of graph. There are currently three instances when this may occur and have been alluded to earlier:

1. When the *eventMaxTime* on an <<EmProReceiveEvent>> expires
2. When an <<EmProReceiveEvent>> node receives a signal before the *eventMinTime* attribute value has elapsed. This currently is not used but may be needed for certain scenarios in the future
3. When a <<EmProNegationEvent>> node has received a signal matching one of its set of trigger signatures before the *eventMaxTime* period has expired

When an interrupt is generated by an EmPro activity graph element, the interrupt is propagated to all downstream successor nodes until it reaches the terminus <<EmProSendTerminateEvent>> node, whereupon the graph itself is disposed by the context-aware system. Thus, the reception of an interrupt signifies the end of execution of the graph.

3.4.1.6 EmPro-Execute Design and Implementation

In this section we describe the design and implementation of the EmPro-Execute module. EmPro-Execute is implemented in the Java programming language, and as a result of this selection of programming language is object-oriented in nature. In Figure 9 we present a UML Package diagram of the EmPro-Execute module.

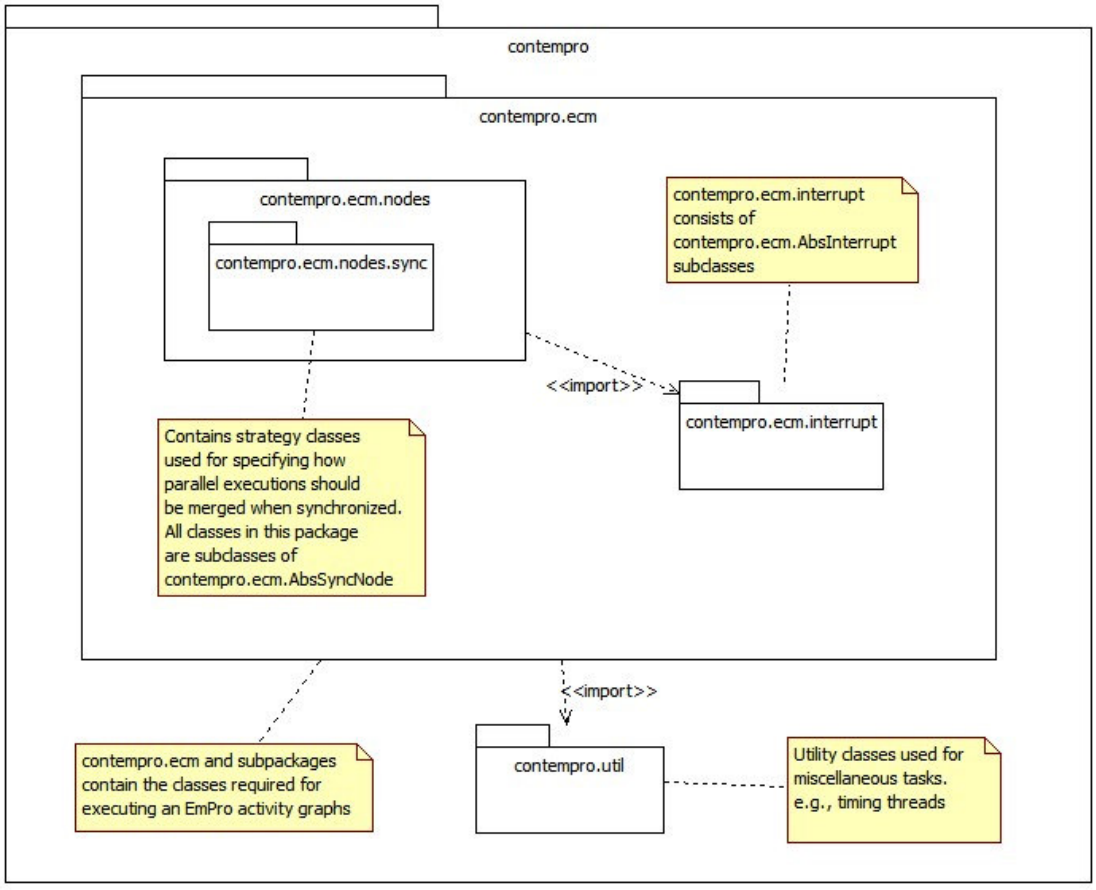

Figure 9: Package diagram contempro

The *contempro.ecm* family of packages contains all the classes required for executing an EmPro activity graph. Figure 10 is a UML class diagram for *contempro.ecm*, the root package for execution engine. This package contains mostly abstract classes and

interfaces to provide the building blocks for all EmPro model elements (all of which implement *IEmProGraphElement*), including nodes (*AbsEmProNode*) and transition. Figure 10 also illustrates the relationship between the EmPro ECM to the context-aware system; the latter must provide an implementation of the *ISignalManager* interface which is responsible for sending signals to the *Activity* class and for receiving incremental and termination signals from the EmPro model.

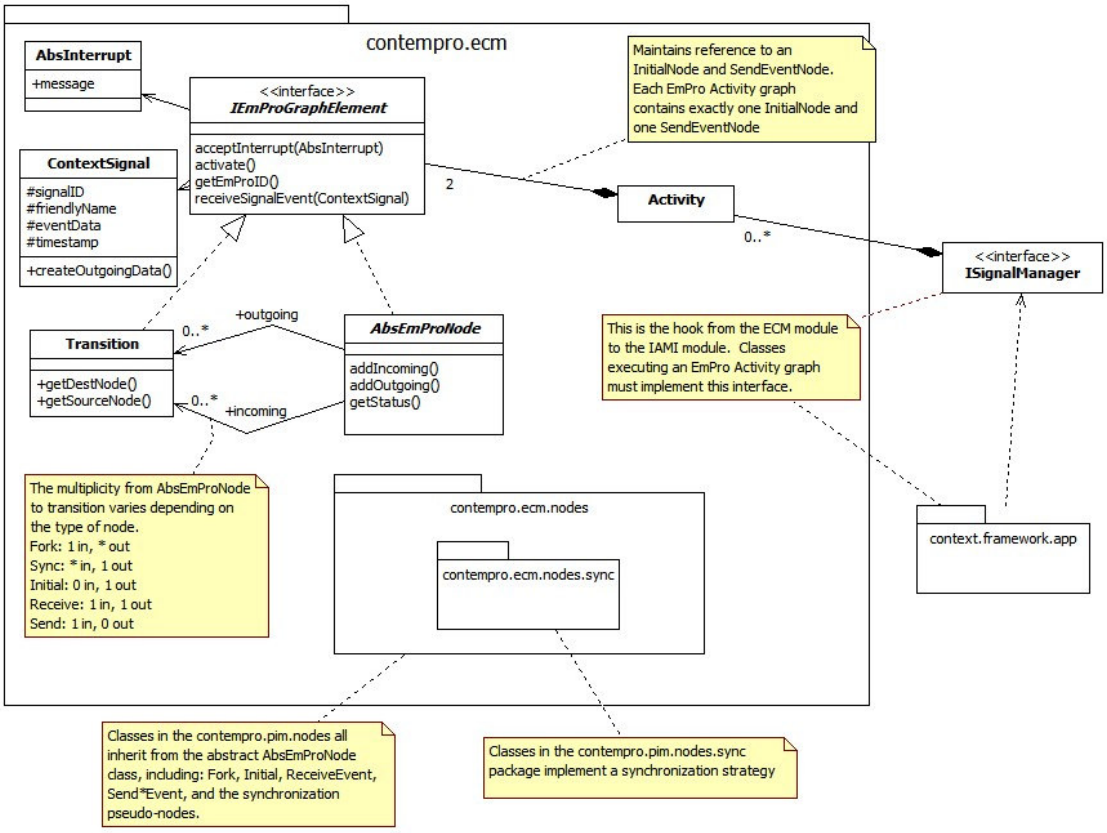

Figure 10: Class diagram for the *contempro.ecm* package

As specified in the interface *IEmProGraphElement*, each EmPro graph element (transitions and nodes) must accept activation notices, interrupts, and signals. EmPro transitions contain very little logic and act primarily as conduits for transmitting interrupts, signals, and activation calls between nodes.

Figure 11 is a UML representation of the *contempro.ecm.nodes* package. All EmPro nodes derive from *contempro.ecm.AbsEmProNode*, which implements generic methods for handling incoming and outgoing transitions. Controller nodes are differentiated from consumer nodes in that consumer nodes must inherit the *AbsSignalConsumer*

implementation. The *AbsSignalConsumer* abstract class specifies behaviours specific to consumers that must be overridden from *AbsEmProNode*, like activate() and a method invoked by the timer when the *eventMaxTime* duration expires.

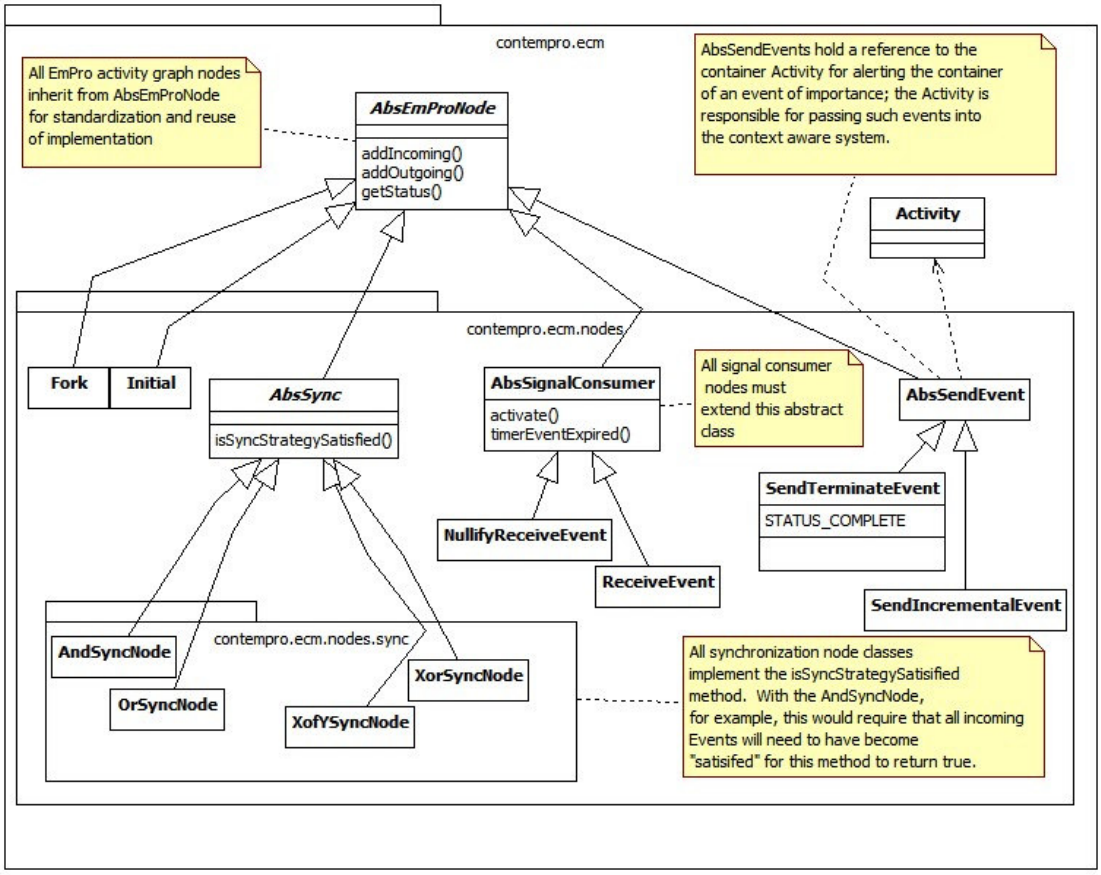

Figure 11: Class diagram for *contempro.ecm.nodes*

The *contempro.ecm.nodes.sync* package holds a collection of strategy classes used for synchronization parallel EmPro activity graph flows. Currently we have implemented four synchronization strategies:

- *AndSyncNode* – all predecessor nodes must be in the satisfied state before this node becomes satisfied
- *OrSyncNode* – any one of the predecessor nodes must be satisfied before this node becomes satisfied. It will become satisfied upon first activation
- *XofYSyncNode* – will become satisfied when at least x number of predecessor nodes is satisfied (y is the total number of predecessor nodes)

- *XorSyncNode* – will become satisfied when exactly one of the predecessor nodes is satisfied. It waits until all predecessor nodes have expired and have sent interrupts.

3.5 Related Works

According to a survey compiled by Strang et. al., in [33],existing context models range from the very simple (e.g., key-value models) to the sophisticated (e.g., ontological approaches). It is evident from this survey and from the body of context-aware literature that existing models do not allow for the explicit representation of the dynamic aspects of context interactions, e.g., process-flow, sequential, or time-based context entity interactions. From a software developer's perspective this would be analogous to depriving the UML of state-chart, collaboration, sequence, and activity modeling diagrams. Most existing context models focus on the structural, static features of entity relationships (e.g., "Building X consists of Y floors"), leaving the responsibility of building time-elapsed entity interaction pseudo-models to a context reasoning system or to an abstract, not-yet implemented context history activity recognizer.

It would appear that our notion of an EmPro closely resembles the established ubiquitous computing research topics of human activity model mining (e.g., [47], [48]) and context history (e.g., [30], [49]). Our approach differs from these in that we advocate a human-initiated model, whereas activity model mining and context history approaches seek to remove the need for human intervention wherever possible by using machine-learning and AI techniques. We advocate a human-initiated design of EmPro's for three reasons. First, it is difficult to incorporate into AI systems expert knowledge from researchers from non-computer related disciplines – experts on human behaviour, for example – who may infuse models with valuable insight. Second, an EmPro may capture an anti-pattern – an interaction pattern that indicates aberrant or contrarian user behaviour, like in a security threat – for which AI training data is not readily available. Third, although AI-based approaches should admittedly play an important role in detecting and recognizing human activity in the future, such approaches at present are not mature and widely available. At the very least, the EmPro modeling system should be a useful tool for prototyping context-based interactions in existing context-aware systems. Furthermore, we expect that future context-aware systems will need both explicit and AI-

based activity modeling methods as they can complement one another; for example, machine-learning may be used to refine a human-designed EmPro and human input may be used to add generality to models synthesized based on observed training-data.

Many existing context-aware projects have utilized user agendas/schedules as well as time data as context information (e.g., [50], [31]). Our project differs from these in that we are more interested in the sequence and temporal proximity in which context events occur rather than the specific time they occur (although the specific time may be important to a particular EmPro modeler).

3.6 Chapter Summary

In this chapter we described our implementation of an ECM capable of detecting activity processes, which we call ContEmPro (Context-aware Embodied Processes). The ContEmPro software system consumes events from a given context-aware system and evaluates whether specific processes are detected based on the sequence and time proximity of the events consumed. The EmPro model, defined in a UML 2.0 profile-extended activity graph, can be constructed using any UML 2.0 compliant design tool. In anticipation of the next chapter, the ContEmPro system was designed to be context-aware middleware agnostic; all inputs are implemented as generic types and are to be type instantiated when ContEmPro is instantiated in 'live' context-aware system. In the next chapter, we describe how we integrated ContEmPro into two separate context-aware middleware systems: the Context Toolkit [6] and the Java Context-Aware Framework [7].

4 The IAMI Approach

4.1 Introduction

In this chapter, we describe our approach to integrating foreign *embodied context models* (ECM) into existing context-aware systems. Many have acknowledged that ubiquitous context-aware computing applications are difficult to implement [5, 6, 51]. Chapter 2 discusses many of the challenges which impede the development of ubiquitous, context-aware applications. Numerous context-aware middleware systems have been proposed to address one or more of the challenges in creating context-aware software applications, including [2, 6-8, 10, 18]. Based on our review of the context-aware and ubicomp related literature, few of these middleware systems are leveraged in the construction of new context-aware systems. This lack of reuse is evident in our review of thirteen of the top-cited (according to Google ™ Scholar on February 14th, 2008) context-aware papers published by the IEEE between 2005 and 2007 (see [52-64]). None of these papers describe the use of third-party or external context-aware middleware. The closest any of these projects came to reusing third party context-aware middleware was the system discussed in [64], which was built on a well-known context-aware middleware (GAIA [65]) developed many years prior by the same research group but with mostly different people involved. Many of the projects described in the thirteen papers were based on multi-purpose (i.e., not context-aware) middleware, such as XML Web services (e.g., [63]), CORBA (e.g., [58]), and OSGi (e.g., [56]). This review of recent literature accords with our readings of context-aware research literature prior to 2005 in how little software is shared between context-aware research initiatives.

We surmise two reasons for the lack of external reuse of context-aware middleware. First, few of these proposed middleware systems become accessible to outsiders. The software artefacts constructed to support the middleware system live and die within the research laboratory – they are constructed solely to achieve a research goal. To our knowledge, the only two exceptions to this trend are the Context Toolkit [6] and the Java Context-Aware Framework (JCAF) [7].

Second, new context-aware middleware systems are constructed to address a perceived or real deficiency in existing middleware approaches. Ubicomp technologies tend to be far more diverse than technologies for desktop/laptop and server computers in terms of operating systems and technical capabilities. A particular context-aware system/middleware may be constructed specifically for a particular hardware type ([66]), a particular programming language ([7]), a architecture design ([57]), a specific context model format (e.g., the OWL ontology used in [2]), reasoning engine [67]), and infrastructure type (e.g., OSGi infrastructure in [68]).

Furthermore, as we pointed out in Chapter 2, there are no widely accepted standards specific to the context-aware computing community. The lack of a supporting foundation for context-aware computing creates significant challenges for those wishing to investigate the context-aware computing research area. It is difficult to prototype new context-aware applications and to test new ideas without this supporting infrastructure. The most common method for dealing with this issue, according to our review of the literature, is to create the middleware infrastructure anew to suit the purpose of the application or the ideas to be investigated, a process that can be very time consuming and requires significant resources.

ECMs, discussed in-depth in Section 2.7, are context entities representing a specific context concern. As sensing technologies become more sophisticated, increasingly complicated ECMs will emerge, and thus it will be near impossible to anticipate all possible ECMs created in the future. The EmPro model, as described in the previous chapter, is also an example of an ECM that is not readily represented in context-aware systems currently implemented.

We propose the IAMI approach (Interpret, Adaption, Model, and Integration) to reusing context-aware middleware, for situations when the target middleware does not functionally support an ECM of interest. In the next section we describe the IAMI approach in detail. In Section 4.3 we describe how we designed our software to support the IAMI approach. In Sections 4.4 and 4.5 we provide specific details on how we implemented the IAMI approach with regard to two context-aware middleware systems, the Context Toolkit and the JCAF.

4.2 The IAMI Procedure

The IAMI procedure is a "bottom-up" approach to modeling ECMs, starting first with an examination of data and services capabilities of an existing context-aware system. The following is a summary of the steps included in the IAMI:

1. Inspection. IAMI gathers meta data on *in situ* devices and sensors within the environment of interest, with particular emphasis on the types of data sensed and events generated by these devices and sensors. The meta data may be gathered at system runtime or at design time. If at system runtime, the information may be gathered from various components of the running context-aware system (e.g., from context ‘widgets’, context ‘entities’, etc.). Otherwise, depending on the design of the context-aware middleware, this information may be obtained from one or more context-aware system configuration files, by inspecting the source code for the context-aware system, or a combination of the two (e.g., viewing the source code for all the Context Toolkit ‘widgets’ utilized in a context-aware environment and examining the build.xml ant [69] configuration file).
2. Adaptation. An adapter layer is created within the context-aware system network. These adapters are created manually by the designer in the format/platform of the context-aware system. IAMI adapters are designed to listen to specific events from one or more context 'widgets' (or other middleware components) and can be translated into something of greater semantic value to an ECM executable. This relieves ECM executables from having to know how to translate middleware-specific events. With respect to the ContEmPro ECM, an adapter for a light intensity widget might be only be interested when the intensity of a particular light is 0%, i.e., when the light is completely switched off, signifying an EmPro activity "Switch off bedroom light". Another adapter may be set up to listen specifically for when the light intensity of that same light source hits 100%, signifying a "Switch on bedroom light" EmPro activity.
3. Model. The adapters created in the previous step are consumable for the ECM. In the example of our EmPro ECM, each adapter is representative of an EmPro

activity, e.g., "Enter bedroom", "Switch on bedroom light", "Put on slippers", etc. These adapters are thus the building blocks of the ECM.

4. Integration. The models are transformed into executable programs and then integrated into the target context-aware system using middleware specific mechanisms. This process includes, (1) embedding the model within middleware specific component so that it can be executed, (2) listening or subscribing to events generated by the adapters, and (3) publishing all the events that can be generated from executing the model which in turn can be consumed by the context-aware middleware of concern.

This provides a high-level view of the IAMI procedure. In the next section, we present a more detailed view of how we implemented the IAMI with respect to the ContEmPro software system.

4.3 Design of the IAMI in ContEmPro

The software design of an implementation of the IAMI approach will vary according to the input requirements of the ECM execution module being represented and the type of context-aware middleware being targeted. It is impossible to envision all the possible combinations of ECMs to target context-aware frameworks. The implementation of ContEmPro provided hereafter serves as an example implementation of the ECM execution module of the IAMI approach.

In a nutshell, the IAMI mitigates communications between a given context-aware system and ECM executables. The IAMI adapter layer subscribes to events generated by the context-aware middleware, affixes or decorates the event with ECM-specific information, then passes the event into one or more ECM executables for evaluation. It is also responsible for sending middleware compliant events generated by the ECM executable. Figure 12 shows the software framework of the ContEmPro IAMI implementation. Within the ContEmPro system, the *AbsEventAdapter* is the base class used for creating all the elements within the IAMI adapter layer. The *contempro.ecm.Activity* class is the ContEmPro implementation of an ECM executable.

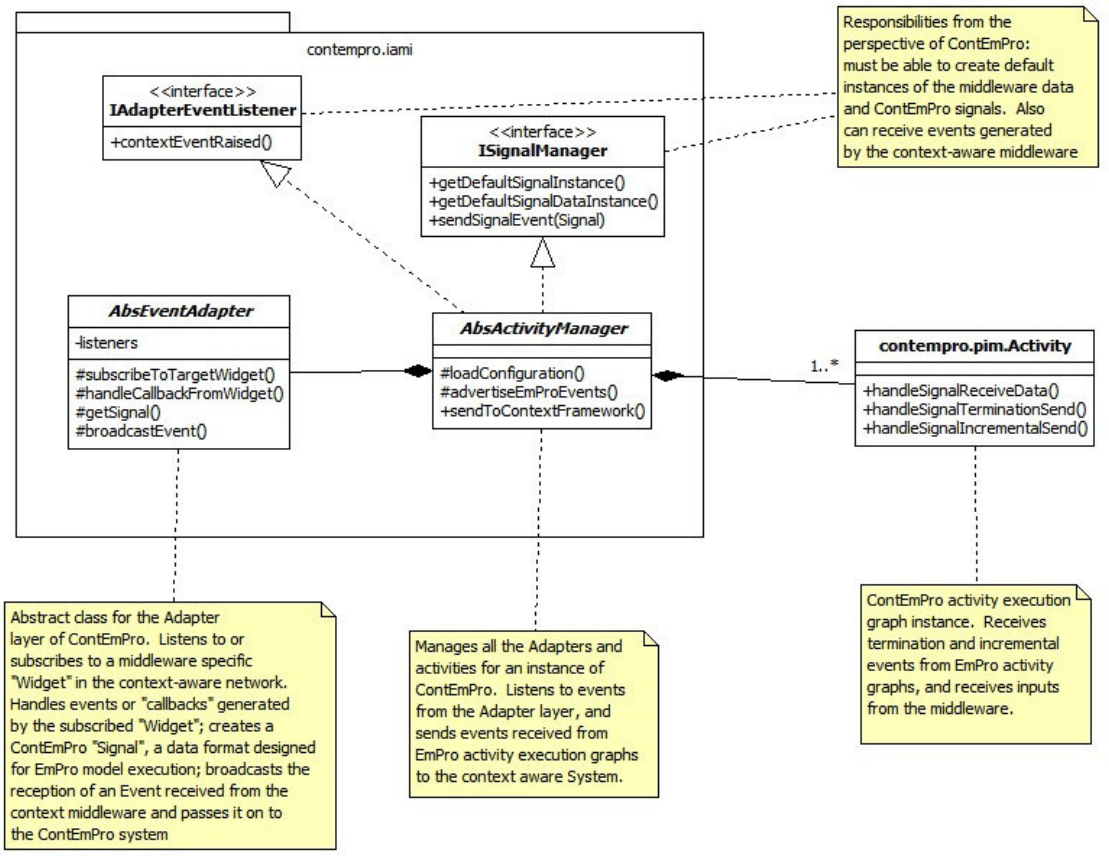

Figure 12: IAMI abstractions in contempro.iami

We summarize the elements of the ContEmPro IAMI implementation below:

- *AbsEventAdapter*: Receives events from a context-aware system and transforms the event (i.e., adds annotations or decorations) into something that can be used by one or more contempro.ecm.Activity instances (i.e., ECM executables).
- *AbsActivityManager*: Manages one or more instances of *contempro.ecm.Activity* (the ContEmPro ECM executable), and listens to events received by AbsEventAdapter instances, then passes events to contempro.ecm.Activity instances. It also sends middleware-compliant messages to the middleware instance as generated by the *contempro.ecm.Activity* instance.
- *contempro.ecm.Activity*: Receives events passed on from the IAMI adapter layer and uses the events to detect EmPro occurrences. When EmPro occurrences are detected, it sends events back to the *AbsActivityManager*, an implementor of the *ISignalManager* interface.

- *ISignalManager*: Implementors of this interface guarantee the ability to create middleware-specific signals/events and message data types

Figure 13: IAMI sequence illustrating an incoming event from the context-aware systemFigure 13 is a sequence diagram illustrating how the ContEmPro IAMI module handles events received from a given context-aware system. The AbsEventAdapter and AbsActivityManager are subclassed to a specific middleware implementation.

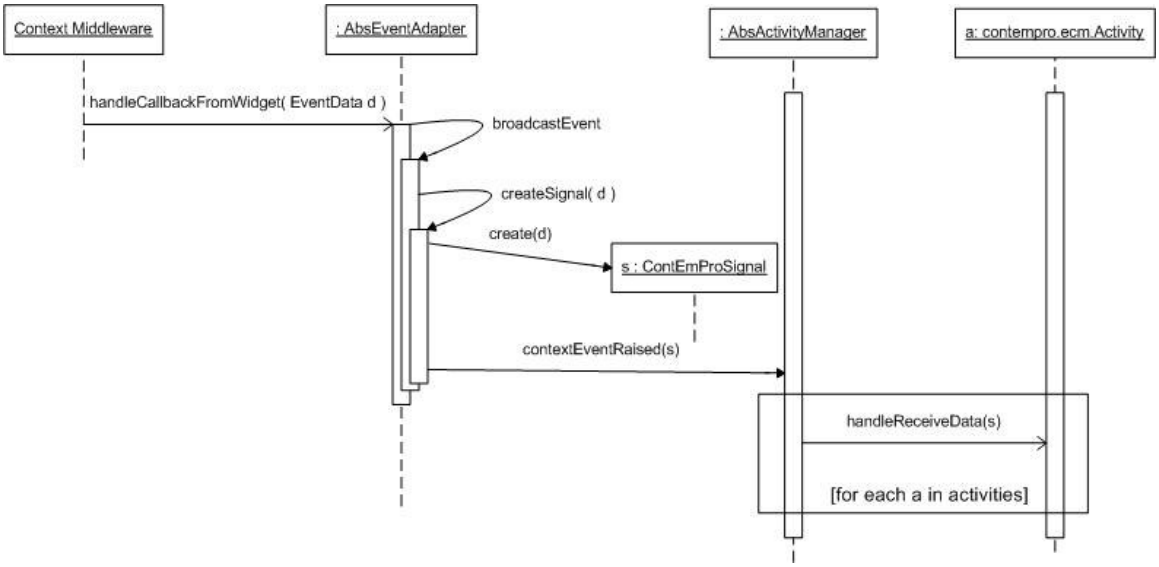

Figure 13: IAMI sequence illustrating an incoming event from the context-aware system

In the following sections we describe how we implemented the IAMI approach for two separate types of context-aware middleware, the Context Toolkit and the Java Context-Aware Framework (JCAF).

4.4 Context Toolkit Integration

The Context Toolkit, designed in the late 1990's and early 2000's, is probably the best-known context-aware computing software initiative. It was designed to promote a separation of concerns between how context information is acquired and how it is ultimately used within a context-aware application. To promote this separation, the Context Toolkit provides a framework for implementing context "widgets", a concept homologous to the graphical user interface control widget (e.g., button, drop down list, etc.). According to [6], context widgets, like graphical user interface widgets, hide the low level details of how information is retrieved and allows context-aware application

designers to focus on application specific details rather than sensor issues. Here is a summary of all first-class entities in the Context Toolkit framework:

- Widget: acquire context data from sensors and provide a uniform interface for accessing this data
- Interpreter: performs transformations on data exposed by one or more widgets, i.e., to give the data more semantic value, such as translating a GPS coordinate to a room number
- Aggregator: aggregates all the context data for a given entity (person, place, or thing)
- Discoverer: a registry of all the components (widgets, aggregators, interpreters) in a given context-aware network

Figure 14 illustrates a basic interaction scenario between applications and components in the Context Toolkit.

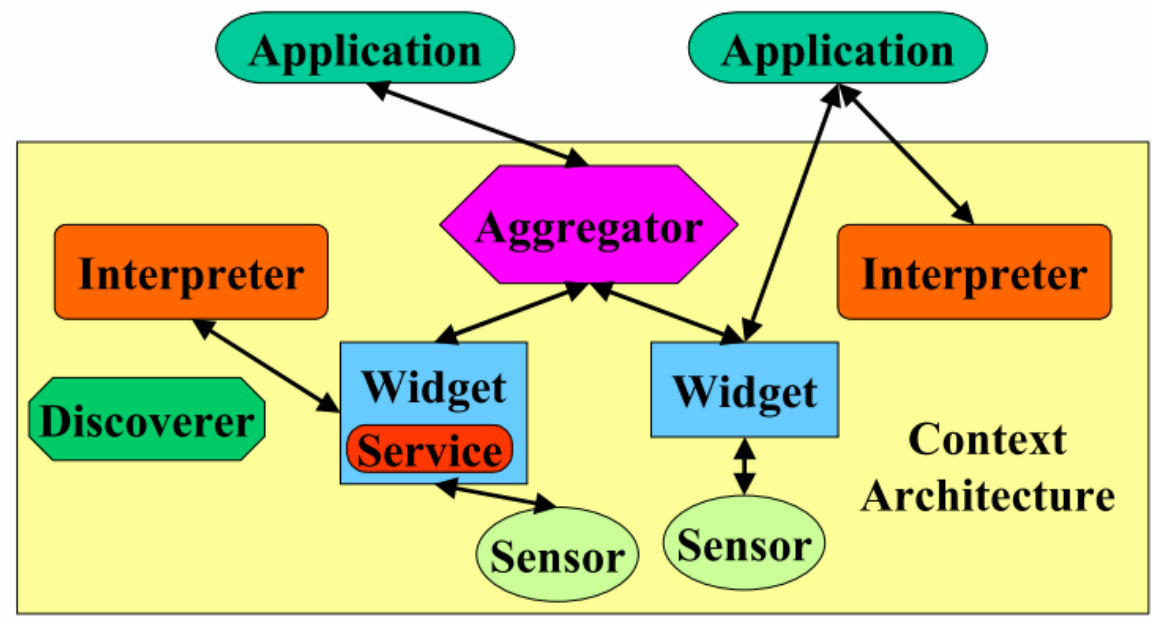

Figure 14: Sample layout of a Context Toolkit application scenario

The design of our ContEmPro IAMI module is depicted in Figure 15.

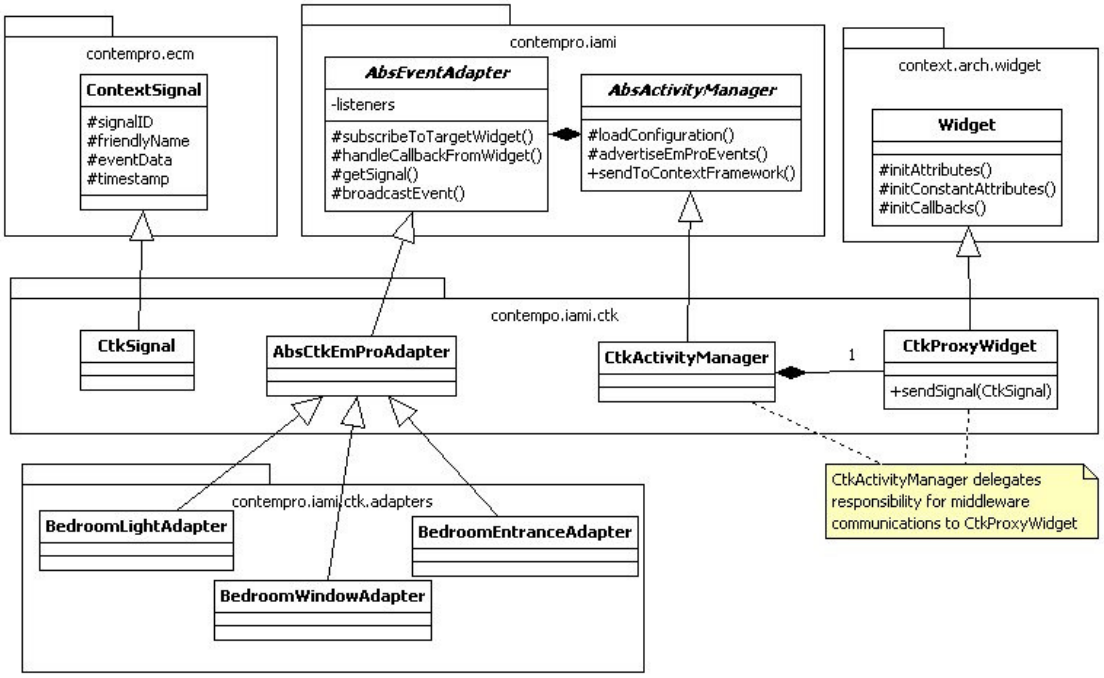

Figure 15: IAMI approach targeting the Context Toolkit

In the following sections we describe aspects of the design depicted in Figure 15 as we step through our implementation of the ContEmPro IAMI module targeting the Context Toolkit.

4.4.1 Context Toolkit-ContEmPro: Inspection

A ContextToolkit *Discoverer* instance maintains a list of all Context Toolkit component instances active in a given physical environment. This list can be accessed by sending a query to the *Discoverer* instance and thereafter inspecting the *ComponentDescription* result of the query. To retrieve a list of all the widgets managed by a *Discoverer,* the code displayed below shows the query that can be issued to the *Discoverer* instance:

```
 // create query item; retrieves all items in the discoverer registry by getting all items
// that are different than "fakeelement" -- should return all
AbstractQueryItem qForWPNP2 = new QueryItem(new IdElement("fakeelement"), new Different() );

// retrieves a vector of context.arch.discoverer.ComponentDescription objects
// server has been initialized earlier with basic context toolkit attributes
// and a discoverer has been initialized and found
Vector res = server.discovererQuery (qForWPNP2);

// obtain iterator
java.util.Iterator iterator = res.iterator();

while ( iterator.hasNext() ) {
```

```
        ComponentDescription cd = (ComponentDescription) iterator.next();

        // just dump out all attributes for this toolkit object
        System.out.println( "Attributes: " + cd.getAllAttributes().toString() );
}
```

Alternatively, the *Inspection* process could be achieved by reading a configuration file, one that contains all the network names, ports, class names, and attributes for each component to be deployed in a physical environment.

4.4.2 Context Toolkit-ContEmPro: Adaptation

The data received from a context-aware middleware is not always at a level of granularity or of sufficient semantic value for use within an ECM executable. The ECM executable is designed to be middleware-agnostic and therefore does not necessarily have the capacity to do logical comparisons on data received from the middleware (e.g., temperature reading is greater than 30 degrees Centigrade). The responsibility for translating context-aware events into something semantically valuable (from the perspective of the ECM) resides within the IAMI adapter layer. For example, an adapter to the Context Toolkit's *context.widgets.WTemperature* sample widget might translate an event received with a reading of 30 degree Centigrade to "Bedroom is hot", or "Bedroom is cold" when the temperature dips below 18 degrees Centigrade. The adapter could then select an event to generate for the ECM executable layer, e.g., generate a "Bedroom is hot" event, "Bedroom is a comfortable" event, or "Bedroom is cold" based on the temperature value.

Since IAMI adapters are designed to listen to middleware-specific events and evaluate specific conditions on the data exposed by the events, the IAMI adapter layer is deployed within the realm of the middleware. That is, adapters must be able to communicate directly or indirectly with the middleware and understands the data structures used within the middleware.

An individual IAMI adapter is responsible for four tasks. First, it *subscribes* to or listens to one or more events generated within the context-aware middleware instance, e.g., adapter x listens for event y on widget z. Second, when an event of interest is generated, an IAMI adapter knows how to *receive* such messages in a context-aware middleware way, e.g., it implements an interface exposed by the context-aware

middleware for handling the reception of such events. Third, an adapter holds the logic for *determining* what information should be passed to the ECM module. Finally, an adapter is responsible for *transmitting* an event into the ECM module, thus completing the 'adaptation' or transformation of the event into the ECM module. We describe each step in the following sub-sections.

4.4.2.1 Subscription

An IAMI adapter overrides the AbsEventAdapter.subscribeToTargetWidget method. It is within this method that an IAMI adapter will subscribe to events generated by one or more context-aware middleware "widgets" or other components (e.g., Context Toolkit *Interpreter* or *Aggregator*, etc.). An IAMI adapter may subscribe to one or more middleware components or no components at all. The code written in this method will be manually entered by the designer using libraries and data structures offered by the middleware. In the example of the ContEmPro project, the code for an event subscription from the Context Toolkit *edu.berkeley.io.context.home.LightWidget* for detecting a light is switch-off is represented in the below code listing.

```
public void subscribeToTargetWidget() {
        context.arch.subscriber.ClientSideSubscriber subscriber =
         new ClientSideSubscriber(this._server.getId(), this._server.getHostAddress(),
                  this._server.getPort(), this.getSubscriptionType( ),
                  this.getQueryItem(), this.getQueryAttributes( ));

        Error response = this._server.subscribeTo(this, this._sourceWidget.getId(),
                           this._sourceWidget.getHostName(), this._sourceWidget.getPort(), subscriber);

        if ( !response.getError().equals(Error.NO_ERROR)) {
                  System.out.println( "Error received: " + response.getError());
                  return;
        }
        this._subscriberID = subscriber.getSubscriptionId();
}
```

The Context Toolkit implementation of the c*ontext.arch.subscriber.ClientSideSubscriber* component shown above allows the subscriber client to specify the conditions for being sent a notification of an event. For example, the adapter could specify that it only wish to receive "Light switched off"

events from a widget handling a specific light source. We could specify the following query conditions, as referred to in the previous code listing.

```
protected Attributes getQueryAttributes() {
        Attributes atts = new Attributes();
        atts.addAttribute( LightWidget.INTENSITY, Attribute.FLOAT);
        return atts;
}

protected AbstractQueryItem getQueryItem() {
        AbstractDescriptionElement intensityCondition =
                new NonConstantAttributeElement( LightWidget.INTENSITY, "0", Attribute.FLOAT );
        return  new QueryItem( intensityCondition, new
        context.arch.discoverer.querySystem.comparison.LowerEqual() );
}
```

The *getQueryAttributes* method specifies the widget attribute in which we are interested, whereas the *getQueryItem* method allows us to define logic the widget will use for evaluating whether the subscriber should be notified of an event. In this case we are saying we are interested in receiving events when the intensity of the light source is lower or equal to zero.

4.4.2.2 Receiving an Event from Widget

When an event is generated from a widget, an IAMI adapter must receive the event in a context-aware middleware specific way. The adapters in the IAMI module targeting the Context Toolkit must directly or indirectly implement the *context.arch.handler.Handler* interface to receive such events. The *context.arch.handler.Handler* interface specifies two methods; the one of interest for our purposes is the *handle* method. In the below code listing, we provide sample code for handling a Context Toolkit event.

```
public DataObject handle(String subscriptionId, DataObject data)
                throws InvalidMethodException, MethodException {

if ( !this._subscriberID.equals(subscriptionId)) {
        String errorMsg = "Subscriber IDs do not match.  Received "  +
                subscriptionId + "; Expected " + this._subscriberID;
        this._server.println(errorMsg);
```

```
            // this may be a resonse from the discoverer and
            // thus isn't necessarily a bad response.
            return data;
    }

    CtkSignal signal = this.determineIncomingSignal(data);
    // if signal is null we aren't interested in this event
    if ( signal == null )
            return data;
    // call the contempro method for handling events
    this.broadcastReceiveEvent(signal);

    // required by ctk, not sure why
    return data;
}
```

Within the implementation of the *handle* method shown above we call two methods, *determineIncomingSignal* and *broadcastReceiveEvent*. These methods correspond with the last two of four responsibilities of an IAMI adapter: determining which signal (if any) to pass on to the ECM module, and, actually transmitting the signal to the ECM module. We discuss these two steps in the next sections.

4.4.2.3 Signal Determination

An IAMI adapter may 'adapt' one or more context-aware system events and may wish to apply logic to determine whether or not an event should be transmitted into the ECM module. The abstract method *determineIncomingSignal* is responsible for determining which *contempro.ecm.ContextSignal* (if any) should be passed into the ECM module. Further, it is responsible for ensuring the appropriate ECM-specific attributes are affixed to the signal and passed on to the ECM layer. In the case of ContEmPro, the only attributes passed from the adapter layer to the ECM layer are: (1) the signal id (i.e., a global unique identifier as a *java.lang.String*), (2) the signal's human readable name (e.g., "Switch off light"), and (3) a time stamp representation of when the signal was received. The code listing below illustrates the code for performing this task within ContEmPro and the Context Toolkit.

```
protected CtkSignal determineIncomingSignal(DataObject data ) {
    Attribute att = new Attribute( data );
    context.arch.storage.Attributes atts = att.getSubAttributes();
```

```
    try {
        context.arch.storage.AttributeNameValue intensity =
                atts.getAttributeNameValue(LightWidget.INTENSITY);
        float val = Float.parseFloat(intensity.getValue().toString());
       // check if intensity is greater than 0
        if ( val == 0f ) {
                return new CtkSignal(data, BedroomLightAdapter.BEDROOM_LIGHT_OFF_ID,
                this._signals.get(BedroomLightAdapter.BEDROOM_LIGHT_OFF_ID) );
        } else {
                return new CtkSignal( data, BedroomLightAdapter.BEDROOM_LIGHT_ON_ID,
                this._signals.get(BedroomLightAdapter.BEDROOM_LIGHT_ON_ID));
        }

        } catch ( Exception ex ) {
        throw new java.lang.RuntimeException( "Could not handle callback");
    }
}
```

In the above code listing, two conditions are tested and two possible *CtkSignal* instances (EmPro event) can be created. There is nothing to restrict the number or types of signals that can be broadcast to the AbsActivityManager(s).

4.4.2.4 Signal Transmit

As shown in the previous code listing, if the signal returned from the *determineIncomingSignal* is null, the event is ignored. But if an instance of *contempro.iami.ctk.CtkSignal* is returned from the *determineIncomingSignal* method, it is transmitted or broadcast to its list of *contempro.iami.IAdapterEventListener* instances. Typically there will be only one listener in this list, of type *contempro.iami.AbsActivityManager*, or in the case of ContEmPro integration with the Context Toolkit, an instance of *contempro.iami.ctk.CtkActivityManager*. The code for implementing the transmission of signals from the adapter to the ECM module resides in the *contempro.iami.AbsEventAdapter* class and is as follows.

```
// each listener implements the contextEventRaised method
protected void broadcastReceiveEvent( S signal ) {
        for ( IAdapterEventListener<S,D> listener : this._listeners ) {
                listener.contextEventRaised(signal);
        }
```

```
}
```

4.4.3 Context Toolkit-ContEmPro: Model

After the IAMI adapter layer has been created, ECM models can be constructed using adapter implementations as building blocks. The modeling of ECMs is less of a software infrastructure issue and more of a visual design issue. Furthermore, the requirements of the Modeling step are ECM-specific. In this section we present our ideas on how to construct EmPro model executables. The elaborateness and sophistication of the tools used for constructing ECM executables is dependent on how complicated the ECMs are to construct and the personal tastes of the ECM designer.

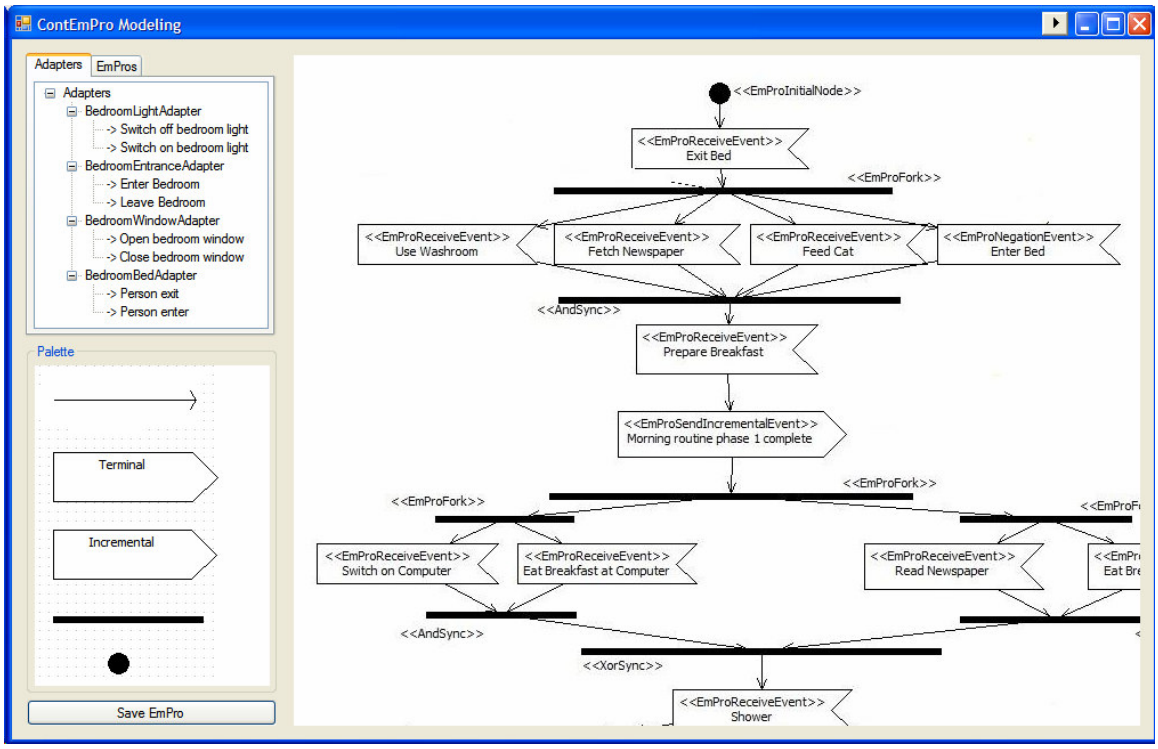

Figure 16: Prototype screenshot of IAMI model step

Figure 16 is a mock-up screen of a tool for creating EmPro model executables. This mockup consists of three areas. On the right side is the EmPro model drawing canvas, which allows users to construct EmPro models by dragging various elements onto the canvas. It also allows various graph elements to be configured, e.g., change an EmPro synchronization from <<XorSync>> to <<AndSync>>. On the top left side under the "Adapters" tab is the list of adapters configured in the previous IAMI step. The child nodes of each adapter are the *CtkSignal* instances that can be generated by that particular

adapter. Directly below the adapters list is a palette of drawable EmPro graph elements, which can be dragged from the palette onto the drawable canvas. The "EmPros" tab, depicted in Figure 17 allows users to drag previously configured EmPro execution models onto the canvas and consume events generated by other EmPro executables.

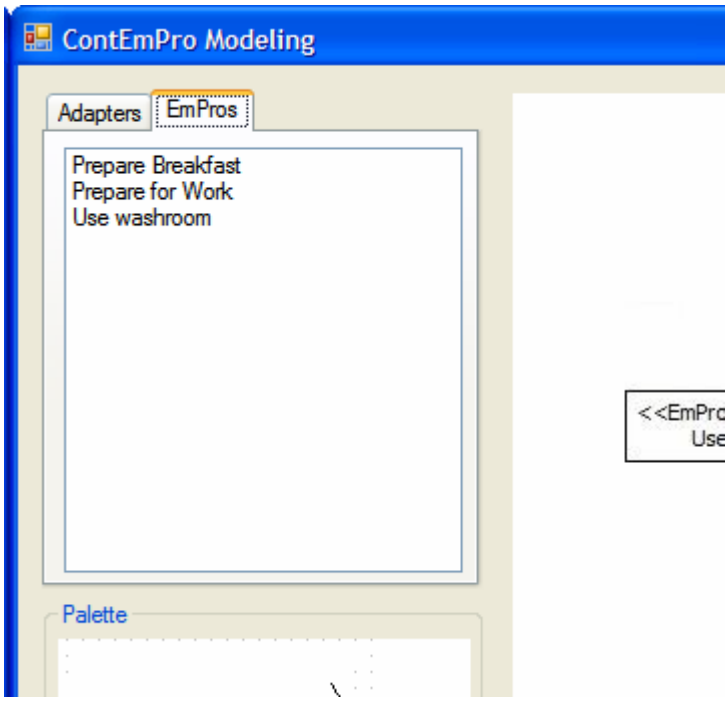

Figure 17: Listing of ECMs (EmPros) in mock-up screen

An important aspect of the EmPro model described in the previous chapter is its foundation on the UML 2.0 and XMI 2.1 specifications. In the planning phase of this thesis work, we anticipated that EmPro models would be designed primarily within a UML design tool. This was part of our larger strategy to approach the modeling requirements of ContEmPro from a so-called *adoption-centric* perspective. The objective of adoption-centric software engineering is, according to Muller and Wong [70], to increase the adoption of research tools by integrating them into existing popular software applications such as Microsoft Visio or Lotus Notes. Muller and Wong hypothesize "that users will more likely adopt tools that work in an environment they use daily and know intimately", thereby leveraging an existing system of cognitive support.

The objectives of adoption-centric software engineering approach are similar to our objectives for the IAMI module – to reuse existing software rather than constructing something from the ground up. Further, we desire to allow EmPro designers to construct

EmPro models within a UML tool of choice. This is theoretically plausible if all UML tools were at least UML 2.0 and XMI 2.1 compliant, and the interface of such tools could be extended. For example, we could create the Model mock-up displayed in Figure 16 in the Eclipse platform, and we would have the additional benefit of having access to the Eclipse Modeling Framework (EMF) for generating code from the EmPro model for perhaps automating the creation of IAMI adapters. (Unfortunately, we could not find a non-proprietary Eclipse UML design plug-in that was both UML 2.0 and XMI 2.1 compliant.) Or, if Microsoft™ Visio was UML 2.0 and XMI 2.1 compliant we would be able to embed the Visio ActiveX control to serve as the EmPro canvas within another Windows™ application. (Unfortunately, Visio 2003 is not UML 2.0 or XMI 2.1 compliant).

We should point out that we did not pursue an implementation of the Model step for either the Context Toolkit or for the JCAF (discussed in Section 4.5).

4.4.4 Context Toolkit-ContEmPro: Integration

Once all the EmPro model executables have been created, the ContEmPro system must perform three different tasks. First, it must inform the context-aware middleware, in this case the Context Toolkit, of new events that can be generated from the collection of EmPro model executables configured in the previous step. To serve this functionality, the *contempro.iami.AbsActivityManager* class defines an abstract protected method called *advertiseEmProEvents*. We implemented this functionality in a utility delegate class called *contempro.iami.CtkProxyWidget* that subclasses from the Context Toolkit *context.arch.widget.Widget* class. The *CtkProxyWidget* accepts a list of *contempro.ecm.nodes.AbsSendEvent* instances, corresponding to all the *contempro.ecm.nodes.SendIncrementalEvent* and *contempro.ecm.nodes.SendTerminateEvent* instances in all EmPro executables. The proxy widget advertises the attributes of all EmPro executable events to the local *Discoverer* instance on behalf of the ContEmPro activity manager. The code below illustrates how EmPro events are published using Context Toolkit components.

```
// constructor; calls Widget superclass constructor which in turn calls
   // the methods below
   public CtkProxyWidget(int port, String id, Collection<AbsSendEvent<CtkSignal, DataObject>>
sendEvents ) {
      super(port, id );
      this._sendEvents = sendEvents;
```

```
    }

    // initialize the non-constant attributes, i.e., what empro signal was raised
    protected Attributes initAttributes() {
       Attributes atts = new Attributes();

       atts.add( CtkProxyWidget.EMPRO_SIGNAL_RAISED);
       return atts;
    }

    // constant attributes = all send events generated by the collection of EmPro executables
    protected Attributes initConstantAttributes() {
       Attributes atts = new Attributes();
       atts.addAttributeNameValue(CtkProxyWidget.EMPRO_MANAGER_ID, this.ID, Attribute.STRING);
       for ( AbsSendEvent<CtkSignal, DataObject > sendEvent : this._sendEvents ) {
          atts.addAttributeNameValue( sendEvent.getFriendlyName(), sendEvent.getEmProID(),
Attribute.STRING);
       }
       return atts;
    }

    // create the callbacks
    protected Callbacks initCallbacks() {
       Attributes atts = new Attributes();

       atts.add(initAttributes());
       atts.add(this.initConstantAttributes());

       Callbacks callbacks = new Callbacks();
       callbacks.addCallback( Widget.UPDATE, atts );
       return callbacks;
    }
```

When the *CtkProxyWidget* instance is created, the super class *Widget* constructor initializes the attributes, finds the *Discoverer* instance, and initializes the callbacks for this object. The *initCallback* method is called by a method in the super class, which registers the capabilities of the widget with the *Discoverer*. After this registration, all widgets in the context-aware network can subscribe to events generated by the *CtkActivityManager* instance.

4.5 JCAF Integration

The other main publicly available implementations of a context-aware middleware, the Java Context-Aware Framework (JCAF) was developed and is currently maintained by Bardram since 2004 [7]. As implied by the name, the JCAF is a context-aware software infrastructure targeting the Java platform. In keeping with its goals of being "lightweight framework with an expressive, compact and small set of interfaces.. [which is] simple and robust", the JCAF has a much smaller code-base than the Context Toolkit. We

present a list of simple metrics in Table 3 to illustrate the magnitude of difference in the amount of source code between JCAF and the Context Toolkit.

Framework	# of Packages	# of Classes	# lines of Code	Size of Byte Code (Jar File)
JCAF	6	38	3465	61KB
Context Toolkit	28	188	16668	380KB

Table 3: Source code differences between the Context Toolkit and the JCAF

A big reason for JCAF's much smaller code-base is because it utilises the Java Remote Method Invocation framework, whereas the Context Toolkit has taken on the responsibility of handling all communications between context components (Widgets, discoverers, aggregators, etc.). Architecturally, the JCAF is separated into three distinct layers as shown in Figure 18: Context Client Layer, the Context Services Layer, and the Context Sensor and Actuator Layer.

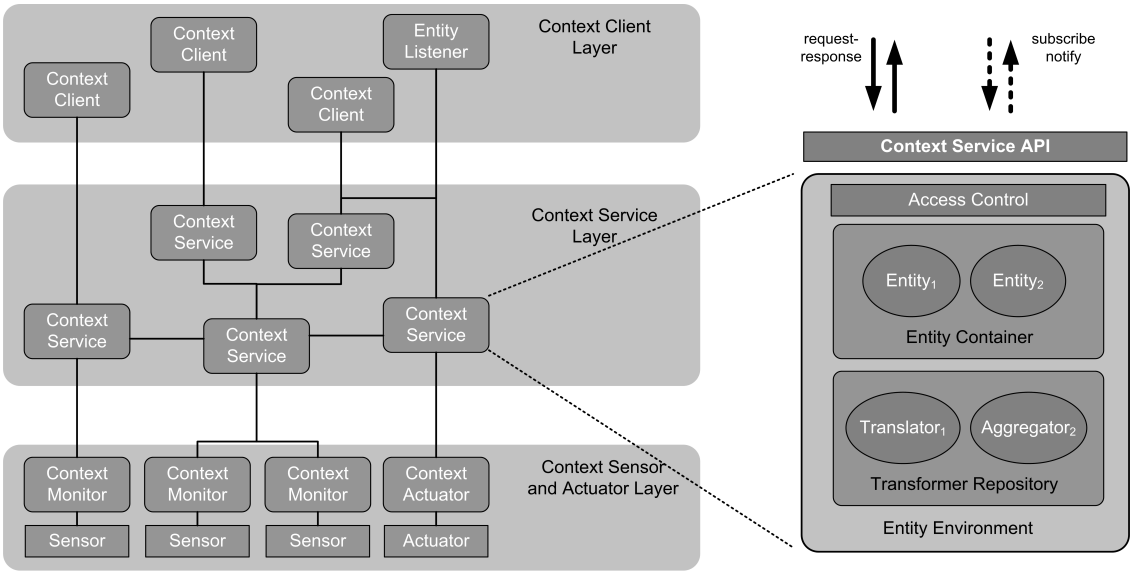

Figure 18: The Java Context-Aware Framework architecture

The Context Toolkit, by comparison, does not distinguish between a "context sensing layer" and a "context service layer". Other important differences between the Context Toolkit and JCAF include:

- The “widget” is the central component within the Context Toolkit, does not exist within JCAF. The functionality of the Context Toolkit widget is split between the JCAF concepts actuators, monitors, and entities. Within JCAF, the central component is the “Entity”, which conceptually maps to the context-aware concept of “entity”, i.e., a person, place, or thing.
- The JCAF, using Winograd’s terminology, is an example of a “network services” [71] context-aware middleware architecture, client-server type of context-aware system; the Context Toolkit, on the other hand, provides a “Widget” architecture (again using Winograd’s terminology).
- JCAF context model is object oriented using the Java programming language. The Context Toolkit context model is key-value-based.

Figure 19 shows a UML class diagram of the ContEmPro-JCAF IAMI module.

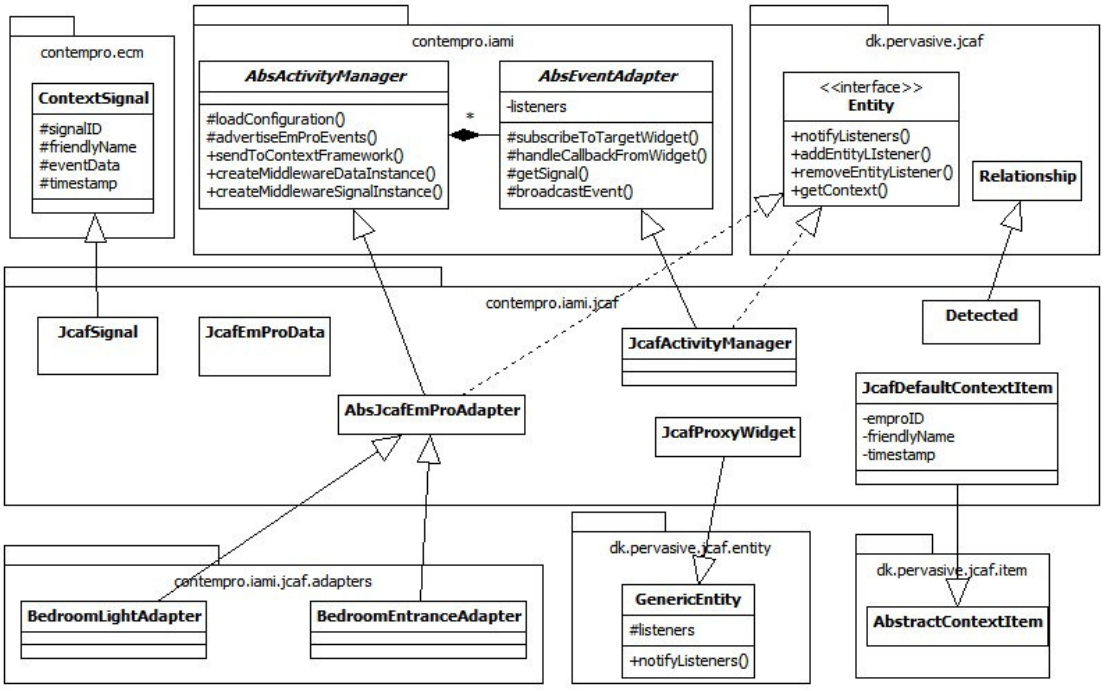

Figure 19: The JCAF-ContEmPro IAMI module

In the following sections we describe the JCAF-ContEmPro IAMI module in more detail and compare its implementation to the Context Toolkit-ContEmPro IAMI module.

4.5.1 JCAF-ContEmPro Inspection

As compared to the Context Toolkit, we found it more difficult to find a standardized way to explore the JCAF context model at runtime. Part of the difference is due to the relative simplicity of the key-value context model of the Context Toolkit, which is conducive for straight-forward information displays. For example, an instance of the Context Toolkit *context.widgets.WTemperature* widget emits five properties (Temperature, Timestamp, Location, Units, and Status) and one service actuator (*setStatusService*). These attributes and service descriptions are easily enumerated and understood for context-aware systems designers. Furthermore, the Context Toolkit provides a data structure to encompass the description of a Context Toolkit element in the *context.arch.discoverer.ComponentDescription* class.

The JCAF context model, on the other hand, is based on object-orientation, which implies the context model is defined in terms of object relationships and behaviours. Furthermore, the fact that the JCAF does not provide a data structure for exploring the capabilities of context elements at runtime (i.e., one similar to the Context Toolkit *ComponentDescription*) makes it more challenging to inspect the context model of a running JCAF-based context-aware system. The JCAF-based context-aware system inspector must either have access to the source code or the Javadoc for the JCAF components deployed in a given environment, or the designer must utilize Java reflection to obtain a listing of all the methods for a given JCAF-based context entity.

Retrieving a list of *dk.pervasive.jcaf.Entity* instances active in a given JCAF-based context-aware system is relatively straight forward. The JCAF provides a utility class, *dk.pervasive.jcaf.AbstractContextClient*, for accessing the main container and manager of JCAF entity instances, an instance of the *dk.pervasive.jcaf.ContextService* class. Once the *AbstractContextClient* instance is created a listing of all the entities is accessible. In the following code listing we illustrate how to obtain a list of all *dk.pervasive.jcaf.entity.Place* instances within a given running JCAF-based system.

```
// obtain reference to ContextService; URI already passed into constructor
ContextService service = this.getContextService();
 // retrieve list of places
Place[] places = (Place[]) service.getAllEntitiesByType(Place.class);
for ( int i = 0; i < places.length; i++ ) {
```

```
        // print out the name of the place
        System.out.println( "Place:" + places[i].getName());
}
```

The following code illustrates how to obtain a list of the methods exposed by all entities active within a given JCAF-based system.

```
// obtain reference to ContextService; URI already passed into constructor
ContextService service = this.getContextService();
Entity[] entities = (Entity[]) service.getAllEntities();

for ( int i = 0; i < entities.length; i++ ) {
        // returns only methods declared by the class, i.e., not inherited methods
        java.lang.reflect.Method[] methods = entities[i].getClass().getDeclaredMethods();
        for ( java.lang.reflect.Method method : methods ) {
                // print out only public methods
                if ( method.getModifiers() == java.lang.reflect.Modifier.PUBLIC )
                        System.out.println( "Method: " + method.getName() );
                } // end if
        } // end for each method
} // end for each entity
```

4.5.2 JCAF-ContEmPro Adaptation

In section 4.4.2 we described four basic elements of a typical IAMI adapter: *subscription*, *reception*, *signal determination*, and *signal transmit*. The implementation of adapters in the Context Toolkit realm focused on receiving events from "widgets". As stated previously, the JCAF does not analogize the "widget" concept. But in many respects the behaviour of the JCAF "entity" implementation resembles the behaviour of the "widget". The JCAF "entity", like the Context Toolkit "widget", is the central figure responsible for handling context information.

We describe how we implemented the *subscription, reception, signal determination,* and *signal transmit* elements in the JCAF-ContEmPro IAMI module in following subsections.

4.5.2.1 Subscription

The subscription process for a JCAF adapter is much simpler than for a Context Toolkit adapter, largely due to the Java RMI foundation of the JCAF. The Java RMI is a distributed computing middleware that simplifies the programming model for distributed

computing applications. Unlike the Context Toolkit, which uses *context.arch.DataObject* class to facilitate all inter-context-aware component communications, the RMI permits method calling on remote objects (i.e., objects residing within another Java virtual machine and/or on another host). The JCAF code for overriding the abstract *contempro.iami.AbsEventAdapter.subscribeToTargetWidget* method is provided below.

```
protected void subscribeToTargetWidget() {
        // this._sourceEntity was set in the constructor
        this._sourceEntity.addEntityListener(this);
}
```

Part of the simplicity of the code, as compared to the Context Toolkit *subscription* process, is due to a limitation in the JCAF subscription approach. The Context Toolkit allows client subscribers to specify conditions on events, e.g., "alert me when temperature falls below 10 degrees Centigrade." To our knowledge we cannot create similar conditions to JCAF subscriptions which thus simplifies the subscription process. The logic for applying these conditions must be processed within the adapter in the *signal determination* adapter step.

4.5.2.2 Receive an Event from an Entity

Receiving an event from a given entity within a JCAF-ContEmPro adapter closely resembles the Context Toolkit implementation. To integrate within the JCAF, we implemented the *dk.pervasive.jcaf.Entity* interface, which dictates an implementation of the *contextChanged* method; this is a listener method which is called by a source JCAF entity when the context has changed for that entity.

```
public void contextChanged(ContextEvent event) {
        JcafEmProData data = new JcafEmProData(event);
        JcafSignal signal = this.determineIncomingSignal(data);
        // don't do anything if null; not interested in this signal if true
        if ( signal != null )
                this.broadcastReceiveEvent(signal);
        }
}
```

As in the Context Toolkit *reception* implementation discussed in Section 4.4.2.2, the methods inherited from *contempro.iami.AbsEventAdapter – determineIncomingSignal* and *broadcastReceiveEvent* – are called to for creating a new ContEmPro signal instance and then transmitting the signal to the ECM module. We discuss these steps next.

4.5.2.3 Signal Determination

Because the JCAF, unlike the Context Toolkit, does not provide the capacity to specify subscription conditions, potentially more logic must be included in the JCAF-ContEmPro adapter to determine whether the event generated by the JCAF-based system is of interest to the adapter. The following code listing illustrates our implementation of a JCAF "MasterBedroomAdapter," *determineIncomingSignal* method, which monitors when a specific person leaves or sits on a bed.

```
// check if this is a person co-located with the bed
if (!data.getItemType().equals(Person.class)) {
        return null;
}
// retrieve the person who has either sat on or left the bed
Person person = (Person) data.getItem();
// return if this is not the person we're listening for
if ( !person.getName().equals("ddahlem")) return null;

// initialize signal to null
JcafSignal signal = null;
// create a signal based on two or more criteria

if ( data.getEventType() == ContextEvent.RELATIONSHIP_REMOVED ) {
   signal = new JcafSignal(data, this.DAVID_LEAVES_BED, this._signals.get(this.DAVID_LEAVES_BED));
} else if ( data.getEventType() == ContextEvent.RELATIONSHIP_ADDED ) {
   signal = new JcafSignal(data, this.DAVID_SITS_ON_BED,
   this._signals.get(this.DAVID_SITS_ON_BED));
}
return signal;
```

The data being processed is of type *contempro.iami.jcaf.JcafEmProData*, a subclass of *dk.pervasive.jcaf.ContextEvent*. The *ContextEvent* class is the carrier of event when the *contextChanged* method is called on entity listeners. From the *ContextEvent* instance we can obtain information regarding what has changed in the context of the *Entity* which caused the event to be raised. We subclass from *ContextEvent* to give additional EmPro properties to such events; this was not necessary in the Context Toolkit implementation because the main agent of communication, the *context.arch.DataObject*, could be extended at runtime.

4.5.2.4 Transmit Signal

The JCAF adapter implementation for transmitting a signal into the ECM module instance is exactly the same as for the Context Toolkit (see Section 4.4.2.4). The code resides exclusively within the *contempro.iami.AbsEventAdapter* class.

4.5.3 JCAF-ContEmPro Model

The JCAF-ContEmPro Model process does not differ significantly from the Context Toolkit-ContEmPro process outlined in Section 4.4.3.

4.5.4 JCAF-ContEmPro Integration

We described in Section 4.4.4 how the Context Toolkit-ContEmPro IAMI module must "advertise" all events that can be generated by an EmPro executable. This is not necessary in the JCAF because the Java RMI handles the registration of JCAF software instances. Further, the RMI also makes the sending of EmPro events to the context-aware system comparatively simple.

```
protected void sendToContextFramework(JcafSignal signal) {
        this._proxy.notifyListeners(signal.getData());
}
```

The 'data' returned from the *signal.getData()* method shown above is of type *contempro.iami.jcaf.JcafEmProData,* which has a context of type *contempro.iami.jcaf.JcafDefaultContextItem*, which in turn has the standard ContEmPro properties of emproID, friendlyName, and timestamp. From the information contained in an instance of *JcafDefaultContextItem*, JCAF components external to ContEmPro can react to events generated by EmPro executables.

4.6 Chapter Summary

This chapter introduced the IAMI approach to reusing existing context-aware systems. We first described the basic architecture of an IAMI implementation module. Then we applied this approach to two specific context-aware middleware systems, the Context Toolkit and the Java Context-Aware Framework (JCAF). Inclusive in our descriptions of how we applied the approach to these two context-aware middleware systems, we provided source code examples of how the integration of an ECM into an existing

context-aware system can be accomplished. In the next chapter, we evaluate the work discussed in this chapter and in previous chapters.

5 Evaluation

5.1 Introduction

In this chapter we evaluate and analyse the main contributions of this thesis: the ContEmPro software system, and a software engineering approach for integrating new context-aware models into existing context-aware systems, the IAMI. In the next section, we analyse the implementation of the ContEmPro activity execution graph model and describe the challenges in evaluating context-aware computing software. Inclusive in this discussion, we provide a listing of the limitations of the ContEmPro software, and provide a road-map for overcoming these limitations. In Section 5.4, we analyse the software engineering approaches we advocate for reusing existing context-aware systems. To this end, we provide a detailed description of how this approach can be used for targeting two additional context-aware system frameworks/middleware systems: the Java Context Aware Framework (JCAF) and the Context Broker Architecture (CoBrA).

5.2 Challenges In Evaluating ContEmPro

Context-aware computing suffers from a number of ailments which make it difficult to evaluate. In this section we describe a summary of the impediments to evaluating the ContEmPro software discussed in this thesis. Later, we provide a listing of the positive and negative/limiting aspects of the ContEmPro system.

5.2.1 The Context-Aware Middleware “Gap”

According to Weiser, the main characteristic and goal of ubiquitous computing (ubicomp) is the metaphorical "disappearance" of computing technology. Computing technology will become so interwoven into our natural human environments that it will become a "constant background presence", something that will "invisibly enhance the world that already exists." To this end, we argued in earlier chapters of this work that computing technology will need to become more aware of, and responsive to, human context. Context-aware computing promotes computing "invisibility" by reducing the intrusiveness of computing, by anticipating the data and service requirements of human

users based on the situation, and delivering results to users where and when they are required.

A typical context-aware research project provides software infrastructure support for creating context-aware applications. The fact that context-aware computing typically targets *implicit* versus *explicit* human-computer user interfaces creates challenges in finding objective evaluations of context-aware middleware. Although there are measurable attributes of any middleware system – such as performance, robustness, security, and scalability etc. – it is the qualitative attributes of a context-aware middleware that are of most interest in context-aware research. According to [51], "traditional user-centered techniques, while appropriate for application design and evaluation, fail to properly support middleware design and evaluation." This is largely due to the separation between the features of the context-aware middleware and the user visible features, and the "indirectness of the coupling between these features." In other words, it is difficult to ascertain a qualitative measurement of a particular context-aware middleware even with a robust case study or sample application because the evaluation information would be available only indirectly.

5.2.2 Creating Real-World Context Aware Environments

Further to the topics discussed in Chapter 2, the technology required to create a simulation environment for testing context-aware applications is either not available or too expensive. For example, in Satyanarayanan's 2001 paper *Pervasive Computing: Vision and Challenges* [1], Satyanarayanan predicted face recognition software would be "feasible" in many work environments within "just a few years", a prediction that has not yet come true. Where the technology was either not available or too expensive to obtain, some context-aware research groups have created *simulated* context-aware environments, whereby some or all of the sensor data is fabricated (e.g., [50]). According to [51], the value of simulated context-aware systems is limited because users can often not get "a good sense of how the applications provided benefit because too much was faked." The authors go on to say:

> There is no point in faking components and data if you intend to test for user experience benefits: By building simulations of aspects of both the middleware and the data it collects, you risk learning nothing about real user

impacts and defeat the purpose of evaluation, wasting precious time and resources.

5.2.3 Lack of Willing Participants for Case Studies

Based on our experience with other context-aware research projects (e.g., [31, 72]), we have found it challenging to find people and organizations who are willing to participate in context-aware related research projects. Perhaps our perceptions are skewed somewhat by the domain area in which we focus, i.e., health-care. Health-care providers in general are notoriously reluctant adopters of information technologies [73]. Yet, the potential advantages of context-awareness in the health-care domain are plentiful [74]. For example, a context-aware system could be used for automatically delivering patient information to health-care professionals using location and proximity information. This would potentially reduce the amount of paperwork that must be lugged around, or reduce the amount of data input that needed to be entered by a care-giver when at a patient's bedside.

We list some of the reasons, based on our first-hand experiences in the health-care domain, why people and organizations have been cool to the idea of adopting context-aware systems:

- The hardware technology is too expensive
- The technology is immature, not proven
- Privacy concerns
- The use of ubicomp/Context-aware technologies will disrupt established practices

5.2.4 Lack of Sharable Research Results

An important part of validating and effectively evaluating your research is providing others the means to replicate your results. In the realm of software engineering this often comes in the form of source code and a way to emulate the conditions for testing. Unfortunately, it is difficult to share results in the realm of context-aware computing research for two main reasons. First, the physical characteristics of the devices used for creating ubicomp/context-aware environments can vary widely. Unlike previous eras of computing as outlined by Weiser in [75], ubicomp (and by association, context-aware computing) technology is not currently dominated by one or two hardware configuration

types (i.e., Intel or AMD) with predicable resource capabilities running two or three different types of operating systems (e.g., Linux, Windows, or Mac). The range of variance of devices and sensors in terms of their resource capabilities and software platforms is wide. Consequently, it is difficult to replicate the testing conditions for a ubicomp research project unless the same types of devices -- with the same resource capabilities and operating systems or software platform -- can be obtained by the outside party for testing.

A second difficulty in sharing results of context-aware research is the lack of software standards. As discussed in Chapter 2, there are no widely accepted software standards specific to the context-aware research community, although it should be noted many have been proposed, for example [2]. Software standards help to simplify the construction of services and components, provide a measure of compliance, and help unify the direction of a research area. Further, they provide a foundation for replicating research results. Since no widely accepted standards exist for context-aware computing it is difficult to compare, for example, the efficacy of one approach versus another, and whether a approach 'A' is appropriate for situation 'B'. To further complicate the issue, very few researchers in the realm of context-aware computing provided access to source code of any kind. The notable exceptions are the Context Toolkit [6], the Java Context Aware Framework [7], and to an extent the Context Broker Architecture (CoBrA) [2].

5.3 Limitations of ContEmPro

In spite of the difficulties in evaluating context-aware software as outlined above, we provide a list of the limitations of the ContEmPro software and EmPro model designs.

5.3.1 Multiple Users

For now the ContEmPro modeling system can only contend for one user, acting in isolation from all other human users. This is a designed limitation as we wanted to limit the complexity of the ContEmPro software in terms of how it integrates with the target context-aware system. The complexity of adding multiple users is mostly a graphical modeling issue and would have been difficult to solve given our so-called 'adoption-centric', UML 2/XMI 2.1 approach to constructing EmPro models. We found it a challenge to find UML graphical editors that allowed customised views of EmPro

models, as what would be required to produce a visual EmPro model for representing more than one human-computer interaction.

5.3.2 Actuators Not Modeled

In Chapter 2 we spelled out three fundamental characteristics of a context-aware system: it senses human activity implicitly, it perceives the context or situation of users, and it adapts its behaviour in accordance with the perceived context. The ContEmPro system concerns itself only with the first two characteristics. Like the multiple user deficiency discussed above, we decided to limit the scope of ContEmPro to only context sensation and perception to reduce the complexity of the ContEmPro system.

5.3.3 Timing Issues

EmPro models are time sensitive. The underlying infrastructure of ContEmPro, however, does not attempt to account for issues which may cause timing delays in the delivery of context events to the ContEmPro model execution engine. This suggests that certain EmPro models could be executed incorrectly due to congestion, latency, or other networking issues unrelated to context-aware activities. In order to make ContEmPro ready for real use this shortcoming has to be addressed.

Another timing-related issue that must be addressed is time-synchronization. ContEmPro assumes that all calculated timings are accurate, or are synchronized (accurately or not) by a central time server to ensure that sequence of events are preserved.

5.3.4 Simple, Static Networks

EmPro models allow the user to specify human-friendly terms to signify an EmPro input event has taken place, e.g., "Shut off bedroom light", or "Enter bedroom". This type of naming allowance seems to work well enough when the EmPro network of nodes is relatively small, e.g., 20 nodes. But we have not attempted to contend for much larger networks where name collisions may force name assignments to become large and cumbersome (i.e., less human-friendly). If for example a bedroom has four light sources, the EmPro modeller might be forced to come up with cumbersome names for EmPro events, such as "Master bedroom light switch near bathroom off", or a less human-friendly description like "Switch light ID234633 off".

In its current state, ContEmPro would also have difficulties handling anything but a static network of embedded devices. ContEmPro could not, for example, dynamically incorporate events generated from non *a priori* sources, like from a personal digital assistant carried by a visitor into a ContEmPro environment. To incorporate a changes to the sensor network (additions, deletions, edits), the whole of the EmPro model needs to be regenerated or recompiled.

5.4 Evaluating the IAMI Approach

In the previous chapter we described our approach to integrating the EmPro ECM into two different context-aware middleware systems. The main method we used for directing the design of the ContEmPro integration system was to actually implement the code necessary to integrate ContEmPro into the target context-aware middleware. We produced two implementations of ContEmPro integration code, one targeting the Context Toolkit [6], the other targeting the Java Context-Aware Framework (JCAF) [7]. Unfortunately, we were bereft of test subjects in this regard, due to the lack of context-aware projects that have allowed public access to source code. In this section, we discuss how this may limit the value of the IAMI module integration software. We attempt to overcome this deficiency with a high-level description of how ContEmPro could be integrated within the Context Broker Architecture (CoBrA), a context-aware middleware with a significantly different architecture than the Context Toolkit and JCAF systems.

5.4.1 Limited Scope of our Test Implementations

In the previous chapter we described how we designed implementations of ContEmPro for use within the Context Toolkit and the Java Context-Aware Framework (JCAF). The Context Toolkit is an example of a peer-to-peer or "widget"-based architecture [71] whereas the JCAF is an example of more centralized approach, a combination of the "network services" and "blackboard" approaches identified by Winograd [71]. We previously declared that a context-aware system is sensitive, perceptive, and proactive. We also argued that a real-world context-aware system, in order to support the perceptive and proactive characteristics of context-aware computing, must possess the capacity to reason about context. Such reasoning is required for determining the context state for a given context entity and deciding how the system should adapt at a given context

conditions. A context-aware system cannot assume that the context information by a system will always be perfect, unambiguous, accurate, and highly precise. Thus, a context-aware system must make inferences about context when the inputs are less than perfect, or when a course of action is not clear-cut.

Regarding the Context Toolkit and the JCAF middleware systems, neither system provides an explicit mechanism for reasoning or deciding what to do about imperfect context information. In both systems, the core responsibility for making decisions about context -- i.e., the determining the current context state and how to adapt -- is given to the client of the context-aware system. The Context Toolkit provides an "Interpreter" component which in some ways resembles a context reasoning mechanism in that is used for translating low level context information into higher level semantics, for example, "when widgets do not or cannot provide context at the level required by applications." [6]. The use of a Context Toolkit Interpreter is not a mandatory institutional service within the system. Within the JCAF system, the designer of the context-aware system is "encouraged" to provide quality metrics of the context information exposed by a JCAF system implementation. But, like the Context Toolkit, the JCAF system does not provide an institutionalized mechanism for reasoning about context; the responsibility for JCAF context reasoning resides in the application layer, i.e., outside of the JCAF system.

5.4.2 Testing IAMI Against Reasoning-Based Context-Aware Systems

To make our evaluation of our IAMI approach towards reusing existing context-aware middleware more complete, we searched for a middleware that possessed mechanisms for context reasoning. We could find only one context-aware computing project that fulfilled this requirement and provided source code. Perhaps not coincidentally, this project is also probably the most influential, most cited context-aware project dedicated to context reasoning: the Context Broker Architecture (CoBrA) [2]. CoBrA is an agent-based system that has a centralized XML context ontology model/"knowledgebase", which is unlike the Context Toolkit and JCAF whereby the context model is embedded within the source code of an implementation. Furthermore, was CoBrA designed to reason about context information that cannot be acquired directly from sensors and, furthermore, make decisions about how to handle imperfect context information. The CoBrA context

knowledge base (model) is encoded in the Web Ontology Language (OWL), a language that is designed to allow computers to communicate in a semantically rich way.

We intended to test how (or whether) our approach could be used within the CoBrA system. We envisioned that the IAMI module would sit atop of the CoBrA "Context Acquisition Module" in the same way the IAMI module works within the Context Toolkit "widget" system and the JCAF "entity" system. The ECM executables would not directly benefit from the CoBrA reasoning engine, e.g., the reasoning engine would not contribute to the execution process.

Although the authors of the CoBrA project have exposed some source code, unfortunately we could not adequately test our integration approach with the source code provided. On the CoBrA main Web site (http://cobra.umbc.edu), source code is provided for a "demo" of a CoBrA-based office meeting system and for an implementation of the temporal reasoning component. In order for us to create a proper ContEmPro implementation test we needed source code relating to the CoBrA "Context Acquisition Module", the module responsible for "acquiring contextual information from sensors, agents, and the Web. It hides the low-level context sensing implementations from the high-level functional components." [2] The IAMI approach is designed to integrate with an existing context-aware system by fusing EmPro model logic onto the existing system by creating a layer of IAMI adapters. We could not find source code nor documentation on how the CoBrA "Context Acquisition Module" interacts with the rest of the CoBrA system and therefore we could not create a test implementation using the approach we took with the Context Toolkit and the JCAF. Unfortunately, we could not find source code from other context-reasoning-based context-aware middleware.

5.4.3 Integrating EmPro into the CoBrA Knowledgebase

The lack of CoBrA source code for the "Context Acquisition Model" does not preclude the possibility of integrating EmPro models into the CoBrA system. As stated above, the CoBrA model/knowledgebase is declaratively specified in an OWL encodings. The CoBrA ontology is called Standard Ontology for the Ubiquitous and Pervasive Application, or SOUPA [76]. Hereafter we discuss whether the SOUPA offers the means for expressing EmPro ECM.

The SOUPA defines a collection of ontologies for representing temporal relations. According to [76], the SOUPA ontology adopts vocabularies from DAML-Time [77], which provides the ability to represent time instants, time intervals, and temporal relations between temporal entities. DAML-Time offers temporal properties such as *before*, *after*, *beforeOrAt*, *afterOrAt*, *sameTimeAs*, *startsSoonerThan*, *startsLaterThan*, *startsSameTimeAs*, *endsLaterThan*, *endsSameTimeAs*, and *endsBeforeStartOf*. Based on these SOUPA, DAML-Time properties, it is likely an EmPro model can be represented the SOUPA ontology.

Assuming it were the case that EmPro models can be sufficiently represented in SOUPA encodings, the IAMI approach still applies up to the *Integration* step (i.e., the last step). There would still be a need to inspect the various properties of the context-aware system. In the case of CoBrA, this would be an evaluation of the SOUPA ontology specified for that system. The adapter layers would also need to be created should the existing SOUPA elements not handle data at a level required by the EmPro model being designed. The IAMI *Model* step would proceed as in the other frameworks. Deploying the EmPro executables would be subject to change, however. In the Context Framework and JCAF implementations of the IAMI module for ContEmPro, the EmPro model executables would be translated into ContEmPro code. Since we desire the CoBrA reasoning engine to replace the functionality of the EmPro executables, we would need to transform the XMI 2.1 information into SOUPA XML, probably using XSLT. Thereafter, the CoBrA reasoning engines would read the generated SOUPA XML, and thus the IAMI *Integration* step is averted. It is unclear how difficult it would be to generate the SOUPA ontology XML required for representing the EmPro model. How difficult it would be to create this XSLT instance has not been evaluated.

5.4.4 The IAMI Approach and Knowledge Sharing

The IAMI approach presented in this work is, we argue, a good approach for integrating new ECMs into a specific instance of context-aware system. However, it is not a good approach for creating sharable knowledge across multiple instances of context-aware systems or applications. This is because the IAMI approach advocates a ‘bottom-up’ construction of ECMs, which is not necessarily conducive for information

sharing because the ECM is defined in terms of sometimes parochial conditions, that is, conditions that are not transferable.

5.5 Evaluation Summary

Evaluating context-aware system research is difficult to accomplish. There are no significant, wide-spread context-aware or ubicomp standards from which to establish a baseline evaluation strategy. Typically in software engineering research, software is evaluated by observing and reporting on how users interact with the software in question. According to [51], this approach does not yield valuable information when the software being considered is context-aware middleware.

In this chapter, we described some of the limitations of the ContEmPro software system, including:

- No actuator behaviours
- Only a single user models
- Requires perfect timing conditions including node synchronizations and no network problems
- Simple, static EmPro models; that is, fewer than 20 nodes or so

We described the limitations of the IAMI approach, as presented in Chapter 4. The main evaluative limitation of our IAMI approach is that we had access to only two context-aware frameworks for evaluating our IAMI integration approach. The Context Toolkit and Java Context-Aware Framework (JCAF) frameworks are similar in that they do not provide a centralized component for reasoning about context. The Context Broker Architecture (CoBrA) context-aware middleware, on the other hand, represents a different class of context-aware middleware in that the context model and reasoning are central components in the middleware architecture. We provided a high level description of how we might integrate EmPro into the CoBrA. From a theoretical perspective, it would be highly advantageous to integrate into the CoBrA system using the SOUPA ontology offered. This would negate the need for the *Integration* step assuming we could transform the EmPro model data into SOUPA XML. Unfortunately, we could not obtain source code from the CoBrA developers to evaluate the different integration approaches. Finally, we discussed the limitation of the IAMI approach in terms of allowing ECM models to be shared within and without a given context-aware system.

6 Conclusions

In this thesis we propose to mitigate one of the many identified difficulties with implementing context-aware applications: the lack of sharing software components across the many context-aware research groups. Based on our review of the context-aware research literature, most context-aware research groups opt to build entirely new context frameworks rather than build on frameworks developed by others. We, on the other hand, propose re-using existing context-aware middleware systems in situations when a new or redefined *embodied context model* (ECM) is developed, but lacks a context-aware middleware to execute the model against sensed information. We provide a detailed description of the approach developed for this decoupling task, called Inspect, Adapt, Model, and Integrate (IAMI). By engaging the steps outlined above we show that a context model need not be specifically tied to a given context-aware middleware, which is usually the case. We submit that this is important in that the number of possible ways of representing and expressing a context model is potentially infinite, but the choice of context-aware middleware systems is limited.

6.1 Contributions

The contributions from this thesis include:

1. A reworked definition of the word "context" as it relates to context-aware computing. The main difference between the definition we offer and the one proposed by Dey in [6] is in how we define the word "entity". Dey defines the word "entity" as "a person, place, or object"; we expand this definition of Entity to "a person, place, object, or an *embodied context model*." An *embodied context model* (ECM) is a model of context that corresponds with a concern or aspect of the context being evaluated in a given context-aware system. This topic is covered in-depth in Chapter 2.
2. Created a model for *embodied processes* (EmPro), a type of ECM. EmPro models can be used for communicating how users interact in a context-aware, ubiquitous environment.

3. Created a UML 2.0 profile to allow EmPro models to be modeled in UML 2.0/XMI 2.1 compliant UML modeling tools.
4. Developed software for executing EmPro models. The software is called Context Aware Embodied Processes (ContEmPro), which allows EmPro models to be executed in a live environment. The software is a deliverable of an open source project, available at http://contempro.tigris.org.
5. Developed the Inspection, Adaptation, Model, and Integration (IAMI) approach to integrating embodied context models into third party context-aware middleware

6.2 Future work

In a given software engineering project or research project there tend to be items that do not get implemented or discussed. Here is a list of initiatives which would extend the work presented in this thesis.

6.2.1 ContEmPro and EmPro Future Work

6.2.1.1 Modeling Actuators

We mentioned in Section 5.3.2 that we did not endeavour to represent context actuators (i.e., instruments which cause changes in the environment like, for example, an automated light switch) within EmPro models and the ContEmPro software system. To do so would not be prohibitively difficult. It would require the inclusion of a new *actuator* concept within the EmPro model (i.e., as represented within the EmPro UML profile) and at all levels of the ContEmPro software system (e.g., new Java class specifications within the *contempro.ecm.** java package). This would also have an effect on the IAMI approach in that adapters have to be created for actuators in addition to adapters for context data acquisition.

6.2.1.2 Modeling Entities

In Section 5.3.3 we discussed how EmPro models are limited to only single-user scenarios in that EmPros currently do not distinguish between different users and the events they generate within a context-aware system. In the current state of the ContEmPro software, this problem can be mitigated within the IAMI adapter layer by implementing the logic to differentiate various users. For example, instead of creating an

EmPro consumer node for "Feed the Cat", we could create a "David Feeds the Cat" EmPro node whereby the adapter is responsible for ensuring that the "David Feeds the Cat" signal is in fact generated by the user David.

In the future, however, it would be advantageous to enrich EmPro models with basic context *entity* representations whereby a specific context entity or entities can be associated with a specific EmPro event. For example, a person entity with a specific identity (e.g., "David") could be assigned to a "Feed the cat" activity. This would add far more power to the modeling capabilities of EmPro, but perhaps at the expense of high complexity.

6.2.2 IAMI Future Work

6.2.2.1 Integration into the CoBrA system

In the previous chapter we discussed how we would go about using the IAMI approach to integrate EmPro models into the CoBrA [2] system. The CoBrA system utilizes an agent-based architecture, and leverages an XML-based Web Ontology Language (OWL) ontology. We suggested in the previous chapter that the IAMI approach would still apply to CoBrA, except the last step *Integration*. It would be interesting to investigate the difficulty (or ease) with which this can be done as compared to targeting a non-reasoning-based context-aware middleware such as the Context Toolkit or JCAF.

6.2.2.2 Top Down Approach

The IAMI approach takes a "bottom-up" approach to integrating ECMs into existing context aware frameworks/systems. It would be interesting to investigate a "top-down" approach whereby the *Model* IAMI step is conducted before the *Inspection* step. To coin yet another initialism, the Model, Inspect, Select, Adapt, Integrate (MISAI) is the top-down version of the IAMI. Briefly, here is a procedure of the MISAI:

1. Model the components, using expected data types of expected semantic value
2. Inspect the capabilities of the target context-aware system, just as in the IAMI
3. Select items in the target context-aware system that directly correspond with items in the ECM being modeled.
4. Adapt items in the ECM that do not have a corresponding semantic data value as inspected

5. Integrate the model into the context-aware system

The main advantage to this approach is that it is more sharable than models created using IAMI. Further, *adaptation* is not an absolute requirement as it is in the IAMI approach; only those items in the ECM that are not semantically similar in the target context-aware system are adapted.

7 Bibliography

[1] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE,* vol. 8, pp. 10-17, 2001.

[2] H. Chen, T. Finin and A. Joshi, "An Intelligent Broker for Context-Aware Systems," *Adjunct Proceedings of Ubicomp,* pp. 12-15, 2003.

[3] M. Weiser, "The Computer for the Twenty-First Century," *Scientific American,* vol. 265, pp. 94-104, 1991.

[4] R. Want and T. Pering, "System challenges for ubiquitous & pervasive computing," *Proceedings of the 27th International Conference on Software Engineering,* pp. 9-14, 2005.

[5] N. Davies and H. W. Gellersen, "Beyond prototypes: challenges in deploying ubiquitous systems," *Pervasive Computing, IEEE,* vol. 1, pp. 26-35, 2002.

[6] A. K. Dey, "Providing Architectural Support for Building Context-Aware Applications," PhD Thesis, Georgia Institute of Technology, 2000.

[7] J. E. Bardram, "The Java Context Awareness Framework (JCAF)-A Service Infrastructure and Programming Framework for Context-Aware Applications," *Pervasive Computing: Third International Conference, Pervasive 2005, Munich, Germany, may 8-13, 2005, Proceedings,* 2005.

[8] B. Schilit and M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *IEEE Network,* vol. 8, pp. 22-32, 1994.

[9] D. A. Norman, *The Invisible Computer.* Cambridge, Massachusetts: The MIT Press, 1998.

[10] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward and A. Hopper, "Implementing a sentient computing system," *IEEE Computer,* vol. 34, pp. 50-56, 2001.

[11] E. H. L. Aarts, *Ambient Intelligence.* Springer, 2003,

[12] J. H. Jahnke, M. D'entremont and J. Steir, "Facilitating the programming of the smart home: Smart homes," *IEEE Wireless Communications,* vol. 9, pp. 70-76, 2002.

[13] A. K. Dey, P. Ljungstrand and A. Schmidt, "Distributed and disappearing user interfaces in ubiquitous computing," *Conference on Human Factors in Computing Systems,* pp. 487-488, 2001.

[14] S. Mann, "Wearable computing: a first step toward personal imaging," *Computer,* vol. 30, pp. 25-32, 1997.

[15] R. T. Azuma, "A survey of augmented reality," *Presence: Teleoperators and Virtual Environments(1054-7460),* vol. 6, pp. 355-385, 1997.

[16] A. Greenfield, *Everyware: The Dawning Age of Ubiquitous Computing.* New Riders Press, 2006,

[17] M. L. Dertouzos, *The Unfinished Revolution: Human-Centered Computers and what they can do for Us.* New York, NY: HarperCollins Publishers, Inc., 2001,

[18] G. Chen, "Solar: Building A Context Fusion Network for Pervasive Computing," PhD Thesis, Dartmouth College, 2004.

[19] OSGi Alliance. (2008, Http://www.osgi.org. *(02/14))*

[20] A. Schmidt, M. Beigl and H. W. Gellersen, "There is more to context than location," *Comput. Graph.,* vol. 23, pp. 893-901, 1999.

[21] A. D. Schmidt, "Implicit human computer interaction through context," *Personal Technologies,* vol. 4, pp. 191-199, 2000.

[22] R. Want, A. Hopper, V. Falcão and J. Gibbons, "The active badge location system," *ACM Transactions on Information Systems (TOIS),* vol. 10, pp. 91-102, 1992.

[23] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research," *Dartmouth Computer Science Technical Report TR2000-381,* 2000.

[24] A. K. Dey and G. D. Abowd, "Towards a Better Understanding of Context and Context-Awareness," *GVU Technical Report GIT-GVU-99-22,* 1999.

[25] K. Van Laerhoven, A. Schmidt and H. W. Gellersen, "Multi-Sensor Context-Aware Clothing," *Proceedings of the Sixth International Symposium on Wearable Computers,* 2002.

[26] J. Maisonnasse, N. Gourier, O. Brdiczka and P. Reignier, "Attentional Model for Perceiving Social Context in Intelligent Environments," *3rd IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI),* pp. 171–178, 2005.

[27] J. Healey and R. W. Picard, "StartleCam: a cybernetic wearable camera," *Wearable Computers, 1998.Digest of Papers.Second International Symposium on,* pp. 42-49, 1998.

[28] R. W. Picard, *Affective Computing.* MIT Press Cambridge, Mass, 1997,

[29] T. Starner, B. Schiele and A. Pentland, "Visual contextual awareness in wearable computing," *Wearable Computers, 1998.Digest of Papers.Second International Symposium on,* pp. 50-57, 1998.

[30] D. H. Wilson, D. Wyatt and M. Philipose, "Using Context History for Data Collection in the Home," *1st International Workshop on Exploiting Context Histories in Smart Environments (ECHISE-2005), Munich, Germany,* 2005.

[31] D. Dahlem, Y. Bychkov, L. Kawasme and J. H. Jahnke, "Towards context oriented web services for smart personal object technologies (COWSPOTS)," in *Workshop on Pervasive Computing, OOPSLA,* 2003,

[32] Wikipedia, "Introduction to Psychology:chpt 4," vol. 2006, 2006.

[33] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey," *Workshop on Advanced Context Modelling, Reasoning and Management Associated with the Sixth International Conference on Ubiquitous Computing (UbiComp 2004),* 2004.

[34] B. Schilit, N. Adams and R. Want, "Context-Aware Computing Applications," *Proceedings of the Workshop on Mobile Computing Systems and Applications,* pp. 85-90, 1994.

[35] P. J. Brown, J. D. Bovey and J. D. X. Chen, "Context-aware applications: from the laboratory to the marketplace," *Personal Communications, IEEE [See also IEEE Wireless Communications],* vol. 4, pp. 58-64, 1997.

[36] T. Erickson, "Some problems with the notion of context-aware computing," *Commun ACM,* vol. 45, pp. 102-104, 2002.

[37] M. S. Malone, "Moore's Second Law: If we don't do something about increasing battery life, we're toast," *Wired Magazine,* April 2004. 2004.

[38] A. K. Dey, G. D. Abowd and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction,* vol. 16, pp. 97-166, 2001.

[39] S. Greenberg, "Context as a dynamic construct," *Hum. -Comput. Interact.,* vol. 16, pp. 257-268, 2001.

[40] A. Dey, J. Mankoff, G. Abowd and S. Carter, "Distributed mediation of ambiguous context in aware environments," *Proceedings of UIST,* vol. 2002, pp. 121–130, 2002.

[41] P. Dourish, *Where the Action is: The Foundations of Embodied Interaction.* Bradford Book, 2004,

[42] Merriam-Webster Incorporated. *2006*pp. 1. Available: http://www.mirriamwebster.com

[43] Object Management Group. www.omg.org, *2008,*

[44] OASIS. (2008, Organization for the advancement of structured information standards. http://www.oasis-open.org, Accessed: *2008(February 14),*

[45] Object Management Group. (2007, 2007-11-02). The UML 2.1.2 superstructure specification.

[46] M. Baldauf and S. Dustdar, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing,* 2004.

[47] O. Brdiczka, P. Reignier and J. L. Crowley, "Supervised learning of an abstract context model for an intelligent environment," *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-Aware Services: Usages and Technologies,* pp. 259-264, 2005.

[48] M. Perkowitz, M. Philipose, K. Fishkin and D. J. Patterson, "Mining models of human activities from the web," *Proceedings of the 13th International Conference on World Wide Web,* pp. 573-582, 2004.

[49] M. Spence, C. Driver and S. Clarke, "Sharing context history in mobile, context-aware trails-based applications," in 2005,

[50] A. K. Dey and G. D. Abowd, "CybreMinder: A Context-Aware System for Supporting Reminders," *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing,* pp. 172-186, 2000.

[51] W. K. Edwards, V. Bellotti, A. K. Dey and M. W. Newman, "Stuck in the Middle: Bridging the Gap Between Design, Evaluation, and Middleware," 2002.

[52] R. Etter, P. D. Costa and T. Broens, "A Rule-Based Approach Towards Context-Aware User Notification Services," *Proceedings of the IEEE International Conference on Pervasive Services, Lyon, France,* pp. 281–284, 2006.

[53] C. Julien and G. C. Roman, "EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications," *IEEE Trans. Software Eng.,* vol. 32, pp. 281-298, 2006.

[54] A. Krause, A. Smailagic and D. P. Siewiorek, "Context-Aware Mobile Computing: Learning Context-Dependent Personal Preferences from a Wearable Sensor Array," *IEEE Transactions on Mobile Computing,* vol. 5, pp. 113-127, 2006.

[55] M. J. van Sinderen, A. T. van Halteren, M. Wegdam, H. B. Meeuwissen and E. H. Eertink, "Supporting Context-aware Mobile Applications: an Infrastructure Approach," *IEEE Communication Magazine, Sep,* 2006.

[56] J. Choi, D. Shin and D. Shin, "Research and implementation of the context-aware middleware for controlling home appliances," *Consumer Electronics, IEEE Transactions on,* vol. 51, pp. 301-306, 2005.

[57] T. Gu, H. K. Pung and D. Q. Zhang, "A service□oriented middleware for building context□aware services," *Journal of Network and Computer Applications,* vol. 28, pp. 1-18, 2005.

[58] H. Kim, Y. J. Cho and S. R. Oh, "CAMUS: a middleware supporting context-aware services for network-based robots," *Advanced Robotics and its Social Impacts, 2005.IEEE Workshop on,* pp. 237-242, 2005.

[59] K. Nishigaki, K. Yasumoto, N. Shibata, M. Ito and T. Higashino, "Framework and Rule-based Language for Facilitating Context-aware Computing using Information Appliances," *Proceedings of the First International Workshop on Services and Infrastructure for the Ubiquitous and Mobile Internet (SIUMI)(ICDCSW'05)-Volume 03,* pp. 345-351, 2005.

[60] N. S. Park, K. W. Lee and H. Kim, "A Middleware for Supporting Context-Aware Services in Mobile and Ubiquitous Environment," *Proceedings of the International Conference on Mobile Business (ICMB'05)-Volume 00,* pp. 694-697, 2005.

[61] M. Raento, A. Oulasvirta, R. Petit and H. Toivonen, "ContextPhone: a prototyping platform for context-aware mobile applications," *IEEE Pervasive Computing,* vol. 4, pp. 51-59, 2005.

[62] J. Wohltorf, R. Cissee and A. Rieger, "BerlinTainment: an agent-based context-aware entertainment planning system," *Communications Magazine, IEEE,* vol. 43, pp. 102-109, 2005.

[63] S. J. H. Yang, B. C. W. Lan and J. Y. Chung, "A New Approach for Context Aware SOA," *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)-Volume 00,* pp. 438-443, 2005.

[64] J. Al-Muhtadi, R. Hill, R. Campbell and M. D. Mickunas, "Context and Location-Aware Encryption for Pervasive Computing Environments," *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops,* 2006.

[65] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell and K. Nahrstedt, "A middleware infrastructure for active spaces," *Pervasive Computing, IEEE,* vol. 1, pp. 74-83, 2002.

[66] L. Barkhuus, "Context information in mobile telephony," in *Mobile HCI 2003,* 2003,

[67] M. Khedr and A. Karmouch, "Negotiating Context Information in Context-Aware Systems," *IEEE Intelligent Systems,* vol. 19, pp. 21-29, 2004.

[68] T. Gu, H. K. Pung and D. Q. Zhang, "Toward an OSGi-based infrastructure for context-aware applications," *Pervasive Computing, IEEE,* vol. 3, pp. 66-74, 2004.

[69] The Apache Software Foundation. (2008, The apache ant project. *2008(February 14),*

[70] H. Müller and K. Wong, "Leveraging Cognitive Support and Modern Platforms for Adoption-Centric Reverse Engineering (ACRE)," *ACSE 2003 3rd International Workshop on Adoption-Centric Software Engineering,* 2003.

[71] T. Winograd, "Architectures for Context," *Hum. -Comput. Interact.,* vol. 16, pp. 401-419, 2001.

[72] J. H. Jahnke, Y. Bychkov, D. Dahlem and L. Kawasme, "Context-Aware Information Services for Health Care," *Revue d'Intelligence Artificielle (RIA),* vol. 19, 2005.

[73] S. Junnarkar. *Law prescribes overhaul of aging system.* http://www.news.com/2030-1070-1001641.html, Accessed *2008(02/15),*

[74] J. Jahnke, Y. Bychkov, D. Dahlem and L. Kawasme, "Context-Aware Information Services for Health Care," *Proceedings of the KI-2004 International Workshop on Modelling and Retrieval of Context,* vol. 114, 2004.

[75] M. Weiser and J. S. Brown, "Designing Calm Technology," *PowerGrid Journal,* vol. 1, pp. 94-100, 1996.

[76] H. Chen, F. Perich, T. Finin and A. Joshi, "SOUPA: standard ontology for ubiquitous and pervasive applications," *Mobile and Ubiquitous Systems: Networking and Services, 2004.MOBIQUITOUS 2004.the First Annual International Conference on,* pp. 258-267, 2004.

[77] J. R. Hobbs, "A DAML Ontology of Time," *Online: Http://www.Cs.Rochester.edu/_ferguson/daml/daml-Time-20020830.Txt], 2002,* 2004.