Using Ontologies for Modeling Context-Aware Services Platforms

Diego Rios[1], Patrícia Dockhorn Costa[1], Giancarlo Guizzardi[1], Luis Ferreira Pires[1], José Gonçalves Pereira Filho[1,2], Marten van Sinderen[1]

Abstract. This paper discusses the suitability of using ontologies for modeling context-aware services platforms. It addresses the directions of research we are following in the WASP (Web Architectures for Services Platforms) project. For this purpose a simple scenario is considered.

Keywords. Context-awareness, services platforms, ontologies, Semantic Web, WASP project.

1. Introduction

Context-aware computing is a new computing paradigm that has brought the possibility of exploring the dynamic *context* of the user in order to provide added-value services or to execute more and complex tasks [1]. Building context-aware systems involves facing several design challenges to cope with highly dynamic environments and changing user requirements. Such challenges are mainly related to gathering, modelling, storing, distributing and monitoring context. These challenges justify the need for proper architectural support.

In the recent years, we have seen research efforts towards service platforms that provide architectural and programming support for context-awareness. This provision is usually done by hard-coding semantics into the underlying system implementation and, as a result of this approach, platforms cannot easily evolve and interoperate. Now we are seeing a movement towards the use of *ontologies* [2] and *Semantic Web* [3] concepts to explicitly formalize the properties and structure of contextual information to guarantee common semantic understanding among different architectural components.

The objective of our research is to evaluate the applicability of the Semantic Web technologies for the modeling and the handling of contextual information within the WASP (Web Architectures for Services Platforms) platform. WASP is a project concerned with the definition and validation of a service platform to support the development and deployment of context-aware integrated mobile speech and data applications based on Web Services technology on top of 3G mobile networks. These 3G or next generation mobile networks interacts with the mobile network and with terminals in this network [3].

2. WASP Architecture

Figure 1 depicts the current version of the WASP architecture, as described in [4].

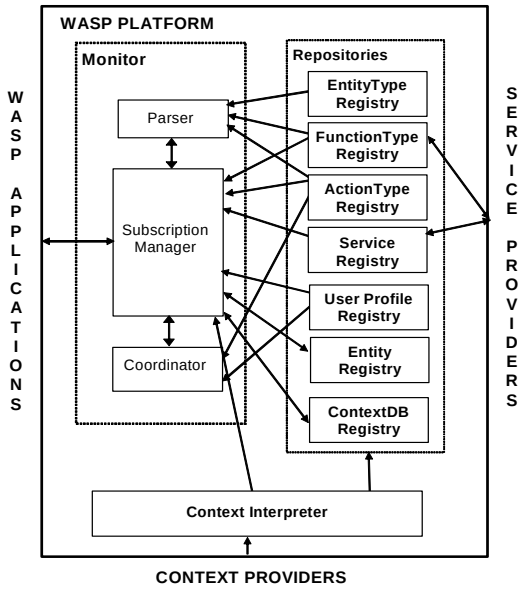

Figure 1. WASP architecture

Interactions between *WASP applications* and the WASP platform are configured during the platform run-time through the addition of *application subscriptions*. Application subscriptions are written in a descriptive language that allows applications to dynamically expose their needs to the platform. With this language it is possible to manipulate the representation of the entities involved in the system (users, museums, restaurants, hospitals, etc.), their attributes, and their context. For instance, it is possible to express that an action involving an entity's context (send an ambulance to John's location) should be taken if an entity (John) enters in a certain context (having a heart attack or a stroke). *Service Providers*

[1] Centre for Telematics and Information Technology, University of Twente. PO Box 217, 7500 AE Enschede, The Netherlands.
{rios, dockhorn, guizzard, pires, filho, sinderen}@cs.utwente.nl.
[2] On leave from Departamento de Informática, Universidade Federal do Espírito Santo, Brazil.
zegonc@inf.ufes.br.

(SP) are business parties that foresee opportunities to profit from offering their services through the platform. *Context Providers (CP)* are the parties responsible for providing contextual information.

The WASP platform architecture is composed by three main components: *i)* the *Context Interpreter (CI)*, which gathers contextual information from the Context Providers, manipulates context and makes it available to the rest of the platform. *ii)* the *Repositories*, which support the *Monitor* with knowledge of the elements involved in the platform (e.g., it contains the entity types, action types, user profiles, etc.). For this purpose, the Repositories collect information from the CI and use services of the SP. *iii)* the *Monitor*, which is the responsible for interacting with the WASP applications and managing their subscriptions, and gathers information from the Repositories and CI.

A distinctive characteristic of the WASP platform is that it enables the dynamic deployment of a large range of context-aware applications that are unanticipated during the design of the platform. Accordingly, we have defined a subscription language - coined *WASP Subscription Language (WSL)* - which applications use to configure the platform to react to a given correlation of events, potentially involving contextual information.

3. The WASP Platform Model

The platform manipulates *data entities*. These entities represent objects of the real world (users, restaurants, museums, roads, vehicles, etc). *Attributes* (age, area, address, etc) and *Context* (location, activity, etc) are associated with *data entities*. In order to effectively and consistently manipulate data entities, attributes and context, we need to organize, represent, and describe them in a model (context model). Once this model is defined, it is used as basis for common understanding of *data entities*, *context* and *attributes* between platform, applications and services providers.

At this moment, the WASP platform model is described by means of UML Class diagrams. The model presents three instantiation levels, a *metamodel*, a *model* and an *object* level. The *metamodel* level is embedded in the platform and it is defined during the platform design-time, being unchangeable during run-time. The lower levels of instantiations are called the model level and the object level. These levels are dynamically changeable during the platform run-rime. They represent instances of the Metamodel and the Model levels, respectively. Ideally, the context model should be extendable to allow the deployment of new kinds of contextual information that have not been anticipated during the platform design. Currently, it does not provide a formal semantic knowledge to allow developing richer functionality in some architectural components. For example, the context information provided by the CP's (sensors or third party context providers) must be processed and made uniformly available to the platform. The existence of a set of ontologies can extend the CI functionality by capturing semantic knowledge and deriving relationships between contextual information, which otherwise could not be directly gathered from the environment. This can greatly increase applications' context-awareness. Likewise, the Service Registry component can use ontologies to enhance the storing, matching and retrieving semantically richer (context-aware) services.

Ontologies are believed to be a key requirement for modeling software system architectures because:

- Ontologies allow architectural components to share knowledge;
- Ontologies allow one to reason about knowledge and check information consistency.

These reasons also apply to context-aware systems, where different entities must share common contextual information representation, and inference and reasoning mechanisms are necessary to allow the derivation of complex contextual information and reason about the context, respectively. Moreover, ontologies not only help to reason about the context, but also help to detect inconsistency in the acquired information since context information can be highly imperfect.

With the emergence of ontology-based reasoning and query engines (especially for Semantic Web ontology languages), like

RuleML [5] and FaCT [6], that exploits different forms of logic (e.g., first order logic, temporal logic, etc.), the platform model can be more expressive and powerful.

We focus on Semantic Web technology, which includes *ontology-based markup languages* for building ontologies, and tools for processing and reasoning over information described by using these ontologies. In particular, we are investigating the suitability of languages such as OWL [8] and DAML+OIL [9].

4. Application Scenario

As an illustration of a possible application scenario we consider a set of CP's (e.g., sensor agents) that communicate with the CI via appropriate domain-specific languages. For instance, a location-sensor device could send a message to the CI with the following content:

```
INSIDE (person15, room52, t1)
```

The semantics of the primitives of this device language are defined in terms of a set of distributed ontologies in which the meanings of Person, Room, Time Interval and the predicate `INSIDE` are specified. These ontologies are represented as a set of logical theories that play the role of the semantic domain for these domain-specific languages. After receiving this message, the CI can interpret its content by accessing the corresponding ontologies. Suppose the following axioms are defined in the semantic domain of this device language:

- The $\subseteq$ (part of) relation is reflexive, asymmetric and transitive;
- Two physical locations X and Y (`room`, `building`, `university`) are disjoint iff there is no physical location `Z` that is both part of `X` and `Y`;
- The predicate `INSIDE(p,l,t)` is defined to hold if a person `p` is at physical location `l` at time `t`;
- A person `p` is inside a physical location `X` in time interval `t` iff `p` is inside a part of `X` at `t`;
- A person `p` cannot be inside two disjoint physical locations at the same time `t`;
- Every Student is a Person;
- A student `s` is present at a university `U` at time interval `t` iff `s` is inside `U` at `t`.

Also suppose that the following information is known to the CI (it was sensed or specified before the current situation):

$$(\texttt{room52} \subseteq \texttt{building5}) \wedge (\texttt{building5} \subseteq \texttt{Utwente}) \wedge (\texttt{building2} \subseteq \texttt{Utwente}) \wedge \texttt{STUDENT(person15)} \wedge \texttt{UNIVERSITY(Utwente)}$$

In this case, the CI could derive, for example, the following information:

$$\texttt{INSIDE(person15,building5,t1)}$$
$$\neg\texttt{INSIDE(person15,building2,t1)}$$
$$\texttt{PRESENT(person15,Utwente,t1)}$$

A similar scenario is being implemented, using the Jena Framework [10] and a generic reasoner, to prove the cited benefits. In the current implementation, the application subscriptions are defined by rules written in terms of a generic rule language. These rules express a certain context and are evaluated by a reasoner, which makes use of the information provided by CPs and the semantic information captured by a set of ontologies defined in OWL to trigger a specific action when the context expressed by the rules becomes true.

5. Conclusions

Many architectural components of context-aware services platforms – and the WASP platform in particular – may profit from Semantic Web technologies. Since semantic knowledge is a cross-cutting concept, it can be exploited by different conceptual layers, from context storage to adaptive interfaces, from service description and discovery to complex service composition. Ontologies play a key role in this scenario, enabling knowledge sharing and providing a model for context reasoning. This paper briefly addresses the suitability of using ontologies in context-aware services platforms.

References

[1] Dey, K., et al. *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*. Human-Computer Interaction Journal 16, 24 (2001), pp. 97-166

[2] Guarino, N., *Understanding, building, and using ontologies*. International Journal of Human–Computer Studies 46:293–310, 1997. [3] *W3C Semantic Web* [http://www.w3c.org/sw]
[4] *WASP project* [http://www.freeband.nl/projecten/wasp/ENindex.html]
[5] Dockhorn Costa, P. *Towards a Services Platform for Context-Aware Applications*. M.Sc Thesis, University of Twente, The Netherlands, 2003.
[6] *Rule Markup Language* (RuleML). [http://www.ruleml.org]
[7] Horrocks, I., et al. *Practical reasoning for expressive description logics*. In Proc. LPAR'99, n. 1705, Lecture Notes in Artificial Intelligence, pp. 161-180. Springer-Verlag, 1999.
[8] *Web Ontology Language (OWL) Guide*, W3C Working Draft 31 March 2003. [http://www.w3.org/TR/owl-guide]
[9] *DAML+OIL*. [http://www.daml.org/2001/03/daml+oil-index.html]
[10] Jena Framework. [http://jena.sourceforge.net]