The current issue and full text archive of this journal is available at
www.emeraldinsight.com/1744-0084.htm

Techniques on developing context-aware web services

Quan Z. Sheng and Jian Yu
School of Computer Science, The University of Adelaide, Adelaide, Australia
Aviv Segev
Department of Knowledge Service Engineering, KAIST, Daejeon, South Korea, and
Kewen Liao
School of Computer Science, The University of Adelaide, Adelaide, Australia

Received 16 December 2009
Revised 15 March 2010
Accepted 24 May 2010

Abstract

Purpose – In the last decade, web services have become a major technology to implement loosely coupled business processes and perform application integration. Through the use of context, a new generation of web services, namely context-aware web services (CASs), is currently emerging as an important technology for building innovative context-aware applications. Unfortunately, CASs are still difficult to build. Issues like lack of context provisioning management approach and lack of generic approach for formalizing the development process need to be solved in the first place for easy and effective development of CASs. The purpose of this paper is to investigate the techniques on developing CASs.

Design/methodology/approach – The paper focuses on introducing a model-driven platform, called ContextServ, and showcasing how to use this platform to rapidly develop a context-aware web application, Smart Adelaide Guide. ContextServ adopts a model-driven development (MDD) approach where a Unified Modeling Language (UML)-based modeling language – ContextUML – is used to model web services and its context-awareness features.

Findings – The paper presents novel techniques for efficient and effective development of CASs using a MDD approach. The ContextServ platform is the only one that provides a comprehensive software toolset that supports graphical modeling and automatic model transformation of CASs.

Practical implications – The proposed approach has been validated in practice by developing various CASs. The experimental study demonstrates the efficiency and effectiveness of the approach.

Originality/value – The paper presents a novel platform called ContextServ, which offers a set of visual editing and automation tools for easy and fast generating and deploying CASs.

Keywords World wide web, Computer applications, Context-sensitive languages

Paper type Research paper

1. Introduction

Over the years, the web has gone through many transformations, from traditional linking and sharing of computers and documents (i.e. "web of data") to current connecting of people (i.e. "web of people"). With the recent advances in radio-frequency identification technology, sensor networks, and web services, the web is continuing

The work reported in this paper has been supported by ARC Discovery Grant DP0878367. The authors would like to thank Sam Pohlenz and Hoi S. Wong for their participation in the implementation of ContextServ.

International Journal of Web Information Systems
Vol. 6 No. 3, 2010
pp. 185-202

1744-0084
DOI 10.1108/17440081011070141

the transformation and will be slowly evolving into the so-called "web of things and services" (Roussos *et al.*, 2009; Sheng *et al.*, 2010a, b; Welbourne *et al.*, 2009). Indeed, this future web will provide an environment where everyday physical objects such as buildings, sidewalks, and commodities are readable, recognizable, addressable, and even controllable using services via the web. The capability of integrating the information from both the physical world and the virtual one not only affects the way how we live, but also creates tremendous new web-based business opportunities such as support of independent living of elderly persons, intelligent traffic management, efficient supply chains, and improved environmental monitoring.

In the last decade, web services have become a major technology to implement loosely coupled business processes and perform application integration. Through the use of context, a new generation of smart web services is currently emerging as an important technology for building innovative context-aware applications. We call such category of web services as context-aware web services (CASs). CASs are emerging as an important technology to underpin the development of new applications (user centric, highly personalized) on the future ubiquitous web. A CAS is a web service that uses context information to provide relevant information and/or services to users (Julien and Roman, 2006; Dey and Mankoff, 2005; Kapitsaki *et al.*, 2009; Sheng and Benatallah, 2005; Sheng *et al.*, 2010a, b). A CAS can present relevant information or can be executed or adapted automatically, based on available context information. For instance, a tour-guide service gives tourists suggestions on the attractions to visit by considering their current locations, preferences, and even the prevailing weather conditions (Liao *et al.*, 2009).

Although the combination of context awareness (CA) and web services sounds appealing, injecting context into web services raises a number of significant challenges, which have not been widely recognized or addressed by the web services community (Sheng *et al.*, 2008, 2010a, b; Yu *et al.*, 2008). One reason is that current web services standards (e.g. Universal Description Discovery and Integration, Web Services Description Language (WSDL), and Simple Object Access Protocol) are not sufficient for describing and handling context information (Keidl and Kemper, 2004; Niu *et al.*, 2008; Yu *et al.*, 2008). 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.

This paper presents how to rapidly build a prototype context-aware web application called "Smart Adelaide Guide" (SAG) using the ContextServ platform (Sheng *et al.*, 2009). SAG recommends tourist attractions based on their current locations, preferred languages, and weather conditions. ContextServ is a platform for rapid development of CASs. ContextServ uses a Unified Modeling Language (UML)-based modeling language – ContextUML (Sheng and Benatallah, 2005) – for formalizing the design and development of CASs. ContextUML provides constructs for:

- generalizing context provisioning that includes context attributes specification and retrieval; and
- formalizing CA mechanisms and their usage in CASs.

ContextServ supports the full lifecycle of developing CASs, including a visual ContextUML editor, a ContextUML to Web Services Business Process Execution

Language (WS-BPEL) translator, and a WS-BPEL deployer working with the JBoss Application Server.

The remainder of this paper is organized as follows. Section 2 gives a brief description of the key functions of the SAG application. Section 3 introduces the ContextUML language and Section 4 introduces the architecture and implementation of the ContextServ platform. Section 5 presents the design of context community, an important concept for optimized selection of context information for CASs. Finally, Section 6 discusses the related work and Section 7 provides some concluding remarks.

2. Smart Adelaide Guide

SAG is a context-aware web application, which offers an interface helping tourists to find interesting places to visit in Adelaide, the capital city of South Australia. SAG recommends attractions based on a user's current location, current weather condition in Adelaide[1], and a user's preferred proximity limitation to the attractions (e.g. 2 km) and language (e.g. French). Figure 1 shows a screenshot of the application.

To use SAG, simply click on any part of the map to indicate a user's current location and a blue balloon will appear on the clicked point with its latitude and longitude. The next step is to choose a proximity from current location and a preferred language for attraction description from the dropdown lists next to the map. After clicking on the Invoke button, the context-aware attraction search web service will be accessed and a list of attractions satisfying the user's current context, including the location, the weather, and the proximity, will appear both as red balloons on the map and as a list of links next to the map. These links contain further information about the attractions. The names and descriptions of the attractions are automatically translated to the selected preferred language. A clear guideline on how to use this application is also given on the web site of the application.

What is happening at the back end is that SAG relies on a context-aware attractions search web service. This web service dynamically adapts its service by considering

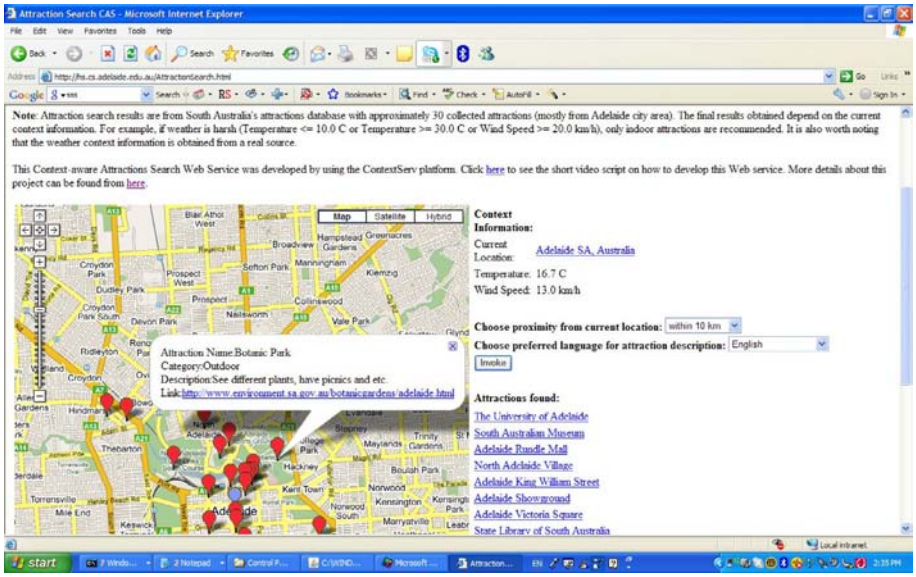

Figure 1.
The SAG a context-aware web applications

several context information of a user. For example, if the weather is harsh (i.e. temperature $\leq 10\,^{\circ}\mathrm{C}$ or $\geq 30\,^{\circ}\mathrm{C}$ or wind speed ≥ 20 km/h), only indoor attractions (e.g. Adelaide Art Gallery) are recommended.

This CAS was developed by using the ContextServ platform (Sheng *et al.*, 2009). The ContextServ platform has been developed from a research project sponsored by Australian Research Council (ARC) (www.cs.adelaide.edu.au/~contextserv). The ContextServ platform offers an environment for rapid and flexible development of CASs. In next sections, we will focus on the description of the relevant novel techniques proposed in the ContextServ platform, as well as explaining how SAG was developed from this platform.

3. ContextUML

In this section, we first introduce the ContextUML, an UML-based language for model-driven development (MDD) of CASs (Sheng and Benatallah, 2005). ContextUML metamodel is shown in Figure 2, which can be divided into two parts: context modeling metamodel and CA modeling metamodel.

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. 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 SAG, temperature and wind speed are atomic contexts because they can be provided by a local weather forecast 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) by collecting, refining, and disseminating context information. To solve the challenges

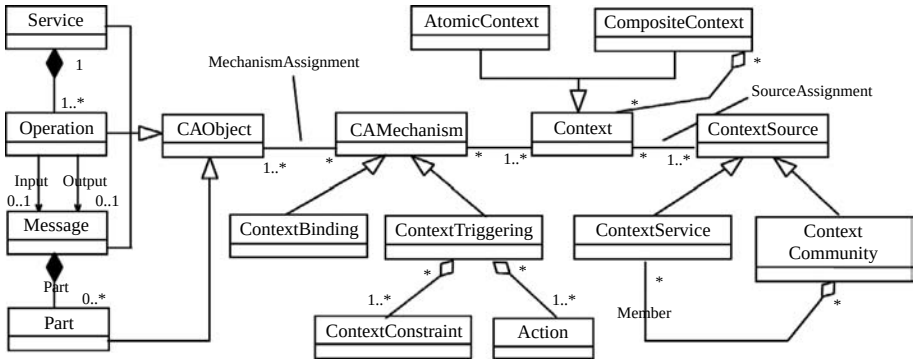

Figure 2.
ContextUML metamodel

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 which was developed by Benatallah *et al.* (2005) and 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 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 and leave the communities at any time.

By abstracting ContextServiceCommunity as one of the 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 (Stolze and Ströbel, 2001; Benatallah *et al.*, 2005) and the criteria used in the function can be a set of quality of context (QoC) parameters (Buchholz *et al.*, 2003). The examples of QoC parameters are:

- precision indicating the accuracy of a context information;
- correctnessProbability representing the probability of the correctness of a context information; and
- refreshRate indicating the rate that a context is updated.

The QoC 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. Since context community is an important concept for CASs, we have a separate section that gives more details on our design (Section 5).

3.2 CA modeling

A CAMechanism is a class that formalizes the mechanisms for CA. We differentiate between two categories of CA mechanisms by subtypes ContextBinding and ContextTriggering, which will be detailed in Sections 3.2.1 and 3.2.2, respectively. CA mechanisms are assigned to context-aware objects – modeled in the type

CAObject – by the relation MechanismAssignment, indicating which objects have what kinds of CA 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 CA mechanism can be assigned to 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.

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. $=$, $\leq$) that compares two values. Examples of context constraints can be:

(1) harshWeather $=$ true; and

(2) windSpeed ≤ 25.

Considering our SAG application, 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(M,R), where M is the output message and R is a transformation rule (e.g. selecting only indoor attractions). Consequently, when weather condition is not good, the output message will be automatically filtered (e.g. removing outdoor attractions) by the service.

4. The ContextServ platform

In this section, we will introduce the ContextServ, a comprehensive platform for simplifying the development of CASs.

ContextServ adopts MDD (Frankel, 2003; Mellor *et al.*, 2003) and the basic idea of MDD is illustrated in Figure 3. Adopting a high 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.

ContextServ relies on ContextUML (Section 3), an UML-based modeling language that provides high-level visual constructs for specifying CASs. In particular, the language abstracts two CA 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) (www.alphaworks.ibm.com/tech/bpws4j).

The ContextServ architecture (Figure 4) features three main components, namely the context manager, the ContextUML modeler, and the RubyMDA transformer, representing the three major steps in developing CASs. All these components are implemented in Java. In the following subsections, we present the details of these three components.

4.1 Context manager

The context manager provides facilities for service developers to specify context provisioning. Current implementation supports the management of atomic context, composite context, and context community.

4.1.1 Managing atomic contexts and composite contexts. As mentioned before, atomic contexts are low-level contexts that can be obtained directly from context sources.

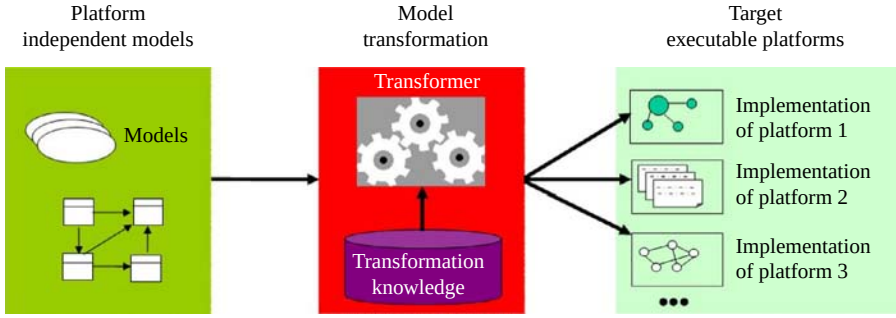

Figure 3.
Model-driven development

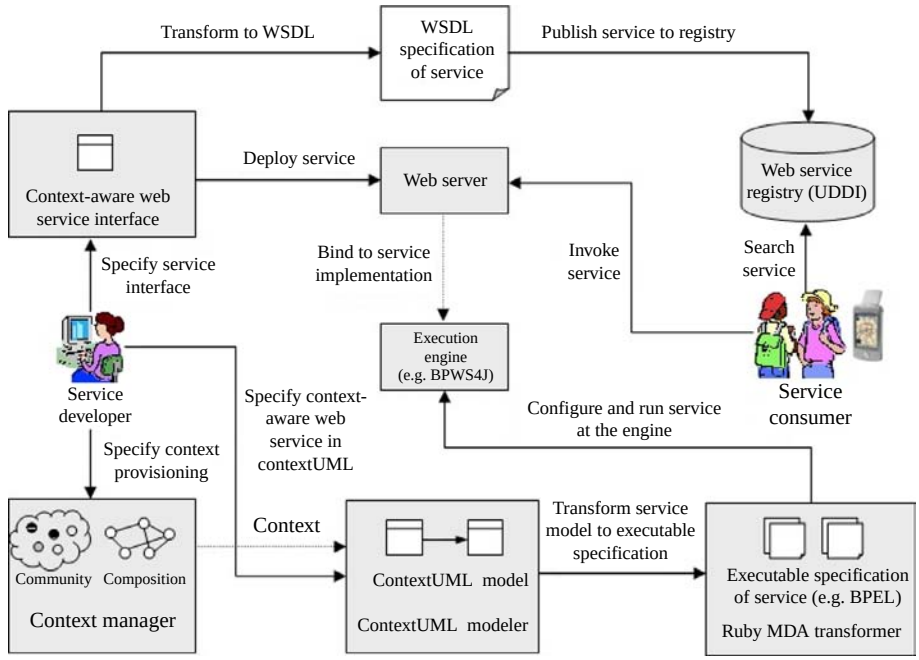

Figure 4.
Architecture of the ContextServ platform

For the ContextServ platform to access context sources, context providers must be registered in the platform. Currently, the platform supports two types of context providers: local context providers and remote context providers. Local context providers are responsible for collecting local context information, such as device memory capacity, CPU usage, etc.

On the other hand, remote context providers gather context information from a remote sensor or device. Every context provider has at least one agent – a piece of program specifying the protocol on how to access the context information. For example, we use a web service agent to access remote contexts that have a web service interface and use a Java class to access local contexts. Figure 5 shows an example of panel for defining an atomic context location. On the left side of the panel, we register a remote context provider containing a web service agent to get the location context, and on the right, side we define the collected context as a location atomic context with type String.

Composite contexts are modeled using statecharts in ContextServ, as shown in Figure 6. Statecharts are 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) (Barnett *et al.*, 2010), an XML-based language for describing generic statecharts, and executed in a SCXML execution engine such as Commons SCXML (http://commons.apache.org/scxml).

4.1.2 Managing context community. A context community implements a common interface (addContextSource(), removeContextSource(), selectContextSource()) for context sources that provide the same context information. The main purpose of a context community is to ensure robust and optimal provisioning of contexts to the context consumer so that on one hand a candidate context source can take the place

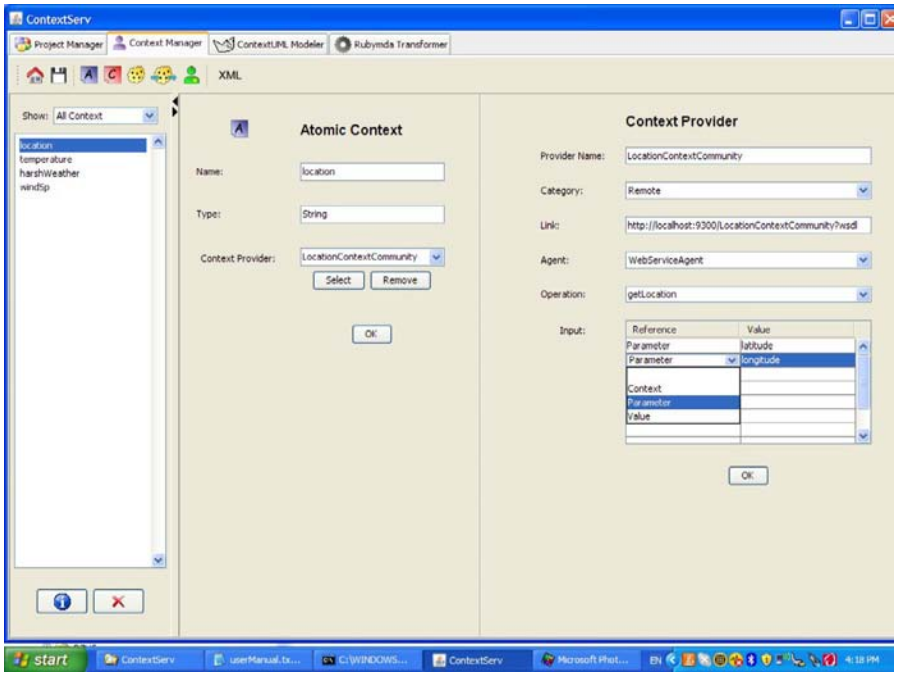

Figure 5.
Panel for defining atomic context

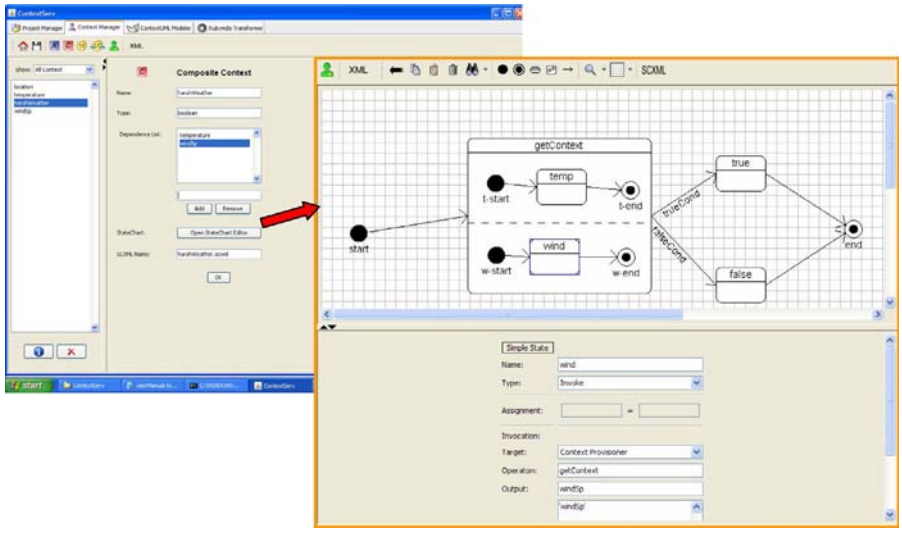

Figure 6.
Specifying composite contexts

of an unavailable context source, and on the other hand contexts having the best quality can be provisioned.

Assume that in a context community, a specific piece of context information is provisioned by n context providers $\{CP_i\}_{1 \leq i \leq n}$. The quality of a context provider

($QoCP_i$) can be modeled by a set of m quality attributes $\{A_j\}_{1 \leq j \leq m}$ such as precision, trustworthiness, availability, response time, etc. Also, each quality attribute can be assigned a weight W_j, so the quality of each context provider can be calculated. More details on the implementation of context communities will be reported in Section 5.

4.2 ContextUML modeler

The ContextUML modeler provides a visual interface (Figure 7) for defining CASs using ContextUML. In the implementation, we extended ArgoUML, an existing UML editing tool (http://argouml.tigris.org), by developing a new diagram type, ContextUML diagram, which implements all the abstract syntax of the ContextUML language (Sheng and Benatallah, 2005).

4.3 RubyMDA transformer

Services represented in ContextUML diagrams are exported as XML Metadata Interchange (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 (http://rubyforge.org/projects/rubygems). 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 an application server and exposes it as a web service. In the implementation, JBoss Application Server is used since it is open source and includes a BPEL execution engine jBPM-BPEL. RubyMDA is developed based on the model transformation rules. The model transformation rules are mappings from ContextUML

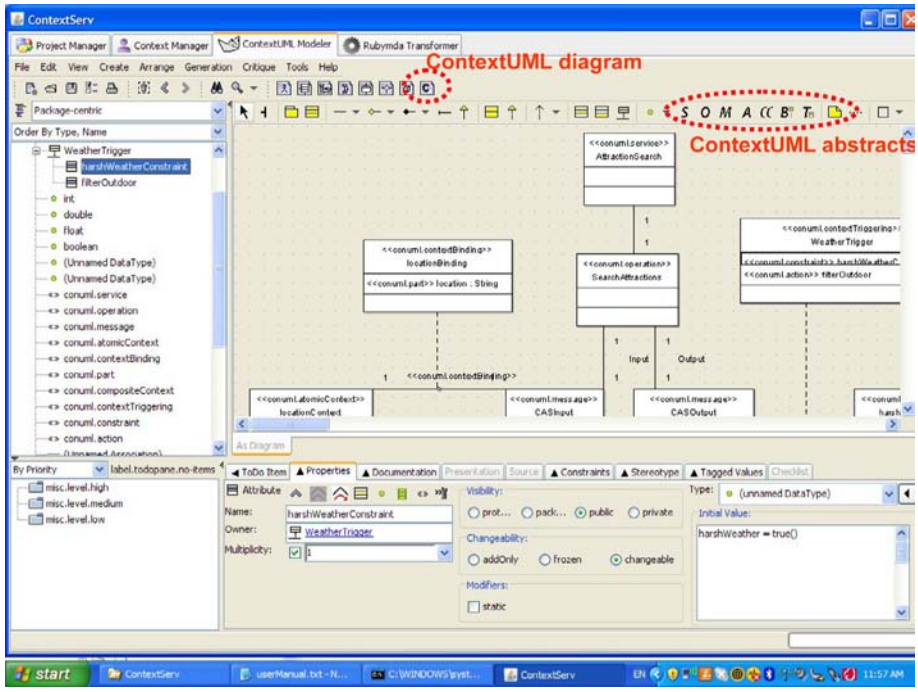

Figure 7.
The ContextUML modeler

stereotypes to BPEL elements. Table I shows a summary of the model transformation rules of RubyMDA.

Figure 8 shows the data flow of RubyMDA model transformer. RubyMDA takes the XMI document as an input, which represents the ContextUML diagram. RubyMDA reads the XMI document and constructs the UML model, which is a set of data structure representing the components in UML class diagram. After the UML model is constructed, RubyMDA transforms it into CAS model, which is a set of data structure representing the CAS described in ContextUML diagram. Finally, RubyMDA generates a BPEL process and WSDL document for a CAS. Moreover, it generates a set of deployment files needed to deploy CAS to a server.

5. Context community: optimization and evaluation

As previously mentioned, context community is a very important concept for optimized and dynamic context information provision, which is of paramount importance to the QoC-aware web services. In ContextServ platform, we have paid a significant attention to the development of context communities. In particular, we developed a module called context community manager (CCM) for developing context communities in ContextServ. In this section, we will describe some technical details of CCM and also presents some performance evaluation. CCM aims at solving the following issues:

- A single context provider may fail to provide the requested context information due to various reasons such as the server is unavailable or the sensor it uses for sensing the raw context information is broken down.

From: UML stereotypes	To: BPEL element
conuml.service	<process>
conuml.operation	<invoke>
conuml.message	<variable>
conuml.atomicContext	<invoke>
conuml.compositeContext	<invoke>
conuml.contextBinding	<assign>
conuml.part	Part attribute in <to>
conuml.contextTriggering	<switch> <invoke>

Table I.
RubyMDA's model transformation rules

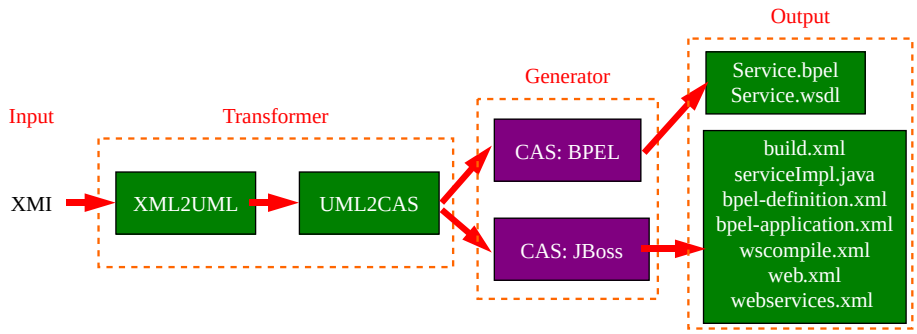

Figure 8.
RubyMDA data flow

- There may exist multiple context providers providing a same piece of context information (usually with different levels of quality). It is hard for a CAS to ensure that the context provider it interacts with provides the optimal context information.

CCM serves as a broker between a CAS and its context providers and will select the context provider who provides optimal context for the CAS from all the context providers. CCM has four main components:

(1) The context retrieval process is implemented as a manager for retrieving and unifying contexts from heterogeneous sources.

(2) The context-monitoring process is responsible for actively monitoring the quality information of the registered context providers by collecting and keeping their quality information obtained through the context-retrieval process.

(3) The context evaluation process is responsible for evaluating the quality information obtained from the context-monitoring process.

(4) The context-selection process is implemented for selecting the best context provider that provides the optimal context information at runtime by using the results obtained from the context-evaluation process.

5.1 Optimal context selection

Suppose that a specific piece of context c (e.g. temperature) can be supplied by a set of context providers, i.e. $\{CP_i\}_{1 \le i \le n}$. These context providers are members of a context community (e.g. a weather context community C_w). The QoCP for each CP_i is modeled by a set of m quality attributes $\{A_j\}_{1 \le j \le m}$, e.g. precision, response time, availability, and a distribution of weights $\{W_j\}_{1 \le j \le m}$ over these attributes, where $\sum_{j=1}^{m} W_j = 1$. To calculate the score of each context provider, we use the following multi-attribute utility function (Benatallah *et al.*, 2005):

$$U(CP_i) = \sum_{j=1}^{m} W_j \times S_{i,j}$$

where $S_{i,j}$ represents the score of A_j of CP_i.

The score matrix S is derived from scaling the initial attributes value matrix I. Since there are positive attributes (e.g. availability) where a greater value indicates a better quality, and also negative attributes (e.g. response time) where a less value indicates a better quality, the score of each A_j is calculated differently for positive and negative attributes:

If A_j is negative:

$$S_{i,j} = \begin{cases} \frac{I_j^{\max} - Ii,j}{I_j^{diff}} & I_j^{diff} \neq 0 \\ 1 & I_j^{diff} = 0 \end{cases}$$

If A_j is positive:

$$S_{i,j} = \begin{cases} \frac{I_{i,j} - I_j^{\min}}{I_j^{diff}} & I_j^{diff} \neq 0 \\ 1 & I_j^{diff} = 0 \end{cases}$$

where: $I_j^{\max} = \max(I_j)$, $I_j^{\min} = \min(I_j)$, $I_j^{diff} = I_j^{\max} - I_j^{\min}$.

In the following, we will use an example to explain how context communities work. Suppose that we have the context providers and quality attributes as listed in Table II. The initial value matrix I is:

$$\mathrm{I} = \begin{bmatrix} 1 & 80 & 30 \\ 2 & 70 & 20 \\ 3 & 60 & 10 \\ 2 & 85 & 15 \\ 4 & 90 & 30 \end{bmatrix}$$

The vectors I^{max}, I^{min}, and I^{diff} are as follows: $I^{max} = [4, 90, 30]$, $I^{min} = [1, 60, 10]$, and $I^{diff} = [3, 30, 20]$. The score matrix S is then calculated as follows:

$$\mathrm{S} = \begin{bmatrix} \frac{(4-1)}{3} & \frac{(80-60)}{30} & \frac{(30-30)}{20} \\ \frac{(4-2)}{3} & \frac{(70-60)}{30} & \frac{(30-20)}{20} \\ \frac{(4-3)}{3} & \frac{(60-60)}{30} & \frac{(30-10)}{20} \\ \frac{(4-2)}{3} & \frac{(85-60)}{30} & \frac{(30-15)}{20} \\ \frac{(4-4)}{3} & \frac{(90-60)}{30} & \frac{(30-30)}{20} \end{bmatrix}$$

If we distributed the weights [0.2, 0.3, 0.5] over the quality attributes, we can get the following utility vector:

$$U = \begin{bmatrix} 1 & 2/3 & 0 \\ 2/3 & 1/3 & 1/2 \\ 1/3 & 0 & 1 \\ 2/3 & 5/6 & 3/4 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.3 \\ 0.5 \end{bmatrix} \approx \begin{bmatrix} 0.4 \\ 0.483 \\ 0.567 \\ 0.758 \\ 0.3 \end{bmatrix}$$

From the above vector, we can see that the context provider D is the best one because its score 0.758 is the highest among all the providers. As a result, the weather context community C_w should select D to supply the context information. More details on

Context provider	Precision	Correctness probability	Refresh rate
A	1	80	30
B	2	70	20
C	3	60	10
D	2	85	15
E	4	90	30

Table II.
Values of quality attributes

modeling quality attributes and implementation of the context community component can be found in Liao (2009).

5.2 Performance evaluation

We also conducted several experiments to evaluate the performance of the proposed CCM. In particular, we conducted experiments to study:

- the performance of our optimal context selection algorithm presented in Section 5.1; and
- the performance of the whole context selection process of CCM.

In the experiments, we constructed a weather context community with 1,000 context providers, and the experiments were conducted on a desktop computer running Windows XP with 3.25 GB of memory and a 2.66 GHz Intel(R) Core(TM)2 Quad CPU.

In the first experiment, we evaluated the performance of our proposed context selection algorithm. In this experiment, the initial value matrix I is already established in the context community (i.e. we only evaluate the context selection process, without involvement of other three components such as the context retrieval process, the context monitoring process, and the context evaluation process). We executed the selection process with different number of context providers and counted the time used in the selection. Figure 9 shows that, in general, the trajectory fits well with the theoretical complexity of the algorithm ($O(n \log n)$). The algorithm is quite efficient. For example, it takes only about 5 ms to execute the algorithm with a context community having 1,000 context providers.

In the second experiment, we used the same context community but tested the performance of whole selection process, which also includes the communication time between the context community and the context providers. As shown in Figure 10, the trajectory is almost linear and the execution time is at the scale of seconds. For example, the whole process used about two seconds for 150 context providers, and about four seconds for 350 context providers, and ten seconds for 1,000 context providers. We believe that the main reason that why the trajectory is linear is because the communication time increases linearly with the increase of the number of context providers, while the time used on executing the selection algorithm is not significant, which is at the level of milliseconds.

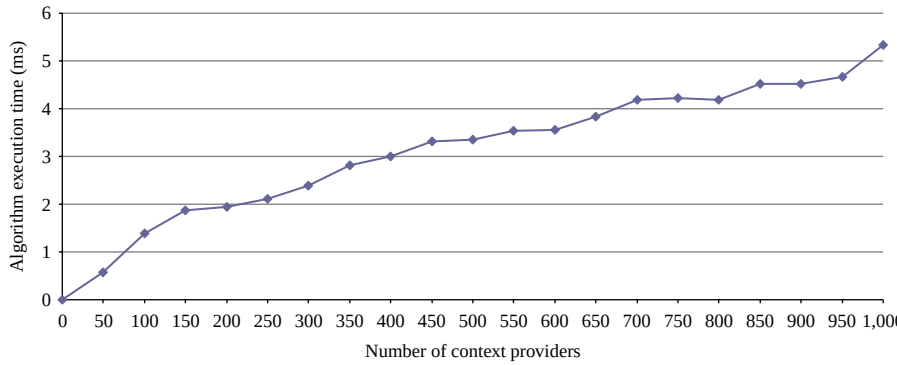

Figure 9.
Performance of the selection algorithm

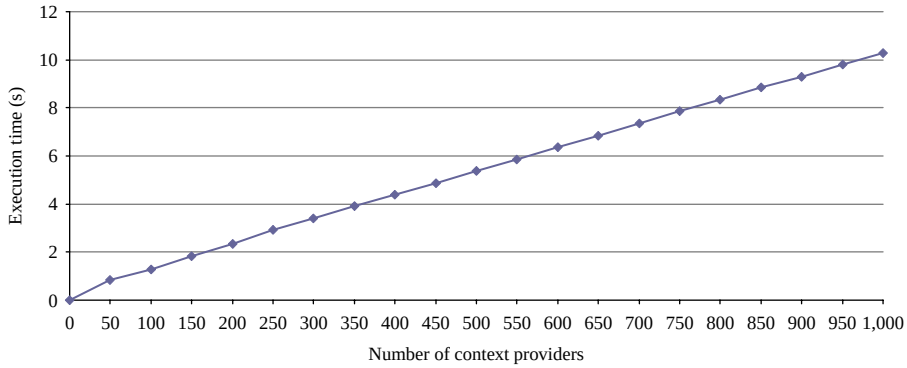

Figure 10.
Performance of the full selection process

From above experiments, we can draw the conclusion that the performance of current implementation of the CCM is acceptable for small to medium-size context communities with hundreds of context providers.

6. Discussions and related work

With the maturing and wide adopting of web service technology, research on providing engineering approaches to facilitate the development of context-aware services has gained significant momentum. Using model-driven paradigm to develop CAS has proven to be a valuable and important strand in this research area considering the quality and efficiency it brings along. Apart from model-driven approaches, in the survey on context-aware service engineering (Kapitsaki *et al.*, 2009), the authors propose other five categories of approach: middleware solutions and dedicated service platforms, use of ontologies, rule-based reasoning, source code level programming/language extensions, and message interception. In general, we agree with their viewpoint that any of the approach has its pros and cons. For example, the source code level approach can give more freedom to developers to do all kinds of context-aware adaptation, but this approach does not separate apart the concerns on CA and suffers from a significant maintenance cost. As to the model-driven approach, apart from its advantages, it requires to keep the consistency between high-level models and low-level executable code at all times, which brings extra complexity. We also agree that some approaches can be used at the same time to bring extra benefits. For example, we are planning to adopt ontologies in the context community to provide enhanced context organization and matching functionality.

In the literature on MDD of context-aware services, the following research work relates to ContextServ in particular. Ayed and Berbers (2006) propose a UML metamodel that supports context-aware adaptation of service design from structural, architectural, and behavioural perspectives. The structural adaptation can extend the service object's structure by adding or deleting its methods and attributes. The architectural adaptation can add and delete service objects of an application according to the context. The behavioural adaptation can adapt the behaviour of the service object by extending its UML sequence diagram with optional context-related sequences. Grassi and Sindico (2007) propose a UML profile considering both MDD and aspect-oriented design paradigms so that the design of the application core can be decoupled from the design of the adaptation logic. In particular, this profile categorize

context into state based which characterizes the current situation of an entity and event based which represents changes in an entity's state. Accordingly, state constraints, which are defined by logical predicates on the value of the attributes of a state-based context, and event constraints, which are defined as patterns of event, are used to specify context-aware adaptation feature of the application.

Table III gives a detailed language capability comparison between ContextUML, the modeling language used in the ContextServ platform, and the above two UML metamodels from the perspectives of context modeling, service modeling, and CA modeling. As we can see from Table III, all languages support the modeling of atomic context, but only ContextUML supports composite context. As to service modeling, only ContextUML directly supports the structure of web services, and the other two languages just use plain UML classes to represent services. Finally, ContextUML does not support behaviour adaptation since it is impossible to change the internal logic of an encapsulated web service. However, Prezerakos *et al.* (2007) extend ContextUML and implement an aspect-oriented transformation technique to transform UML models to AspectJ.

It is worth noting that our ContextServ platform is the only one that provides a comprehensive software toolset that supports graphical modeling and automatic model transformation for CAS development. ContextServ provides a comprehensive platform where CASs are specified in a high-level modeling language and their executable implementations are automatically generated and deployed, thus contributing significantly to both design flexibility and cost savings.

7. Conclusions

In recent years, CASs are emerging as an important technology for building innovative context-aware applications. Unfortunately, CASs are still difficult to build, due to lack of context provisioning management approach and lack of generic approach for formalizing the development process.

In this paper, we have presented techniques for efficient and effective development of CASs. In particular, we introduced ContextUML, an UML-based modeling language, and the ContextServ platform that implements ContextUML, for MDD of CASs. We used an example to showcase how a context-aware web application can be built using the platform. We also introduced the context community, an important concept for dynamic and optimized context provisioning for CASs. Our ongoing work includes developing more applications to validate the system and conducting more experiments to study the system performance. We also plan to extend the context provisioning by supporting semantics of contexts.

Table III.
Language capability comparison

		ContextUML	AyedUML	GrassiUML
Context modeling	Atomic context	+	+	+
	Composite context	+	−	−
	Context quality	+	+	−
	Context collection	−	+	−
Service modeling	Web service	+	−	−
CA modeling	Context binding	+	+	+
	Context triggering	+	+	+
	Behaviour adaptation	−	+	−

Note

1. It is retrieved from a really simple syndication feed, available at: http://rss.weather.com.au/sa/adelaide

References

Ayed, D. and Berbers, Y. (2006), "UML profile for the design of a platform-independent context-aware applications", *Proceedings of the First Workshop on Model Driven Development for Middleware* (*MODDM'06*), *Melbourne, Australia*, pp. 1-5.

Barnett, J., Akolkar, R., Auburn, R.J., Bodell, M., Burnett, D.C., Carter, J., McGlashan, S., Lager, T., Helbing, M., Hosn, R., Raman, T.V., Reifenrath, K. and Rosenthal, A.M. (2010), "State chart XML: state machine notation for control abstraction", available at: www.w3.org/TR/scxml/

Benatallah, B., Dumas, M. and Sheng, Q.Z. (2005), "Facilitating the rapid development and scalable orchestration of composite web services", *Distributed and Parallel Databases, An International Journal*, Vol. 17 No. 1, pp. 5-37.

Buchholz, T., Küpper, A. and Schiffers, M. (2003), "Quality of context: what it is and why we need it", *Proceedings of the 10th Workshop of the Open View University Association* (*OVUA'03*), *Geneva, Switzerland, July*.

Dey, A.K. and Mankoff, J. (2005), "Designing mediation for context-aware applications", *ACM Transactions on Computer-Human Interaction*, Vol. 12 No. 1, pp. 53-80.

Frankel, D.S. (2003), *Model Driven Architecture™: Applying MDA™ to Enterprise Computing*, Wiley, New York, NY.

Grassi, V. and Sindico, A. (2007), "Towards model driven design of service-based context-aware applications", *Proceedings of the International Workshop on Engineering of Software Services for Pervasive Environments: in Conjunction with the 6th ESEC/FSE Joint Meeting, Dubrovnik, Croatia*, pp. 69-74.

Julien, C. and Roman, G.-C. (2006), "EgoSpaces: facilitating rapid development of context-aware mobile applications", *IEEE Transactions on Software Engineering*, Vol. 32 No. 5, pp. 281-98.

Kapitsaki, G., Kateros, D., Prezerakos, G. and Venieris, I. (2009), "Context-aware service engineering: a survey", *Journal of Systems and Software*, Vol. 82 No. 8, pp. 1285-97.

Keidl, M. and Kemper, A. (2004), "Towards context-aware adaptable web services", *Proceedings of the 13th International World Wide Web Conference* (*WWW'04*), *New York, NY, May*.

Liao, K. (2009), "Optimal context provisioning in web service environments", Honours thesis, School of Computer Science, The University of Adelaide, Adelaide.

Liao, K., Sheng, Q.Z., Yu, J. and Wong, H.S. (2009), "Smart Adelaide guide: a context-aware web application", *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services* (*iiWAS 2009*), *Kuala Lumpur, December*.

Mellor, S., Clark, A.N. and Futagami, T. (2003), "Special issue on model-driven development", *IEEE Software*, Vol. 20 No. 5, pp. 14-18.

Niu, W., Shi, Z., Wan, C., Chang, L. and Peng, H. (2008), "DDL-based model for web service composition in context-aware environment", *Proceedings of the IEEE International Conference on Web Services* (*ICWS'08*), *Beijing, September*.

Prezerakos, G.N., Tselikas, N. and Cortese, G. (2007), "Model-driven composition of context-aware web services using ContextUML and aspects", *Proceedings of the IEEE International Conference on Web Services* (*ICWS'07*), pp. 320-9.

Roussos, G., Duri, S.S. and Thompson, C.W. (2009), "RFID meets the internet", *IEEE Internet Computing*, Vol. 13 No. 1, pp. 11-13.

Sheng, Q.Z. and Benatallah, B. (2005), "ContextUML: a UML-based modeling language for model-driven context-aware web service development", *Proceedings of the 4th International Conference on Mobile Business* (*ICMB'05*), *Sydney, July*.

Sheng, Q.Z., Yu, J. and Dustdar, S. (2010a), *Enabling Context-aware Web Services: Methods, Architectures, and Technologies*, CRC Press, Boca Raton, FL.

Sheng, Q.Z., Zeadally, S., Luo, Z., Jung, J.-Y. and Maamar, Z. (2010b), "Ubiquitous RFID: where are we?", *Information Systems Frontiers*, available at: www.springerlink.com/content/b642w1021mwuu247/

Sheng, Q.Z., Nambiar, U., Sheth, A.P., Srivastava, B., Maamar, Z. and Elnaffar, S. (2008), "WS3: international workshop on context-enabled source and service selection, integration and adaptation", *Proceedings of the 17th International World Wide Web Conference* (*WWW'08*), *Beijing, April*.

Sheng, Q.Z., Pohlenz, S., Yu, J., Wong, H.S., Ngu, A.H. and Maamar, Z. (2009), "ContextServ: a platform for rapid and flexible development of context-aware web services", *Proceedings of the 31st International Conference on Software Engineering* (*ICSE'09*), *Vancouver, May*.

Stolze, M. and Ströbel, M. (2001), "Utility-based decision tree optimization: a framework for adaptive interviewing", *Proceedings of the 8th International Conference on User Modeling* (*UM'01*), *Sonthofen, July*.

Welbourne, E., Battle, L., Cole, G., Gould, K., Rector, K., Raymer, S., Balazinska, M. and Borriello, G. (2009), "Building the internet of things using RFID: the RFID ecosystem experience", *IEEE Internet Computing*, Vol. 13 No. 3, pp. 48-55.

Yu, Q., Liu, X., Bouguettaya, A. and Medjahed, B. (2008), "Deploying and managing web services: issues, solutions, and directions", *The VLDB Journal*, Vol. 17 No. 3, pp. 537-72.

Corresponding author
Quan Z. Sheng can be contacted at: qsheng@cs.adelaide.edu.au