A Rule-Based Approach Towards Context-Aware User Notification Services

Richard Etter, Patricia Dockhorn Costa, Tom Broens

Abstract— **Pervasive Computing is the vision of technology that is invisibly embedded in our natural surroundings. Users are offered unobtrusive services that require minimal attention. In this paper the Awareness and Notification Service (ANS) is presented that enables to rapidly build applications that inform users about their environment. Additionally, the service offers notifications tailored to the user's preferences and current context. ANS takes a rule based approach based on the Event-Condition-Action pattern. Users specify when and what should be notified to them by using a convenient ANS rule language. This flexible mechanism allows to rapidly develop applications that provide context-aware notifications without the need to write programming code to activate rules, nor to implement personalized notifications.**

I. Introduction

In pervasive computing environments, users are surrounded with ubiquitous communication and information technology. In this paradigm, the users and their requirements are central. Pervasive computing environments offer unobtrusive services that assist users in their daily live. The services automatically adapt to them and provide support in their everyday activities.

Context awareness [1] has emerged as a key element in pervasive computing. It facilitates applications that are aware of their environment and enables them to adapt to the current context. A key application that can benefit from context awareness is a notification service. With context awareness a notification service is able to keep track of changes in the environment and can appropriately communicate these changes to applications that users are currently using. The notification of the users can even depend on their current situation. For example, when users have a meeting, they receive a less intrusive notification than in a non-meeting situation.

This work was partially funded by the Amigo project in Ambient Intelligence. The Amigo project is funded by the European Commission as an integrated project (IP) in the Sixth Framework Programme under the contract number IST 004182.

Richard Etter, Fraunhofer Institute IPSI, Germany; division AMBIENTE – smart environments of the future. e-mail: richard.etter@ipsi.fraunhofer.de
Tom Broens, Centre for Telematics and Information Technology, University of Twente, the Netherlands, e-mail: broens@ewi.utwente.nl
Patricia Dockhorn Costa, Centre for Telematics and Information Technology, University of Twente, the Netherlands, e-mail: p.dockhorncosta@ewi.utwente.nl

In order the enable such a notification service, a flexible mechanism is required that allows user-applications to easily specify what changes in the environment their users are interested in. Current middleware for context-aware systems [2][3][4] provides unified ways to subscribe to and manage context data. But they fall short providing efficient and convenient ways for application developers to specify what context data they are interested in. Besides this, the notification process is generally not tailored to the users' context.

In this paper, we present a context-aware notification service (coined Awareness and Notification Service – ANS) which is based on a rule-based approach and provides notifications depending on the users' context. The following section (II) gives an overview of the Awareness and Notification Service. Section III describes the architectural design of the service. Section IV introduces the ANS rule language. Section V gives conclusions.

II. The Awareness and Notification Service

ANS provides the basic functionality required to develop applications that allow applications and users to stay aware of any significant change in their environment with minimal effort. Changes that ANS is able to keep track off can be of various nature such as the activities or presence of remote people.

From the system perspective, ANS makes applications aware of context changes by notifying them. Applications do not have to take care about managing and monitoring context data. Applications only have to register monitoring rules that specify what changes in context should be notified to them. In our approach, changes in the environment are modeled using Event-Condition-Action (ECA) rules [5]. The developed domain specific language explicitly defines the concepts of context and context events that facilitates the specification of context-aware reactive behaviors. Application developers write ECA rules using this language in a scripting format.

From the user perspective, ANS provides notifications with appropriate rendering of intensity, based on a user's preferences and current context. In order to be notified appropriately, users create an individual user notification profile. The profile describes how and when a user wants to be notified. Before ANS sends a notification, the service checks the notification profile of the user that is to be notified. Based on this profile, ANS sends a notification with an appropriate rendering of intensity. The application receiving the ANS notification implements the notification of the user according to the intensity. An example is an application that changes the color of an ambient light in order to deliver a notification with a low intensity.

III. Architectural Design

The design of ANS is based on the principles of a service oriented architecture. The service is implemented as a webservice and is based on the standards SOAP, WSDL, XML, and UDDI. All external services that ANS depends on are implemented as webservices, too. Internally ANS follows a component based approach based on the OSGi component based framework [6]. Figure 1 shows the architectural design of ANS.

Conceptually three main parts can be distinguished following the Event-Control-Action Pattern [5]. The

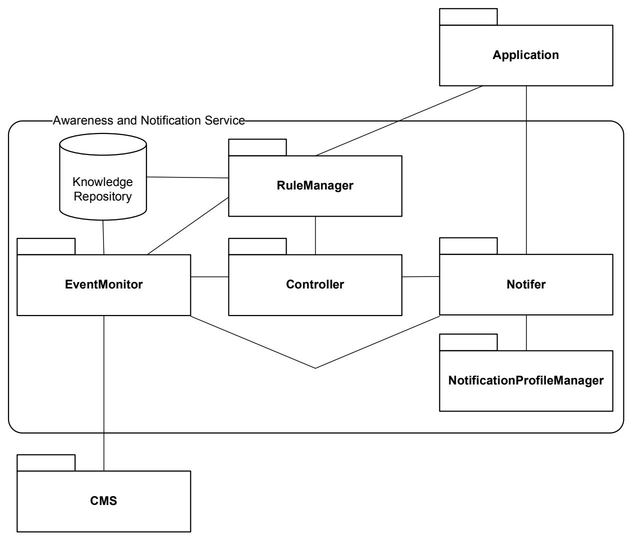

Fig. 1. Architectural Design of ANS

EventMonitor provides the context data events received from context sources offered by a Context Management Service (CMS). The Controller monitors these events and evaluates notification rules. If a notification rule evaluates to true the Notifier is triggered, which represents the action part.

A. The EventMonitor

The EventMonitor is responsible to provide easy access to context data. This includes searching for context sources, selecting a context source, registering to the context source eventing mechanism, and deregistering when a context source is no longer needed.

The EventMonitor allows other ANS components to easily subscribe to or query for context data. If a component needs to know the location of a certain person, the Event Monitor connects to a Context Management Service (CMS) and subscribes to the requested location data. From then on the Event Monitor provides the component with events containing the requested data.

For every event that an ANS component has subscribed to, the EventMonitor maintains a subscription to the corresponding context source component. It keeps an overview of the events it has subscribed to and makes sure that in case of an overlap in requirements (i.e. when two different events can be served by the same context source) no redundant subscriptions are registered.

B. The Controller

The controller is the central entity in the ANS service. It is responsible for retrieving and parsing user specified rules that define a notification. Additionally, it monitors context events that it receives from the EventMonitor. Every time, it receives an event, it evaluates the notification rules and eventually, when the rules condition turns true, initiates the notification of a user.

The controller consists of two sub-components:

- RuleManager: this component is responsible for translating entered user rules to notification rules which can be handled by the controller.
- Controller: the task of the Controller is to monitor the received context events and to evaluate the notification rules. This is mainly done by applying a generic rule-engine like JESS [7] (which is currently used in ANS), Mandarax [8], or SweetRules [9]. The Controller starts the notification process when a rule is triggered.

In more detail, the RuleManager offers two main functionalities: (i) (un)subscription of new rules and (ii) management of existing rules. When a new rule is entered, the rule manager checks if the rule is well-formed (using the language meta-model). It extracts the relevant context variables from the condition part of the rule and subscribes to changes of these context variables via the EventMonitor. The EventMonitor generates context events when these context variables change.

The RuleManager transforms the rules expressed in the ANS domain-specific ECA language (see section IV) into a rule that can be handled by the underlying rule-engine (in this case JESS). These mappings are defined modularly which offers a flexible mechanism to plug-in other rule-engines. The management of rules consists of starting, stopping, updating, and querying rules.

The Controller wraps a general purpose rule-engine. Its main functionality is to transform events received from the EventMonitor into changes in the knowledge base used by the rule engine. Furthermore, it instantiates the rules received from the RuleManager in the rule-engine. When a rule is triggered a notification event is generated and send to the Notifier. This notification event triggers the notification of users.

C. The Notifier

The task of the Notifier is to send notifications to applications and users. For each notification it determines the appropriate level of intensity before sending the notification. The intensity of notifications is based on the current context of users and their personal notification preferences.

If users want to receive personalized notifications, they create individual user notification profiles. A user notification profile defines which level of intensity is appropriate in which context. The intensity of notifications can be based on availability of the user that is to be notified, or where the user is currently located. A further option is to take the co-presence of other persons into account. In general, the notification profiles are dynamic and allow users to take into account all available context data. When editing a personal user notification profile, a user is free to combine the different parameters in order to specify the appropriate level of intensity for certain situations. In case a user defines settings that are conflicting, an implemented conflict strategy implemented in ANS resolves the issue. A user can use one of the predefined profiles or refine one the profiles. If a user has not created a notification profile, a standard profile is used.

The notification of a user works as follows. If a rule in the Controller evaluates to true, the Controller sends a notification event to the Notifier. The Notifier transforms the event into a notification. An event encompasses the message, the UserIDs of the users that are to be notified and references to applications. First the Notifier determines the right intensity for the notification. It does so by retrieving the relevant user notification profiles. In case additional context parameters are necessary in order to determine the intensity of the notification, the Notifier queries the EventMonitor. This is for example the case, if a user has specified not to be notified at home. In this case the Notifier queries the EventMonitor for the location of the user. Once the notification profiles are evaluated and the intensity of the notification is determined, the Notifier sends the notification to the application. It is then the task of the application that receives the notification to interpret the level of intensity, e.g. to change the color of an ambient light in order to send a notification with a low level of notification to a user.

IV. The ANS Rule Language

The ANS rule language allows application developers to conveniently enhance their applications with reactive context-aware behavior by using a scripting format. This relieves the developer from writing programming code inside his application to deal with notifications. This is handled by the ANS service when initiating the rules.

Notification rules follow the Event-Condition-Action (ECA) pattern [5][10]. A *Context Data Event* models an occurrence of interest (e.g., a change in context), which may or may not establish results; A *Condition* specifies what must hold prior to the execution of the action; and Actions consist of notification services.

When designing the ANS ECA rule language, high attention has been paid to the following qualities:

- Expressive power: the language permits the specification of complex event relations. It allows the use of relational operator predicates (e.g., < , >, =), and the use of logical connectives (e.g., AND, OR, NOT) on conditions to build compound conditions.
- Convenient use for application developers: It provides high-level constructs that facilitate event compositions.
- Extensibility: The language allows the addition of new predicates to accommodate events being defined on demand.

A. Basic Concepts

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 [11].

Facts define current "state of affairs" in the user's environment. Example of a Fact is *Jerry is married to Maria*. The situation context abstraction allows application developers and users to leverage on the fact abstraction in order to derive high-level context information. Example of a situation is *isOccupied*, derived from the fact "Maria is cooking" or "Jerry is working". Situations may be built upon other situations, for example, isAvailable may be defined as not isOccupied and isReachable. Facts and situations are defined as part of the overall information models (ontologies), which are not discussed in this paper due to lack of space.

There are three possible states (true, false and unknown) and six state transitions (e.g., TrueToFalse and FalseToUnknown). The unknown state accommodates uncertainty of context information (when the value of context information is unknown). Notification invocations are associated with sequences of transitions and the validation of pre-conditions.

B. Syntax and Semantics

The condition part of ECA rules comprises two parts: an event part that defines a relevant situation change; and a pre-condition part that defines a logical expression that must hold following the event and prior to the execution of the notification. Both events and pre-conditions are defined in terms of situation and facts.

Each trigger rule is associated with a lifetime, which can be `once`, `from <start> to <end>`, `to <end>`, `<n> times`, `frequency <n> times per <period>`. Events, pre-conditions and notifications are prefixed by the clauses **`Upon`**, **`When`** and **`Do`**, respectively. The clause **`scope`** has been included to parameterize an ECA rule. A **`scope`** clause defines a collection of entities for which the rule should be applied. The the clause **`select`** has also been included, which returns a collection of entities respecting a given filtering expression.

Fig. 2 depicts the ECA language Metamodel in UML. In the following some examples of rules are presented:

1) Maria would like to be notified when her kids enter home with friends.

```
Upon EnterTrue (Pablo.isAtHome) OR EnterTrue
(Roberto.isAtHome)
When (Pablo.isAtHome AND Pablo.withFriends) OR
(Roberto.isAtHome AND Roberto.withFriends)
Do Notify (Maria, "kids are home with friends")
Always
The situations isAtHome and withFriends are defined
as follows:
```

- `entity.isAtHome` = entity's current location is within the home boundaries;
- `person.withFriends` = person is close to (e.g., within 20 meters) friends. In this scope, "friend persons" may be known or unknown to the system.

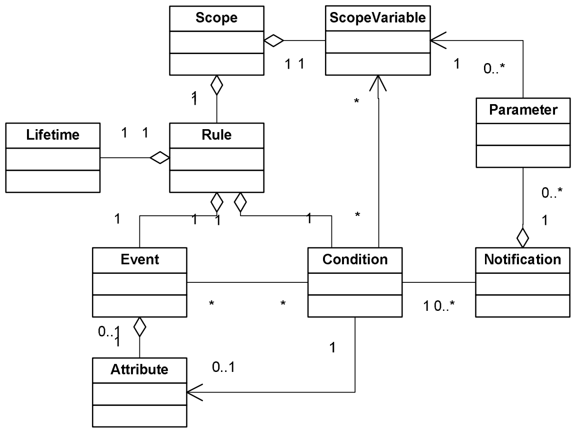

Fig. 2: ECA language metamodel

2) Notify all family members (except Jerry) that Jerry is arriving home.

```
Scope (Select (person.family.*, member,
member.isAtHome & "member.name <> Jerry"); p))
{
  Upon EnterTrue (Jerry.isAtHome)
  When True
  Do Notify (p, "Jerry is home")
  Always
}
```

The previous scope clause defines a dynamic set of family members (the ones inside the home), for which this rule should be applied. The scope variable "p" has been defined in the scope clause and has been used in the **DO** clause.

V. CONCLUSION

Pervasive computing environments provide new opportunities to inform users about their environment. In this paper, a rule-based notification service was presented. The service enables developers to rapidly implement applications that allow users to be aware of their environment. Applications register monitoring rules (i.e. specification of conditions based on context and corresponding notification actions) that specify what their users are interested in, by using a convenient rule language. Hereby, the management or monitoring of context data is delegated to ANS. The entered rules are evaluated continuously by ANS and relevant users are notified when one of the rules evaluate to true. Additionally, the service provides notifications that are tailored to the user's situation (context). Before the notification service sends a notification, the appropriate intensity for the notification is determined. This ensures that the notification service provides notifications that are as unobtrusive for users as possible and as intrusive as necessary.

An initial prototype of ANS has been implemented using Java and the generic rule engine JESS [7]. The implementation is based on the OSGi component based framework [6] to get a clear separation of concerns between the ANS sub-components. Additionally, OSGi facilitates the exposure of the service to other services and applications. Remotely ANS can be accessed using standard webservice technologies.

The development of ANS is ongoing and the following two future research issues will be tackled. First *temporal aspects* will be considered. The ANS rule language currently does not provide means to specify temporal ordering of events. The language will be extended to support temporal aspects, such as sequencing and concurrency of events. Secondly c*ontext model*s will be considered. In parallel to the definition of the ANS rule language, context models and context modeling abstractions will be defined that facilitate the representation of context information in a meaningfully and unambiguously manner.

Besides enabling rapid development of context-aware applications, the developed Awareness and Notification Service successfully represents a pervasive service that requires only as much attention of users as necessary. It is a foray in an age where users are ubiquitously surrounded with technology that unobtrusively assists them in their everyday live.

REFERENCES

[1] Dey, A.K., and Abowd, G., "Towards a Better Understanding of Context and Context-Awareness", CHI Workshop, 2000.

[2] Bardram, J.E., "Applications of Context-Aware Computing in Hospital Work–Examples and Design Principles" in Proceedings of the ACM Symposium on Applied Computing, 2004, pp 1574-1579

[3] Chen, H., "An Intelligent Broker Architecture for Context-Aware Systems", PhD proposal in Computer Science, University of Maryland, Baltimore County, USA, 2003.

[4] Dey, A.K., "Providing Architectural Support for Building Context-Aware Applications", Ph.D. thesis, College of Computing, Georgia Institute of Technology, 2000

[5] Dockhorn Costa, P., Pires, F., Sinderen, M., "Architectural Patterns for Context-Aware Services Platforms" in Proceedings of the Second International Workshop on Ubiquitous Computing (IWUC 2005), Miami, May 2005, pp 3-19

[6] OSGi consortium website, http://www.osgi.org/

[7] Jess – the Rule Engine for the Java Platform. Available at http://herzberg.ca.sandia.gov/jess/

[8] Dietrich, J., The Mandarax 3.0 Manual, Version December 8th 2003, Institute of Sciences & Technology, Te Kura Putaiao o Hangarau-a-Mohiotanga, Massey University, Palmerston North, New Zealand.

[9] jDREW website, http://www.jdrew.org/jDREWebsite/jDREW.html

[10] Ipina, D., and Katsiri, E., "An ECA Rule-Matching Service for Simpler Development of Reactive Applications". Published as a supplement to the Proc. of Middleware 2001 at IEEE Distributed Systems Online, Vol. 2, No. 7, November 2001.

[11] Henricksen, K. and Indulska, J., "A software engineering framework for context-aware pervasive computing" in Proc. of the 2nd IEEE Conference on Pervasive Computing and Communications (Percom2004), Orlando USA, 2004, pp 67-77.