Modeling Context-Aware Behavior by Interpreted ECA Rules

Wolfgang Beer[1], Volker Christian[1], Alois Ferscha[1], and Lars Mehrmann[2]

[1] Johannes Kepler University Linz, Department for Practical Informatics,
Altenbergerstrasse 69, 4040 Linz Austria
wolfgang.beer@jku.at, {voc, ferscha}@soft.uni-linz.ac.at
http://www.soft.uni-linz.ac.at

[2] Siemens AG, CT SE2, Otto-Hahn-Ring 6, 81730 Munich, Germany
mehrmann@mchp.siemens.de

Abstract. The software architecture of distributed systems is about to change due to new requirements of modern mobile devices. New network techniques, like ad-hoc radio communication or peer-to-peer networks allow mobile devices to sense their environment and to interact with other devices dynamically. This paper presents a cutting-edge way to describe objects and their interaction, also called context, as well as the possibility to configure such interaction scenarios. A lookup mechanism collects information about the environment and a role-based classification is responsible for identifying possible interaction partners. Furthermore the configuration of scenario behavior with context rules is introduced. Finally a comparison with already existing context frameworks is given and a practical emergency scenario configuration is shown.

1 Introduction

More and more radio based wireless networks seem to become the standard for mobile device interaction. For many tasks it is not necessary to be a participant of a global area network, but to connect to an ad hoc network between several mobile devices. Even personal area networks (PANs) get more and more popular, e. g. in an ad hoc communication between a cellular phone, a handheld organizer and a headset. Depending on fixed networks, traditional software architecture for distributed applications contains aspects that harden its use in ad hoc communication environments. Client server architecture depends on a reliable central server that offers services. In ad hoc communication the network structure changes rapidly and normally it is not possible to set up a globally accessible server. Therefore a mobile device has to act as a server and as a client at the same time, to provide services like voice phone, or to use services like calendar function or headset output.

The first question that comes to our mind, when using ad hoc communication, is how to find other entities and how to describe their functionality. Moreover, it is necessary to know how to describe the kind of interaction, which is possible with this new entity

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 1064–1073, 2003.

that appeared. A set of devices is possibly able to solve higher-level problems that the single device is not able to fulfill. Therefore, the single device has to be able to get context information about its environment, i.e. to find compatible partners, in order to solve joint problems. Embedded or personal mobile devices have to exchange sensor and actuator information, to model a digital representation of the real world. With this representation, digital devices should get sensitivity for the context of a human user, e. g. the location or the environment noise and the problems the user would like to solve, e. g. to switch to a louder sound profile. The digital devices could analyze the historical development of this context representation, to propose solutions for future problems. So, embedded wireless devices offer the chance to improve the kind of interaction between human users and their digital helpers [10][1].

2 Description of Digital and Non-digital Objects

In traditional applications the programmer is responsible for connecting client and server programs, to solve a certain kind of problem. The programmer is aware of the interfaces, the network addresses, the functionality and maybe the reliability of the different participants.

In modern peer-to-peer, ad-hoc communication environments the programmer does not automatically get this information. If a device appears in another communication range, it is totally unknown, which device it is and what services it offers, or demands. Therefore it is necessary to get a semantic description of the unknown objects attributes and services. Various research projects deal with the problem of semantic information modeling and distribution, often with the goal to extend the possibilities of smart software agents and reasoning mechanisms.

Originally, the XML based W3C standard RDF (Resource Description Framework) [8] and its schema language were developed to meet the requirements of semantic information retrieval on the web. Various research projects deal also with the topic of joint terminology, like the definition of ontology [11].
Generally, we distinguish between two ways of describing and classifying objects:

- ***Closed world assumption*:** Data and services are completely known at design time. Therefore, it is easy to define a suitable semantic model, in order to handle this information. It is not possible to add any unknown element in a closed world assumption. Moreover, parts of information are explicitly excluded from the context world model, in order to simplify reasoning [13].
- ***Open world assumption*:** Data and services are not known at design time. Therefore, it is necessary to derive all the information from the semantic description of the participating agents. This issue is hard to realize and a major problem in AI research [9].

Another important aspect, when we are speaking about the description of objects, is the classification of objects. There are two different methods of classifying objects:

- ***Static classification*:** The class hierarchy is completely known at design time. Once an object appears, its type either is part of the known class hierarchy, or its type is unknown. The class hierarchy cannot change at runtime.
- ***Role based classification*:** An unknown object is classified by a library of attribute information. The object's set of attributes is compared with attribute sets that are known. This library of attributes and known attribute sets (we call them attribute templates) can be extended at runtime. Naturally an object is able to fulfill more than one attribute template, so it acts in more than one role. Objects can extend their functionality at runtime, i.e. their attribute template is able to change at dynamically.

To support role-based classification with an open world assumption, our framework is based on the work of XEROX Parc in 1995 [9]. Every object, called entity, is a container for a set of attributes. This set of attributes fully describes the entity's functionality. A specific set of attributes is called an attribute template. It is responsible for classifying an entity at runtime. An entity is likely to act in more than one role. As it is shown in Figure 1, an entity, which owns the attributes {Name, Birthday, SocNumber} would be classified as a person in the European union and in the United States.

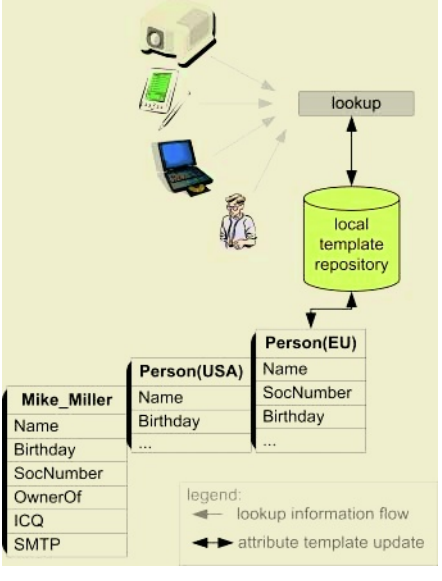

Fig. 1. Entity classification with attribute templates

Without the social security number, the entity would be classified as a person in Europe but not in the United States. So the role based classification of entities is based on the set of attribute templates that are registered locally on the mobile device. The classification of objects is not a task of the context middleware but of the applications that run on it. Imagine a chat application that lists all persons in the local environment. Therefore, the chat application itself has to provide an attribute template that defines how the chat application specifies a person object.

3 Framework Architecture Overview

As already mentioned in section 2, an entity is classified by the set of attributes it provides at runtime. The set of attributes possibly changes at runtime, when an entity loads or unloads attributes over the network or from a local storage. Therefore an entity has to reference a class loader and a transport layer, to load and to deploy new attributes. The class loader itself retrieves class information, either from a local attribute repository or through the transport layer. Because of the recursive nature of the attribute-entity architecture, it is possible for an entity to react as an attribute within another entity, as it is shown in Figure 2.b.).

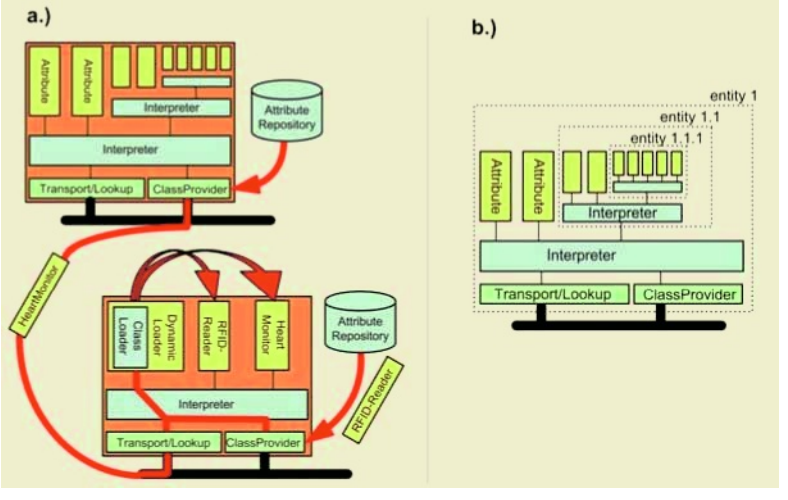

Fig. 2. *a.)* Shows dynamic attribute loading at run time, while *b.)* shows a simple recursive attribute-entity architecture

Between every context information flow, including the loading or unloading of attributes, the interpreter checks whether any additional actions should be performed. These additional actions are specified in context rules. *ECA* (**E**vent **C**ondition **A**ction) rules are able to react on certain entity states, where the event specifies the context event an attribute throws [3]. Context rules are described more detailed in Section 4. Figure 2.a.) shows how an entity is able to load an attribute, called Heart Monitor, and how the entity deploys it to another entity over a transport layer. The entities contain a set of attributes, which may be updated at runtime. In this upgrading process special attributes are loaded, which are necessary to fulfill a certain context scenario, see Section 6. The transport layer does not depend on any specific protocol, it can be changed at runtime. At the moment a TCP socket implementation is available as well as a SOAP-HTTP implementation. We plan to implement an additional JXTA transport layer, as far as JXTA provides reasonable performance on mobile devices.

3.1 XML Configuration of Entities

Entities exist in containers, which manage their lookup mechanism, transport and lifecycle. A container has the possibility to host a performance-optimal collection of entities. A desktop computer would probably host a larger set of entities, while a mobile device is limited to a small set. Containers can exchange entities at runtime in order to optimize performance or to minimize network load. To minimize network load, it is possible to host a personal entity on a desktop computer when the user is in the office. When the user leaves his office, his entity description travels with him, hosted on his PDA. At the container startup time, a set of entities is loaded by the *Dynamic Loader* attribute of the container. XML configuration files inform the container, which initial set of entities should be loaded and which initial sets of attributes the entities own. The attribute-specific configuration is also located in separate XML configuration files, which were referenced by the entity configurations. The following figure shows how a simple container configuration is modeled in XML on a mobile device.

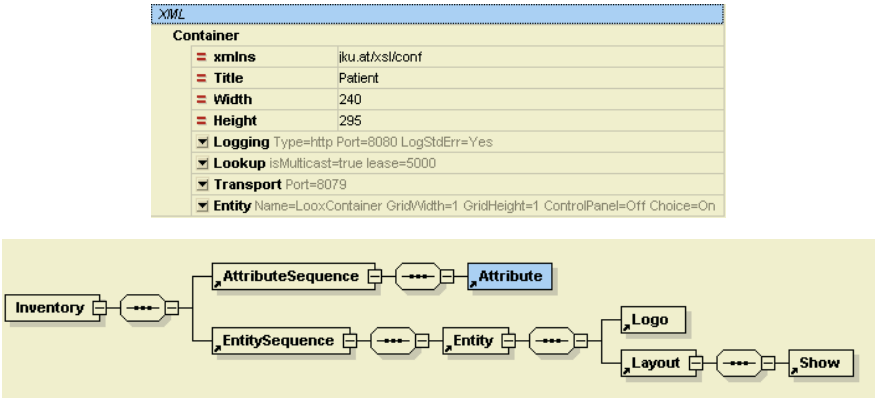

Fig. 3. XML configuration of containers and entities

3.2 Lookup

Communication in ad-hoc mode requires a mechanism that discovers entities, which are inside the communication range of an entity. In order to find a service provider in an ad-hoc network it is necessary to announce the presence of mobile devices in the environment. For a small number of service providers, it is possible to use Broadcast or Multicast IP solutions to announce their presence in the network. Broadcasting of service provider access information is not scalable and therefore not directly used for larger networks. Discovery protocols like SLP (Service Location Protocol) and the discovery mechanism of JINI (Java Intelligent Network Infrastructure) introduce hybrid solutions, where Multicast IP is used for searching in local networks and fixed addresses are used for global service discovery [12].

The lookup mechanism implemented in context framework also uses a hybrid solution where fixed service provider addresses are used for global service discovery. Alternatively, it is possible to use Broadcast or Multicast IP service provider announcement for local area discovery. A context container is responsible for announcing and receiving the presence of entities and their services. Additionally, the context lookup informs about the set of attributes of an entity, which enables the role-based classification. A difference to JINI is, that our solution runs on Java Personal edition version 1.1.8 and is lightweight enough to run on mobile devices. Another difference is, that our lookup solution sends XML based lookup information. The use of XML is a performance drawback, but one of the major disadvantages of JINI is, that only Java based service providers are supported, through the use of Java Object Serialization.

4 Dynamic Interaction of Entities through Context Rules

In any context scenario, it is important that the entities can update or change their relations at runtime. As it is described in Section 2, an entity is able to dynamically classify unknown objects that appear in its environment with attribute templates. In this connection we defined a context rule:

"A context rule specifies the reaction of an entity on a specified context state and defines therefore the event based interaction between entities"

An entity is able to sense its environment with sensors and to change the information in the scenarios world model, as shown in Figure 3.a.). The sensing mechanism could be either event-triggered or time-triggered, as it was already observed in [2]. The framework has to transform the lower level sensor input into information that fits in the framework's context world model.
In the context framework the set of entity attributes contain the high-level context information about an object. So the values of these attributes provide an entity state. A context event is triggered by an attribute, if its state changes and it would like to inform its entity. To react on a specific entity state, it is necessary to define a context rule inside the entity. Context rules define an ECA (Event-Condition-Action) matching service [3], as it is already known from active database research [7].

4.1 Context Rule Syntax

We defined a configuration syntax, which is easy to understand for humans and more compact than XML-coded ECA definitions:

```
Rules = "rules" [ Targets ] "{" { Rule } "}".
Targets = "for" EntityOrTemplate { "," EntityOrTemplate }.
Rule = "on" Event [ "if" Condition ] Action
```

The non terminal symbol EntityOrTemplate describes an entity or a template which defines a set of similar entities as target for the rule. Thus it is possible to bind the rule to a specific entity or to a specific role, specified with a template name. Because of

very limited space, the non terminal symbols Event, Condition and Action are not shown in EBNF here. The following example shows a set of rules, that is bound to all entities, which act in the role of a ***Person***:

```
rules for <Person> { ... }
```

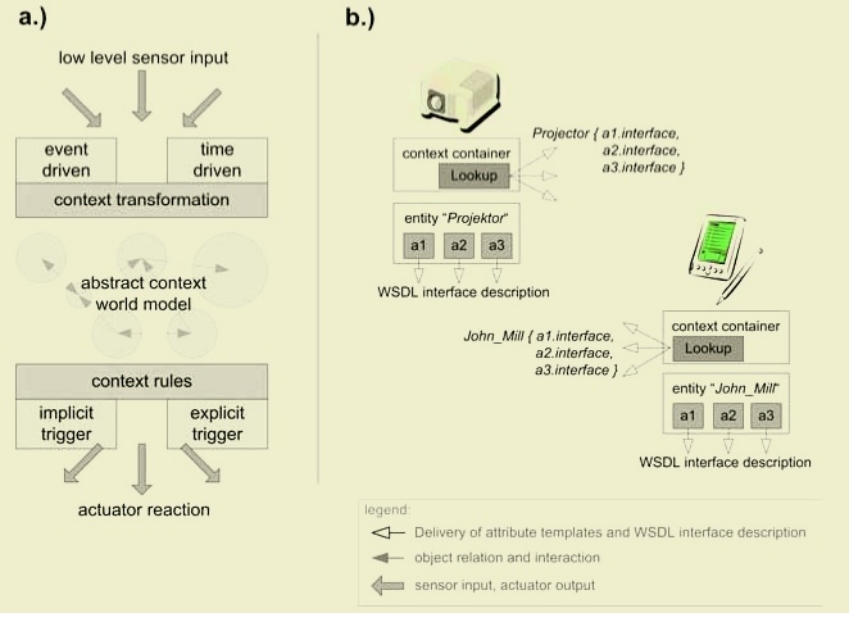

Fig. 4. *a.)* Shows the context information transformation and event triggering, while *b.)* shows the lookup and interface WSDL description delivery.

An entity owns a rule interpreter, which is responsible for catching specific events and reacting on them. Also the attribute loading events or the lookup events can be caught by the rule interpreter. The context rule interpreter is able to load an initial set of rules from a file, referenced by the entity's XML configuration. After the container startup, the entity is able to receive new context rules over the transport layer. These new context rules are registered with the interpreter. Considering the easy plain-text syntax of our context rules, it is also possible to register new context rules from mobile devices like cellular phones or PDAs. Distributed rule deployment is a convenient feature to develop a context scenario on a laptop computer and deploy it to different mobile devices.

Attribute Interface Description with WSDL

Distributed scenario development and deployment is also supported by the interface description of attributes with WSDL (Web Service Definition Language). As the lookup service collects information, which set of attributes an unknown entity provides, the entity is also able to deliver WSDL information for its attributes, as it is

shown in Figure 3.b.). With these types of information it is easy to generate context rules automatically or to use a visual builder tool to generate context behavior.

Rule Consistency
One major problem of the context rule management and deployment is to maintain the consistence of rules. This problem statement is already solved by expert systems, or generally by artificial intelligence research. In fact expert system shells like CLIPS or Jess [5] offer the possibility to reason about entities and their relations. For this purpose it is planned to integrate the Jess library into the context framework, in order to manage entity relationship reasoning. At the moment the interpreter itself triggers actions according to incoming events, but it is not able to check the consistence of new rules.

Advantages of Interpreted Context ECA Rules
One of the most important advantages of controlling complex interaction scenarios with interpreted ECA rules is the high degree of flexibility to change the interaction topology at runtime. Therefore, it is possible to change the interaction partners and the interaction itself at runtime. With the use of attribute templates the rules for handling specific events can be routed to a group of entities that fulfill a specific role. The current implementation of our ECA rule syntax enables the dynamic delivery and lifecycle management of rules at runtime. The rules are coded with the use of a proprietary rule syntax, which enables short and concise creation of new rules, also on mobile devices. Another possible module would be a visual builder tool, to support the creation of context rules, or even the creation of whole interaction scenarios without programming. These aspects, of context modeling through ECA rules, allow nontechnical people to control smart environments and devices without knowledge of a complex system, which a smart environment definitely is. As an example for an abstract view on complex scenario modeling, the Washington's Department of Bioengineering developed a tool called Labscape. With Labscape it is possible to model complex scenarios in a laboratory [6].

5 Example Context Scenario

To demonstrate the abilities of the context middleware, a complex emergency scenario simulation was built (aware of the fact that real-time applications normally are not implemented in Java). For an *compact, out of the box demonstrator* we simulated indoor location sensors with RFID transponders and active readers (in a real world outdoor scenario we would use differential GPS and GPRS as transmission protocol). The scenario starts with an automatic emergency call, triggered by a heart patients PDA-HeartMonitor device:

```
rules for <HeartPatients> {
    on HeartMonitor.Alarm() {
        <EmergencyDispatcher>.EmergencyCall:Alarm($Patient);
    }
}
```

The lookup identifies all entities, acting in the role of an *EmergencyDispatcher*, to which the emergency call is delivered to. When the emergency dispatcher receives an emergency call a rule catches this call and delivers some reaction possibilities to the human emergency dispatcher:

```
rules for <EmergencyDispatcher> {
    on EmergencyCall.Alarm(Patient p) {
        Map.ShowPatient(p);
        Map.ShowNearestAmbulances(p.location);
        Map.ShowNearestHospital(p.location);
    }
}
```

When the human emergency dispatcher reacts on the emergency call, the chosen ambulance car is informed:

```
rules for <EmergencyDispatcher> {
 on EmergencyCall.SendAmbulance(Ambulance a, Patient p, Hospital h) {
   a.Alarm:SendTo(p, h);
 }
}
```

The three context rules, that are shown above, are a small and primitive subset of the mass of context rules that form the complete emergency scenario. They should just explain the purpose of such an abstraction layer.

Fig. 5. RFID reader with PDA device and a model emergency car with sensor equipment.

5.1 Emergency Scenario Hardware Setup

The hardware setup is based on three Siemens Loox devices for hosting the patient, the mobile doctors and the ambulance car entities. In order to receive sensor information about the proximity location of entities RFID (Radio Frequency Identification)

readers were used. For network ad-hoc communication purposes, wireless IEEE802.11b compact flash cards are used. The opened device is shown in Figure 4. To get proximity location information, the presentation floor material is tagged with RFID transponders. The devices are now able to find themselves with the lookup mechanism and to communicate over the context containers transport layer. The emergency scenario is modeled with ECA context rules, in order to define the behavior and the interaction between the emergency entities.

6 Conclusion

The goal of this work was, to show how complex ad-hoc scenarios could be modeled by defining ECA rules for each entity, which is relevant for the scenario. A very flexible and stable context middleware software framework was implemented and tested within an example scenario. Many different simulated entities (emergency cars 1-10, mobile doctors 1-5, 1-10 ambulance drivers and nurses) were hosted in mobile context containers, to show the flexibility of our context middleware. The scenario is not bound to any fixed behavior, but could be changed at runtime through distributed ECA rule deployment. Major future issues will be the visual composition of ECA context rules and a lightweight security model to restrict access to the context model.

References

1. A. K. DEY, G. D. ABOWD: Toward a Better Understanding of Context and Context-Awareness. GIT, GVU Technical Report GIT-GVU-99-22, June 1999.
2. A. FERSCHA, S. VOGL, W. BEER: Ubiquitous context sensing in wireless environments, 4th DAPSYS, Kluwer Academic Publishers 2002
3. Diego López de Ipiña: An ECA Rule-Matching Service for Simpler Development of Reactive Applications, Published as a supplement to the Proceedings of Middleware 2001 at IEEE Distributed Systems Online, Vol. 2, No. 7, November 2001
4. J. PASCOE, N. Ryan, D. Morse: Issues in developing context-aware computing. Handheld and Ubiquitous Computing, Nr. 1707 in Lecture Notes in Computer Science, pages 208–221, Heidelberg, Germany, September 1999. Springer-Verlag
5. JESS (Java Expert System Shell), http://herzberg.ca.sandia.gov/jess/
6. L. ARNSTEIN et al., „Labscape: A Smart Environment for the Cell Biology Laboratory", IEEE Pervasive Computing, July-September 2002, pp. 13–21.
7. PATON N.W. and Diaz O.: Active Databases Survey, ACM Computing Surveys, Vol. 31 No. 1, pp. 63–103, March 1999
8. RDF (Resource Description Framework), http://www.w3.org/RDF/
9. R. WANT, W. SCHILIT, et. al.: The PARCTAB Ubiquitous Computing Experiment, Technical Report CSL-95-1, Xerox Palo Alto Research Center, March 1995.
10. SENTIENT COMPUTING, AT&T Laboratories, http://www.uk.research.att.com/spirit/
11. SEMANTIC WEB Organisation, http://www.semanticweb.org
12. SLP (Service Location Protocol), http://www.ietf.org/rfc/rfc2608.txt
13. T. KINDBERG, et al.: people, places, things: web presence for the real world