Context Generation and Structuralization for Ambient Networks

Tomasz Szydło
tszydlo@agh.edu.pl

Robert Szymacha
szymacha@agh.edu.pl

Krzysztof Zieliński
kz@ics.agh.edu.pl

Departament of Computer Science
AGH-University of Science and Technoloogy
Al. Mickiewicza 30, 30-059 Kraków, Poland
+49 12 617 39 02

ABSTRACT

This paper presents a model of context information data source structuralization, having in mind requirements of Ambient Networks applications. The proposed solution extends the ContextWare architecture, which is a general framework for context information dissemination. It assumes that the transfer of context information which represents a temporal state of sensors is accompanied by semantic information. This is a key concept in achieving true interoperability between heterogeneous system domains in mobile applications. A system for semi-automatic generation of data sources in the form of Web Services wrapping the sensors is presented. The input data of this system consists of information about sensors and an ontology describing their semantics. A proposed notation, describing the mapping of values from sensors to the ontology used in the Ambient Networks project, was employed at this stage. The outcome of the system consists of a Web Service which exposes data and semantics of context information.

Categories and Subject Descriptors

D.2.12 [**Interoperability**]: Data Mapping.

General Terms

Management, Design, Standardization.

Keywords

component; Ambient Networks; context; context sensors; context management; triggering; context modeling

1. INTRODUCTION

Pervasive computing systems pose new challenges for software developers dealing with software and hardware, as well as their development. Interoperability and adaptability are the most important ones, but new technologies such as metadata and semantic representation are needed to bridge the gap between heterogeneous platforms and applications. A very good survey of the challenges and state of the art in software technologies applicable to pervasive computing environments can be found in [12]. The main challenges of pervasive software include a uniform and adaptive middleware technology as well as interoperability of services and networks.

Pervasive computing enhances the use of computers by making computers effectively available throughout a physical environment and, at the same time, making them invisible to the user. Mark Weiser [15] expressed this goal as achieving the most efficient technology and making computing as ordinary as electricity. Pervasive computing [13] is another term used in the same context but from a different point of view. Pervasive computing emphasizes mobile data access, smart spaces and context awareness. The Ambient Networks project [7] explores a similar area, placing more stress on ad-hoc communication and execution environment mobility.

Modern Web computing and Web service technologies are the driving force behind development of interoperable system applications. Full power of this technology may be achieved with semantic extension [8] of data exchange between system components. This goal was recognized a few years ago by the REST (Representational State Transfer) architecture [4], supported by the Web. The basic concept of the REST architecture is that of a resource. A resource is any piece of information that can be uniquely identified. In REST, requesters and services exchange messages that typically contain both data and metadata. Another important point is that services in REST do not maintain the state of an interaction with the requestor, so if an interaction requires a state, it must be part of the messages exchanged. This simplifies mobility and recovery processes and improves scalability of such services. Web technology bases on simple generic interfaces, which is not true for custom-defined Web Services. This makes SOA [15] far more complex than REST. Nevertheless, the fundamental concept of REST may be used for a SOA-based system. The goal of this paper is to present how the concept of REST may be practically used for structuralization of the context information in AN (Ambient Network) [7] systems. The proposed solution assumes that the transfer of context information which represents the temporal state of the AN execution environment is accompanied by semantic information. This is the only way to achieve true interoperability between system domains in a mobile environment. As a consequence of this approach, a suitable wrapping of sensors by generating data with context sources, adding semantic

information, has been proposed. This concept extends the ContextWare architecture proposed by the AN Project [7].

The structure of the paper is as follows. First, in Section 2, the Ambient Networks ContextWare architecture is briefly summarized. Next, in Section 3, the requirements of a context information structuralization are specified and illustrated by a case study. The proposed context source model is described in Section 4, while its construction process is presented in Section 5. The paper ends with conclusions.

2. AMBIENT NETWORKS CONTEXTWARE ARCHITECTURE

Ambient Networks (AN) are aimed at enabling cooperation of heterogeneous networks on demand, transparently to potential users, and without the need for pre-configuration or offline negotiation between network operators. Context awareness, facilitated by collecting context information, dissemination and management will play an essential role in the operation of AN. The general architecture of the system providing such functionality is ContextWare, proposed by the AN Project [7] and depicted in Figure 1. ContextWare could be considered a middleware model for context processing, dealing with sources of context localization and identification, acquiring context data from various sources, distributing context information to interested consumers, as well as caching and storing context in a database.

AN consist of base components, called Functional Entities (FE). FE is a high-level building block which exposes a Web Service interface for communication with other FEs and applications from outside the AN. For ContextWare architecture, ANs have identified the FEs i.e. Context Coordinator (ConCoord) FE, Context Manager FE, and Triggering FE which will be described in subsequent sections.

In addition to FEs, there are also other entities, called Context Sources and Context Sensors, which provide context information for AN (described in Section 4).

Ambient Networks have identified the use of a common ontology as a requirement for Functional Entities to exchange context information in a consistent and unambiguous way. Composition/decomposition taking place in Ambient Networks requires efficient knowledge transfer and automatic decision-making. This in turn demands proper conceptualization of the context domains, having roles in the process of composition/decomposition. We have decided to use the Resource Description Framework (RDF) [9], a W3C specification for defining metadata models.

Every piece of context information provided in an AN is identified by a Universal Context Identifier (UCI). UCIs are a new type of Uniform Resource Identifiers (URI) [14] and uniquely identify a given context object, but not its location within the network. A UCI is the conceptual rendezvous point between client and sources, i.e. whenever a client wants to obtain specific context information, it has to know the UCI of the object representing this information. Similarly, a context source is assumed to know the UCIs of the context objects it wants to publish.

A fully qualified UCI looks as follows:

ctx://domain.org/path?options

where: *"ctx"* is the new URI scheme; *"domain.org"* is the DNS domain name within which the context object exists; *"path*" is a sequence of words separated by slashes ('/'); *"options"* specifies further modifiers like the data encoding format on the wire.

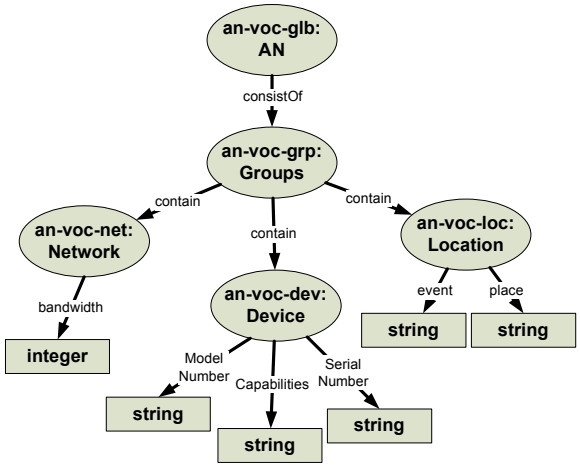

Figure 2. Simple ontology

2.1. Context Coordinator FE

The Context Coordinator FE (ConCoord) is a distributed registry that maps Universal Context Identifiers to the location of context objects. The ConCoord maintains this registry by receiving REGISTER requests from Context Sources (entities providing context) and Context Managers. Hence, Context Sources actively disseminate pointers to their context objects to the ConCoord. The ConCoord FE is the first point of contact for Context Clients. When Context Clients want to access context information, they send RESOLVE requests which contain one or more UCIs to the ConCoord. The ConCoord then responds by returning the locations of the corresponding context objects. Clients contact the located context sources directly to GET the information, or to SUSBSCRIBE to context change events by receiving NOTIFICATIONs, using the Triggering FE. If a Context Source no longer wants to be registered with the ConCoord it removes its entry in the ConCoord by sending a DEREGISTER request.

ConCoord is built on a Distributed Hash Table [5], which provides a robust and efficient distributed registry for storing context information.

2.2. Triggering FE

The Triggering FE has been identified in Ambient Networks to handle notifications between different Functional Entities in an asynchronous way. Because of the properties of the AN architecture, there are many FEs which require this type of communication; using the Triggering FE enables this to be performed in a standardized way.

A trigger producer is an entity which is able to send triggers (asynchronous notifications) using the Triggering FE. In ContextWare, trigger producers are ConCoord FE, Context Manager FE, and Context Source. If a trigger producer wants to

send a trigger, it has to REGISTER in the Triggering FE, and then send triggers using the SEND primitive.

A trigger consumer is an entity, which is able to receive triggers from the Triggering FE. The consumer subscribes to the Triggering FE (using the SUBSCRIBE primitive) for a specific type of triggers. It is also possible to add filtering expressions for the trigger, if the consumer does not want to receive all triggers of a specific type.

If a trigger is sent by a trigger producer, the Triggering FE sends this trigger to the trigger consumer (if trigger filters apply).

In ContextWare, the Triggering FE is used to send notifications about context changes in the network.

2.3. Context Manager FE

As opposed to the ConCoord FE which maps UCIs to the location of context information, the Context Manager (CM) FE manages context within and across domains. It therefore provides the means to:

- allow Context Sources to delegate the distribution of the context information they provide,
- allow recursive aggregation and processing of context information to be done once and on behalf of many clients,
- schedule interactions between Context Sources and Context Clients, monitoring these interactions, re-allocating channels of interaction as needed.

Context Managers, once created, register their output type and capabilities with the ConCoord. The ConCoord's registry therefore maintains mappings to locations of context sources and also of context managers. This enables recursive multi-pipe establishment in a distributed way. The ConCoord locates the final context manager, which locates the managers for its input, which in turn locate the managers for their input, and so on, until the inputs are all initial objects (i.e. basic Context Sources).

Context Managers may also provide additional functionality, such as caching context fetched from source to reduce network load. It should be noted that the depicted tree is rebuilt automatically when a new CM appears in the network or when it disappears (this is the consequence of the highly dynamic structure of Ambient Networks).

2.4. Context Sensors

Context Sensors play a key role in the management of AN, since all context or networking information – whether raw, derived, aggregated, inferred or archived – can ultimately be traced to an original physical or logical source. The programs that interface with these sources of context or networking information and provide the abstractions for integration with MangementWare are called network sensors. Network sensors can be pre-deployed or dynamically deployed on demand in the AN underlay or overlay for a specific use. A number of types of network sensors are under development within the project: P2P context event sensors, Quality of Service sensors, Node and device context sensors and flow context sensors. The information collected through sensors can be used for network and service self-adaptation, triggering network services, implicit QoS signaling, mobility support, in context-based flow classification, in content delivery, in mitigating network attacks, and in controlling malicious or resource-wasting flows such as spam and spit, interplay and optimization of the underlay with the overlay.

3. CONTEXT INFORMATION STRUCTURALIZATION

There are two different approaches to structuralization and processing context information. The first, more traditional approach assumes that Context Sources (CS) have implicit semantic information about the context they provide. A CS only produces data in a CS-specific format, without any description, e.g. as name-value pairs. In this approach, every Context Client which retrieves context from CS has to know what each value means – e.g. that data called "Load" means "% of load for the CPU" and so on. We call this data-centric approach.

The second approach assumes that semantic information is explicitly added to data gathered from CS. This in turn means that CS provides sensor values and additional descriptions, which tell the client how to interpret this data. This novel approach is much more generic than the first one. We call it semantic-data-centric approach.

In case of a CS change (e.g. change of data units), in the data-centric approach the Context Client must be changed as well, because of the fact that all context information is hardcoded inside. In the semantic-data-centric approach, the Context Client may be written in a generic way, and, by using ontologies, be able to discover this change and adapt to the current context provided by the CS. For example, one network sensor may provide throughput in kilobits per second, whereas another one may refer to megabits per second. By using ontologies for context description we can add information on how these units are related; thus the client will be able to understand these two different values, and translate them to its internal units.

Ambient Networks constitute a highly dynamic environment, built of different devices, context sources, and clients. Because of this fact, the Context Client in an AN must be able to understand context information provided by different Context Sources. Context information cannot be hardcoded in the Context Client, so we have decided to use the semantic-data-centric approach.

4. CONTEXT SOURCES

Context information consist of raw data from different type of sensors and the semantic layer which enriches it and give them meaning. In Ambient Networks context data is provided by various sets of Sensors which provide it using Sensors-specific protocols. For example, it may be a plain socket protocol with specific string or binary values defined. Thanks to this fact, sensors may be executed on machines with limited hardware and software capabilities, if needed. Semantic meaning of the values are added by the Context Source layer. ContextWare operates on context information, which is stored in the RDF ontology format. This is a flexible and high-level description format which assures interoperability. Context Source is an entity which is able to communicate with a sensor using that sensor's protocol and expose its interface (i.e. sensor values) in the ContextWare format. Figure 3 depicts how the semantic layer is added to the values acquired from sensors. This cannot be done automatically because, as pointed out previously, sensors typically provide only raw values. User interaction is necessary to map these values from sensors onto their semantic meaning.

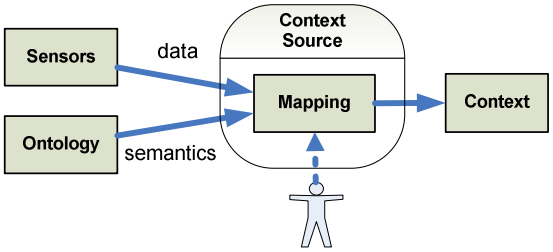

Figure 3. Adding semantic meaning to sensor values

The ContextWare implementation is based on the following assumptions:

a) exposing context interface as a Web Service to assure interoperability

b) adding semantic information using the RDF ontology format allows us to create generic clients, which are not aware of the full context information specification. A client is able to discover a context and learn how to access it;

c) ConCoord registration and optionally generating triggers for context information changes.

Sensors may be executed on limited hardware (e.g. light sensor), which is not even aware of ANs, so it may not be possible to access the full Ambient Networks infrastructure. Context Source is a kind of delegate, which registers a sensor in the ConCoord Functional Entity and the Triggering Functional Entity. Context Source knows its UCI and registers this UCI with the Service Access Point in the ConCoord FE. In addition, CS registers itself as a trigger producer in the Triggering FE. If there is a context client, or Context Manager, which should be asynchronously notified about context changes, it may subscribe in the Triggering FE for a specific kind of Trigger; subsequently the Context Source will send the client notifications about every context change.

The Context Source interface consists of the following primitives:

- GET, which may be called by a client to get context information in the RDF format;
- SUBSCRIBE, which is called by a client, when it needs to be notified by a source about context changes. This primitive returns trigger to the client, which should subscribe in the Triggering FE. Following subscription, the Context Source sends triggers, and clients get information that they should refresh context information (trigger contains information about the Context Source UCI where the context has changed, and after receiving this trigger client should call the GET primitive on the Context Source);
- UNSUBSCRIBE, which is called by a client to unsubscribe from notifications about context changes.

5. CONTEXT SOURCE CONSTRUCTION PROCESS

This section describes the process of constructing context source in the Ambient Networks. Figure 4 depicts the necessary steps for building the *SensorContextSource*. Most of the steps are performed automatically. User interaction is only necessary for choosing sensors to collect information from, write simple wrappers and map them onto the AN ontology. In the most cases, some sensors will be widely used in the Ambient Network so user can get mapping from the repository, and customize parameters when there is need for it.

During bootstrap of an AN, the distributed sensors in the network start collecting the type of context they are responsible for. For instance, a network utilization sensor starts monitoring throughput on the node's local interfaces. As mentioned before, sensors have their own specific protocol for providing collected data. There is the need to map between elements manually.

As stated before, we are using the RDF ontology format which is processed by the Jena framework [6]. The simplest way to map sensor values with RDF is to hardcode them in the source code. The ContextWare architecture prototype in Ambient Networks is implemented using the Web Services technology. AN implementations may change, but mapping between ontology elements and sensors will remain the same.

To avoid problems with further enhancement of prototypes we have developed a notation for mapping values from sensors to the ontology used in the AN project. This allows us to separate the mapping from the source code. Listing 1 shows a simple file which maps the *cpusensor* running on the node to an ontology-derived value (file with *.src* extension in Figure 4). The CONFIGURATION keyword sets the name of the sensor wrapper class. Variables *$(…)* are the fields in that class. To parse the mapping file we use JavaCC [10]. For the prototype we have developed the *src2java* translator which generates for us a static mapping in the source code from the ContextSource configuration file.

```
//What is the UCI of this SensorContextSource
UCI ctx://AmbientNetworks/Devices/an1lt;
//Which of the Sensors you want to use.
CONFIGURATION FreeBSDDeviceSensors;
AN {
  DEVICES {
    DEVICE an1lt {
      cpuUsage     = $(cpusensor);
      serialNumber = "12312312";
      width        = "1024";
      height       = "576";
      videoBitRate = "48000";
      frameRate    = "25.00";
    }
  }
}
```

Listing 1. Context Source configuration file

When these steps are completed, i.e. when we have the *.src* and *SensorWrapper.java* files, the *SensorContextSource* WebService is built automatically and is deployed into the Axis server. In the case of a change of implementation technology, we have to change the part of the process which is responsible for building automation. The file with ontology mapping and sensor wrapper remains unchanged.

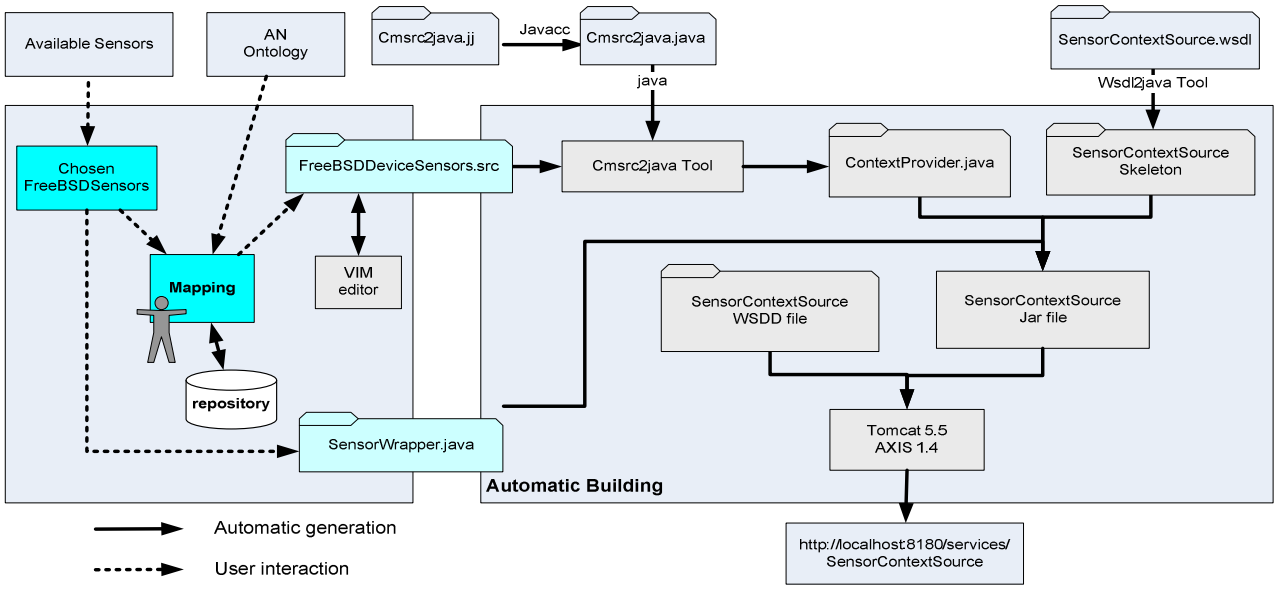

Figure 4. Building process of SensorContextSource

6. SUMMARY AND FUTURE WORK

The software components for the AN system require business process logic that can be configured just in time in accordance with the context information provided by the execution environment. The heterogeneity of context information sources makes approaches which assume inbuilt component knowledge about the context data semantics unfeasible. The proposed solution bases on annotation of the context data with semantic information generated by the context source and is much more flexible. This approach is similar to the REST technology which is a foundation of the Web execution architecture.

The process of context source construction requires the definition of ontology for domains over the AN application which is being executed. The most suitable representation of this information is RDF. Hence, an RDF description of context data must be provided for each context source. This process has to be performed manually as only a system designer is able to decide about data semantics. One this is done, system activity is fully autonomic.

The presented solution enables very late binding of the context information to the business logic thus improving adaptability of applications required by AN systems. The process of wrapping context sources as Web Services may be performed automatically using standard Java tools, with user-defined mapping of sensor-generated values onto ontology items.

7. REFERENCES

[1] Bałos, B., Szydło, T., Szymacha, R., Zieliński, Z. “Context Dissemination and Aggregation for Ambient Networks. Jini Based Prototype” , EuroSSC 2006, 1st European Conference on Smart Sensing and Context

[2] Christopoulou, E., Goumopoulos, C., Kameas, A. “An ontology-based context management and reasoning process for UbiComp applications”. In Proceedings of the 2005 joint conference on Smart objects and ambient intelligence, pages: 265 - 270, Grenoble, France, October 2005

[3] Coutaz, J., Crowley, J., Dobson, S., Garlan, D.: “Context is key”, Communications of the ACM 48(3). March 2005

[4] Fielding, T. R. “Architectural Styles and the Design of Network-based Software Architectures”, UCI Ph.D. Thesis 2000.

[5] http://bamboo-dht.org/, The Bamboo Distributed Hash Table

[6] http://jena.sourceforge.net/ , Jena Semantic Web Framework

[7] http://www.ambient-networks.org, Ambient Networks Web Page

[8] http://www.w3.org/2001/sw/, W3C Sematic Web

[9] http://www.w3.org/RDF/ , Resource Description Framework (RDF)

[10] https://javacc.dev.java.net/ , Java Compiler Compiler - The Java Parser Generator

[11] McGuinness, D. L., Fikes, R., Hendler, J., Stein, L.A. “DAML+OIL: An Ontology Language for the Semantic Web”. In IEEE Intelligent Systems, Vol. 17, No. 5, pages 72-80, September/October 2002.

[12] Niemela, E., Latvaskoski, J. Survey of Requirements and Solutions for Ubiquitous Software ACM International Conference Proceeding Series; Vol. 83, pp 71 – 78, Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia, College Park, Maryland, 2004

[13] Niemela, E., Vaskivuo, T. "Agile Middleware of Pervasive Computing Environments," presented at Middleware Support for Pervasive Computing Workshop, Orlando, Florida, USA, 2004.

[14] Uniform Resource Identifier (URI): Generic Syntax, RFC 3986

[15] Weerawarana, S., Curbera, F., Leymann, F., Story, T., Frguson, T.F. “Web Service Platform Architecture”, Prentice Hall 2005.

[16] Weiser, M. "The computer for the twenty-first century," Scientific American, pp. 94-104, 199