ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services

Quan Z. Sheng and Boualem Benatallah

School of Computer Science and Engineering
The University of New South Wales, Sydney, NSW 2052, Australia
{qsheng, boualem}@cse.unsw.edu.au

Abstract

Context-aware Web services are emerging as a promising technology for the electronic businesses in mobile and pervasive environments. Unfortunately, complex context-aware services are still hard to build. In this paper, we present a modeling language for the model-driven development of context-aware Web services based on the Unified Modeling Language (UML). Specifically, we show how UML can be used to specify information related to the design of context-aware services. We present the abstract syntax and notation of the language and illustrate its usage using an example service. Our language offers significant design flexibility that considerably simplifies the development of context-aware Web services.

1. Introduction

The proliferation of ubiquitous, interconnected computing devices (e.g., PDAs, 3G mobile phones) is fostering the emergence of environments where Internet applications and services made available to mobile users are a commodity [7]. Users expect that Web services are aware of their current personal environments (e.g., their locations, the activities they are engaged in, and personal preferences) and provide more intelligent services. We call such kind of Web services *context-aware services* (CASs) [12, 8, 4]. A CAS provides users with a *customized* and *personalized* behaviour. For example, a `dining-at-uni` service gives users suggestions on where to have lunch in university's campus, depending on their food preferences, their locations on campus, and even the weather conditions.

To date, CASs are still hard to build. One reason is that current Web services standards (e.g., UDDI, WSDL, SOAP) are not sufficient for describing and handling context information. Although currently there exist many tools for Web services development (e.g., Java2WSDL utility in Apache Axis[1] can generate WSDL descriptions from Java class files), the development of CASs can not benefit directly from such tools. CAS developers must implement everything related to context management—including the collection, dissemination, and usage of context information—in an ad-hoc manner. Another reason is that, to the best of our knowledge, there is a lack of generic approaches for formalizing the development of CASs. As a consequence, developing CASs is a very cumbersome and time consuming activity, especially when these CASs are complex.

Model Driven Architecture (MDA) [9] is an approach that supports system development by employing a model-centric and generative development process. MDA increases the quality of complex software systems based on creating high level system models and automatically generating system architectures from the models. In this paper we show how this paradigm can be specialized to CASs development. In particular, we present a UML-based modeling language—ContextUML—for formalizing the design and development of CASs. ContextUML provides constructs for i) generalizing context provisioning that includes context attributes specification and retrieval; and ii) formalizing context awareness mechanisms and their usage in CASs.

Model-driven CAS development approach has a number of advantages. First, it improves the productivity and quality of CASs development. By exploiting ContextUML, services are formally and clearly documented. More importantly, the implementation of services can be generated automatically by transforming the service design models to particular target platforms (e.g., BPEL [1] specifications). The conformance of the implementation with service models is guaranteed and the time and effort of service development can be reduced. Second, it eases system maintenance and evolution. Any changes can be easily made at the service model level and propagated automatically to the implementation. Finally, it enhances the portability of service design due to technical independency of service models. The service models can be migrated to new technologies (e.g., new Web service languages or protocols) by simply developing new transformation rules.

[1] http://ws.apache.org/axis.

Proceedings of the International Conference on Mobile Business (ICMB'05)
0-7695-2367-6/05 $20.00 © 2005 IEEE

The work described in this paper is the first step for creating a complete model-driven approach for CASs development. With the approach, CASs can be formally modeled and their executable implementations can be automatically generated and deployed. We believe that as mobile computing devices become widely adopted and the needs for mobile services continue to grow, such an approach will be an important part of any mobile business development environment.

The remainder of the paper is organized as follows: Section 2 briefly overviews some basic concepts used in the paper and then gives an example CAS. Section 3 presents abstract syntax of ContextUML. Section 4 illustrates the notation of ContextUML and shows how to design CASs using the example CAS. Finally, Section 5 discusses and concludes the paper.

2. Background

In this section, we first briefly overview some basic relevant concepts, namely *context*, *context-aware service* (CAS), and *UML*. We then present a CAS that will serve as a running example throughout this paper.

2.1. Context

Dey and Abowd have defined context—which is widely used in the literature today—as "*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 applications themselves*" [8].

In a CAS environment, context contains any information that can be used by a Web service to adjust its execution and output. Examples of contexts are: i) contexts related to a service requester (mostly it is the client who invokes a service), including the requester's identification information, personal preferences, current situation (e.g., location), and other information (e.g., friends list, calendar); ii) contexts related to a Web service, such as service location, service status (e.g., available, busy), and its QoS attributes (e.g., price, reliability); and iii) other contexts like time and weather information.

It should be noted that some contexts are application specific. Forecasted weather, for instance, could be a context in a vacation planning service, but not in a currency conversion service.

Some context information can be *sensed* directly (e.g., locations and temperatures using physical sensors), while others have to be *derived* from available context information. Contexts are provided by *context providers*. It is interesting to mention that more and more context providers advertise their services—called *context information services*[2]—over the Web that can be seamlessly integrated into CASs. Recently, quite a few research efforts on modeling the context provisioning services are proposed [15, 4, 11].

[2]E.g., US National Weather Service, http://www.nws.noaa.gov.

2.2. Context-Aware Service

A service is *context-aware* if it uses context information to provide relevant information and/or services to users, where relevancy depends on the users' task [8, 14, 16]. A CAS can present relevant information or can be executed or adapted automatically, based on available context information. For example, a service can display restaurants that are around a user's current location, and if the weather is good, the service even suggests some restaurants that customers can sit outside.

To develop CASs, two important issues need to be considered. The first is the provisioning of context information. CAS developers have to identify what kind of context information will be used and how to derive it. Due to heterogeneity of context providers, sensor imperfection, quality of context information, and dynamic context environments, context provisioning is not trivial [15]. In particular, various context providers may provide a same piece of context information (usually with different quality and data formats) and it is difficult to specify– at design stage–which context provider should be contacted for the provision of a particular context. Further, some context required by a CAS may not be able to find any context provider who can supply the context directly.

The second issue is the mechanisms that can be used by CASs to adapt their behaviours based on corresponding context information without explicit user intervention. In other words, it is the problem of how to use context information to achieve *context awareness* of services. In addition, the abstracted context awareness mechanisms should guarantee the efficiency of the development and maintenance of CASs. For instance, it should be possible to use legacy Web services to develop CASs without changing their implementation.

2.3. Unified Modeling Language

The Unified Modeling Language (UML)[3] is considered as the industry de-facto standard for modeling software systems and plays a central role in Model Driven Architecture (MDA) [9]. In UML, the structural aspects of software systems are defined as classes, each formalizing a set of objects with common services, properties, and behaviour. Services are described by methods. Properties are described by attributes and associations. Object Constraint Language (OCL) can be used to express additional constraints.

UML can also serve as foundation for building domain specific languages by specifying *stereotypes*, which introduce new language primitives by subtyping UML core types, and *tagged values*, which represent new properties of these primitives. Model elements are assigned to such types by labeling them with the corresponding stereotypes. In addition, UML can also be used as metamodeling language, where UML diagrams are used to formalize the abstract syntax of another modeling language.

[3]http://www.omg.org/technology/documents/formal/uml.htm.

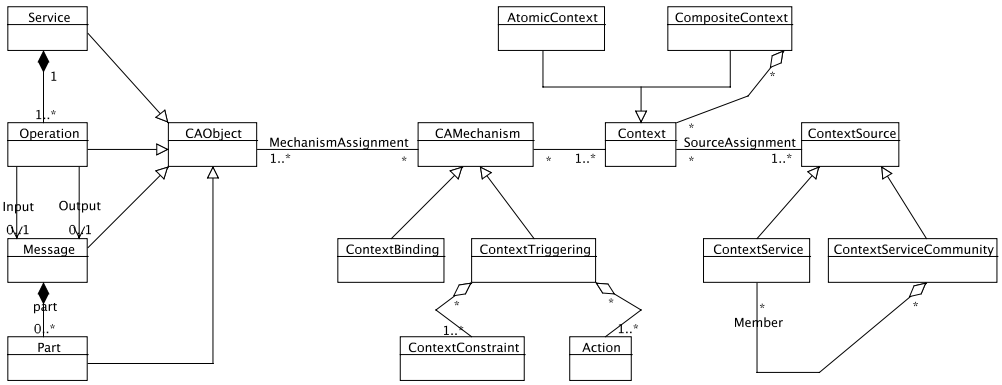

Figure 1. ContextUML metamodel

2.4. Example: Attractions Searching Service

Suppose that there is a context-aware attraction searching service that is offered by a mobile network operator. Mobile users subscribed to the network operator can invoke this service using their mobile devices to get the recommended attractions when they visit new cities. The service works like the following:

- Users can subscribe their personal preferences to the service. For example, a user can specify what kinds of attractions (e.g., historical sites) she likes, and which language (e.g., Chinese) the description of attractions should be.
- The service recommends attractions according to a user's location (e.g., the city that the user is currently in).
- During the recommendation, the service also considers other contexts like weather and user preferences. If the weather is *harsh*, the service will only suggest indoor attractions (e.g., Sydney Aquarium). The definition of weather to be harsh depends on a couple of contexts like the temperature (e.g., above 40 degree Celsius) and the likelihood of rain (e.g., more than 80%). The recommended attractions will also reflect the user's preferences, e.g., translating the attraction descriptions to user's preferred language.

In the following of the paper, we will introduce our modeling language, ContextUML, which provides the foundation for designing such a CAS.

3. ContextUML

We now define the syntax of ContextUML, including a *metamodel* and a *notation*. The metamodel defines *abstract syntax* of the language, while the notation defines the concrete format used to represent the language (also called *concrete syntax*). We will give examples of concrete syntax in Section 4.

Figure 1 shows the metamodel of ContextUML. We will introduce the abstract syntax of ContextUML from two aspects: *context modeling* and *context-awareness modeling*.

3.1. Context Modeling

3.1.1 Context Type

A Context is a class that models the context information. In our design, the type Context is further distinguished into two categories that are formalized by the subtypes AtomicContext and CompositeContext. Atomic contexts are low-level contexts that do not rely on other contexts and can be provided directly by context sources (see Section 3.1.2). In contrast, composite contexts are high-level contexts that may not have direct counterparts on the context provision. A composite context aggregates multiple contexts, either atomic or composite. The concept of composite context can be used to provide a rich modeling vocabulary.

For instance, in the scenario of attraction searching service, `temperature` and `rainLikelihood` are atomic contexts because they can be provided by e.g., `GlobalWeather`[4] Web service. Whereas, `harshWeather` is a composite context that aggregates the former two contexts.

3.1.2 Context Source

The type ContextSource models the resources from which contexts are retrieved. We abstract two categories of context sources, formalized by the context source subtypes ContextService and ContextServiceCommunity, respectively. A context service is provided by an autonomous organization (i.e., context provider), collecting, refining, and disseminating context information. To solve the challenges of heterogeneous and dynamic context information, we abstract the concept of context service community, which enables the dynamic provisioning of optimal contexts. The concept is evolved from *service community* we developed in [3] and the details will be given in Section 3.1.3.

It should be noted that in ContextUML, we do not model the acquisition of context information, such as how to collect *raw* context information from sensors. Instead, context services that we abstract in ContextUML

[4] http://www.capescience.com/webservices/globalweather.

encapsulate sensor details and provide context information by interpreting and transforming the sensed information (i.e., raw context information). The concept of context service hides the complexity of context acquisition from CAS designers so that they can focus on the functionalities of CASs, rather than context sensing.

3.1.3 Context Service Community

A context service community aggregates multiple context services, offering with a unified interface. It is intended as a means to support the dynamic retrieval of context information. A community describes the capabilities of a desired service (e.g., providing user's location) without referring to any actual context service (e.g., `WhereAmI` service). When the operation of a community is invoked, the community is responsible for selecting the most appropriate context service that will provide the requested context information. Context services can join, leave communities at any time.

By abstracting ContextServiceCommunity as one of context sources, we can enable the dynamic context provisioning. In other words, CAS designers do not have to specify which context services are needed for context information retrieval at the design stage. The decision of which specific context service should be selected for the provisioning of a context is postponed until the invocation of CASs.

The selection can be based on a *multi-criteria utility function* [18, 3] and the criteria used in the function can be a set of *Quality of Context* (QoC) parameters [5]. The examples of QoC parameters are: i) `precision` indicating the accuracy of a context information; ii) `correctnessProbability` representing the probability of the correctness of a context information; and iii) `refreshRate` indicating the rate that a context is updated.

The quality of context is extremely important for CASs in the sense that context information is used to automatically adapt services or content they provide. The imperfection of context information may make CASs *misguide* their users. For example, if the weather information is outdated, our attractions searching service might suggest users to surf at the Bondi Beach although it is rainy and stormy. Via context service communities, the optimal context information is always selected, which in turn, ensures the quality of CASs.

It should be noted that *ontologies* play a pivotal role in understanding the semantics of the context services. An ontology refers to the formal, explicit description of a shared conceptualization of a domain of interest that are often conceived as a set of entities, relations, instances, functions, and axioms [10]. Context services described in ontologies possess explicit semantic representations, which makes the automatic selection of context services possible. The description of the detailed ontology model of context services is outside of the scope of this paper.

3.2. Context Awareness Modeling

A CAMechanism is a class that formalizes the mechanisms for context awareness (CA for short). We differentiate between two catergories of context awareness mechanisms by subtypes ContextBinding and ContextTriggering, which will be detailed in Section 3.2.1 and Section 3.2.2, respectively. Context awareness mechanisms are assigned to context-aware objects—modeled in the type CAObject—by the relation MechanismAssignment, indicating which objects have what kinds of context awareness mechanisms.

CAObject is a base class of all model elements in ContextUML that represent context-aware objects. There are four subtypes of CAObject: Service, Operation, Message, and Part. Each service offers one or more operations and each operation belongs to exactly one service. The relation is denoted by a composite aggregation (i.e., the association end with a filled diamond). Each operation may have one input and/or one output messages. Similarly, each message may have multiple parts (i.e., parameters). A context awareness mechanism can be assigned to either a service, an operation of a service, input/output messages of an operation, or even a particular part (i.e., parameter) of a message. It is worth mentioning that the four primitives are directly adopted from WSDL, which enables designers to build CASs on top of the previous implementation of Web services.

3.2.1 Context Binding

A ContextBinding is a subtype of CAMechanism that models the automatic binding of contexts to context-aware objects. By abstracting the concept of context binding, it is possible to automatically retrieve information for users based on available context information. For example, suppose that the operation of our example CAS has an input parameter `city`. Everyone who wants to invoke the service needs to supply a city name to search the attractions. Further suppose that we have a context `userLocation` that represents the city a user is currently in. A context binding can be built between `city` (input parameter of the service) and `userLocation` (context). The result is that whenever our CAS is invoked, it will automatically retrieve attractions in the city where the requester is currently located.

An automatic contextual reconfiguration (i.e., context binding) is actually a *mapping* between a context and a context-aware object (e.g., an input parameter of a service operation). The semantics is that the value of the object is supplied by the value of the context. Note that the value of a context-aware object could be derived from multiple contexts. For the sake of the simplicity, we restrict our mapping cardinality as one to one. In fact, thanks to the introduction of the concept of composite context, we can always model an appropriate composite context for a context-aware object whose value needs to be derived from multiple contexts.

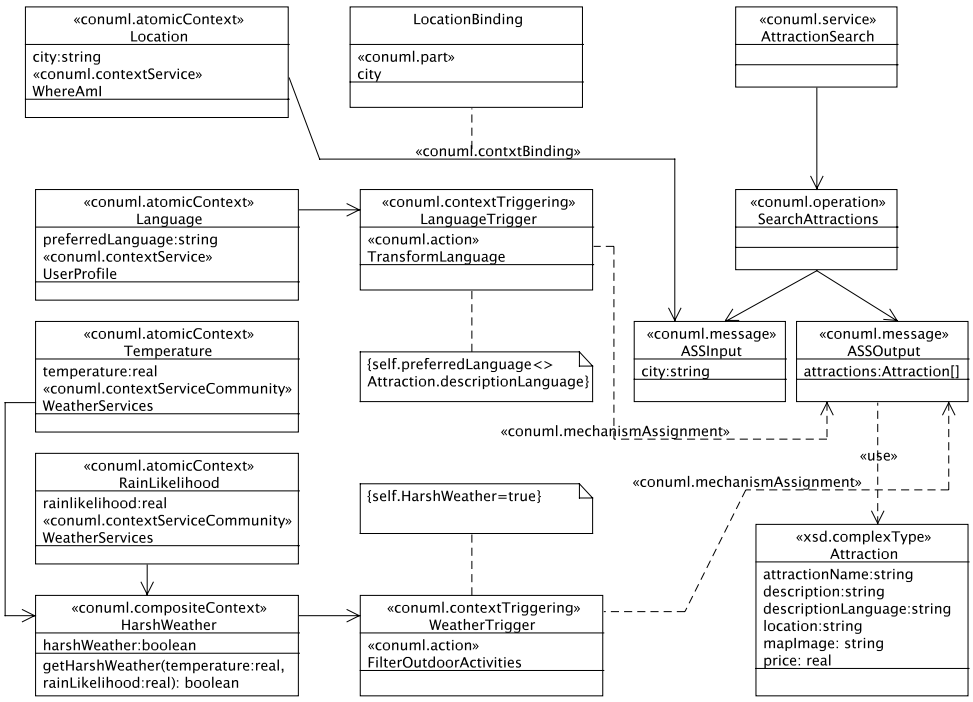

Figure 2. Attractions searching service

3.2.2 Context Triggering

The type ContextTriggering models the situation of contextual adaptation where services can be automatically executed or modified based on context information. A context triggering mechanism contains two parts: a set of *context constraints* and a set of *actions*, with the semantics of that the actions must be executed if and only if all the context constraints are evaluated to true.

A context constraint specifies that a certain context must meet certain condition in order to perform a particular operation. Formally, a context constraint is modeled as a predicate (i.e., a Boolean function) that consists of an operator and two or more operands. The first operand always represents a context, while the other operands may be either constant values or contexts. An operator can be either a prefix operator that accepts two or more input parameters or a binary infix operator (e.g., =, $\leq$, $\not\geq$, and $\ni$) that compares two values. Examples of context constraints can be: i) `harshWeather`$= true$; ii) `rainLikelihood`$\geq 80\%$.

Considering our attractions searching service, we can have a context triggering mechanism assigned to its output message. The constraint part of the mechanism is `harshWeather`$= true$, and the action part is a transformation function `filter`$(\mathcal{M}, \mathcal{R})$, where $\mathcal{M}$ is the output message and $\mathcal{R}$ is a transformation rule (e.g., selecting only indoor attractions)[5]. Consequently, when weather condition is not good, the output message will be automatically filtered (e.g., removing outdoor attractions) by the service.

4. ContextUML Notation

As introduced before, a notation of ContextUML is used to help designers create intuitive, readable CAS service models. In this section, we explain our UML-based notation (i.e., concrete syntax) and illustrate the semantics of ContextUML using our example CAS.

Figure 2 formalizes the attractions searching service, which in particular i) suggests attractions of the city where user is currently located, ii) transforms the results to the user's preferred language, if not expressed in this language, and iii) suggests *only* indoor attractions if the weather is not good.

4.1. Context

We start by declaring the contexts used in the service, namely `Location`, `Language`, `Temperature`, `RainLikelihood`, and `HarshWeather` (see Figure 2). The former four are atomic contexts that are represented by UML classes with the stereotype *conuml.atomicContext* [6]. Each atomic context class has two attributes. One is the context name and the other is an assigned context source (a context service or a community). The assigned context source should be contacted for the retrieval of the context. As Figure 2 suggests, the value of `Location` will be derived by a context service named `WhereAmI`, whereas the retrieval of

[5] It should be noted that the transformation can be achieved by e.g., applying the eXtensible Stylesheet Language Transformation (XSLT, http://www.w3.org/TR/xslt).

[6] We use *conuml* to indicate ContextUML in the stereotype.

Temperature and RainLikelihood will be cared by a context service community WeatherServices.

On the other hand, HarshWeather is a composite context represented by a UML class with the stereotype *conuml.compositeContext*. Unlike atomic context class, a composite context class has one attribute (i.e., context name). The business logic of the aggregation (i.e., how to compute the value of a composite context from its aggregated contexts) is implemented via an operation of the composite context class. For instance, HarshWeather aggregates Temperature and RainLikelihood using the operation getHarshWeather().

4.2. Context-Aware Objects

We then define the context-aware objects that—as discussed before—are basically service components (e.g., operation, input message). From Figure 2 we can see that our service AttractionSearch is represented as a class with the stereotype *conuml.service*. The service has one operation searchAttractions represented by a class with the stereotype *conuml.operation*.

The operation's input and output messages are represented by classes with stereotype *conuml.message*. The input message of the operation is a simple type (i.e., city:string), while the output message is an array of Attraction, which is a complex type that includes a couple of attributes like attractionName, location, description etc. It should be noted that we use WXS (W3C XML Schema) [7] for the declaration of data types in the operation messages. The prefix *xsd* is used in the stereotypes to identify WXS.

4.3. Context Awareness Mechanisms

Now we specify several context awareness mechanisms, each formalizing a requirement of the example CAS. The first requirement of our example CAS is modeled in a context binding called LocationBinding (Figure 2). A context binding is represented as an association class with the stereotype *conuml.contextBinding*, connecting a context with a UML class representing a context-aware object. The attribute of the class represents the binding object to the context. For example, LocationBinding connects context Location and input message of SearchAttractions, where the binding object is an input parameter (city, represented as *conuml.part*) of the message.

The latter two requirements of the service are formalized by the context triggering called Language-Trigger and WeatherTrigger (Figure 2). A context triggering is drawn as a class with the stereotype *conuml.contextTriggering*. Each attribute of the class represents an action of the context triggering. Each context triggering is associated with a context constraint. Context constraints are expressed using UML's *Object Constraint Language* (OCL). The constraint $self.harshWeather = true$ of the context triggering WeatherTrigger, for example, restricts the context awareness mechanism to cases where the weather is not good. The assignment of a context triggering mechanism is defined using a dependency relationship (dashed line with an arrow) with the stereotype *conuml.mechanismAssignment*.

[7] http://www.w3.org/TR/xmlschema-2.

5. Discussions and Conclusion

The work presented in this paper is related to model-driven development of context-aware Web services. Regarding the CASs development, very few research proposals have been presented in the literature. A representative effort is the work done by Keidl and Kemper in [12]. The authors propose a context framework for the development and deployment of context-aware adaptable Web services. In the framework, contexts are limited to the information of service requesters and are embedded into the SOAP messages. Further, the issue of context retrieval leaves open in their work. In contrast, our language provides not only context processing mechanisms for CASs development, but also rich primitives for modeling contexts and their retrieval.

Although model-driven software development is a well established practice [13], the technology is still in the early stages in terms of context-aware Web services development. To the best of our knowledge, the work presented in this paper is the first effort in modeling CASs development based on UML language. Recently, there are a few efforts emerged for model-driven Web services development. In [17], authors propose a UML-based model-driven method for Web services composition. While in [2], authors discuss a framework that supports model-driven development of Web services and show how a completed executable service specification of a Web service can be generated from its external specifications (e.g., protocol specifications and interface). Both efforts, however, do not address the development of CASs. In [6], authors propose some solutions for conceptual modeling of multi-channel, context-aware Web applications. The work is based on a modeling language called WebML, which was initially designed for the development of Web page (i.e., hypertext) based applications. Further, their context model does not cover the heterogeneity of contexts.

In this paper, we present ContextUML, a UML-based language for model-driven CASs development. We introduce the metamodel and notation of the language and illustrate its usage using an example CAS. Our ContextUML offers significant design flexibility to CAS designers. Firstly, ContextUML separates the modeling of context and context awareness from service components. A couple of context awareness mechanisms are abstracted and context awareness is achieved by assigning the mechanisms to relevant service components like service oper-

ations and their input/output messages. This separation eases both development and maintenance of CASs. If new context-aware requirements are needed in a CAS, the CAS designer only needs to modify/create context awareness mechanisms and (possibly) redo their assignments. The designer does not, however, need to adjust the implementation of the service components. Another benefit is that it is possible to use legacy codes of Web services in the development of CASs.

Secondly, the abstraction of context service communities provides a significant flexibility for context provisioning by dynamic binding of context services, and ensures the quality of context information by enforcing e.g., QoC based selection policy. CAS designers do not have to decide—and even do not have to know—which context services will be used at CAS design time. Communities also make context provisioning more robust. For example, if a selected context service from a community becomes unavailable, another context service can be selected from the community.

Finally, the concept of composite context improves the modeling power of context information to CAS designers. By applying composite contexts, service designers can model any high-level context attributes that are useful in CASs.

We view our work presented in this paper as a first step towards formalized development of context-aware Web services. Our ongoing work includes the generation of complete, executable implementation of CASs from their ContextUML service models. For example, it is possible to transform a CAS design model to a BPEL specification where context processing, service invocation, and context awareness handling are processed sequentially. The resulted BPEL specification can be executed in any BPEL execution engine. Another plan is to extend our modeling language to support the development of context-aware composite Web services.

In summary, we believe that, once the research and development work on the aspects described above has been completed, the approach will result in a comprehensive platform that can substantially reduce CASs development effort, and therefore, foster the flourish of electronic businesses in mobile and pervasive environments.

References

[1] T. Andrews et.al. Business Process Execution Language for Web Services 1.1. `http://www-106.ibm.com/developerworks/library/ws-bpel`.

[2] K. Baina, B. Benatallah, F. Casati, and F. Toumani. Model-Driven Web Service Development. In *Proc. of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04)*, Riga, Latvia, June 2004.

[3] B. Benatallah, M. Dumas, and Q. Z. Sheng. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. *Distributed and Parallel Databases, An International Journal*, 17(1):5–37, 2005.

[4] T. Buchholz, M. Krause, C. Linnhoff-Popien, and M. Schiffers. CoCo: Dynamic Composition of Context Information. In *Proc. of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, Boston, Massachusets, USA, August 2004.

[5] T. Buchholz, A. Küpper, and M. Schiffers. Quality of Context: What It Is And Why We Need It. In *Proc. of the 10th Workshop of the OpenView Univeristy Association (OVUA'03)*, Geneva, Switzerland, July 2003.

[6] S. Ceri, F. Daneil, and M. Matera. Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In *Proc. of the 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03)*, Roma, Italy, December 2003.

[7] Y. Chen and C. Petrie. Ubiquitous Mobile Computing. *IEEE Internet Computing*, 7(2):16–17, 2003.

[8] A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. Technical Report GIT-GVU-99-22, GVU Center, Georgia Institute of Technology, June 1999.

[9] D. S. Frankel. *Model Driven Architecture^{TM}: Applying MDA^{TM} to Enterprise Computing*. John Wiley & Sons, 2003.

[10] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[11] K. Henricksen and J. Indulska. A Software Engineering Framework for Context-Aware Pervasive Computing. In *Proc. of the Second IEEE Annual Conference on Pervasive Computing and Communications (PerCom'04)*, Orlando, Florida, USA, March 2004.

[12] M. Keidl and A. Kemper. Towards Context-Aware Adaptable Web Services. In *Proc. of the 13th International World Wide Web Conference (WWW'04)*, New York, USA, May 2004.

[13] S. Mellor, A. N. Clark, and T. Futagami. Special Issue on Model-Driven Development. *IEEE Software*, 20(5):14–18, 2003.

[14] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. In *Proc. of the 2nd International Symposium on Wearable Computers*, Pittsburgh, USA, October 1998.

[15] D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proc. of the Conference on Human Factors in Computing Systems (CHI'99)*, Pittsburgh, PA, USA, May 1999.

[16] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proc. of the 1st International Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, USA, December 1994.

[17] D. Skogan, R. Gronmo, and I. Solheim. Web Service Composition in UML. In *Proc. of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC'04)*, California, USA, September 2004.

[18] M. Stolze and M. Stoebel. Utility-based Decision Tree Optimization: A Framework for Adaptive Interviewing. In *Proc. of the 8th International Conference on User Modeling (UM'01)*, Sonthofen, Germany, July 2001.