Smart Interactions and Services - Adding Process-Awareness to Context-Aware Middleware

J. H. Weber-Jahnke and D. Dahlem

University of Victoria
Department of Computer Science
Victoria BC Canada

jens@uvic.ca

Abstract
Context-aware (CA) computing has not seen broad industrial adoption, because of the lack of robust, reusable software components in this domain. This paper summarizes our work on Embodied Processes (EmPros) and their enactment in CA middleware.

1 Introduction: Expectations Unmet

Smart interactions and services require, among other things, context-aware (CA) computing – a field that has been investigated for at least two decades. Mark Weiser’s vision of ubiquitous computing (ubicomp) [1] has inspired a flurry of research activity, resulting in theories, models and prototypes of CA systems [2]. He predicted that his vision would be realized within 20 years, i.e., by 2011. Still, with two years left until that deadline, industry seems to be nowhere near to attaining Weiser’s vision of economically realizing smart interactions using CA services. Why is this so? Davies and Gellerson suggested that software engineering issues lie at the heart of the problem [3]. They point out that, while computer hardware and networks have advanced as predicted, ubicomp lacks middleware to allow “us [to] combine components to form applications unforeseen at the time of their deployment” [3]. Want and Pering point out that the limitations on ubicomp progress have moved to the higher levels of abstraction, primarily into the software area [4]. They argue that software capabilities have not advanced at a pace that can take full advantage of the hardware and networking infrastructure.

One problem is to find a common agreement on how to define, represent and implement “context” in software. While Schilit and Theimer are credited with to be the first to coin the phrase context-aware (CA) in the ubicomp sense [5], many different definitions and interpretations of this term exist. A frequently cited definition has been provided by Dey: “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]. Context information may have physical nature, e.g., light, visual context, audio, motion & acceleration, location, touch, temperature, etc. or it may be non-physical, e.g., the time of the day or the current appointment on a user’s calendar.

Most prototype CA systems have been developed with idiosyncratic context model implementations. Despite the absence of an agreed upon definition of the concept of a CA computing system, the following two capabilities are common to current prototypes:

I. the capacity to sense context information automatically, i.e., without requiring a human to enter it explicitly,

II. the capacity to change or adapt system behaviour depending on the context information sensed.

We hypothesize that the emergence of economically reusable context management system (CxMS) software components will act as catalyst for smart interaction & services, analogously to what database management systems (DBMS) have done for the eBusiness sector.

2 What is wrong with existing CA middleware?

Numerous CA middleware systems have been proposed to address one or more of the challenges in creating CA software applications, including [5-10], and a few of our own, e.g., [11-17]. None of these papers describe the use of third-party CA middleware components. The closest any of these
projects came to reusing third party CA middleware was the system discussed in [18], which was built on a well-known context-aware middleware (GAIA [19]) developed many years prior by the same research group but with mostly different people involved. Many of the projects described in the other papers were based on conventional (i.e., not CA) middleware, such as XML Web services, CORBA, and OSGi.

We suggest two reasons for the lack of external reuse of CA middleware. First, few of these proposed middleware systems become accessible to outsiders. The software artifacts 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) [8].

Second, CA middleware systems are constructed to be “complete” with respect of the context models they implement, e.g., location, proximity, user context etc. This assumption of “completeness” impedes their extension in cases where additional or different context models are required. The most common method for dealing with this issue is to create the middleware infrastructure from scratch to suit the purpose of the application - a time-consuming and expensive undertaking.

3 Interactions between Context Data and Process

The context models implemented in many existing CA middlewares fail to properly model interactions between contextual data and services processes. Strang et. al. surveyed approaches for modeling and reasoning about context [20]. They identified the following six categories of context models. The same categories have been used in a more recent survey paper by Baldauf et al. [21]:

- **Key-value models**: context data is stored in form of key-value pairs.
- **Markup scheme models**: The model is organized hierarchically in SGML/XML.
- **Graphical models**: The model is organized visually, primarily to make it understandable to a human. Thereafter the model is transformed into executable code.
- **Object-oriented models**: Context models are defined using object-oriented design. Sensors and information are encapsulated as objects.
- **Logic based models**: Context is defined as facts, expressions, and rules.
- **Ontology based models**: The models use Semantic Web standards such to model and reason about context (e.g., OWL).

These models have in common a data-centric approach to representing context. Workflows and processes are typically added on top of these data representations using application code. However, existing CA middleware frameworks/systems use idiosyncratic ways of implementing this aspect. Therefore, their workflow inference and execution engines cannot normally be reused as an independent component. Our objective is to improve on this situation. We propose an approach to specify and interpret processes as part of context models, extending existing CA systems.

4 Our Approach

In the rest of this paper, we summarize our approach to decoupling context data representations from the modelling of contextual processes. These two aspects are commonly entangled in current CA systems – a fact that impedes reuse of context model components. The specific contributions of this research are the concept of Embodied Processes (EmPro) for defining contextual workflow, a UML profile for modeling Embodied Processes and a software component called Context Aware Embodied Processes (ContEmPro) that can be used for executing EmPro models. Furthermore, we summarize an approach to integrating EmPros into existing applications, called IAMI (Inspection, Adaptation, Model, and Integration).

We have evaluated our approach with two separate context-aware middleware systems, the Context Toolkit [6] and the Java Context-Aware Framework (JCAF) [8].

4.1 Embodied Processes

Little previous CA computing research addresses how to explicitly model and reason about high-level CA interactions. In the following, we summarize the Context-aware Embodied Process system (ContEmPro) in an attempt to respond to this gap. Much of everyday human activity can be described by, and are governed by, processes. Processes can be constructed to represent how context entity interactions unfold from the perspective of a larger, multi-step activity. We declare a process as embodied when the process is perceivable by an agent and indicative of 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. Upon recognizing that a user is engaged in a particular Embodied
Process (EmPro) EmPro, a CA system could generate conjectures about what information and services the user will likely need. In order to represent an EmPro within a CA system, we require the following characteristics:

- The EmPro representation must allow for the specification of ordered as well as unordered events. Sometimes, order is important for context understanding, other times it is not.
- Temporal aspects are important. There are many situations when the timing, or what we refer to as the *temporal proximity* of events, is significant.
- Negation events should be supported. The occurrence of a negation event effectively invalidates the detection of a particular EmPro.
- EmPro models should be composable based on other EmPro models.

### 4.2 Modeling EmPros

There are many languages for expressing and executing processes and workflows. Our current approach is based on a UML 2 profile extension of Activity Diagrams, implementing the features mentioned above.

An example EmPro is given in Fig. 1. This process embodies the morning routine of a fictitious person. The EmPro is initiated when the person leaves the bed, at which time she carries out several actions (in any order) before preparing breakfast. Entering the bed again serves as a negation event (e.g., on a holiday, when she does not work). After preparing breakfast, she may eat it at the computer or while reading the paper at the table. Then, she will take a shower, dress, prepare for departure, exit the house and enter the car. Any *EmProReceiveEvent* may be composed by other, more primitive EmPros. Each EmPro sends a *terminal event* (*EmProSendTerminateEvent*) upon completion in order to update any context database that may be integrated with the ContEmPro system. Moreover, EmPros can send incremental events (*EmProSendIncrementalEvent*) to update context databases about progress.

Note that tagged values (e.g., time constraints) are not shown in Fig. 1, because the UML tool used does not display them graphically.

### 4.3 Using EmPros

The component developed to interpret EmPro models is called *EmPro-Execute* and its purpose is to augment existing or new CA systems with "process-awareness". EmPro-Execute can be integrated in a CA system as a *logical sensor*, which is, according to Baldauf et al., a sensor that combines physical and virtual information to solve higher tasks [21].

We use a four-step method for integrating EmPro-Execute with existing CA middleware. The method, called IAMI (*Interpret, Adaption, Model, and Integration*) is summarized below:

1. **Inspection** gathers meta data on *in situ* devices and sensors within the environment of interest. The information may be gathered from various components of the CA system (e.g., from context 'widgets', context 'entities', etc.).

2. **Adaptation.** An adapter layer is created within the CA middleware. IAMI adapters are designed to listen to specific events from one or more context 'widgets' (or other middleware components). The adapter may translate original data into a form required by the embodied processes of interest. For example, an adapter for a light intensity widget might only be interested in binary events (lights on/off) rather than a gradual value.

3. **Model.** The adapters created in the previous step are consumable by EmPros, i.e., 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 for modeling the EmPros.

4. **Integration.** The models are transformed into executable programs and then integrated into the target CA system using middleware specific mechanisms. This process includes, (1) embedding the model within middleware specific component so that it can be executed, (2) 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 CA middleware.

We have validated the IAMI method with by integrating EmPros with two different CA middleware architectures, the *Context Toolkit* [6] and the *Java Context-Aware Framework (JCAF)* [8]. These systems were chosen because they are well known and publicly available. We have also designed a third integration example, using the Context Broker Architecture (*CoBrA*) [7]. Details on these experiments and are documented in [2].

### 5 Conclusion

Awareness of Embodied Processes (EmPros) is an important aspect of smart interactions. However, there is little research on how to systematically integrate EmPros in CA middleware. This paper has summarized the current progress toward our objective of providing EmPro awareness in form of a reusable software component. Our future work is on translating these ideas to BPEL4People.
References


