Towards a Rule-Based Approach for Context-Aware Applications*

Laura Daniele, Patrícia Dockhorn Costa, and Luís Ferreira Pires

Centre for Telematics and Information Technology,
University of Twente, Enschede, The Netherlands
{l.m.daniele, p.dockhorncosta, l.ferreirapires}@ewi.utwente.nl

Abstract. Context-aware applications can sense and explore the users' context in order to provide proper and useful services to these users. These applications can react intelligently upon changes in the user's context, performing actions relevant to the user, the application itself, and the interaction between user and application. Context-aware reactive behaviors can be expressed by using rules written in a Domain-specific Language, coined ECA-DL, specially developed for context-aware applications. This paper proposes support for the development of a generic component capable of executing rules written using ECA-DL. This component executes these rules by using Jess, which is a well-known tool for developing rule-based systems.

Keywords: Context-awareness**,** ECA pattern, ECA-DL, Rule-based systems.

1 Introduction

Context-awareness is a computing paradigm in which applications can determine their behavior by sensing and exploring the users' context without explicit user intervention. These applications can react intelligently upon changes in the user's context by performing actions relevant to the user, the application itself, and the interaction between user and application. We can express this by using a language specially developed for context-aware applications, coined Event-Control-Action Domain-specific Language (ECA-DL) [13].

This paper proposes support for the development of a rule engine component capable of processing context-aware applications behaviors expressed in ECA-DL. This component can be implemented by using Jess, a well-known tool for developing rule-based systems. However, since the Jess rule engine reasons in terms of a specific language, we have to map ECA-DL statements onto the Jess language in order to have these statements being executed. The ultimate goal of this work is to show how this mapping can take place.

The structure of the paper is the following: Section 2 describes context-aware applications and the Event-Control-Action (ECA) pattern, which is an architectural

* This work is part of the projects Freeband AWARENESS and A-MUSE (http://awareness.freeband.nl, http://a-muse.freeband.nl). Freeband is sponsored by the Dutch government under contract BSIK 03025.

A. Pras and M. van Sinderen (Eds.): EUNICE 2007, LNCS 4606, pp. 33–43, 2007.

pattern that facilitates the development of context-aware applications, Section 3 presents the ECA-DL language, Section 4 describes the Jess architecture, Section 5 discusses the mapping from ECA-DL onto Jess by means of a case study, Section 6 discusses some related work on context-aware applications based on ECA rules, and Section 7 presents our conclusions and identifies topics for future work.

2 Context-Awareness

In the area of ubiquitous and pervasive computing, context is considered as key in the efforts to disperse and enmesh computation into people's lives. Context-aware applications aim to acquire and utilize information about the context of a device and its user to provide services that are appropriate to particular people, place, time and events [1].

Context-awareness implies intelligence that enables an application to discover, reason and predict a situation, and adapt to it in a dynamically changing environment. Applications operating in distributed environments also had to become mobile, in particular when servicing people on the move. In order to produce real awareness in ubiquitous and pervasive computing, programs with embedded intelligence also had to become mobile and retrieve context-related information in different locations. Thus, mobility aids in the intelligent acquisition of context [2].

2.1 Context-Aware Applications

In our work we start from the definition of context given in [3], which is: *"Context is a collection of interrelated circumstances in which something exists or occurs"*.

In the development of context-aware applications we have to cope with context discovery, sensing, extraction, manipulation and interpretation. Actually, we need to create a correspondence between objects in the real world and objects in the applications, since circumstances sensed by the environment in real world cannot be directly used by the applications. Therefore, it is necessary to represent these *circumstances* for the real world, which form the context, in terms of *context information*, characterized by specific values, which we call *context conditions*.

Fig. 1 shows the context of a person (application user) in real world and context-aware applications that can only refer to this context through context information [4].

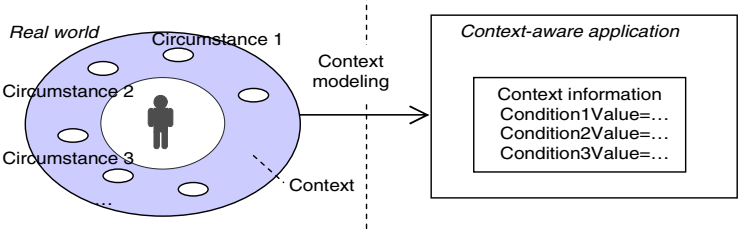

Fig. 1. Context in real world versus context information in context-aware applications

We can define many different kinds of context circumstances. For example, the geographical location in which the user can be found, environmental circumstances of the physical environment of the user, such as temperature, humidity, light, etc., or the user's vital signs like the heart beat or the blood pressure.

2.2 Event-Control-Action (ECA) Pattern

Whenever some specific circumstances change in the user's context, the applications should be able to consequently adjust their behavior. For this purpose we can use the Event-Control-Action (ECA) pattern [5]. The Event-Control-Action (ECA) pattern is an architectural pattern that can facilitate the development of context-aware applications, since it presents solutions for recurring problems associated with managing context information and reacting upon context changes.

Fig. 2 shows that the ECA pattern divides the tasks of gathering and processing context information (*Event* module), from tasks of triggering actions in response to context changes (*Action* module). These separate tasks are realized under the control of an application behavior description (*Control* module), in which reactive context-aware application behaviors are described in terms of ECA rules, also called *condition rules*. These rules have the form *if*<*condition*> *then* <*action*>. The condition part of an ECA rule specifies the situation under which the actions are enabled, and it consists of logical combinations of *events*. An event models some happening of interest in our application or its environment. The action part of the rule consists of one or more actions that are triggered whenever the condition part is satisfied [5].

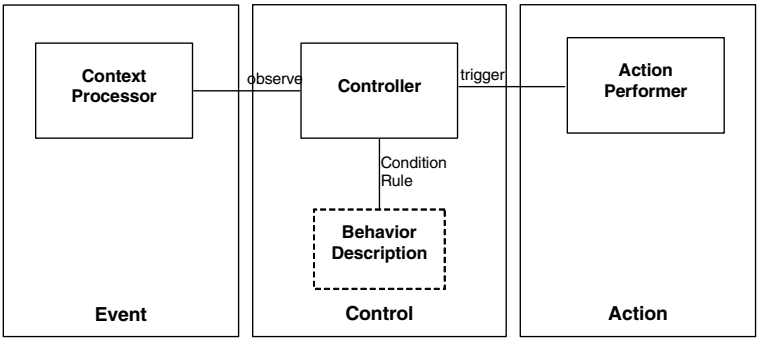

Fig. 2. Event-Control-Action pattern in context-aware applications

Therefore, the ECA pattern reflects the reactive nature of context-aware applications, whose behaviors can be expressed in ECA rules. In order to allow ECA rules to be expressed and manipulated, we have developed a language to define them. This language has been developed in the scope of the Freeband AWARENESS project [6] and is coined ECA Domain-specific Language (ECA-DL).

3 ECA-DL

ECA-DL is a *Domain-specific Language* specially targeted to context-aware applications. Rules in ECA-DL consist of an *Event* part that models an occurrence of interest in the context, a *Condition* part that specifies a condition that must hold prior to the execution of the action, and an *Action* part to be executed when conditions are fulfilled. Often the Action part of a rule consists of the invocation of a notification service, but it could also be any operation needed by the application. ECA-DL rules follow the ECA pattern, and, therefore, they can be used for specifying ECA rules.

ECA-DL has been developed with the following requirements in mind [14]:

- *Expressive power*, in order to enable the specification of complex event relations. ECA-DL allows the use of relational operator predicates (e.g., <, >, =) and the use of logical connectives (e.g., AND, OR, NOT) on events, allowing compound conditions to be built;
- *Convenient use*, in order to facilitate its utilization by context-aware application developers. ECA-DL provides high-level constructs that facilitate the definition of event compositions;
- *Extensibility*, in order to allow extension of predicates to accommodate events being defined on demand, as well as event properties.

In ECA-DL, context changes are described as changes in situation states. *Situations* represent specific instances of context information, typically high level context information. Situations may be defined upon other situations or facts. *Facts* define current "state of affairs" in the user's environment.

Facts and situations are defined as part of information models, which we have defined using UML class diagrams. Our models define entities, context, and mutual relationships between each entity and its context.

4 Jess

In order to find a suitable technology to execute ECA-DL rules, we compared some available tools for developing rule-based system, namely CLIPS [7], Jess [8], jDREW [10] and Mandarax [11], and Jess appeared to be the most appropriate choice. Jess (Java Expert System Shell) is a fast and powerful rule engine that supports the development of rule-based systems and runs on the Java platform.

A rule-based system basically consists of *facts* and *rules*. Facts represent all the pieces of information the rules work with. The general form of a Jess rule is:

```
(defrule RuleName "comment"
(fact_1). . . (fact_N) => (action_1) . . . (action_M))
```

Jess rules have two parts: a left hand side (LHS) and right hand side (RHS). The LHS is strictly defined for matching fact patterns. The RHS defines a list of actions to be performed if the pattern(s) of the LHS is (are) satisfied. Actions are typically method calls. Fig. 3 shows the Jess architecture.

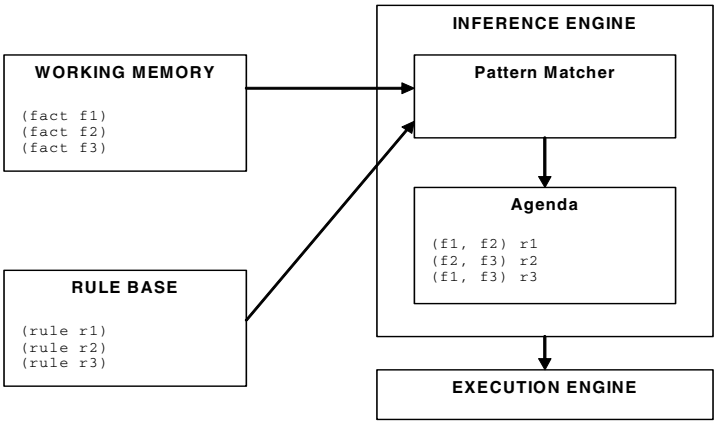

Fig. 3. The Jess architecture

The *working memory* contains facts, which can be used as both LHS and RHS of the rules. The Jess working memory is similar to a relational database: facts are like rows of a database, with indexes to speed up searching in the working memory. The *rule base* contains all the rules the engine knows. The *inference engine* decides what rules to fire and when, and consists of the *pattern matcher* and the *agenda*.

The *pattern matcher* decides which rules to activate based on the contents of the working memory. A rule is activated when the pattern matcher finds facts that satisfy the LHS of this rule, assuming a forward chaining reasoning. The *agenda* stores the list of rules that could be potentially fired. The agenda consists of an ordered list of rules, whose RHS can be executed. The agenda has to decide which rules have the highest priority and should be fired first. This process is called *conflict resolution strategy* and usually it takes into account the specificity or complexity of each rule, and the relative age of the LHS of each rule in the working memory.

Finally, the *execution engine* fires the rules, by executing the RHS of each rule that the inference engine has decided to fire [9].

5 Mapping ECA-DL to Jess

Jess can only process rules expressed in the Jess language. Therefore, since we want to use Jess to execute ECA-DL rules, we need to define mappings from ECA-DL onto the Jess language. We have performed some case studies in order to identify these mappings. Based on our experience with these cases studies, we defined guidelines for these mappings, which can be used as input for automated translation.

5.1 General Approach

Fig. 4 shows the general approach we have taken for designing the mappings.

An information model in ECA-DL consists of a UML class diagram that depicts entities and contexts, reflecting the knowledge that the target context-aware

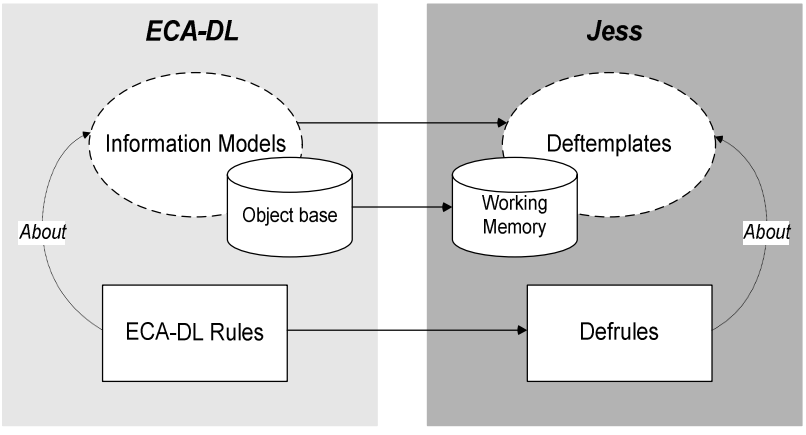

Fig. 4. Design of the mapping from ECA-DL onto Jess: general approach

application manipulates. Entities and context are represented as classes, and the relationships between them are defined as associations between these classes. A `deftemplate` in Jess is the static structure to define the structure of facts. We need to define `deftemplate`s before asserting any facts in the working memory of the rule engine. The first step in our approach has been to provide a mapping from the ECA-DL information models to `deftemplate`s in Jess.

We can create instances of the classes represented in an ECA-DL information model. These instances are the objects contained in the object base shown in Fig. 4. Analogously to defining objects in ECA-DL, we can also assert facts with specific values in Jess. These facts reflect the structure of the `deftemplate`s and they are contained in the working memory of the Jess engine. Therefore, the operation to create objects in ECA-DL corresponds to the operation to assert facts in Jess. The second step in our approach has been to provide a mapping from one or more ECA-DL objects in the object base to facts in the working memory of Jess.

Fig. 4 shows that ECA-DL rules are based on information models. Analogously, Jess rules, defined with the `defrule` command, are based on `deftemplate`s. ECA-DL rules use objects that are instances of the entity and context classes of the information model. Likewise, `defrule` constructs in Jess use facts asserted on previously defined `deftemplate`s. The third and last step in our approach has been to provide a mapping from ECA-DL rules to `defrule`s by investigating the correspondences between ECA-DL specific constructs to Jess constructs.

5.2 Case Study

Consider the following scenario:

"During the hot season, when the temperature in a building of the University of Twente is more than 30 degrees and it is later than 14:00 hours and earlier than 17:00 hours, all the persons in the building should be notified to go home".

We have expressed this scenario by using the following ECA rule:
If <During the hot season the temperature in a building of the University of Twente is more than 30 degrees AND it is later than 14:00 hours AND it is earlier than 17:00 hours > *then* <Notify (all the persons in the building), "You can go home."">

Fig. 5 represents the mapping from the ECA-DL information model corresponding to this rule onto `deftemplates` in Jess.

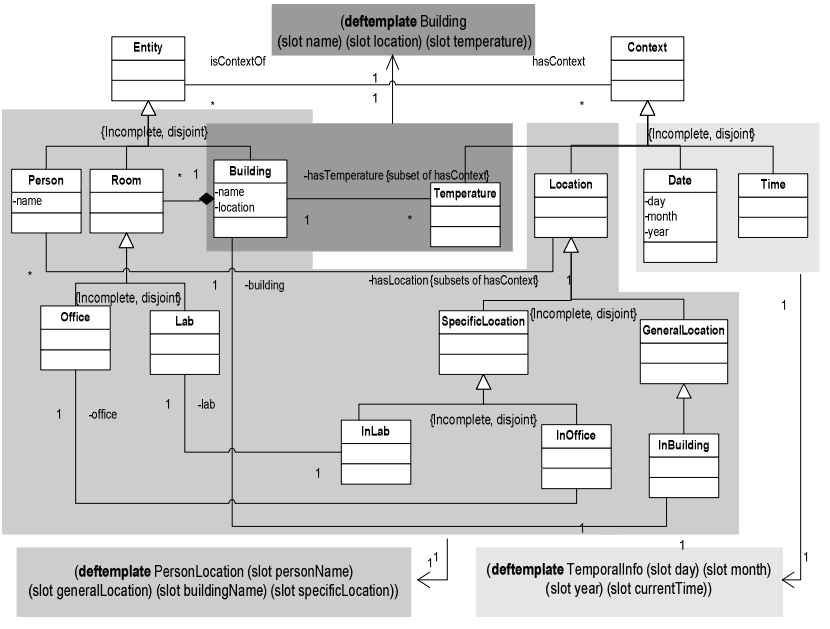

Fig. 5. General mapping from an ECA-DL information model to Jess `deftemplates`

In Fig. 5, the association hasLocation between the entity Person and the context Location has been mapped onto a PersonLocation `deftemplate`. The association hasTemperature between the entity Building and the context Temperature has been mapped onto a `deftemplate` called Building. A Building has a name (slot name in the template), a location (slot location), which in our case is the University of Twente, and a temperature (slot temperature). The context Date and Time have been mapped onto a `deftemplate` called TemporalInfo. Date has attributes day, month, year that are mapped, respectively, onto the slots day, month, year of the `deftemplate`, while Time has been mapped onto the slot currentTime.

Fig. 6 presents the ECA-DL rule we have used to describe the desired application behavior, and the Jess rule onto which this ECA-DL rule has been mapped. In the ECA-DL rule, the **`Select`**`(building.*, build, build.inUT)` clause defines all building located in the University of Twente and the **`Scope`** clause stores this set of buildings in a variable `b`. The rule is executed upon the event `EnterTrue (b.temperature>30)`, i.e., when the temperature in a building of the University of Twente is more than 30 degrees, and when the additional conditions `currentTime >`

14 and currentTime < 17 are fulfilled. The **Do** clause selects all the persons in a building b, i.e., in the building of the University of Twente where the temperature is more than 30 degrees, in order to notify them to go home. Finally, since the rule should be executed during the hot season, the lifetime associated with the rule is **from** <May> **to** <September>, which are the hottest months of the year.

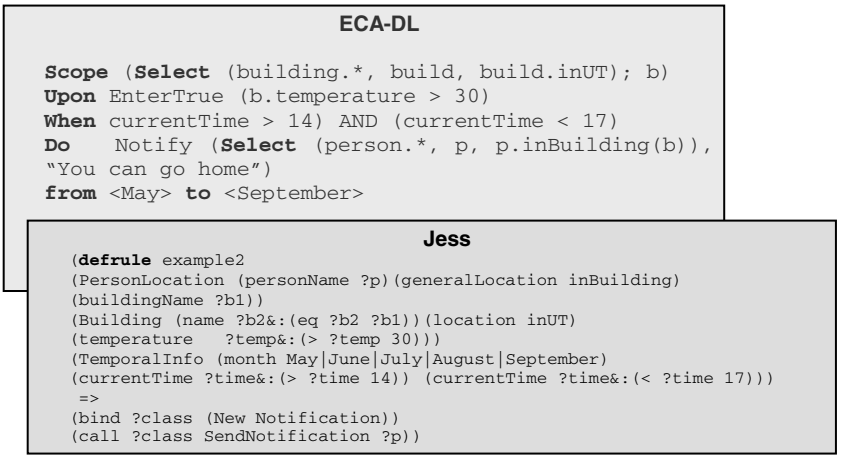

Fig. 6. Mapping from an ECA-DL rule onto a Jess rule

In the Jess rule, the **defrule** command checks in the Jess working memory for facts PersonLocation with slot generalLocation with value inBuilding, and stores the values of the slots personName and buildingName, in variables ?p and ?b, respectively. Then, it checks for facts Building with a slot name with the same value of the slot buildingName, a slot location with value inUT, and a slot temperature with a value higher than 30 degrees. Finally, it checks for a fact TemporalInfo with a slot months with value May or June or July or August or September, and a slot currentTime with a value between 14 and 17.

If the engine finds all these facts, it executes the RHS of the rule, which creates an object named ?class by instantiating the Notification class and calls a method SendNotification on this object in order to notify ?p (i.e., all the persons that have location in a building of the UT with a temperature higher than 30 degrees).

The clauses **Scope** (**Select** (building.*, build, build.inUT); b)) and **Upon** EnterTrue (b.temperature > 30) have been mapped onto the slots (location inUT) and (temperature ?temp&:(> ?temp 30)) of the fact Building in the LHS of the **defrule**.

The **When** clause has been mapped onto the slots (currentTime ?time&:(> ?time 14)) and (currentTime ?time&:(< ?time 17)) of the fact TemporalInfo in the LHS.

The **Do** clause has been mapped onto the RHS of the **defrule**, but the clause **Select** (person.*, p, p.InBuilding(b)) corresponds to the following code in the LHS:

```
(PersonLocation(personName ?p)(generalLocation inBuilding)
(buildingName ?b1))
(Building (name ?b2&:(eq ?b2 ?b1))
(location inUT)(temperature ?temp&:(> ?temp 30)))
```

Finally, the lifetime **from** <May> **to** <September> has been mapped onto the slot (month May|June|July|August|September) of the fact TemporalInfo in the LHS.

6 Related Work

Although considerable efforts have been made in rule-based context-aware applications to map ECA rules to a language of choice, none of the efforts we found in the literature use Jess as software environment to support the mapping.

In [15], a CORBA-based ECA Rule Matching Service is presented. This service complements the Standard CORBA Notification Service with a Composite Event Matching Engine based on CLIPS. This service highly simplifies the development of reactive applications by alleviating the programmer from the implementation of complex composite event handling mechanism. Although CLIPS provides a proper rule-based environment to execute ECA rules, Jess is a better choice for our purpose as we have discussed in [12]. Basically, Jess is the Java evolution of CLIPS, which is written in C. Both engines support a high level of extensibility and integration with code written in other programming languages. An important requirement for our work has been to be able to extend the rule engine's standard functionality by using Java, in order to process ECA rules expressed in the ECA-DL language, and Jess fulfils this requirement. Moreover, although both engines provide interactive development environments, Jess comes with JessDE, which is an Eclipse-based development environment that allows the developer to increase productivity. Finally, concerning the language used by these systems and its ease of use, CLIPS language is considered as inconvenient for the programmer because the overuse of parenthesis and the need to use inverse polish notation for building arithmetic and conditional expressions.

In [16], the modeling of complex ad-hoc context-aware scenarios is discussed. These scenarios are defined in terms of a set of ECA rules for each entity that is relevant for the scenario. Towards this aim, a very flexible and stable context middleware software framework was implemented and tested for an example scenario. Nevertheless, the drawback of this framework is that it cannot guarantee the consistence of rules by reasoning about entities and their relationships. In the reported implementation, the task to trigger actions according to incoming events is performed by an interpreter component within the framework, which is not able to check the consistence of new rules. Therefore, in order to manage entity relationship reasoning, the authors plan to integrate the Jess library into the context framework that they have realized. In our work, we considered that the usage of an available expert system shell like Jess reduces the cost and time of development compared with writing the expert system from scratch, as it has been done in [16].

7 Conclusions and Future Work

In [12] we have reported on the mapping of ECA-DL rules onto the Jess language and, based on case studies similar to the one above, we have been able to identify patterns and generalize these mappings. More detail on the case studies and mapping guidelines can be found in [12].

Future work consists of the improvement, generalization and automation of the mapping in order to enhance productivity and provide an automatic translation of ECA-DL rules (i.e., of the **`Upon`**, **`When`**, **`Do`**, **`Select`**, **`Scope`** clauses) to Jess `defrules`. Currently, work is being done to specify and implement this automatic translation as a transformation based on the metamodels of the ECA-DL and the Jess languages. This way, changes in the ECA-DL specifications can automatically reflect to the `deftemplates` structures in Jess.

In addition, it would be interesting to provide a mapping from ECA-DL to a generic rule engine model, which is not specific to any particular technology. This generic model could be mapped onto different engines, such as the ones that we have studied in [12], with little effort. In this way the mapping effort concentrates on creating a generic model of an application that can be mapped straightforwardly to specific technologies.

References

1. Moran, T.P., Dourish, P.: Introduction to This Special Issue on Context-Aware Computing. Human Computer Interaction 16, 87–95 (2001)
2. Zaslavsky, A.: Mobile Agents: Can They Assist with Context Awareness? In: MDM'04. IEEE International Conference on Mobile Data Management, p. 304. IEEE Computer Society Press, Los Alamitos (2004)
3. Context: Merriam-Webster Online Dictionary page. Available at [http://www.m-w.com/dictionary/context]
4. Dockhorn Costa, P., Ferreira Pires, L., van Sinderen, M.: Architectural Support for Mobile Context-Aware Applications. In: Handbook of Research on Mobile Multimedia, pp. 456–475. Idea Group Inc. (2006)
5. Dockhorn Costa, P., Ferreira Pires, L., van Sinderen, M.: Architectural Patterns for Context-Aware Services Platform. In: Proceedings of the Second International Workshop on Ubiquitous Computing (IWUC 2005), Miami (May 2005)
6. Freeband AWARENESS project. Available at [http://awareness.freeband.nl]
7. CLIPS website. Available at [http://www.ghg.net/clips/CLIPS.html]
8. Jess website. Available at [http://herzberg.ca.sandia.gov/jess/]
9. Friedman-Hill, E.: Jess in Action: Java Rule Based Systems. Manning Publications Co. (2003)
10. jDREW website. Available at [http://www.jdrew.org/jDREWebsite/jDREW.html]
11. Mandarax website. Available at [http://mandarax.sourceforge.net/]
12. Daniele, L.M.: Towards a Rule-Based Approach for Context-Aware Applications. Thesis for a Master of Science Degree in Electronic Engineering from the University of Cagliari, Italy (May 2006), Available at [http://asna.ewi.utwente.nl/education/Student%20assignments/completed%20bachelor%20and%20master%20assignments/daniele.html]

13. Dockhorn Costa, P., Ferreira Pires, L., van Sinderen, M., Broens, T.: Controlling Services in a Mobile Context-Aware Infrastructure. In: CAPS 2006. Proceedings of the Second Workshop on Context Awareness for Proactive Systems, Kassel, Germany (June 2006)
14. Etter, R., Dockhorn Costa, P., Broens, T.: A Rule-Based Approach Towards Context-Aware User Notification Services. In: Proceedings of the IEEE International Conference on Pervasive Services, Lyon, France, June 2006, pp. 281–284. IEEE Computer Society Press, Los Alamitos (2006)
15. de Ipiña, D.L.: An ECA Rule-Matching Service for Simpler Development of Reactive Applications. In: Proceedings of Middleware 2001 at IEEE Distributed Systems Online 2(7) (November 2001)
16. Beer, W., Christian, V., Ferscha, A., Mehrmann, L.: Modeling Context-Aware Behavior by Interpreted ECA Rules. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 1064–1073. Springer, Heidelberg (2003)