Controlling Services in a Mobile Context-Aware Infrastructure[1]

Patrícia Dockhorn Costa, Luís Ferreira Pires, Marten van Sinderen, Tom Broens

Centre for Telematics and Information Technology, University of Twente, The Netherlands
{dockhorn, pires, sinderen, broens}@cs.utwente.nl

Abstract. Context-aware application behaviors can be described as logic rules following the Event-Control-Action (ECA) pattern. In this pattern, an Event models an occurrence of interest (e.g., a change in context); Control specifies a condition that must hold prior to the execution of the action; and an Action represents the invocation of arbitrary services. We have defined a Controlling service aiming at facilitating the dynamic configuration of ECA rule specifications by means of a mobile rule engine and a mechanism that distributes context reasoning activities to a network of context processing nodes. In this paper we present a novel context modeling approach that provides application developers and users with more appropriate means to define context information and ECA rules. Our approach makes use of ontologies to model context information and has been developed on top of web services technology.

1. Introduction

The dynamic nature of context-aware applications, and the increasing integration of these applications into our daily tasks in a variety of domains (e.g., home, work and leisure), generate rapid changes in the requirements for the technology to support these applications. Although it is not possible to fully predict these changes, the supporting technology can be designed in such a manner that it can be configured to match changing requirements, preferably at runtime. This calls for a high level of flexibility. We aim at coping with these issues by means of a shared Context Handling Infrastructure to support context-aware applications. This infrastructure comprises, among others, reusable context processing and managing services, which facilitate context-aware application development. It provides building blocks that can be combined and specialized to satisfy application-specific requirements. A central building block in our Context Handling Infrastructure is the Controlling Service. This service takes application-specific rules and information (context) models as input in order to carry out application-specific adaptation within the infrastructure, at runtime.

This paper aims at presenting our Controlling service, which facilitates the configuration of application-specific behaviors. Application requirements, expressed

[1] This work is part of the Freeband AWARENESS project (http://awareness.freeband.nl). Freeband is sponsored by the Dutch government under contract BSIK 03025.

in terms of pieces of application behaviors, are written in a scripting format following the Event-Control-Action (ECA) pattern. In this pattern, an Event models an occurrence of interest (e.g., a change in context); Control specifies a condition that must hold prior to the execution of the action; and Actions represent the invocation of arbitrary services. The Controller component facilitates the configuration of the infrastructure by taking ECA rules and application-specific context models as input to (i) subscribe to context sources, (ii) perform context reasoning, and (iii) trigger actions on behalf of applications, in response to context changes. We have developed a scripting language for the purpose of writing context-aware ECA rules. This language is composed of an information part, defined by our context models, and a behavior part, defined by the language metamodel. Since ECA rules are written in a scripting format, application developers do not need to write programming code.

Furthermore, we propose a generic context model that incorporates a novel context categorization scheme that classifies context according to its nature, providing application developers and users with more appropriate means to define context information and ECA rules. Our approach provides a generic context model that captures general concepts and allows domain-specific and application-specific extensions.

We focus on mobile context-aware applications that are widely distributed and are typically offered by telecommunication providers. Examples of such applications are healthcare tele-monitoring applications, tourism applications and communication applications. We ignore sensory issues in this paper, but rather focus on the service infrastructure that leverages on the sensor network to provide appropriate context information to a large range of context-aware applications. Since the nature of applications is diverse, a rich set of context information is exploited by the infrastructure, including location of people and devices, vital signs and user activity, amongst others.

The remainder of this paper is structured as follows: Section 2 presents the Controlling service, and identifies the challenges to realize such services; Section 3 presents our context model; Section 4 discusses our language to describe ECA rules; Section 5 presents an overview of the infrastructure architecture and our prototyping activities; Section 6 discusses related work; and Section 7 gives final remarks and conclusions.

2. The Controlling Service

A Controlling Service accepts ECA rule specifications and activates them within the infrastructure. ECA rule activation occurs at infrastructure runtime, which requires runtime discovery and composition of context and action services. Context services aim at providing context information and action services implement the actions to be triggered when context conditions are satisfied. Fig. 1 depicts a typical usage flow of the Controlling Service.

The following phases are identified:

– Phase 1 initiates with end-users defining application behaviors by means of a graphical interface.

- Phase 2 consists of performing the mapping of an end-user rule specification to a less abstract specification to be provided to the infrastructure, in a scripting format (e.g., XML). The translation from users' inputs to a rule specification in some notation that can be accepted by the infrastructure is a responsibility of the application components. It is also possible that application developers specify application rules, as opposed to end-user rules. End-user and application rules are equally treated in this paper.
- Phase 3 consists of the actual invocation of the Controlling Service after the rule specification has been provided to the infrastructure. The Controlling Service verifies whether the specification is well-formed and separates it into events, conditions and actions.
- Phase 4 corresponds to the attempt of the Controlling Service to find event sources capable of providing context event notifications of interest. The Controlling Service decides whether or not to subscribe to one of more of these Context Provisioning services.
- Phase 5 consists of the exchange of a subscribe request and eventual event notifications. The Controlling Service determines whether the conditions are satisfied by the context event notifications being generated.
- Phase 6 is entered typically when a certain condition is satisfied. At this moment, an action should be triggered, and, therefore, its actual implementation needs to be found. For that purpose, the Controlling Service makes use of the Action Discovery Service.
- Phase 7 emcompasses the actual execution of an Action Service.

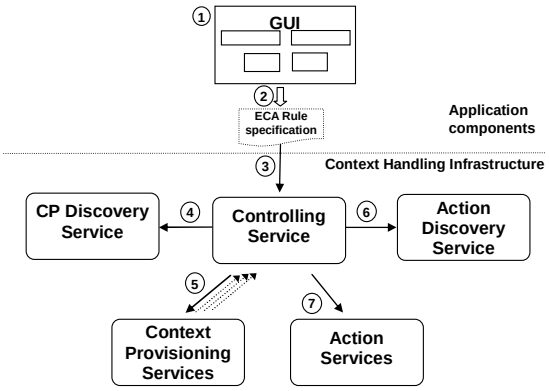

Fig. 1. Typical usage flow of the Controlling Service

Although much study has been carried out in each of the topics mentioned above, the following research questions remain open: (i) how expressive should the ECA rule language be to accommodate user's and developer's requirements? (ii) what are the context abstractions needed to effectively compose application behaviors? (iii) what elements should be included in ECA rule specifications to provide enough information to perform infrastructure configuration? (iv) how to dynamically discover context provisioning services based on ECA rule specifications? and (v) how to

invoke action services on behalf of application components? This paper provides answers for the three first questions in terms design solutions.

3. Context Modeling

A shared context model formally defines context information concepts and their relationships, such that context information can be distributed and unambiguously interpreted by interacting system parts. Our approach requires a context model to (i) provide application users and developers with appropriate means to describe context information and application behaviors; (ii) allow applications, infrastructure and third-party service providers to agree upon syntax and semantics of context information, thus enabling interoperation; and (iii) provide context processing components with proper means to perform context information reasoning. We have used ontologies to model context information in our infrastructure.

3.1 Characteristics of context information

We use the context modeling abstractions of *facts* and *situations* [4][5] to provide application developers and users with more appropriate means to define context information. A *Fact* defines a current "state of affairs" in the user's environment, such as "Bob has access to PDA and PC" and "Bob and his PDA are co-located in room A". The *situation* context abstraction allows application developers and users to leverage on the *fact* abstraction to derive high-level context information, such as `isOccupied`, derived from the fact that Bob is engaged in an activity and `isReachable`, derived from the fact that Bob is near to a device that supports a given communication channel. Situations may be built upon other situations, for example, `isAvailable` may be defined as Bob not being `isOccupied` and being `isReachable`. Application behaviors are defined at runtime as ECA rules, using our context models. Section 5 elaborates on ECA rules.

Since our service infrastructure supports a large number of mobile context-aware applications, a rich set of context information is exploited. However, it is not possible to define a complete context model that is accepted by all applications, since each application may define context information in a different way. For example, the context information `near` could mean *within 10 meters* in one application and mean *within 5 kilometers* in another. It may also be possible that certain context information types are domain-specific, rather than application-specific. For example, `heart-rate` and `body-temperature` are types of context information concepts shared among applications in the medical domain, but may be useless concepts in other domains. We suggest in our approach a general context model that contains concepts shared by applications we deal with. This model should be extended with application-specific concepts (facts and situations) on demand, at infrastructure runtime.

3.2 Context models

Fig. 2 depicts some parts of our general context model. This model is a context ontology that captures general concepts and allows domain-specific and application-specific extensions. We have used OWL-DL [8] to define this ontology.

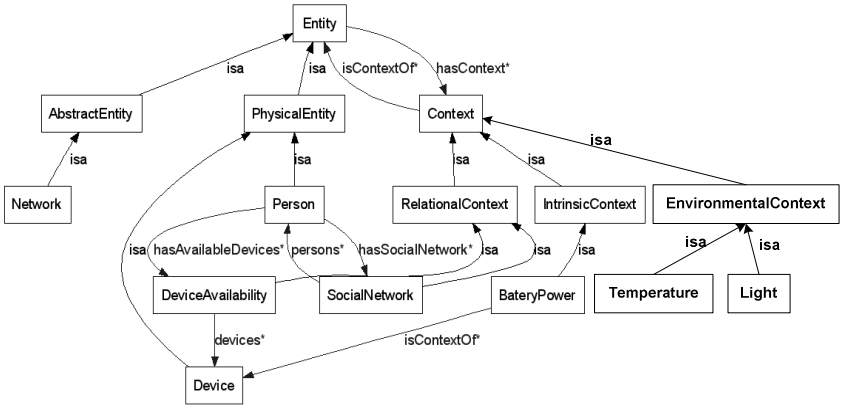

Fig. 2. Selected concepts of the general context model

We distinguish three fundamental categories of context information, namely intrinsic context (`IntrinsicContext`), relational context (`RelationalContext`) and environmental context (`EnvironmentalContext`):

- Intrinsic context defines a type of context information that belongs to the essential nature of an entity and does not depend on the relationship with other entities. An example of intrinsic context is the `location` of a `PhysicalEntity`, such as a `Person` or a `Building`.
- Relational context defines a type of context information that represents a relation between distinct entities. An example of relational context is `Containment`, which defines a containment relationship between entities, such as an entity `Building` contains a set of entity `Persons`. Another example of relational context is `Distance`, which represents the spatial distance between two physical entities.
- Environmental context defines a type of context information that belongs to the physical environment of an entity. Examples are `Light`, `NoiseLevel`, `Pressure` and `Temperature`.

Relational context may be used to relate an entity to the collection of entities that play a role in the entity's context. Examples of relational contexts that can be used for this purpose are `DeviceAvailability`, and `SocialNetwork` (Fig. 2). The `DeviceAvailability` of a person captures the person's current accessible devices and the `SocialNetwork` of a person captures the collection of persons interacting with that person by any communication channels.

The main benefit of this general categorization of context is that it explicitly separates the concepts of entity and context. The relational context type allows us to traverse from an entity to the entities that are related by the relational context. This

has enabled us to recursively define the relationship between entity and context, facilitating navigation in the model.

We have extended the general context model to support a tele-monitoring application [2]. In this application, patients' vital signals are processed to detect abnormalities, such as the possibility of having an epileptic seizure, within seconds. Several actions may be taken upon an epileptic alarm, such as contacting volunteers capable of providing first aid and streaming patient's bio-signals to doctors at real time.

We have specialized entity person to `AidPerson`, `Patient` and `HealthCarePerson`. In addition, to accommodate the new types of intrinsic context information presented by the tele-monitoring scenario, we have included `BodyPressure`, `HeartRate`, `hasSeizure`, `isOccupied`, `isAvailable`, `isReachable`, `isAtHome` and `isAtWork`. The first two types of context information (`BodyPressure` and `HeartRate`) are facts, meaning that they are not derived information but can be sensed directly from the environment. The situation `hasSeizure` is derived from the persons `HeartRate` and `BodyPressure` by means of a rather complex algorithm. The situations `isOccupied`, `isReachable`, and `isAtHome` are defined in OWL-DL (using our ontology) as follows:

- `isOccupied ≡ ∀ isContextOf (Person ⊓ (∃ hasContext Activity))`
- `isReachable ≡ ∀ isContextOf (Person ⊓ (∃ hasContext (DeviceAvailability ⊓ ((∃ hasContext ChannelAvailability)))))`
- `isAtHome ≡ ∀ isContextOf (Person ⊓ (∃ hasContext (Containment ⊓ (∃ container Home))))`

4. ECA Rules

The dynamic aspects of applications, i.e., application behaviors, are defined following the Event-Control-Action (ECA) pattern mentioned previously. We have developed an expressive language that enables the specification of ECA rules. These rules carry enough information to allow the Controller component to autonomously configure the infrastructure (composition of context and action) with no need for further intervention from the application developer. The behavior part of the ECA language we are proposing is based on the situation-based triggering approach presented in [4][5]. We have extended and adapted this approach to satisfy our infrastructure requirements. The information part of the ECA language is based on our context models, which have been presented in the previous sections.

Context changes are described as changes in situation states. There are three possible states (*true*, *false* and *unknown*) and six state transitions. The unknown state accommodates uncertainty of context information. Action invocations are enabled by sequences of transitions and the validation of pre-conditions. 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 action. Both events and pre-conditions are defined in terms of situations and facts. Each rule is associated with a lifetime, which can be `once`, `from <start> to <end>`, `to <end>`, `<n> times` and `frequency <n> times per <period>`.

Events, pre-conditions and actions are prefixed by the clauses `Upon`, `When` and `Do`, respectively. In our approach, we have included the clause `scope` to parameterize an ECA rule. A scope clause defines a collection of entities for which the rule should be applied. We have also included the clause `select`, which returns a collection of entities respecting a given filtering expression[2]. Consider the following ECA rule partially modeling the tele-monitoring scenario:

```
Scope (Select (entity.patient.*, pat, isIncluded
              (pat.medConditions, epilepsy)))
{
  Upon EnterTrue(pat.hasSeizure)
  When pat.hasSeizure.Accuracy > 50%
  Do critical
     Foreach (Select (entity.patient.aidPersons, aidP,
                      aidP.isAvailable ^ aidP.isNear(pat)))
     Contact (aidP)
always
}
```

This ECA rule defines a scope that includes all patients suffering from epilepsy. The function `isIncluded` is part of the standard library to manipulate collections. The `Upon` clause defines a situation state transitions (`EnterTrue`) in which the action should be invoked. The `When` clause defines a pre-condition for the action to be invoked, which is the minimum accuracy required (50%) for the seizure alarm. The `Do` clause defines that the patient's designated aid persons who are near and available (not occupied and reachable) should be contacted on their preferred channel. The term `critical` indicates that the rule should be pre-fetched, meaning that no time should be spent with action lookup requests.

The scope clause defines a dynamic group of epileptic patients. Epileptic patients may enter and leave the system, and the scope clause maintains the actual list of patients, creating and removing rules for each patient that enters and leave the system, respectively.

Conditions are asserted based on information contained in the knowledge base, which is kept up-to-date with context information events originated from the network of Context Sources.

Other examples of notification ECA rules are (in a smart home setting):

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

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

Notify Bob when the average temperature of the house goes beyond 30^{o}C.

```
Upon EnterTrue(house.temperature > 30)
When True
```

[2] For the sake of readability, the select clause is described in an OQL-like language, instead of OWL-QL

```
Do Notify (Bob, "House temperature is beyond 30ºC")
Always
```

Without the use of the Controlling service, context-aware application developers would have to write programming code to implement the behaviors described above with no automated support for (i) subscribing to context sources, (ii) receiving context event notifications, (iii) performing context reasoning, (iv) implementing a monitoring function to check conditions and (v) implementing action invocations. Our approach facilitates the development process by reducing development efforts and time. However, application developers using the Controlling service are limited to the expressiveness of our language when describing application behaviors.

5. Infrastructure Architecture and Prototyping Activities

Fig. 3 depicts an example configuration of the infrastructure services.

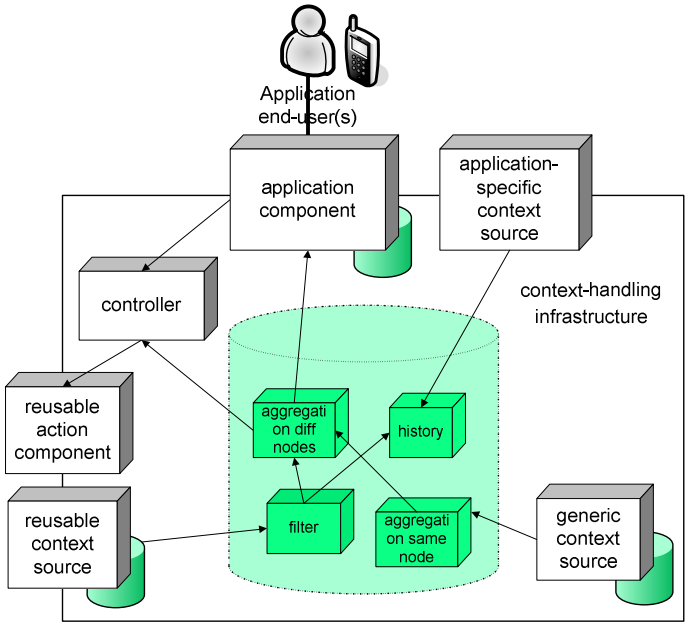

Fig. 3. Example configuration of the context handling infrastructure

Application components may either gather context information directly from the network of context sources or use the service offered by the Controller component. In the latter, applications provide the infrastructure with an application-specific context model and ECA rules, and the Controller makes sure action services and notifications are invoked and delivered appropriately through the reusable action components. Reusable context sources, application-specific context sources and generic context sources are components developed by third-party developers, application developers and the infrastructure, respectively. These components maintain their own specific

extensions of the infrastructure general context model, and therefore their own knowledge bases. These particular knowledge bases are represented by database symbols in Fig. 3. The other context processing nodes are dynamically created, according to the layers of reasoning that are required to execute an ECA Rule.

We have prototyped a previous version of this architecture using Web Services technologies and the Java programming language. This prototype has been experimented in various scenarios in the tourism domain [3][11]. It includes a limited version of the Controller and a number of location-based Reusable Context Sources (GPS sensors). The Controller interface is offered as a web service end-point, allowing the operations to be remotely called by the application components. Application components have also been implemented as a web service end-point to allow callbacks from the Controller. We have defined an XML Schema that represents a limited version of the ECA rule language such that application ECA rules can be written as XML documents and validated using this WSL XML Schema. Our ECA rule parser reads application rules in XML format and maps them into Java classes, which are automatically compiled and executed at runtime. We have used an object-based context model and context reasoning is based on hard-coded algorithms. For service discovery, we have used the industry standard UDDI.

Our current prototyping activities include developing a Controller component using the Jess Rule Engine [9]. We are working on the mapping between the ECA language constructs and Jess constructs. In general, a simple ECA rule needs to be mapped to a set of fact assertions and rule definitions in Jess. The mapping of the Scope clause onto Jess rules is particularly complex, since the scope clause requires runtime rule definitions and deletions for each entity entering and leaving the scope, respectively.

Furthermore, we are also working on the runtime creation and distribution of context processing nodes. A newly defined ECA rule may require context reasoning activities that do not exist at the time the rule has been included. When this occurs, the Controller needs to create context processing nodes that are capable of performing the pieces of context reasoning that are required. In order to create such node, the Controller provides (i) the reasoning algorithm itself (derived from the context model); (ii) the type of context input(s) and where this information can be gathered from, and (iii) the type of the expected outputs.

Context processing nodes only exist when they are needed (limited lifetime). There are two types of context processing nodes, namely the stateless and stateful context processing nodes. Stateless context processing nodes perform a piece of reasoning that does not require persistence of context information, while stateful context processing nodes typically maintain history information. Once context processing nodes are running properly, the Controller component is regularly fed with (high-level) context information provided by both pre-defined context sources and dynamically defined context processing nodes.

6. Related Work

Various approaches to address ECA services for context-aware systems have been proposed. The ECA rule matching service [6] proposes an extension to the standard CORBA Notification services with a composite Event Matching Engine, using CLIPS to implement event correlation and the aggregation process. This approach does not address context information representation, therefore being limited to a predefined set of context types. In addition, there is no support for context information management activities, such as gathering, processing and distribution.

The work presented in [12] discusses a structured framework to design and implement context-processing modules. This work uses ontologies for context information specification and composition. However, as opposed to our approach, no context categorization is provided. Application developers using this framework need to (i) provide a description of context in RDF/XML format and (ii) find and/or implement context processing components. Our approach differs from this work since we take a top-down strategy to perform configuration of context processing modules, which is based on ECA rule descriptions. Application developers provide ECA rules and context models as input, and the Controlling service takes care of dynamically finding and/or implementing context processing components that are required.

The framework presented in [1] proposes a rule-based sentient object model to facilitate context-aware development in an ad-hoc environment. The main functionality is offered in a tool that facilitates the development process by offering graphical means to specify context aggregation services and rules. Although this approach introduces useful ideas on how to easily configure rules and aggregation services on a sentient object, it is based upon a simple model of context that is both informal and lacks expressive power.

The Context Management framework presented in [7] defines a framework and a tool for facilitating end-user customization of context-aware features. This work concentrates on a tool that allows users to combine context and actions in order to define ECA rules. The context and action options provided by the tool reflect the concepts defined in a context ontology. Differently from our approach, there is no use of discovery mechanism to find and match context sources and action providers. Furthermore, there is no strong support for application rules (as opposed to end-user rules) and parameterization of rules. The mechanism presented in this paper focuses on application rules that may be applied to a collection of users. Therefore, as shown in this paper, the use of parameterization through the *scope clause* is important. However, although we present an alternative mechanism to gather context, process rules and invoke actions, our approach would benefit from a user friendly tool such as the one presented in [7].

7. Conclusions

We have discussed in this paper our current efforts towards a flexible context handling infrastructure. A central element of this infrastructure is the Controller component, which takes application-specific rules and context models as input in

order to carry out runtime application-specific adaptation within the infrastructure. We have discussed (i) important aspects on context modeling, (ii) a mechanism to define ECA rules, and (iii) a general overview of the infrastructure services. As opposed to various related works [4][10][12], our proposal is based on a top-down approach towards automatic configuration of ECA rules. Based on ECA rules and models of context, the Controller is capable of autonomously configuring the infrastructure accordingly. This approach facilitates the development process, since application developers do not need to write programming code to (i) activate rules; (ii) find and compose context sources; (iii) implement context reasoning activities; and (iv) invoke actions.

We have discussed a generic context model that can be specialized with domain and application specific concepts. This creates application-specific virtual knowledge bases, permitting specific requirements to be addressed in a general Controller architecture. In addition, our context categorization allows us to define context information recursively, which conforms to the recursive nature of context information. For example, it is possible to define that a device is part of a person's context, and that communication channels are part of the device's context, and so forth. Our context model allows us to traverse from an entity to the entities that are related to this entity by the relational context. This has enabled us to recursively define the relationship between entity and context, facilitating navigation in the model.

We have defined an ECA language to specify rules that are used by the Controller to create and compose context processing nodes. This language allows us to specify ECA rules that consider a scope in which these rules should be applied, as opposed to cumbersomely defining an individual rule for each entity instance.

By using the Event-Control-Action pattern we have decoupled context concerns from action concerns, under the control of application-specific rules, enabling effective distribution of responsibilities among various parties within the infrastructure. This approach has greatly improved extensibility and flexibility of the infrastructure's generic functionality, since rules, actions and context information can be added on demand, at infrastructure runtime.

As part of our ongoing research, we are developing additional context-aware applications with the support of the infrastructure's Controlling service. We are also investigating effective approaches to distribute Controller components, while tackling synchronization of rules and potentially conflicting issues.

References

[1] Biegel, G., and Cahill, V.: A Framework for Developing Mobile, Context-Aware Applications. In: Proc. of the 2nd IEEE Conference on Pervasive Computing and Communications (Percom2004). USA (2004) 361-365.

[2] Dockhorn Costa, P., Ferreira Pires, L., van Sinderen, M.: Architectural Patterns for Context-Aware Services Platforms. In: Proc. of the Second International Workshop on Ubiquitous Computing (IWUC 2005 at ICEIS 2005). USA (2005) 3-19.

[3] Dockhorn Costa, P., Ferreira Pires, L., van Sinderen, M.: Designing a Configurable Services Platform for Mobile Context-Aware Applications. In: International Journal of Pervasive Computing and Communications (JPCC), 2005, Troubador Publishing.
[4] Henricksen, K., and Indulska, I.: A Software Engineering Framework for Context-Aware Pervasive Computing. In: Proc. of the 2nd IEEE Conference on Pervasive Computing and Communications (Percom2004). USA (2004) 77-86.
[5] Henricksen, K.: A Framework for Context-Aware Pervasive Computing Applications. PhD thesis, School of Information Technology and Electrical Engineering, The University of Queensland (2003).
[6] Ipina, D., and Katsiri, E.: An ECA Rule-Matching Service for Simpler Development of Reactive Applications. In: Proc. of Middleware 2001 at IEEE Distributed Systems Online, Vol. 2, No. 7, November 2001.
[7] Korpipää, P., Malm, E., Salminen I., Rantakokko, T.: Context Management for End User Development of Context-Aware Applications. In: Proc. of the 6th International Conference on Mobile Data Management. Cyprus (2005) 304-308.
[8] McGuinness, D., and van Harmelen, F.: OWL Web Ontology Language – Overview, W3C Recommendation (2004). Available at http://www.w3.org/TR/owl-features/.
[9] Jess – the Rule Engine for the Java Platform. Available at herzberg.ca.sandia.gov/jess/
[10] Przybilski, M.: Distributed Context Reasoning for Proactive Systems. In: Floreen, P., et al. (eds.): Proc. of the Workshop on Context Awareness for Proactive Systems (CAPS 2005). Finland (2005) 43-54.
[11] Pokraev, S., Koolwaaij, J., van Setten, M., Broens T., Dockhorn Costa, P., Wibbels, M., Ebben, P., Strating, P.: Service Platform for Rapid Development and Deployment of Context-Aware, Mobile Applications. In: Proc. of International Conference on Webservices (ICWS'05). USA (2005).
[12] Sbodio, M. and Thronicke, W.: Specification and Design of Framework-Based Context Processing Modules. In: Floreen, P., et al. (eds.): Proc. of the Workshop on Context Awareness for Proactive Systems (CAPS 2005). Finland (2005) 79-92.