A Context Middleware Using an Ontology-Based Information Model

Iris Hochstatter[1], Michael Duergner[2], and Michael Krause[3]

[1] Munich Network Management Team, Information Systems Laboratory (IIS), University of Federal Armed Forces Munich, 85577 Neubiberg, Germany
Iris.Hochstatter@unibw.de
[2] Ludwig-Maximilians-Universität München
Michael.Duergner@stud.ifi.lmu.de
[3] BearingPoint, München
Michael.Krause@ifi.lmu.de

Abstract. For the adaptation of services to the current situation of a user, the services are in need of specific context information. The acquisition of context in highly dynamic environments is a complex process as the appropriate context sources are not known in advance. Moreover, to realize Mark Weiser's vision of ubiquitous computing, many services on the one hand and a good deal of context information on the other hand have to be combined. Hence, we follow a middleware approach to automate context retrieval for services. For the exchange over domain boundaries, services in need of and services offering context information have to agree on a common description of the information. Therefore, a flexible and extensible information model is a basic requirement. This paper describes in detail the integration of those two important foundations of context-aware computing.

1 Introduction

In highly dynamic environments with a multitude of mobile entities, it is important for (i) context-aware services to find and for (ii) context information services to provide context information to many other systems. A restaurant finder service for example looks for venues close to the user's location and may depend on many other information describing the user's situation. Which context information services, i.e. context sources it has to query will not be known until the actual query is made. On the other hand, context information about any entity, as for example the user's location can be used in many different context-aware services. To relieve context-aware services of the intricacies of context retrieval and composition as well as to facilitate the reuse of context information at the same time, in [1] we proposed infra-structural services, namely the CoCo Infrastructure. Context information is exchanged over domain boundaries, thus all involved actors have to agree on how to express and interpret context information. Therefore, an information model has to balance expressiveness and inferential efficiency. In [2] we introduced a modeling approach based on ontologies that takes into account the special characteristics of context information.

A. Pras and M. van Sinderen (Eds.): EUNICE 2007, LNCS 4606, pp. 17–24, 2007.

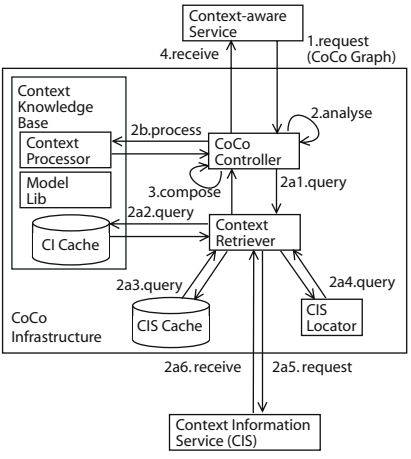

Fig. 1. Middleware: The CoCo infrastructure

In this paper we will go into details about the integration of this ontology-based information model into our context middleware.

The paper is structured as follows: section 2 and 3 presents the basic building blocks CoCo Infrastructure and Context Meta-Model. Section 4 discusses the integration of the ontology-based information model into the context middleware. We conclude the paper with a short summary and an outlook to future work.

2 The CoCo Infrastructure

The *CoCo Infrastructure* (see fig. 1) acts as a broker between context-aware services (CAS) that request context information and context information services (CIS) that provide context information. It therefore relieves the context-aware services from the burden of discovering context information services, data transformation, or derivation of high-level context information from low-level context information.

CoCo stands for *Composing Context*. The CoCo Infrastructure does not only support the request for a single piece of context information (like 'the current position of Alex') but also the request for composed context information (like 'the temperature at the place where Alex currently is'). This requires the ability to describe such compositions. Therefore, we have developed a language that describes the request for composed context information in so-called CoCo graphs [1]. The graph-like structures are expressed in XML.

Basically a CoCo Graph is made out of two types of nodes: *Factory Nodes* describing the requested piece of context information and *Operator Nodes* containing instructions how to process one or several pieces of context information. The procedure is as follows: (step 1) First, the request in form of a CoCo graph,

is sent from the context-aware service to the CoCo infrastructure where it is (step 2) analyzed by the *CoCo Controller*.

For every Factory Node it sends (step 2a1) a request for context information to the *Context Retriever*, which at first queries the *Context Cache* (step 2a2), whether it already has got the requested information. In case it is not available there, the *CIS Cache* is asked whether it already knows an appropriate context information service to retrieve the information from (step 2a3). If not it then instructs the *CIS Locator* to find appropriate context information services (step 2a4), to which the retriever sends related context requests. After having received context information for each request (step 2a5 and step 2a6) the retriever matches this information against the request of the controller, selects the most appropriate piece of context information and returns it to the controller.

For each Operator Node, the controller instructs the *Context Processor* to execute its operation (step 2b), e.g. adaptation, selection or aggregation. Context Processor and Context Cache are part of the *Context Knowledge Base*, which also includes the *Model Lib*. The Model Lib contains the ontology-based information model the middleware is based upon, this will be discussed in the next section.

3 The Context Meta-Model

To facilitate a common understanding and a uniform representation of context, we developed an information model as foundation for context interoperability. This model was introduced in [2]. Context information poses special requirements on an information model as literally every information can be used as context information. It is therefore not sufficient to have a single context model but we propose a Context Meta-Model (CMM) that can be used by application developers to design their own application specific context models, reuse existing models and combine both possibilities. We base our information model on ontologies and thus gain the possibility to perform reasoning and an important formal basis. Description logics are a set of logic-based formalisms used to specify ontologies. They identify a subset of first order logic that offers a good trade-off between expressiveness on the one hand and determinable and efficient inference on the other. To account for flexibility, the CMM incorporates rules that are based on Horn formulae.

Basic building blocks for representing (context) knowledge in our context meta-model are entity classes, datatype classes, and properties with their associated quality classes (see figure 2):

- *Entity class*: base construct for representing a group of entities (persons, places, things, events etc.) that belong together because they share some properties
- *Datatype class*: base construct for representing a datatype (temperature, noise level, position etc.)
- *Property*: base construct for representing a type of relationship between an instance of an entity class and an instance of either an entity or a dataype class. An example for a *property* as a relation between two entities on the

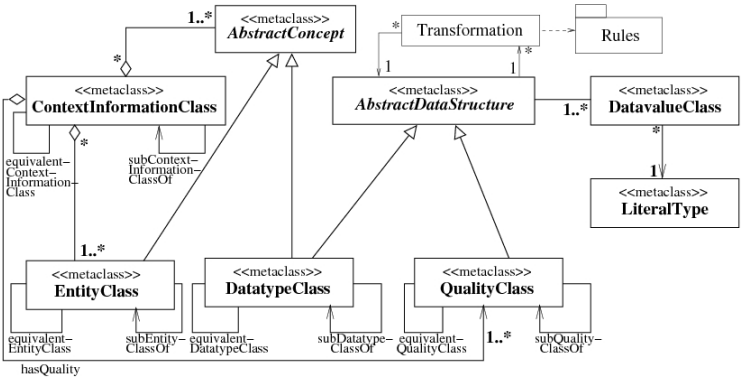

Fig. 2. The Context Meta-Model

model layer is: `Person` "owns" `MobilePhone`. `Person` "hasPhoneNumber" `PhoneNumber` relates an entity with a datatype (more details are given in the next subsection).

– *Quality class*: base construct for representing specific quality aspects of dynamically acquired information (certainty, precision, resolution etc.) also known as *Quality of Context*.

In order to represent *temporal history* information, for every property the acquisition time is captured as a timestamp. It is a mandatory quality class for every property. *Dependencies* between properties are expressed as rules in the form of Horn clauses. Each rule expresses an implication between an antecedent and consequent: whenever the conditions specified in the antecedent hold, the conditions specified in the successor must also hold. This allows to specify consistency conditions as well as derivation rules. Conditions can reference entity classes and datatype classes as well as properties and their associated quality classes. This way a rule can take into account quality information and also specify the quality of the deduced properties.

In addition, there are two special constructs for the semantically rich specification of datatypes: datavalue properties and transformation rules. A *Datavalue Class* is a base construct for specifying data structures, i.e. datatype classes and quality classes. Each datavalue class associates a data structure (*AbstractDataStructure*) with a literal type and thus allows to compose complex data structures from literals. E.g. the coordinates for a position are composed from `longitude` and `latitude`. The *Transformation* is the base construct for representing a transformation from values of one data structure to values of another data structure. An example is the transformation between a position in Gauss-Krueger coordinates into a WGS-84 format, the transformation function itself is given or described in form of a rule on class level in the *Rules* and the identifier for the rule is given in the model itself.

Further modeling constructs are *specialization-relations* that may be specified between two classes of the same kind in order to organize them in (separate)

specialization hierarchies. Finally, there are *equivalence-relations*. Their semantics is that the first node represents the equivalent concept or role as the second node and should therefore be interpreted equivalently. They are useful for mapping context models that have been developed separately in order to enable interoperability.

4 Integrating the Context Meta-Model into the CoCo Infrastructure

As context information services and context-aware services as well as a context composition middleware will be operated by different providers, who may not even know each other directly, the main goal behind the implementation was to enable the use of different context models nearly automatically to facilitate interoperability. This can only be achieved if the middleware does not care about the specific context model it deals with at the moment. CoCo itself operates in most parts just within the structure given by the context meta model (CMM). This allows an easy integration of new context models into CoCo on-the-fly at runtime without even stopping the service. Furthermore it should enable us to delegate the retrieval of context model specific code to either a third-party service or maybe even use in Java integrated mechanism.

The identification of a specific context model and `EntityClass` or `ContextInformationClass` of this model is performed on the XML layer by the namespace and the name of the XML tag, i.e. by its qualified name. The Model Lib of CoCo therefore has an integrated mechanism to translate this qualified name into a Java package name and class name to actually load the correct Plain Old Java Object (POJO). The only prerequisite for this to work is, that the needed classes have to be within the classpath of the CoCo service. This is right now done by copying the JAR packages there but should be replaced by the possibility to automatically load the classes via the Internet directly from the vendor's location.

In this section, we will first give a detailed view on the process that is executed in the CoCo middleware, we then describe the integration of the ontology-based information model CMM into the CoCo Infrastructure and show the achievements of this work.

4.1 Parsing and Binding

The first action that takes place when the service is invoked is that it tries to parse the submitted CoCo Graph and determines if it is syntactically and semantically correct, as far as CoCo is able to understand the semantics. Parsing is done in a two step approach, i.e. in the first step we create a DOM tree out of the XML document and do the syntax checks here. The second one is to parse this DOM tree and translate the elements to the appropriate POJOs, e.g. a factory-node DOM element creates a FactoryNode Java object. Afterwards one of the most important actions takes place, i.e. the different Input-, Factory-,

Operator- and OutputNodes are bound to each other according to the dependencies specified by the user. E.g. the Factory Node which is responsible to retrieve the temperature for the location where Axel is, is bound to the Factory Node which gets Alex's location as this location is a prerequesite for the other one.

As Factory Nodes normally are just bound to one other node OperatorNodes may be bound to a theoretically nearly infinite number of other nodes. Due to the structure of Operator Nodes there are also bindings within the OperatorNode itself, e.g. the output of an Operator Node may be the outcome of a calculation done inside the node or may be a fixed value depending on the outcome of the calculation. The power and possibilities of Operator Nodes are not completely visible at the moment as they depend closely on the context models available.

4.2 Starting and Running the Nodes

After the binding step is complete the CoCo Controller searches for these nodes which are not bound to any other and starts them. Afterwards, the Node tries to fulfill its task, i.e. either invoke the CoCo Retriever to fetch context information in case of a Factory Node or to hand over to the CoCo Processor in case of an Operator Node. In both cases the involved components report either the sucessful execution or any error to the Node which has invoked them. In case of success the result is returned, i.e. the outcome of the operation or the retrieved context, and the Node is then responsible to inform all nodes which are bound to it about the fulfillment of the task. If an error occured during execution, e.g. if there is no way to retrieve the requested context information for whatever reasons, the Node is responsible to inform the CoCo Controller about this problem which afterwards has to deal with this issue.

This mechanism goes on as long as there are any nodes left that need to be executed. In case the node is an OutputNode it either has the value already in case of a fixed value, or it retrieves it from the node it is bound to and informs the CoCo Controller that it is ready to return its value. After all output nodes have reported to the CoCo Controller it is its task to compose the XML document which is returned to the user.

4.3 Integrating Jena with the CoCo Infrastructure

The Jena framework [3] is a Java framework for developing Semantic Web [4] applications. It implements the modeling languages RDF, RDFS and OWL, and provides a rule-based inference engine. The Jena database system [5] uses the JDBC to connect to a relational database like MySQL, Oracle or PostgreSQL. RDF triples are stored with subject, predicate and object and each line corresponds to one RDF statement. Jena allows to manage different RDF models simultaneously by assigning an own triple table to each of the models. Jena also includes a SPARQL engine (SPARQL Protocol And RDF Query Language) [6]. SPARQL is a data-oriented query language that searches the model and returns relevant information as a graph or a set of variables. Its syntax is similar to an SQL statement and supports four different request types. Jena is therefore well

suited for introducing our semantic information model to the CoCo Infrastructure and allows us to store context information persistently in a database while retaining the semantics as it supports OWL DL.

To integrate Jena with the CoCo Infrastructure, we added an interface `ContextInformationCacheJena` to the CoCo Infrastructure that includes the procedures `insert` and `query`. The Jena database subsystem uses the JDBC driver to connect to a PostgreSQL database. After connecting to the database, a persistent model has to be created with the `ModelFactory`. Whenever the Context Retriever receives a request for context information, it first queries the Context Cache via the new interface. The `query` procedure proceeds in five steps: First, from a list of all available models the appropriate has to be chosen and opened. Next, an ontology model is created from the model: the ontology model also contains specification information regarding the ontology language, reasoner and storage location. A SPARQL query searches for the entity and its relevant context. As an entity can have multiple identities, it has to be looked for each of them. When the right entity has been found according to its identity, the sought-after context information can be retrieved in a next step. The result is a model, that has to be converted to a DOM element. Usually the DOM parser should be able to convert the model to a DOM element. Unfortunately, the parser could not resolve namespaces, so in our case the `query` procedure calls a conversion procedure and the model is first converted to a JDOM element and then to a DOM element. The DOM element is finally returned to the Context Retriever.

If the query fails because the context information was not in the Context Cache, the Context Retriever looks for an adequate context information service and queries it. The response is then stored in the Context Cache via the insert procedure. The `insert` procedure has an `EntityClass` object as parameter. This object is first converted to a DOM object and then stored as an RDF model in the database.

5 Related Work

Various approaches to infrastructural support of CASs as well as to context modeling exist. In previous works on those topics, existing approaches to context provisioning (cp. [1]) and context information models (cp. [7] and [8]) have been evaluated thoroughly. In this paper, we present only the most important findings from this extensive research regarding the modeling of context information. In terms of expressiveness, there is no approach that captures all features of context information so far. Most of the existing approaches restrict their generality by stipulating semantic categories and almost none provide constructs to express meta information which is crucial to determine whether the given context information is useful for a particular service. Many of the approaches, in particular earlier ones, lack a formal foundation that is necessary to enable efficient inference, extensibility and distribution of models. In addition, support for interoperability is not explicitly given. Shortcomings in terms of expressiveness and structure result in difficult applicability of an approach in practice.

6 Conclusion

The complex task of brokering context information between all involved actors has been discussed in this paper. Based on the description of a middleware and an ontology-based information model, the concrete integration of both has been described in great detail. Due to the combination of relatively new technologies it has to be dealt with unexpected challenges but the approach shows the feasibility and with proceeding development, the advantages will be even more substantial.

Though this conceptual change in the CoCo Infrastructure works quite well there remains room for improvement. At the moment, context information is stored in the database but never erased or moved. While historical context information surely is useful, databases will get out of hand without a fitting algorithm to clear the database possibly relying on the Quality of Context. Secondly, with the improvement of OWL databases and inference engines a lot more efficient solutions will be possible.

Acknowledgments. The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments. The MNM Team founded by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, the University of Federal Armed Forces Munich and the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences. Its web-server is located at http://www.mnm-team.org. This work has been performed partially in the framework of the EU IST Network of Excellence EMANICS Management of Internet Technologies and Complex Services (IST-NoE-026854).

References

1. Buchholz, T., Krause, M., Linnhoff-Popien, C., Schiffers, M.: CoCo: Dynamic Composition of Context Information. In: Proceedings of the First Annual International Conference on Mobile and Ubiquitous Computing (MobiQuitous) (August 2004)
2. Fuchs, F., Hochstatter, I., Krause, M., Berger, M.: A Meta–Model Approach to Context Information. In: Proceedings of 2nd IEEE PerCom Workshop on Context Modeling and Reasoning (CoMoRea) (at 3rd IEEE International Conference on Pervasive Computing and Communication (PerCom 2005)) (March 2005)
3. Jena Semantic Web Framework, `http://jena.sourceforge.net/`
4. World Wide Web Consortium: Semantic Web, `http://www.w3.org/2001/sw/`
5. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: Aberer, K., Koubarakis, M., Kalogeraki, V. (eds.) Databases, Information Systems, and Peer-to-Peer Computing. LNCS, vol. 2944, pp. 131–150. Springer, Heidelberg (2004)
6. World Wide Web Consortium: SPARQL Query Language for RDF
7. Fuchs, F.: A Modeling Technique for Context Information Master's Thesis, Ludwig Maximilian University Munich (2004)
8. Strang, T., Linnhoff-Popien, C.: A Context Modeling Survey. In: Proceedings of the Workshop on Advanced Context Modeling, Reasoning and Management (2004)