An Architecture for Semantics-Based Interaction of Spontaneously Connecting Software

Teemu Vaskivuo

VTT Electronics, P.O. Box 1100, 90571 Oulu, Finland
teemu.vaskivuo@vtt.fi

Abstract. Generally, software entities such as components or objects are made to interact with each other according to information that is known at design-time. Spontaneously interacting software elements have to overcome this, as they are not necessarily designed together. This paper presents an approach for connecting software functionality by utilising semantically linked information about the capabilities and features of software entities, instead of utilising their strictly defined software interfaces. The approach is presented through a software architecture for a design that performs suggested kind of functionality.

1 Introduction

Generally, software entities such as components or objects are made to interact with each other according to information that is known at design-time. The design-time information used in connecting software functionality is commonly present in forms of interfaces, class descriptions, or similar information structures. Those structures carry very little information according to which the functionality represented by them could be analysed or categorised. By complementing the presence of software entities with additional, semantically linked information, software interactions can be made understandable on a higher conceptual level. Additional semantic information provides the software entities the means to access the meaning related to the actions that they commit, the events that take place, and the services they provide and use.

In a spontaneous environment, connections and disconnections between software entities take place in an unplanned manner. Such connectivity requires that the software entities are able to extract information about each other's properties. Enabling technologies for spontaneous networking such as Jini, UPnP, and Salutation [3] provide limited capabilities to express the semantic relations between software entities that they connect together. This paper presents a model and an implementation of a software framework that provides spontaneously connecting software entities the ability to utilise knowledge about the semantic connections there are between them and other entities (such as software entities, conceptual entities, or information).

Annotations about the semantic connections between pieces of information should be able to be understood by several individual parties. Such consistency can be achieved by agreeing on certain rules that are used when annotating information. In the computerised world, an ontology is an explicit specification of a conceptualisation [1]. By agreeing upon the used ontology, the information presented about a piece of information can be understood correctly at the specified level of conceptualisation.

J.M. Cueva Lovelle et al. (Eds.): ICWE 2003, LNCS 2722, pp. 403–406, 2003.

Different ways to represent [5] and to use conceptual semantic information and ontologies in order to describe content of information plays an important part in research efforts related to agents [2], and the semantic web [4]. Also the presented approach utilises ontologies in specifying the used conceptualisation.

2 Model and Implementation

Spontaneous networking of software that bases on semantically linked information requires a mechanism that helps in representing the semantic relations between information and functionality. With the help of such mechanism, the software can be made to handle generic occurrences, each of which can be defined on a conceptually high level. The conceptual definitions can then be applied to many practical cases.

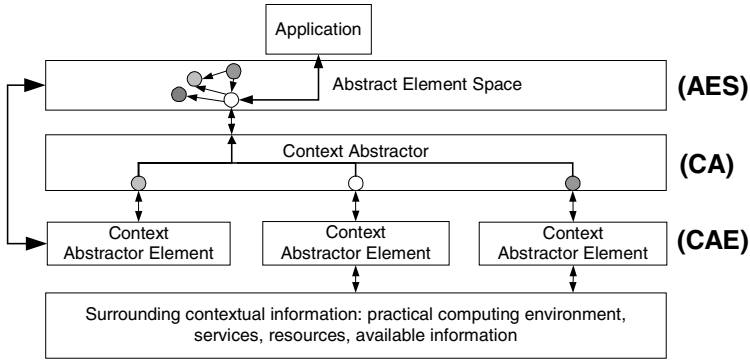

Fig. 1. Conceptual architecture of semantic middleware

Here, "semantic middleware" is proposed to perform the task of creating an abstract representation of the environment. Semantic middleware can be considered as an architectural layer that abstracts the practical functionality of a service environment to a conceptually higher level. Practical tasks of semantic middleware are to:

- Produce a semantically cross-linked *view* of the elements in the surrounding environment (or context) of a software entity.
- Enable efficient queries to explore the content of the produced *view*.
- Provide means to interact with the environment presented in the *view*.

The semantic view can be considered as an automatically updated space that represents the state of the surrounding context with semantically cross-linked elements. Information in the semantic view is then further utilised by applications in various tasks.

Presented semantic middleware comprises a layered architecture. The layers, visible in Figure 1, are *abstract element space, context abstraction, and context sensing*. Abstract Element Space (AES) embodies the semantic view. Applications utilise semantic middleware by interacting with the AES. Context Abstractor (CA) is the layer that produces information to the AES according to ontologies that specify the semantic relations between pieces of information. Context Abstractor Elements (CAEs) sense the context and create an abstracted representation of it. Together several CAEs produce the context-sensing layer. CAEs are add-on components. They can be con-

nected to the CA to sense some specific feature, ability, service, resource, sequence, event, or any other element from the surrounding computing environment.

Some of the CAEs can abstract information that already exists within the AES. Thus it is possible to create a taxonomy where more atomic elements are used in constructing larger conceptual entities within the AES. An example of such operation is visualised by the leftmost CAE in Figure 1. That particular CAE only abstracts information that is already present in the AES.

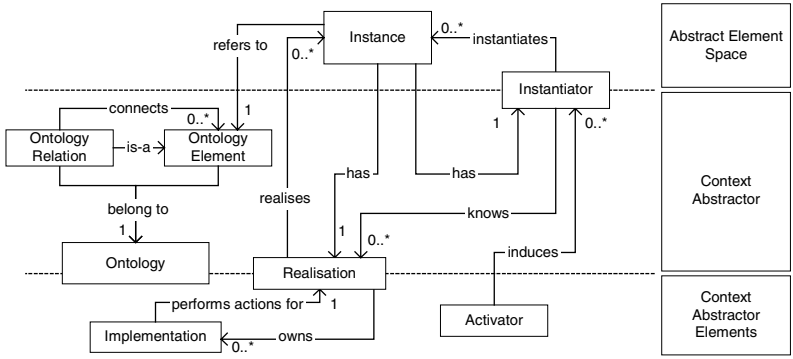

Fig. 2. A diagram representing the most important class relations within the implementation

A demonstration implementation of semantic middleware was created as a part of the presented work. Figure 2 shows the classes that form the core framework of the implemented system. From Figure 2 it can be seen that the CAE layer comprises instances of *Activators* and *Implementations. Activators* function when CAE-initiated active operation is required, for example when a CAE senses something. *Implementations* represent the CAE's closest relationship to resources. Each CAE has to embody the ways to interact with the resources it abstracts.

In the CA layer, *Realisations* represent realisable *Instances* of different conceptual entities that are defined within an *Ontology*. *Realisations* are provided by CAEs and used by *Instances* through the CA layer. The CA layer stores representations of *Ontologies.* An *Ontology* comprises (not shown in Figure 2) a certain *BaseOntology*, and special *CAE Specific Ontologies* that accompany each CAE in order to represent the information provided by each particular CAE. Information within the *Ontology* is described with *OntologyElements* that represent the elements within the ontology, and *OntologyRelations* that represent the relations between the interconnected elements.

The AES layer incorporates the actual storage for instantiated elements. The instantiated elements are represented by instances of the *Instance* class. Each *Instance* is connected to a certain *OntologyElement*. Accordingly, the semantic relations of an *Instance* can be examined through the relations its representative *OntologyElement* has within the *Ontology*. An *Instantiator* represents an entity that is in charge of instantiating *Instances*. *Instantiators* control the instantiations and keep count of the elements instantiated by them. For example, CAEs or applications can act as *Instantiators*. Utilisation of the semantic middleware takes place through the AES layer, which provides its users an efficient set of operations for examining and interacting with the contents of the AES.

The demonstration implementation was created with Java and C++. The CAEs in it were limited to a *Bluetooth Abstractor* element, a *Visual Abstraction* element, and a *Primitive Abstractor* element. By utilising these abstractor elements, a test case of context abstraction was carried out. In the test case, a demonstrative and simple text-based communication was made to establish itself between spontaneously connected devices each time the communication ability was available. The Bluetooth abstractor element provided a source of spontaneous connections and disconnections between devices. The Visual Abstractor Element was made to abstract a part of the Java class library in order to produce a generic tool for displaying information. The Primitive Abstractor Element was made to provide abstractions of certain basic resources, such as primitive data types for time, textual data, numeric data, location, and conceptual input/output operations.

3 Summary and Further Work

Software entities of a spontaneous network have a need for expressing and acquiring semantic information about other software entities and resources. In this paper, a conceptual approach of a semantic middleware has been presented for performing the provision of semantic information. Through the experiences brought by the conceptual design and the implementation, the approach has been experienced as feasible in enhancing connectivity between spontaneously connecting software entities.

Despite of the conceptually high-level information structures within semantic middleware, its use does not provide any intelligence to any application per se. Instead, the ability to access the information about the surrounding context and the semantic relations in it as presented, may provide a better basis for such intelligent information processing. The task of further reasoning and intelligent operation is in the case of semantic middleware always left to the applications.

The applicability of the approach is currently limited by the restricted amount of CAEs. A new CAE has to be produced for acquiring the ability to sense a new element from the context. Efficient and simple ways for integrating legacy resources and applications to operate with the presented concept would provide a change to leverage their already developed potential, making it a strong topic for further research work.

References

1. T. R. Gruber, ”Toward principles for the design of ontologies used for knowledge sharing”, Technical Report KSL 93-04, Stanford University, Stanford, CA, August 1993.
2. Foundation of Intelligent Physical Agents (FIPA), http://www.fipa.org
3. S. Helal, “Standards for Service Discovery and Delivery”, IEEE Pervasive Computing, Vol.1, Iss.2, 2002, pp. 95–100.
4. The Semantic Web Initiative, http://www.w3.org/2001/sw/
5. A. Gomez-Perez, O. Corcho, “Ontology languages for the semantic web”, IEEE Intelligent Systems. Vol.17, Iss.1, 2002, pp. 54– 60.