UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

ISSN 0103-2569

A Web Service Integrating Ubiquitous Computing Applications

Carlos Roberto E. de Arruda Jr
Renato de Freitas Bulcão Neto
Maria da Graça Campos Pimentel

Nº 208

RELATÓRIOS TÉCNICOS

São Carlos – SP
Ago./2003

A Web Service integrating ubiquitous computing applications

Carlos Roberto E. de Arruda Jr
Renato de Freitas Bulcão Neto
Maria da Graça Campos Pimentel

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
Av. do Trabalhador São-Carlense, 400 - Centro
Caixa Postal 668
CEP 13560-970 São Carlos, SP

{credajun, rbulcao, mgp}@icmc.usp.br

Resumo. *Ubiquitous computing applications exploit context information to adapt their services in accordance with users needs based on information sensed both from the physical and computational environment. We advocate that context-aware applications can take advantage of the benefits provided by Web Services in particular, alleviating differences in heterogeneous ubiquitous environments and apportioning responsibilities between applications and infrastructures. We present the Context Kernel Web Service that illustrates how such a service could be built at the same time that leverages dimensions for context information proposed in the literature. The Context Kernel as a Web Service allows applications not only to store and retrieve, but also to exchange context information through the Web.*

Keywords: Ubiquitous Computing, Context-awareness, Web Services, Context Kernel.

Contents

List of Figures

1. Introduction

Ubiquitous computing is an emergent human-computer interaction paradigm where users interact with everyday devices which seamlessly embed applications [Weiser, 1991]. The main goal is to support users in their everyday tasks usually carried out away from the desktop platform. Pervasive and mobile computing are technologies which have profoundly contributed to the evolution of ubiquitous computing [Lyytinen and Yoo, 2002].

Context-aware computing is a research theme on ubiquitous computing with focus on the ability of a computational entity to adapt its behavior based on context information sensed both from the physical and computational environment. Context is any relevant information about the user-application interaction, including the user and the application themselves [Dey, 2001]. The relevance of the service provided depends on the current task. Thus, context-aware applications handle context information in order to provide applications with context-aware services.

Many context-aware applications, such as *PARCTab* [Schilit et al., 1993], *Active Badge* [Want et al., 1992] and *CampusAware* [Burrell et al., 2002], implement their own mechanisms for capturing, storing and processing context information. Recent efforts have been geared towards providing support for the construction of services dedicated to capture, store and process context information. Examples include the *Context Fabric* [Hong and Landay, 2001] service infrastructure, the *Aura* [Garlan et al., 2002] architectural framework and the *GaiaOS* [Hess et al., 2002] meta-operating system.

Developers have to face two challenges for building context-aware applications: first, the support for several levels of heterogeneity; second, the distribution of responsibilities between applications and infrastructures. An alternative to deal with these two issues is to take advantage of the benefits provided by Web Services.

Recently, Web Services have become a key success factor as infrastructure for several development and integration projects, including gateway servers, APIs and middlewares [Arsanjani et al., 2003]. The essence of Web Services [W3C, 2002] is the use of the Internet infrastructure to bridge a myriad of Internet systems transparently and independently of differences in network technologies, devices, operating systems and programming languages. Web Services are largely based on HTTP as the application-level protocol and open XML-based specifications, such as XSD (*XML Schema Definition Language*) [W3C, 2001], WSDL (*Web Service Description Language*) [W3C, 2003b] and SOAP (*Simple Object Access Protocol*) [W3C, 2003a].

In this paper we discuss how the Web Services technology could be used to allow the storage and retrieval of context information by means of the Context Kernel [Arruda Jr et al., 2003], a Web Service that allows applications to handle context information based on the *who*, *where*, *when*, *what*, *why* and *how* context dimensions discussed in the literature [Abowd et al., 2002] [Truong et al., 2001]. As a Web Service, the Context Kernel allows applications not only to store and retrieve, but also to exchange context information via the Web.

Section 2 presents current research in context-aware computing. Section 3 advocates the use of Web Services as an approach for building context-aware infrastructures. This is illustrated through the Context Kernel service presented in Section 4. Section 5 describes a scenario in which context-aware applications make use of the Context Ker-

nel. Section 6 discusses some limitations of the Context Kernel implementation. Finally, Section 7 presents our concluding remarks and future work.

2. Related Work

Making applications context-aware is one of the challenges pointed out by [Abowd et al., 2002] in ubiquitous computing. Recent research on context-aware reports results on mobile and context-aware systems, sensor-based context information, context modeling, development methods for toolkits, frameworks and service infrastructures. This section presents some of those endeavors on context-awareness focusing on how they deal with context information.

The *Xerox PARCTab* [Schilit et al., 1993] and *Olivetti Research Lab's Active Badge* [Want et al., 1992] are pioneering demonstrations of context-aware systems. Based on indoor location mechanisms, both systems provide users with valuable services, such as locating individuals within a building or remembering whom they met and any phone calls they made or received. In both cases, the systems themselves are responsible for storing context information.

The *CampusAware* [Burrell et al., 2002] is a location-aware campus tour system that allows users to make textual notes on their handhelds about locations they are visiting or find locations they are interested in. The *CampusAware* system uses a central repository with three databases for context and social information, such as users' notes and locations visited. This system relies on the facilities provided by the *Global Positioning System* (GPS) for outdoor location information.

The *Portable Help Desk* (PHD) [Salber et al., 2001] is a location-aware system that allows a mobile student to locate her colleagues and find useful resources, such as printers, restaurants and vending machines. The PHD system relies on a highly accurate location service enabling students to instantaneously locate one another in order to manage their team meeting schedules and projects. This system stores all context information on a relational database shared among multiple services and accessed through user-configurable web pages and customizable real-time maps.

The *Context Toolkit* [Dey et al., 1999] offers a framework with generic services and abstractions — context widgets, interpreters and aggregators — in order to overcome the lack of support for a standard development of sensor-based context-aware applications. The history of each type of context information is stored on a relational database by its corresponding context widget for providing any application interested with that valuable information. For instance, a particular context widget has been used to sense the presence of a user and be able to identify him/her in a whiteboard capture application for informal meetings [Brotherton et al., 1999] by means of a sensor that captures that kind of context information. Thus, *Context Toolkit* provides applications with capture, storage, conversion, aggregation, access and distribution of context information.

The *Context Fabric* [Hong and Landay, 2001] is a service infrastructure for context-aware applications focusing on a storage model for flexible and distributed context information. Similarly to the *Context Toolkit*, its infrastructure is responsible for the storage of context information. However, the *Context Fabric* differs from *Context Toolkit* in two significant ways [Hong, 2002]. First, *Context Fabric* takes the architectural model in the

Context Toolkit and generalizes it to two services, an event service and a query service, handling events and queries as main abstractions. Second, the *Context Fabric* separates the specification of context needs from its processing. This allows applications to take changes in the environment into account when processing context needs, such as when someone moves from one place to another.

The *GaiaOS* [Hess et al., 2002] is a component-based middleware designed to provide support for a *User Virtual Space*, an abstraction that comprises context information, tasks and devices associated to each user. A user virtual space is characterized as a proactive entity in a system software, because it must reconfigure itself while its components — the current location or the device being used — change. The *GaiaOS* aims at addressing some inherent challenges in ubiquitous computing environments, such as context management, binding, mobility and adaptability. The storage service is provided by a context-aware data management system that organizes user virtual spaces in distributed file servers, each file server managing its own data. Given its architecture, other services can be implemented on top of the *GaiaOS* middleware.

The *Aura* project [Garlan et al., 2002] provides an architectural framework for user mobility in ubiquitous computing environments that allows the adaptation to users context and needs. Its main component includes a representation of users tasks as a collection of service descriptions which can be mapped on *Service Suppliers* currently available from the environment. The *Task Manager* is responsible for allocating tasks to available resources; the *Environment Manager* is aware of which service suppliers are currently available from the environment; finally, the *Context Observers* provide information on the physical context to the environment manager. These components are placed at the three layers defined by Aura's architecture [Cheng et al., 2002].

These related works register that the challenges for building context-aware applications include the support for several levels of heterogeneity and the distribution of responsibilities between applications and infrastructures. An alternative to deal with such issues is to take advantage of the benefits provided by Web Services, as discussed next.

3. Context-aware infrastructures as Web Services

The essence of Web Services [Stal, 2002] is the use of the Internet infrastructure to allow applications to communicate seamlessly and independently of heterogeneous hardware and software. Web Services are accessed using standard Internet protocols as HTTP operating on top of TCP and can be defined by self-describing messages referencing information to understand the message. Open XML-based specifications — XSD, WSDL and SOAP — are the building blocks of the basic Web Services architecture [Burner, 2003].

The XML language is used for representing the data inside messages using element and attribute names. The XML Schema (or XSD) specification provides a common collection of data types for describing XML attributes and elements in order to achieve a syntactic level of interoperability across the Internet.

For the exchange of messages, Web Services may use the SOAP protocol, which has two main characteristics: simplicity and extensibility. The SOAP protocol defines the XML-based syntax, the semantics and the order of messages exchanged between peers

(applications or services). SOAP messages include an optional Header element and a mandatory Body element, all wrapped by an Envelop element.

Web Services may take advantage of the WSDL specification to define the service contract, i.e., the collection of messages which a service accepts and produces. Moreover, WSDL describes the mapping between abstract messages to particular objects or methods to be used by applications requesting the services [Fremantle et al., 2002]. The WSDL language defines:

- *types* as optional elements used to build messages;
- the request and response *messages*;
- *portTypes* to map messages to abstract operations;
- *bindings* to map *portTypes* to concrete protocols;
- *ports* to define the real communication addresses of services;
- and *services* as collections of ports.

We advocate that infrastructures aimed at supporting context-aware applications should take advantage of the benefits leveraged by Web Services. In particular, infrastructures would allow applications to exchange data with communication efficiency, loose coupling and asynchrony [McKusick, 2003]. This means that applications built upon Web Services-based infrastructures would not only store and retrieve, but also exchange information through the Web.

In the next section we discuss how the Web Services approach could be used to allow the storage, retrieving and exchange of context information.

4. Context Kernel

The Context Kernel is a Web Service that allows applications to handle context information based on the dimensions *who*, *where*, *when*, *what*, *why* and *how* discussed in the literature [Abowd et al., 2002] [Truong et al., 2001] by formalizing a set of XML-based operations associated to those dimensions. The Context Kernel Web Service classifies those dimensions as follows [Arruda Jr et al., 2003]:

- *who*, *where*, *when* and *how* are *primitive dimensions*; they are handled independently of other dimensions;
- *what* and *why* are *derivative dimensions*; they are obtained by relating other dimensions — primitive or derivative.

The Context Kernel stores *primitive dimensions* as a triple containing *type, value* and an optional *qualifier*. The following example represents an instance of the dimension *when*:

(type=date, value=2003-06-04T16:19:29, qualifier=datetime)

A *derivative dimension* is defined by means of a *rule* that contains at least one *premise* and one *inference*. *Premise* is a *dimension-type-value-optional_qualifier* quadruple as in

(dimension=where, type=latitude, value=33) (dimension=where, type=longitude, value=70).

Inference is a *dimension-value* pair as in

(*dimension=what, value=Santiago*).

Moreover, *rules* are grouped in *schemas* and a *context* contains definitions of *schemas*. The following XML excerpt (Example 1) illustrates the vocabulary defined by Context Kernel.

```
<!--            Example 1            -->
<context>
 <schema>
  <rule>
   <premiseS>
    <premise dimension="where" type="latitude" value="33"/>
    <premise dimension="where" type="longitude" value="70"/>
   </premiseS>
   <inferenceS>
    <inference dimension="what" value="Santiago"/>
   </inferenceS>
  </rule>
 </schema>
</context>
```

4.1. Context Kernel workflow

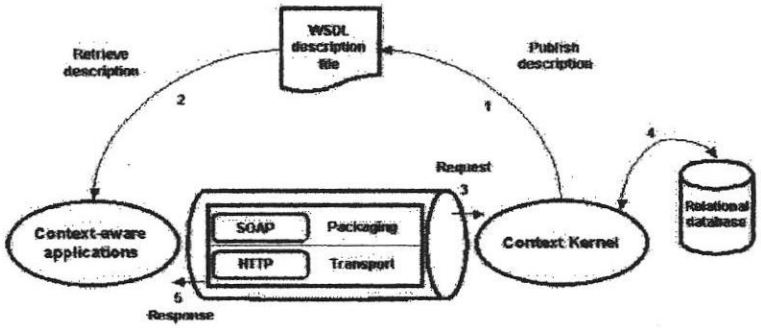

Figure 1: Context Kernel workflow [Arruda Jr et al., 2003].

Figure 1 depicts the Context Kernel workflow and the necessary steps so that a context-aware application makes use of the Context Kernel service as follows.

- In Step 1, the Context Kernel publishes the WSDL document definition that contains the description of the service;
- In Step 2, a context-aware application retrieves the WSDL document. By parsing it, the application obtains the service contract and the network endpoints that honor this contract. In the Context Kernel environment, endpoints are Java servlets used to extend the capabilities of servers in a *request-response* programming model;
- Based on the service contract, applications can send HTTP messages (POST or GET) using the SOAP packaging protocol requesting (Step 3) the available methods provided by Context Kernel;
- In Step 4, the Context Kernel stores/retrieves context information provided by applications on/from a database;
- The result of processing context information is also returned to the requesting application (Step 5) via SOAP messages.

4.2. Context Kernel API

The software platform used in the implementation of the Context Kernel includes: the Linux operating system, the Apache web server extended with the Tomcat servlet container, the Java programming language, open W3C specifications — XML, XSD, WSDL and SOAP — and the MySQL database where the context information is stored.

The Context Kernel API offers four categories of methods: *registry*, *status*, *storage* and *retrieval*. They are invoked by applications using the SOAP protocol encapsulating XML-based messages similarly to the message illustrated in Example 1.

An application invokes the method InfoApp() to *register* its metadata, such as *name*, *description* and *developers*, and receives back one public and one private identifier. The former is used to store context information while the latter is used by third-party applications to access the context information in a read-only basis. Once an application holds the public identifier of a third-party application, it is able to query the Context Kernel API to obtain information stored by that third-party application. The following XML excerpt (Example 2) describes the metadata of the *iClass* system[1] registered with the server by means of a method InfoApp().

```
<!--            Example 2            -->
<info>
 <name>iClass</name>
 <description>Capture & access application</description>
 <url>http://catuaba.icmc.usp.br/eclass/</url>
 <lastRelease>2002-08-07</lastRelease>
 <version>1.0</version>
 <status>academic</status>
 <developers>
  <author email="renan@icmc.usp.br">Renan</author>
  <author email="carlos@icmc.usp.br">Billy</author>
  <author email="aandrade@icmc.usp.br">Andrea</author>
 </developers>
</info>
```

An application can retrieve *status* information stored by any other application. This can be achieved by invoking the method StatusApps() which returns the public identifiers (*<publicID>*) and metadata of all applications registered with the server as shown in Example 3. The *iClass* application invokes the method StatusApps() and, in this case, receives back the metadata of the *CoTeia*[2] and *iClass* applications. The element *<toCK>* stores the particular date and time in which an application has registered with Context Kernel.

```
<!--            Example 3            -->
<applications>
 <application>
  <name>CoTeia</name>
  <description>Tool for web page authoring</description>
  <url>http://coweb.icmc.usp.br/</url>
  <lastRelease>2001-01-06</lastRelease>
  <version>1.0</version>
  <status>academic</status>
  <publicID>0LdWc7sw405zSSkLO8OEiYsuczAcSa</publicID>
  <toCK>2003-01-03T14:36:20</toCK>
```

[1] *iClass* is a capture and access application that allows the access to multimedia material captured from living lectures in the form of web hyperdocuments [Cattelan et al., 2003].

[2] *CoTeia* is a collaborative application for asynchronous remote authoring of web pages [Arruda Jr and Pimentel, 2001].

```
  <developers>
   <author email="credajun@icmc.usp.br">Carlos</author>
  </developers>
 </application>
 <application>
 <name>iClass</name>
 <description>Capture & access application</description>
 <url>http://catuaba.icmc.usp.br/eclass/</url>
 <lastRelease>2002-08-07</lastRelease>
 <version>1.0</version>
 <status>academic</status>
  <publicID>CSagsqFgO7bmP4NBgwd6mUPOhyGI5e</publicID>
  <toCK>2003-01-04T14:36:20</toCK>
 <developers>
  <author email="renan@icmc.usp.br">Renan</author>
  <author email="carlos@icmc.usp.br">Billy</author>
  <author email="aandrade@icmc.usp.br">Andrea</author>
 </developers>
 </application>
</applications>
```

The Context Kernel offers methods which allow applications to *store* and *retrieve* context information relative to contexts containing *schemas*, *rules*, *premises* and *inferences*, as already shown in Example 1. It is worth noting that Context Kernel relies on context-aware applications regarding the validity of the data and rules being stored. Therefore, context-aware applications themselves are responsible for the specification of which kinds of data and rules are particularly relevant to them, i.e., relevance is prerogative of applications.

The method PutData() allows applications to *store* context information relative to *primitive dimensions*; in this case, the element *<context>* includes primitives. In the Example 4, an application requests the method PutData() to store that "a *person* called *Camacho* was at *lab3 room* on *June 4th* at *10:32:04 AM*".

```
<!--            Example 4            -->
<context>
 <primitive>
  <whoS>
   <who type="person" value="camacho"/>
  </whoS>
  <whereS>
   <where type="room" value="lab3"/>
  </whereS>
  <whenS>
   <when type="date" value="2003-06-04T10:32:04" qualifier="datetime"/>
  </whenS>
 </primitive>
</context>
```

The method PutRules() allows the storage of context information relative both to *primitive* and *derivative dimensions*. The Example 5 shows an application requesting the method PutRules() to store the rule "a *person* called *Renata* is at *lab4 room* all *Friday* because she attends a *software engineering class*". Applications may associate an expiration date for their rules; in this case, the rule is valid only for the *first semester year 2003*. When the validation date of a rule expires, it will continue stored on the server, but with solely historical purpose, not being used for querying any more.

```
<!--            Example 5            -->
<context>
 <privateID>BTbhtrGhP8anQ5MChve7nVQPizHJ6f</privateID>
 <schema>
  <rule begin="01/01/2003" end="01/07/2003">
```

```
    <premiseS>
     <premise dimension="who" type="person" value="Renata"/>
     <premise dimension="where" type="room" value="lab4"/>
     <premise dimension="when" type="date" value="Friday AM"/>
    </premiseS>
    <inferenceS>
     <inference dimension="why" value="software engineering class"/>
    </inferenceS>
   </rule>
  </schema>
</context>
```

The Context Kernel service makes available the methods GetRules() and GetAny() for querying context information. The method GetRules() allows applications to retrieve context information based either on the public identifier of an application or the value of *premises* or *inferences* associated to *primitive* and *derivative dimensions*.

The method GetAny() also allows applications to retrieve context information based on the value of *premises* or *inferences* associated to *primitive* and *derivative dimensions*. Moreover, it is possible to specify responses based on the dimensions themselves. Applications may also specify the maximum number of answers or even combine premises using boolean operators as illustrated in Example 6: an application requests "the three most recent (*last value=3*) activities (*dimension=what*) which a *person* called *Claudia* has done in the *kitchen* on *June 04th, 2003*".

```
<!--            Example 6            -->
<context>
 <publicID>URQjzIK7hQ9boR6NDixf8oCUciusHi</publicID>
 <last value="3"/>
 <premiseS>
  <boolean type="AND">
   <premise dimension="who" type="person" value="claudia"/>
   <premise dimension="where" type="room" value="kitchen"/>
   <premise dimension="when" type="date" value="2003-06-04"/>
  </boolean>
 </premiseS>
 <inferenceS>
  <inference dimension="what"/>
 </inferenceS>
</context>
```

4.3. Context Kernel architecture

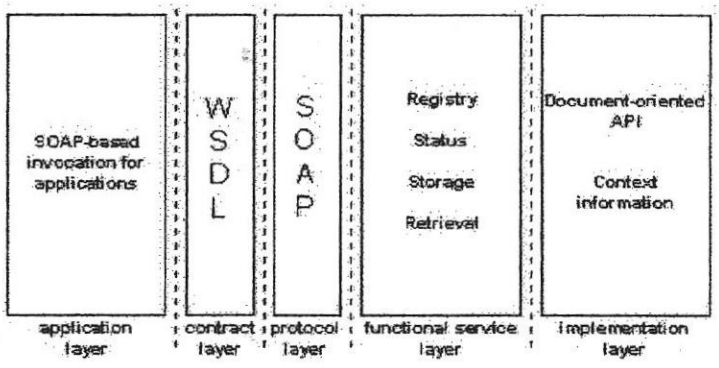

Figure 2: Context Kernel architecture.

The Context Kernel architecture can be thought as five distinct layers as depicted in Figure 2. In the *application* layer resides the set of ubiquitous computing applications

making use of the Context Kernel service by invoking SOAP messages. The next layers refer to the *contract* layer using WSDL for the service description, and the *protocol* layer using SOAP on top of HTTP, respectively. The *functional service* layer describes the service interface, i.e., the main functions provided by the Context Kernel service, such as registering and getting status of applications and storing and retrieving context information.

The *implementation* layer refers to the Context Kernel logic implementation. The Context Kernel Web Service adopts a document-oriented API implementation [Burner, 2003], abstracting the system architectures and creating a loosely-coupled connectedness that withstands changes to the underlying implementation. Document-oriented approaches use *Document* as style of message and *Literal* as serialization format — rather than *RPC* style and *Encoded* format. The following WSDL excerpt (Example 7) describes the *binding* between a *portType* element (*portTypeGetAny*) and the HTTP communication protocol using a *Document/Literal* approach.

```
<!--            Example 7            -->
<binding name="getAnyBinding" type="tns:portTypeGetAny">
 <soap:binding style="document" transport="http://www.w3.org/2002/12/soap/bindings/HTTP"/>
 <operation name="getAny">
  <input>
   <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none"
                                                                    use="literal"/>
  </input>
  <output>
   <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none"
                                                                    use="literal"/>
  </output>
 </operation>
</binding>
```

In the next section we illustrate a scenario in which two context-aware applications make use of the Context Kernel service.

5. Scenario of use

This section illustrates how two applications can take advantage of the benefits leveraged by Context Kernel. First, we briefly present the applications in respect with the context information they handle. We describe a scenario in which those applications make queries based on context information stored on Context Kernel.

5.1. Context-aware applications

CoTeia is an application for collaborative authoring of hyperdocuments [Arruda Jr and Pimentel, 2001] that has been used as a collaborative learning tool roughly for 100 courses during the three last years at our institute (`http://coweb.icmc.usp.br`). It is implemented using PHP as server-side scripting language, XML standards for structuring and presentation of web pages and a MySQL database.

CoTeia could use the following *primitive* context information:

- *user data (who)*: username, password, email, preferences and permissions;
- *location by IP (where)*: remote computer of users and server;
- *timing (when)*: login and logout.

iClass is a capture and access application that has been used to capture classes and lectures since mid-year 2002 at our institute (`http://catuaba.icmc.usp.br/iclass`). The *iClass* infrastructure includes electronic whiteboard, projector, tablet, microphone, networked computer and software modules for capture, synchronization, storage and automatic generation of web hyperdocuments. *iClass* is implemented using Java, XML standards for structuring and presentation of web pages and the Xindice database [Apache XML Project, 2002].

iClass could use the following *primitive* context information:

- *user data (who)*: username, password;
- *type of objects captured (how)*: slides and handwritings (default), audio, audio/video, web logging;
- *location (where)*: classroom's name;
- *timing (when)*: creation date of a session, period (semester and year) of a discipline, duration time of visit of each slide.

Once defined their primitive context information, both *CoTeia* and *iClass* may store them on the Context Kernel infrastructure or infer some derivative information from them.

5.2. Integration with Context Kernel

Once defined their derivative information, applications may store their corresponding rules on the Context Kernel database. Moreover, applications can exchange context information through the Web, for instance, in order to provide users with a customized service.

In our scenario, *Renan attends a lecture all Thursday mornings at lab4 room. Those lectures have been captured by the iClass application and made available on the Web for later perusal.* The requesting Java code in *iClass* so that Context Kernel stores that rule is presented as follows. Notice that *iClass* invokes the method PutRules(). This scenario assumes that all applications involved in this scenario have already been registered with the Context Kernel, thus they hold their public and private identifiers.

```
//            Example 8
public class PutRulesRequest {

public static void main(String args[]) throws Exception {

 // an instance of class PutRules
 PutRules ptRules = new PutRules();

 // primitive context information
 String premises[][] = new String[2][4];
 premises[0][0] = "who";
 premises[0][1] = "person";
 premises[0][2] = "Renan";
 premises[0][3] = "";   //qualifier as optional parameter
 premises[1][0] = "where";
 premises[1][1] = "room";
 premises[1][2] = "lab4";
 premises[1][3] = "";
 premises[2][0] = "when";
 premises[2][1] = "date";
 premises[2][2] = "thursday AM";
 premises[2][3] = "";

 // derivative context information
 String inferences[][] = new String[1][2];
 inferences[0][0] = "what";
```

As a result of the integration with Context Kernel, *CoTeia* and *iClass* — or any application — are able to apply the results obtained from queries to the Context Kernel towards search, customization of contents and navigation, and automation of services. For instance, *CoTeia* may infer activity information, such as the authoring of hyperdocuments from usernames and URIs. The *iClass* application may infer the class to be captured from temporal and location (e.g. room) information of a general schedule.

This scenario also illustrates that non context-aware applications can be extended to make use of context information without having to implement the whole storage and exchange infrastructure.

6. Limitations of Context Kernel

The Context Kernel Web Service is an ongoing project. Therefore, Context Kernel has still some limitations mainly related to its on-demand behavior and query mechanism as follows:

Lack of a preemptive event notification. The communication between the Context Kernel and applications is explicit, i.e., applications must explicitly make calls to the Context Kernel API to take some action. However, it is essential that Context Kernel is able to detect and respond to all relevant events occurring in ubiquitous computing environments, turning it into a Web Service with preemptive features;

No support for elaborate queries. The query mechanism of Context Kernel is based on the boolean operators OR and AND. Other query predicates and delimiters — NOT, ALL, GROUP BY, ORDER BY, ASC and DESC — and the support for nested queries are not implemented yet. For instance, despite providing the filtering mechanism *last value*, presented in Example 6, the Context Kernel API can not request ALL occurrences of some type of context information or GROUP and ORDER queries results, such as "what are the names of ALL students who attended the HCI class this morning?" or "GROUP tasks from my to-do list BY status of emergency and ORDER them BY date ASC";

No support for queries using temporal intervals. Time information has been widely used as indexing mechanism in ubiquitous computing applications. However, the Context Kernel does not handle continuous events, only discrete ones, such as dates and days of the week. It can not be inferred, for instance, whether a student has attended a class or not from his presence in the classroom, since he could have attended for few minutes only. Therefore, comparison operators — AFTER, BEFORE and BETWEEN — combined with *beginTime*, *endTime* and *durationTime* attributes must be provided to tackle this limitation. In the Example 8, the method PutRules() might store the amount of time necessary to infer that *Renan* had really attended the lecture. In addition, if *Renan* left the classroom to drink some water, a location-aware application might keep the times he left and came back to the classroom, add all duration times that he was into the classroom, and store that information on Context Kernel. When *iClass* or *CoTeia* applications made a query on the Context Kernel, they could infer with more accuracy whether *Renan* has attended the lecture or not.

7. Conclusions and future work

Context information has been widely adopted in ubiquitous computing applications so that they may be able to adapt their behavior in conformance to users needs. Related work reports that building context-aware applications demands support for both platform heterogeneity and distribution of responsibilities between applications and infrastructures — the Web Services approach can be exploited in both issues.

We present the Context Kernel Web Service, a work in progress that allows applications to handle context information based on semantic dimensions by formalizing a set of XML-based operations associated to those dimensions. We present a sample scenario in which applications integrated to Context Kernel may not only store and retrieve context information, but also exchange it through the Web towards search, customization of contents and navigation and automation of services. As a Web Service, Context Kernel allows applications to interoperate seamlessly in heterogeneous settings of hardware and software such as the Web. In addition, being responsible for the storage and retrieval of context information, Context Kernel also contributes to keep the integrity of context information.

Although the Web Services approach has some limitations — security, routing, re-configurability and performance issues — new technologies [IBM, 2001], architectures [Kleijnen and Raju, 2003] and methods [Arsanjani et al., 2003] are maturing to achieve acceptable service level characteristics and migrate to a full service-oriented architecture that makes available useful business services. However, provided the access to context information itself is a concern, context-aware applications that would run in restricted environments could also make use of a solution as that adopted by Context Kernel.

At the present time we are working on the limitations described in Section 6, firstly on the querying support. As future work we aim at extending the Context Kernel infrastructure by implementing an enhanced and evolutionary model for storing and retrieving context information through web-based ontologies in order to achieve a greater level of semantic interoperability among context-aware applications. Moreover, some applications have been chosen for integration with the Context Kernel infrastructure, such as *CoTeia* and *iClass*.

References

Abowd, G., Mynatt, E. D., and Rodden, T. (2002). The human experience. *Pervasive Computing*, 1(1):48–57.

Apache XML Project (2002). Apache Xindice. Available online in `http://xml.apache.org/xindice/`.

Arruda Jr, C. R. E., Bulcão Neto, R. F., and Pimentel, M. G. C. (2003). Open context-aware storage as a web service. In *Proceedings of the International Workshop on Middleware for Pervasive and Ad-Hoc Computing held as part of the ACM/IFIP/USENIX International Middleware Conference*. 7p.

Arruda Jr, C. R. E. and Pimentel, M. G. C. (2001). Projeto e implementação de um sistema colaborativo de edição. *Revista Eletrônica de Iniciação Científica da Sociedade Brasileira de Computação (REIC)*, 1. In Portuguese.

Arsanjani, A., Hailpern, B., Martin, J., and Tarr, P. (2003). Web services: Promises and compromises. *ACM Queue (Web Services)*, 1(1):48–58.

Brotherton, J. A., Abowd, G. D., and Truong, K. N. (1999). Supporting capture and access interfaces for informal and opportunistic meetings. Technical Report GIT-GVU-99-06, Georgia Institute of Technology.

Burner, M. (2003). The deliberate revolution. *ACM Queue (Web Services)*, 1(1):28–37.

Burrell, J., Gay, G. K., Kubo, K., and Farina, N. (2002). Context-aware computing: A test case. In *Proceedings of the International Conference on Ubiquitous Computing*, pages 1–15.

Cattelan, R. G., Andrade, A. R., Rocha, C. F. P., and Pimentel, M. G. C. (2003). iClass: um sistema para captura e acesso de sessões em ambiente educacional. *Revista Eletrônica de Iniciação Científica da Sociedade Brasileira de Computação (REIC)*, 3(1). 19p. (In Portuguese).

Cheng, S., Garlan, D., Schmerl, B., Sousa, J. P., Spitznagel, B., Steenkiste, P., and Hu, N. (2002). Software architecture-based adaptation for pervasive systems. In *International Conference on Architecture of Computing Systems (ARCS'02): Trends in Network and Pervasive Computing*, pages 67–82.

Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7.

Dey, A. K., Abowd, G. D., and Salber, D. (1999). A context-based infrastructure for smart environments. In *Proceedings of the International Workshop on Managing Interactions in Smart Environments*, pages 114–128.

Fremantle, P., Weerawarana, S., and Khalaf, R. (2002). Enterprise services. *Communications of the ACM*, 45(10):77–82.

Garlan, D., Siewiorek, D., Smailagic, A., and Steenkiste, P. (2002). Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2):22–31.

Hess, C. K., Román, M., and Campbell, R. H. (2002). Building applications for ubiquitous computing environments. In *Proccedings of the International Conference on Pervasive Computing*, pages 16–29.

Hong, J. I. (2002). The Context Fabric: an infrastructure for context-aware computing. In *Proceedings of the ACM Conference on Computer-Human Interaction*, pages 554–555.

Hong, J. I. and Landay, J. A. (2001). An infrastructure approach to context-aware computing. *Human-Computer Interaction (HCI) Journal*, 16(2-3).

IBM (2001). Web Services Invocation Framework. Available online in `www-106.ibm.com/developerworks/library/ws-wsif.html`.

Kleijnen, S. and Raju, S. (2003). An open web services architecture. *ACM Queue (Web Services)*, 1(1):38–46.

Lyytinen, K. and Yoo, Y. (2002). Introduction. Special Issue: Issues and challenges in ubiquitous computing. *Communications of the ACM*, 45(12):62–65.

McKusick, K. (2003). Interview with Adam Bosworth. *ACM Queue (Web Services)*, 1(1):12–21.

Salber, D., Siewiorek, D. P., and Smailagic, A. (2001). Supporting mobile workgroups on a wireless campus. In *Proceedings of the Third International Workshop on Human Computer Interaction with Mobile Devices*.

Schilit, B. N., Adams, N., Gold, R., Tso, M., and Want, R. (1993). The ParcTab mobile computing system. In *Proceedings of the Workshop on Workstation Operating Systems*, pages 34–39.

Stal, M. (2002). Web Services: beyond component-based computing. *Communications of the ACM*, 45(10):71–76.

Truong, K. N., Abowd, G. D., and Brotherton, J. A. (2001). Who, What, When, Where, How: Design issues of capture & access applications. In *Proceedings of the International Conference on Ubiquitous Computing*, pages 209–224.

W3C (2001). XML Schema, W3C Recommendation. Available online in `http://www.w3.org/TR/xmlschema-0/`.

W3C (2002). Web Services Activity. Available online in `http://www.w3.org/2002/ws`.

W3C (2003a). Simple Object Access Protocol (SOAP) 1.2, W3C Proposed Recommendation. Available online in `http://www.w3.org/TR/soap12-part0/`.

W3C (2003b). Web Services Description Language (WSDL) 1.2, Working Draft. Available online in `http://www.w3.org/TR/wsdl12/`.

Want, R., Hopper, A., Falco, V., and Gibbons, J. (1992). The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, pages 94–104.