ContextServ: A Platform for Rapid and Flexible Development of Context-Aware Web Services

Quan Z. Sheng[1], Sam Pohlenz[1], Jian Yu[1], Hoi S. Wong[1], Anne H.H. Ngu[2], and Zakaria Maamar[3]

[1] School of Computer Science
The University of Adelaide
Adelaide, SA 5005, Australia
{qsheng, sam, jyu, hoi}@cs.adelaide.edu.au

[2] Department of Computer Science
Texas State University
San Marcos, TX, USA
angu@txstate.edu

[3] College of Information Technology
Zayed University
Dubai, UAE
zakaria.maamar@zu.ac.ae

Abstract

Context-aware Web services are currently emerging as an important technology for building innovative context-aware applications. Unfortunately, context-aware Web services are still difficult to build. This paper describes ContextServ, *a platform for rapid development of context-aware Web services. ContextServ adopts model-driven development where context-aware Web services are specified using ContextUML, a UML based modeling language. The platform also offers a set of automated tools for generating and deploying executable implementations of context-aware Web services. This paper presents the motivation, system design, implementation, and usage of ContextServ.*

1 Introduction

Context awareness, which has been identified as one of the key challenges for the next decade, refers to the capability of an application or a service being aware of its physical environment or situation (i.e., context) and responding proactively and intelligently based on such awareness [1, 4, 5]. With recent developments in computer hardware, software, networking, and sensor technologies, context awareness becomes one of the most important trends in computing today that holds the potential to make our daily lives more productive, convenient, and enjoyable. For example, a dining service gives users suggestions on where to have lunch by considering their current locations, food preferences, and even the prevailing weather conditions.

Context-aware Web services (CASs) [2, 8] are recently emerging as an important technology for building innovative context-aware applications. Unfortunately, despite the active research into, and development of, Web services over the last few years, CASs are still difficult to build [2, 6, 9]. One reason for this difficulty is that current Web services standards, such as the Web Services Description Language (WSDL) and the Simple Object Access Protocol (SOAP) [3], are not sufficient for describing and handling context information [6, 9]. CAS developers must implement everything related to context management, including collection, dissemination, and usage of context information, in an ad hoc manner. Due to heterogeneity of context providers, quality of context information, and dynamics of context environments, context provisioning is not trivial [4]. In particular, various context providers may provide the same piece of context information (usually with different quality and data formats) and it is difficult to specify, at service design stage, which context provider should be contacted for the provision of a specific context. Sometimes, the context required by a CAS may not even be directly available through any context provider. Furthermore, Web service platforms often evolve rapidly, with continuous updating of existing platforms, as well as the regular appearance of new alternatives [9]. Service developers must dedicate considerable effort to manually porting service code to different platforms or new versions of the same platform. As a consequence, developing and maintaining CASs is a very cumbersome, error-prone, and time consuming activity, especially when these CASs are complex.

Motivated by these concerns, we have developed the ContextServ platform for rapid development of CASs. One innovative feature of ContextServ is to use a model-driven approach that offers significant design flexibility by separating the modeling of context and context awareness from service components, which eases both development and maintenance of CASs. Another feature of ContextServ is that it supplies a set of automated tools for generating and deploying executable implementations of CASs. As a result, development costs can be significantly reduced. In the following sections, we overview the design and implementation of ContextServ, and sketch the proposed demonstration.

2 System Overview

ContextServ adopts model-driven development (MDD) and the basic idea of MDD is illustrated in Figure 1. Adopt-

ing a higher-level of abstraction, software systems can be specified in platform independent models (PIMs), which are then (semi)automatically transformed into platform specific models (PSMs) of target executable platforms using some transformation tools. The same PIM can be transformed into different executable platforms (i.e., multiple PSMs), thus considerably simplifying software development.

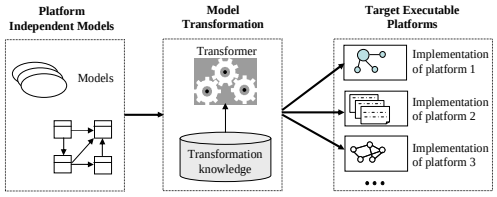

Figure 1. Model-driven development

ContextServ relies on ContextUML [7], a Unified Modeling Language (UML) based modeling language that provides high-level, visual constructs for specifying CASs. In particular, the language abstracts two context awareness mechanisms, namely *context binding* and *context triggering*. The former models automatic contextual configuration (e.g., automatic invocation of Web services by mapping a context onto a particular service input parameter), while the latter models contextual adaptation where services can be dynamically modified based on context information. Service models specified in ContextUML are then automatically translated into executable implementations (e.g., WS-BPEL specifications) of specific target service implementation platforms (e.g., IBM's BPWS4J[1]).

ContextServ distinguishes between *atomic* and *composite* contexts. The former are low-level and independent contexts that do not rely on any other contexts and can be provided directly by context providers. By contrast, the latter are high-level contexts that may not have direct counterparts on the context provision and have to aggregate multiple contexts, either atomic or composite. For instance, in the scenario of attraction searching service, `temperature` and `rainLikelihood` are atomic contexts because they can be provided by e.g., `GlobalWeather` Web service[2]. Whereas, `harshWeather` is a composite context that depends on the former two contexts. For example, if `temperature`>40°C or `rainLikelihood`>80%, the value of `harshWeather` may be set to "true". 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.

ContextServ exploits the concept of *context community* to address the issue posed by heterogeneous and dynamic context information. A context community is essentially

[1]http://www.alphaworks.ibm.com/tech/bpws4j.

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

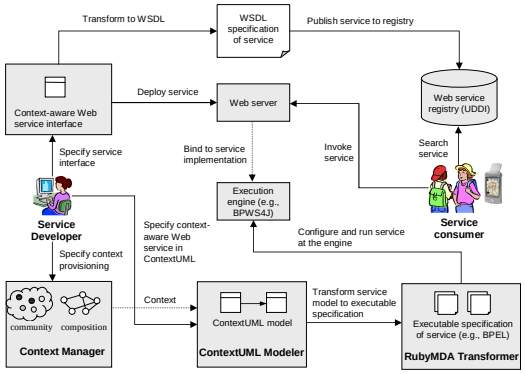

Figure 2. Architecture of ContextServ

a container where multiple context sources—from which contexts are retrieved—are aggregated and a unified interface is offered. The abstraction of context communities provides a significant flexibility for context provisioning through dynamic binding of context resources, and ensures the quality of context information by enforcing Quality of Context (QoC) based selection policies. CAS designers do not have to decide and even do not have to know, at service design time, which context providers will be used.

3 Implementation

The ContextServ platform provides an environment (Figure 2) where a service developer specifies the required contexts and context-aware Web services using high-level and visual modeling languages. The service model is automatically transformed, using a set of transformation rules, to the executable specification of the target platform, which is then deployed to the corresponding execution engine. At this point, the service provider also needs to create a WSDL specification for the service and publish it (e.g., to UDDI registry) for free location and invocation.

The ContextServ architecture features three main components, namely the *context manager*, the *ContextUML modeler*, and the *RubyMDA transformer*. The context manager provides facilities for service developers to specify context provisioning. Current implementation supports atomic context, composite context, and context community. In ContextServ, composite contexts are modeled using statecharts, a widely used formalism that is emerging as a standard for process modeling following its integration into UML. The statechart of a composite context is then exported into State Chart Extensible Markup Language (SCXML), an XML based language for describing generic statecharts, and executed in a SCXML execution engine such as Commons SCXML[3]. A context community implements a common interface (e.g., `addContextSource()`, `removeContextSource()`, `selectContextSource()`) for con-

[3]http://commons.apache.org/scxml.

text sources that provide same context information.

The ContextUML modeler provides a visual interface for defining context-aware Web services using ContextUML. In the implementation, we extended ArgoUML, an existing UML editing tool[4], by developing a new diagram type, ContextUML diagram, which implements all the abstract syntax of the ContextUML language [7]. Services represented in ContextUML diagrams are exported as XMI files for subsequent processing by the *RubyMDA transformer*, which is responsible for transforming ContextUML diagrams into executable Web services, using RubyGems 1.0.1[5]. The ContextServ platform currently supports WS-BPEL, a de facto standard for specifying executable processes. Once the BPEL specification is generated, the model transformer deploys the BPEL process to a Web server and exposes it as a Web service. In the implementation, JBoss Application Server is used as the Web server since it is an open source and includes a BPEL execution engine, jBPM-BPEL 1.1. A set of mapping rules—from ContextUML diagram to BPEL and WSDL specifications—has been developed for the transformation purposes.

4 Demonstration Scenario

Several context-aware applications have been developed using ContextServ platform. In this demonstration, we focus on a tourism service that provides "intelligent" recommendations of attractions to tourists. The service works as follows: (i) the service recommends attractions according to a user's location (e.g., the city that the user is currently in), and (ii) during the recommendation, the service also considers other contexts like weather. If the weather is harsh, the service will only suggest indoor attractions (e.g., South Australian Museum). The definition of weather being harsh depends on a couple of contexts like temperature (e.g., above 40 degree Celsius) and likelihood of rain (e.g., more than 80%).

The ContextServ platform provides an integrated environment where service developers can specify and deploy context-aware Web services. We will demonstrate: (i) how to specify contexts, (ii) how to define an intelligent attractions search service using ContextUML language, and (iii) how to automatically transform the service model into an executable Web service.

Defining context provisioning. The context manager offers facilities that service developers can use to specify different kinds of contexts involved in a CAS. The specifications of the contexts, including context name and context retrieval mechanism, are stored in an XML document, for subsequent usage in the specification of CASs. The contexts of attractions search Web service are specified as shown in Figure 3. The context `location` is an

[4]http://argouml.tigris.org.

[5]http://rubyforge.org/projects/rubygems.

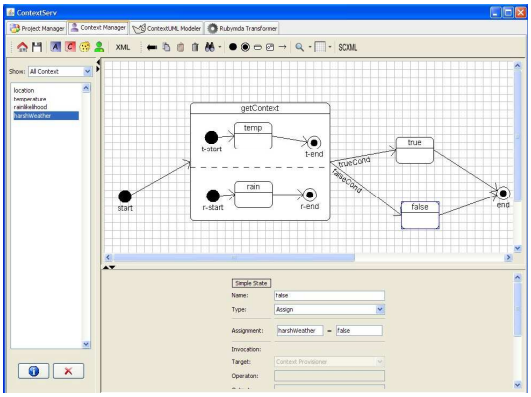

Figure 3. Specifying contexts

atomic context, provided by a location-based service that returns the location information of a user with a GPS enabled mobile device. The contexts `temperature` and `rainlikelihood` are provided by a context community that aggregates several weather forecast services. Finally, the context `harshWeather` is a composite context, which specification is displayed in the right panel of Figure 3.

Defining a context-aware Web service using ContextUML. The ContextUML modeler offers a graphical user interface (GUI) (Figure 4) allowing service developers to specify context-aware Web services using ContextUML language. We developed a ContextUML diagram that implements all ContextUML constructs and integrated the module into ArgoUML (toolbars in the top of Figure 4), so that service developers can use them. A context-aware Web service is defined by drawing a ContextUML diagram (top right panel of Figure 4). The information associated with each element of the diagram (e.g., attributes of a class) can be specified in the bottom left panel of Figure 4.

Transforming service model into an executable Web service. Once a context-aware Web service has been defined using the CAS modeler, the model transformer comes into play during the model transformation process. This process takes input the XMI document of the service model—produced by the CAS modeler—and performs the following tasks: (i) converting the service model into executable Web service specifications, including BPEL and WSDL specification and the relevant configuration files; and (ii) deploying the BPEL process to the Web server and exposing it as a Web service.

The model transformer provides a set of automated tools performing the transformation. In particular, `XML2UML` and `UML2CAS` are used to transform the XMI document into a Ruby structure representing the corresponding context-aware Web service. `CAS:BPEL` is used to generate BPEL and WSDL files of the service, while `CAS:JBoss` is used to produce specific files that are required to deploy the BPEL process to an execution engine (jBPM-BPEL in our case) that is running on a Web server (JBoss Application

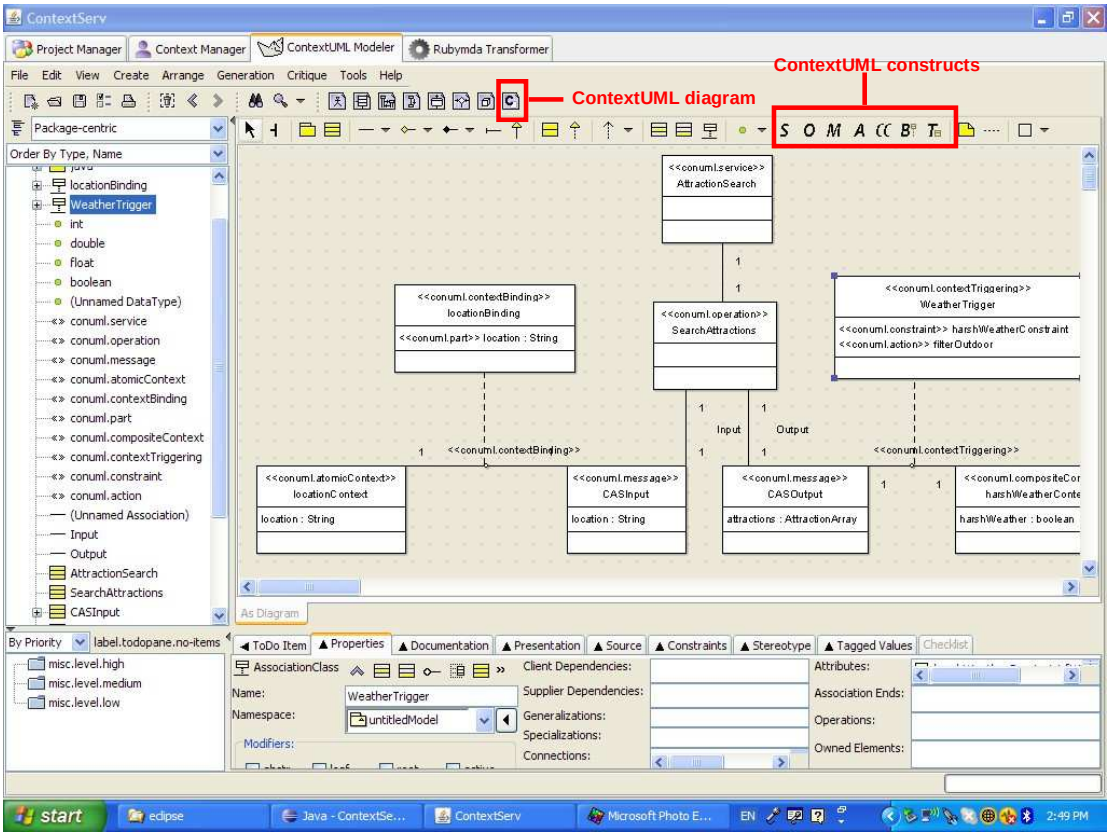

Figure 4. Defining context-aware Web services using ContextUML

Server in our case). A Web service can be published to the UDDI registry, from which an end user can locate and invoke it.

5 Conclusion

In this paper, we have presented ContextServ, a comprehensive platform for simplifying the development of context-aware Web services. ContextServ adopts model-driven development where context-aware Web services are specified in a high-level modeling language and their executable implementations are automatically generated, thus contributing significantly to both design flexibility and cost savings. The platform has been validated by successfully creating a number of context-aware Web services. Currently, we are extending the platform to: i) support more context triggering mechanisms and ii) introduce semantic support of context provisioning. Interested readers are referred to the project website[6] for more details.

References

[1] G. D. Abowd et al. Context-Aware Computing. *IEEE Pervasive Computing*, 1(3):22–23, 2002.

[2] D. Benslimane and Z. Maamar. Special Issue on Context-Aware Web Services. *Distributed and Parallel Databases*, 21(1):1–3, 2007.

[3] F. Curbera et al. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, March 2002.

[4] A. K. Dey and J. Mankoff. Designing Mediation for Context-aware Applications. *ACM Trans. on Computer-Human Interaction*, 12(1):53–80, 2005.

[5] C. Julien and G.-C. Roman. EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications. *IEEE Trans. on Software Engineering*, 32(5):281–298, 2006.

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

[7] Q. Z. Sheng and B. Benatallah. ContextUML: A UML-Based Modeling Language for Model-Driven Context-Aware Web Service Development. In *Proc. of the 4th Intl. Conf. on Mobile Business (ICMB'05)*, Sydney, Australia, July 2005.

[8] Q. Z. Sheng et al. WS3 - International Workshop on Context-Enabled Source and Service Selection, Integration and Adaptation. In *Proc. of the 17th Intl. World Wide Web Conf. (WWW'08)*, Beijing, China, April 2008.

[9] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed. Deploying and Managing Web Services: Issues, Solutions, and Directions. *The VLDB Journal*, 17(3):537–572, 2008.

[6] http://www.cs.adelaide.edu.au/∼contextserv.