ContextXML: A data processing framework for Ubiquitous Computing

D. Wildschut*, T. Riedel*, C. Decker*, D. Roehr†, M. Beigl† and M. Isomura‡
*TecO,University of Karlsruhe
{wildschut,riedel,cdecker}@teco.edu
†DUS,University of Braunschweig
{roehr,beigl}@ibr.cs.tu-bs.de
‡KDDI R&D Laboratories Inc.
isomura@kddilabs.jp

Abstract

In this paper we propose a framework for developing ubiquitous applications allowing the programmer to concentrate on the application logic. Starting from a lightweight layered system architecture we realize a system by implementing its components using existing technology like JXTA, XML Schema, XPath and XSLT and mapping it to ubiquitous technology like networked sensors. By building a sensor mapping application on top of it we demonstrate the usability and extensibility of our approach.

1. Introduction

Due to the dynamic and often distributed nature of ubiquitous systems, application development for them has always been a complex process. Developing ubiquitous applications is all about making information accessible, interpreting this information in context and generally making ”things” communicate. Therefore in addition to the functional aspects, the application designer has to deal with many issues: heterogeneous platforms, communication in ad-hoc networks and application deployment. An underlying framework as introduced in this paper can provide support for such functionality.

The world of ubiquitous computing systems is modeled here as a dynamic network of data flow and processing with sources and sinks in the real world. As the model shown fig. 1 illustrates, an ubiquitous computing application consists of an interconnected set of nodes providing first level contextual information from sources like networked sensors and smart artifacts. Within a loosely coupled network this information is aggregated and logic is performed on it to finally display it based on the context of the user.

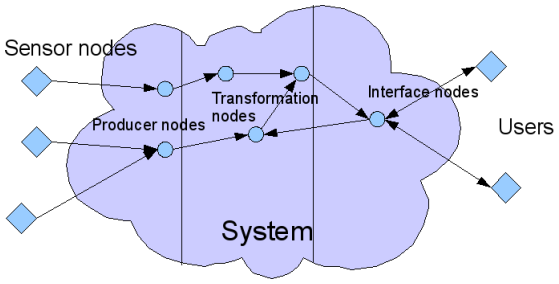

Figure 1. Overview of a typical Ubiquitous Computing System

We believe that a ubiquitous system framework has to provide three key elements to support such applications:

- **Transparency** of networking is needed for heterogeneous artifacts to connect to the system from any place.
- **Standardization** of information representation is needed for unacquainted entities to communicate.
- **Simplicity** of the API is needed for it to be used ubiquitously.

The framework presented here is based on an intuitive subscription model implemented on top of a P2P overlay network. Viewing the world as a hierarchically structured object model the nodes can transparently request information using XPath selectors as subscriptions. By using a transformation-based logic, the nodes process the data and extract new information, which can then be provided to other nodes.

To identify data in the network ContextXML wraps information in so-called XML contexts, XML messages, which contain the data and a header describing its context. This contextual information encompasses message type, contextual attributes like location or time of creation and an extensible set of attributes

978-0-7695-3297-4/08 $25.00 © 2008 IEEE
DOI 10.1109/SAINT.2008.102

describing data. This context is used to interpret the data and most importantly to identify and filter the data a node wants to subscribe.

We use standardized XML technology like XML Schema, XPath and XSLT to process data requests, encode the information and even to implement our transformation based logic. The system integrates with other approaches to semantically structure data by using a subset of RDF-XML and supports the Dublin Core attributes.

Our main focus lies in providing a lightweight portable, modular and extensible middleware for real-life ubiquitous computing applications. In this paper we show how starting from a layered system architecture we realize a system by implementing its components using existing (de facto ubiquitous) technology. Additionally, by applying it to create a sensor mapping application we show the usability and extensibility of our approach.

2. Related Work

With the Sensor Web [2] the OGC has introduced an extensive set of standard building on top of XML and Internet technology. In contrast to our system their framework is designed for static infrastructures, not for ubiquitous ad-hoc networking and rapid application development. Standards like SensorML require a high degree of detailed standardized information about the sensors, which make them difficult to use in the context of smart artifacts.

The graphical programming framework EQUIP [1] is targeted at a similar application spectrum and user group as ContextXML. It uses a tuple space to allow communication between components which are implemented as Java Beans and can be executed on different hosts. A graphical client can then be used to connect to the tuple space and instantiate individual Beans. The running Beans are linked using typed pipes feeding data from one property into another. One problem is that the runtime system and the typing relies to a great extent on Java technology and does not allow heterogeneous settings. Another problem, which is a problem shared by the Sensor Web, is the lack of support for dynamic coupling between context producers and consumers. All entities need to connect to a common data space and be linked manually. Our goal, however, is the dynamic composition of application logic based on current availability of contextual information.

3. Overview of the ContextXML System

This section provides an overview of the ContextXML architecture, starting with the structure of the ContextXML nodes. A ContextXML application consists of a set of nodes with each node containing a program, filters to collect data for the program and a description of the resulting message (this is used for the request evaluation).

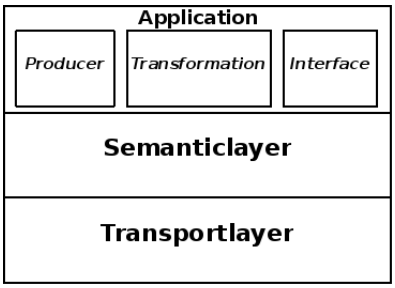

Figure 2. Node layers

Each node shares the basic architecture, which is divided into three layers communicating using XML encoded messages as seen in Fig. 2. The modular architecture and XML based communication makes it possible to replace layers with different implementations.

3.1. Transport layer

The transport layer is responsible for the network communication and handles sending and receiving of XML messages. It has two different ways of sending data, multicast and unicast. Multicast is used to announce a node's subscription request. Unicast messages are used to respond to subscriptions. Our current implementation uses the Java implementation of the P2P protocol JXTA [3].

3.2. Semantic layer

The semantic layer is responsible for routing the information based on contextual properties. It evaluates filters and tries to find a transformation capable of satisfying the request. If such a transformation is found the request is then passed to the application layer. In the current implementation these requests consist of a XPath expression for the produced XML message (See Fig.4 for an example message from our application). A special feature of our system is the lazy execution of the application. Instead of immediately publishing a filter as soon as it arrives from the application layer,

it is stored until a matching request comes for the output of the transformation owning the filter. This also allows us to modify the filter to accurately fit the request, which avoids requesting unnecessary data. These two factors cut down on the necessary traffic and processing required by the application.

3.3. Application layer

The application layer contains the logic for each node. ContextXML models the application seen in 1 by dividing the nodes into three distinct roles based on their function.

3.3.1. Producer. The *Producer* nodes are responsible for introducing external information into the system. An example of a *Producer* node would be a sensor hub collecting and converting sensor data into our XML-based format for further processing. In the ContextXML system a *Producer* node creates messages out of external sensor data, which can be live data or data sets stored in a database. The *Producer* nodes are used to interact with other Ubicomp systems such as sensor networks.

3.3.2. Transformation. *Transformation* nodes take care of all in-network data processing, which includes the classification, aggregation, and transformation of data. Since the filter mechanism handles aggregation ContextXML *Transformation* nodes only perform data transformation, taking a set of data provided by other nodes and extracting new information which is then made available to other nodes.

3.3.3. Interface. The *Interface* nodes provide an interface for external applications. This can be a graphical interface, generated content like a website or even a remote control for a robot. In addition to generating an externally accessible output, *Interface* nodes can also be used as *Producer* nodes and accept input, which is converted to messages and made available to the rest of the application. In contrast to the *Producer* nodes *Interface* nodes are used to interact with classical infrastructures (e.g. business systems via web services) and human users via mapping the interfaces to human computer interfaces (e.g. web sites or door plates).

4. Implementation of a Geocoding Sensor Map

In this section we outline our implementation of ContextXML based on a transportation layer using JXTA 2.5 and a Saxon-B 9.0.0.2 XSLT/XPath 2.0

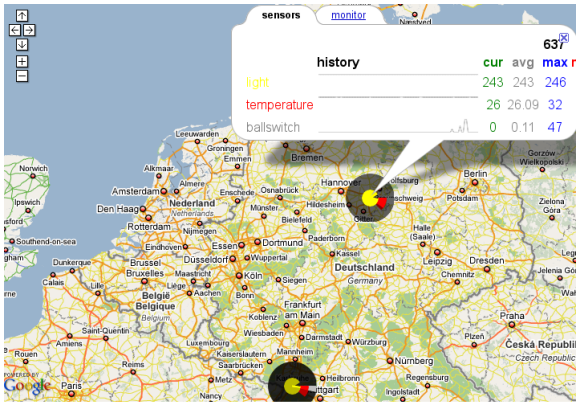

Figure 3. Screenshot of the P2P Sensor Map application

implementation of the semantic layer. Additionally we use the libparticle ConCom frontend to access real-world data and a Jetty based Ajax web server for users to interface information.

We have ported our Sensor Map application presented in [5] to the ContextXML platform. This geocoding application visualizes Particle [4] sensor nodes on map using a Google Maps application and allows the user to create logic using the map interface.

While the previous application collected all data on the same TCP network, making it difficult to visualize sensors in a different physical location, the new implementation allows a potentially unlimited number of locations to feed data into the system. We successfully used the JXTA transport layer to connect nodes from test sites in Brunswick, as seen in Fig. 3, and Tokyo to the application without additional configuration.

The application consists of producer node at each location providing live data from Particle sensor nodes annotated with contextual information and transformation nodes aggregating a sensor history locally. The Sensor Hub in the local network will annotate and format the data and add a GPS location identifier associated with the source.

The processing node collects all Particle contexts from the different locations and transforms them using XSLT (as seen in Fig.5), calculating average, maximum and minimum of the three different sensors for each Particle. XSLT allows us to define a transformation completely in XML, which enables us to easily deploy the programs using the same encoding used for data. Because the transformations are stateless we can shift them around within in the network. Doing aggregation closer to the data source allows us to save bandwidth. We can also easily generate transformations to be deployed from within the pro-

gram context. One of the ideas of geocoding was that the user could define logic using the map interface. Generating and deploying new transformations allows us to generate user specified events like temperature alerts from within the application.

The resulting XML document is then fed into the Google Maps JavaScript application running on Jetty, an AJAX-capable Java web server, which generates a graphical interface in the user's web browser. The web server opens a new session for every connecting user. The JavaScript application determines the viewport displayed in the web browser and generates a corresponding filter.

The semantic routing layer dynamically connects data sources and sinks by publishing and evaluating filter requests. Based on the viewport we define an XPath expression to filter out outside locations. The request is published to the *Transformation* nodes, which then generate a filter for the necessary data. This filter is evaluated by the *Producer* nodes. If the filter expression evaluates to true on the produced message a JXTA message pipe is established. The combination of all subscriptions forms a semantic routing network.

```
<msg:message>
  <msg:header>
    <msg:id>msg:id</msg:id>
    <msg:type>uPartData</msg:type>
    <msg:timestamp>2008-04-23T12:00:00
    </msg:timestamp>
    <msg:location>DE/KA/TeCO/Room1
    </msg:location>
  </msg:header>
  <msg:data>
    <msg:ballswitch>120</msg:ballswitch>
    <msg:light>100</msg:light>
    <msg:temperature>23</msg:temperature>
  </msg:data>
</msg:message>
```

Figure 4. Example message from our application

```
<ambientlight type='uint8_t' unit='lx'>
<xsl:attribute name="avg">
  <xsl:value-of select="avg((light,\
    for $x in \
    tokenize($oldnode/ambientlight/@arr,',')\
     [position() lt 100] return number($x)))"/>
</xsl:attribute>
```

Figure 5. Excerpt from the XSLT transformation used in our application

5. Conclusion & Outlook

In this paper we presented ContextXML, a programming framework for ubiquitous computing. The main feature of ContextXML is the underlying subscription mechanism for data communication, allowing efficient execution of the components of the application. The modular design allows easy replacement of layers to adapt ContextXML to different requirements. In the current implementation we have implemented the layers using only standard XML technology reducing the complexity and allowing high portability. We used it re-implemented our Geocoding Sensor Map application in ContextXML, demonstrating and evaluating the ContextXML programming process.

In future work, we are planning to extend the system, allowing dynamic deployment of code written in other programming languages like Java, Python or Ruby based on node capabilities. We also plan to add a database node to allow us to process past data. We are currently evaluating the framework with applications like our AwareOffice system. Another important concern being worked on is the implementation of security and privacy policies for data flow and code deployment.

References

[1] Greenhalgh, C. (2002). EQUIP: a Software Platform for Distributed Interactive Systems, Equator IRC Technical Report Equator-02-002, Nottingham, 2002.

[2] OGC SWE website, http://www.opengeospatial.org/projects/groups/sensorweb

[3] JXTA Java implementation website, https://jxta.dev.java.net/

[4] Decker, C., Krohn, A., Beigl, M., Zimmer T. "The Particle Computer System" Proceedings of the ACM/IEEE Fourth International Conference on Information Processing in Sensor Networks 2005, Los Angeles, USA

[5] Christian Decker, Till Riedel, Phillip Scholl, Albert Krohn, Michael Beigl, Graphically Geo-Coding of Sensor Systems Information, 4th International Conference on Networked Sensing Systems (INSS), Braunschweig, Germany, June 6-8, 2007, Proceedings of the INSS 2007.