UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

POCAp: a Software Process for Ontology-Based Context-Aware Applications

Renato de Freitas Bulcão Neto
Taciana Novo Kudo
Maria da Graça Campos Pimentel

Nº 273

RELATÓRIOS TÉCNICOS

São Carlos – SP
Abr./2006

POCAp: a software process for ontology-based context-aware applications

Renato de Freitas Bulcão Neto[1], **Taciana Novo Kudo**[1]
Maria da Graça Campos Pimentel[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brazil

Abstract. *While research in context-aware computing has produced useful results in the form of devices, software infrastructures and applications, there has been a lack of research investigating software engineering issues for context-aware computing. This paper presents the POCAp process, a novel process for the development of context-aware applications that make use of ontological context information. POCAp addresses the need for software engineering solutions regarding the development of ontology-based context-aware applications.*

1. Introduction

Weiser pointed out the need for researches to focus on the development of applications [Weiser 1993]. As one of the building blocks of ubiquitous computing, context-aware computing investigates problems related to supporting applications in acting autonomously on behalf of users based on context information sensed from instrumented environments [Abowd and Mynatt 2000].

Abowd points out that the context-awareness feature of ubiquitous computing requires investigation into software engineering to build general support for the development of context-aware applications [Abowd 1999]. His observations include a demand for the design of architectures based on the separation of concerns with respect to infrastructures and applications. Similarly, it has been reported the demand for software engineering techniques addressing the inherent complexity and time-consuming characteristics of developing context-aware systems [Helal 2005] [Kortuem et al. 2006].

Generic approaches to context modeling have been demanded in order to capture various features of context including the many types of context information [Abowd et al. 2002] and context histories [Henricksen et al. 2002]. In this scenario, ontologies [Gruber 1993] play a key role as context modeling technique mainly due to their formality, expressiveness, portability, implementation of abstraction capability, as well as for their intrinsic facilities for enabling systems to reason about context.

In the past few years, ontology-based context models have been used to allow seamless cooperation among heterogeneous devices [Chen et al. 2004b] as well as context acquisition, discovery, reasoning and distribution [Chen et al. 2004a] [Gu et al. 2005] [Ranganathan et al. 2004] [Tan et al. 2005]. However, those efforts have not been tackled considering software engineering techniques concerns.

In this paper, we propose a novel software process oriented to support ontology-based context-aware applications. The POCAp software process (*Process for Ontological*

Context-aware Applications) assumes that such applications are built using a *context service infrastructure* capable of processing context information, and that the semantics of context information is provided by an *ontology-based context model.*

This paper details the implications of each POCAp activity in the lifecycle of ontology-based context-aware applications from the analysis and specification stage to the verification and validation stage. We observe that the development of applications using POCAp must take into account, among others: (a) the expressiveness of the ontology language underlying a context model; (b) the configurability level of the services provided by a semantic-enhanced software infrastructure; and (c) the capabilities of the reasoners provided by an inference making service. Using the POCAp software process, we address the need for software engineering solutions that make the development of ubiquitous computing applications more predictable, specially of ontology-based context-aware applications.

The remaining of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the POCAp process to supporting the development of ontology-based context-aware applications. Section 4 shows an instantiation of the POCAp process. Section 5 presents our concluding remarks and future work.

2. Related work

This section presents software engineering methods, processes and tools that specifically address the challenges of producing ubiquitous computing systems.

Anind et al. identify a design process for building sensor-based context-aware applications [Dey et al. 2001]. However, their design process has not been formalized with respect to process modeling and notation. For instance, the artifacts produced by each process phase as well as the possible iterations between the process phases are not documented. On the other hand, their process discusses low-level aspects which are not covered by our POCAp process.

In order to make it easier to describe context information, it was proposed a UML-based notation for context specification to be integrated into the project of interactive systems [Van den Bergh and Coninx 2005]. Their work focuses on the analysis and specification activities, while POCAp also includes support with respect to design, development and verification & validation.

Weis et al. present a process that supports the development of customizable ubiquitous applications, which combine component-based software, visual programming, and self-configuration [Weis et al. 2006]. In comparison to POCAp, Weis' process primarily focuses on the development stage and does not take into account the importance of the analysis and specification activity for the development of context-aware applications.

Overall, the literature reports a demand for the investigation of software engineering techniques to support the development of context-aware systems [Abowd 1999] [Helal 2005] [Kortuem et al. 2006].

Using a formal specification, we have modeled POCAp to frame and organize the different factors and issues related to the development activities of context-aware applications. Compared with related work, our software process copes with more stages, from analysis and specification to verification & validation.

3. The POCAp software process

This section describes the POCAp software process in two perspectives: first, we outline the process as a set of coherent activities (or stages); second, we provide further details about how those activities have to be carried out.

3.1. An overview of POCAp

In syntony with many software engineering researchers, Fuggetta states that the activities of a software process typically include: analysis and specification, design, development, verification and validation, deployment, operation, maintenance, and retirement [Fuggetta 2000].

The POCAp process encompasses the first four activities proposed by Fuggetta: from analysis and specification to verification and validation. We do not tackle the four remaining activities given their dependence on the organization where the system is to be deployed.

In general, each POCAp activity is intended to achieve the same objectives with respect to a traditional software process stage[1]:

(1) Analysis and specification is the activity of establishing what a context-aware system should do (functional requirements), the constraints on the system's operation and development (non-functional requirements), and the set of context information needed;

(2) Design is the activity of producing a context-aware software structure (e.g. architectural design) that realizes the system specification;

(3) Development is the activity of translating the context-aware software structure into an executable program;

(4) Verification and validation is the activity of checking, reviewing and testing whether a context-aware system conforms to its specification and meets the requirements of its customer.

In order to model the POCAp process, we have used the Software Process Engineering Metamodel (SPEM) formal specification [OMG 2006], which provides a UML-based framework capable of modeling software development processes.

Figure 1 presents the POCAp process with focus on its main four activities[2]: (1) analysis and specification, (2) design, (3) development, and (4)[3] verification and validation.

As shown in Figure 1, the POCAp process aims at supporting four different actors' roles during the development of a context-aware system: analysts, designers, developers and validators. Analysts are guided throughout the analysis and specification activity to produce artifacts describing both customer's and system's demands. Designers use the requirements artifacts to plan the system implementation activity. Developers use the design artifacts to understand what system is to be developed. Finally, validators or

[1] We have borrowed Sommerville's definitions for each POCAp activity [Sommerville 2004].

[2] The legend in the figure indicates that the arrow-shaped symbol stands for an activity. The other symbols indicate initial and final stages, decision stage and flow control.

[3] We use numbered activity labels in the figures and in the text, as in "(4)", to facilitate the presentation of the process.

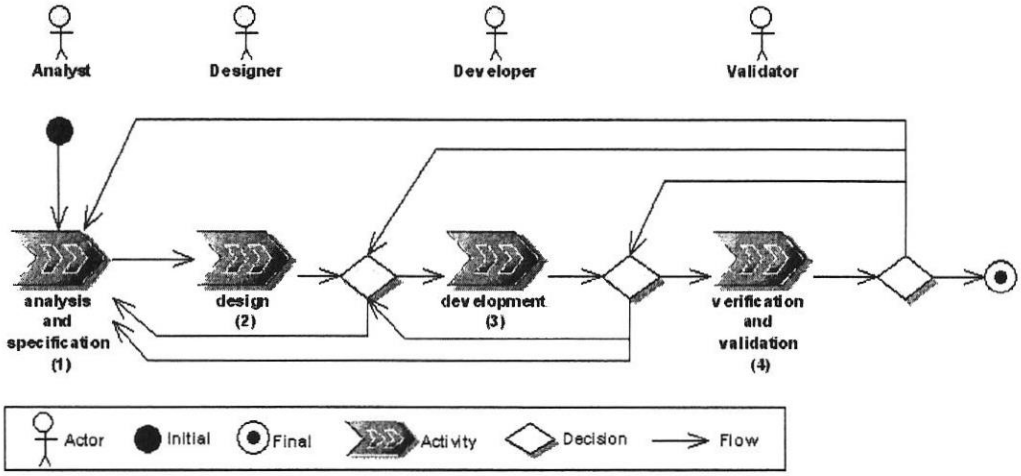

Figure 1. The POCAp software process.

system test engineers use the requirements document and the system implementation to perform validation tests for a context-aware system.

Using the SPEM notation, we have represented all activities as well as all connections and possible iterations between POCAp's internal activities. The arrows shown in Figure 1 indicate that the iterations coming after each activity (except the analysis and specification stage) indicate that it is possible to change aspects of a context-aware system in response to changing demands, such as new services and types of context information, and software optimizations.

In order to simplify the upper-level view of the POCAp process, we omit the input and output artifacts generated from each activity in Figure 1. On the other hand, those artifacts are represented in the next sections, where we detail all POCAp activities.

3.2. The (1) analysis and specification activity

Figure 2 depicts the POCAp (1) analysis and specification activity, which is composed of four subactivities: (1.1) requirements analysis and specification, (1.2) analysis and specification of context information, (1.3) analysis and specification of model reuse and (1.4) analysis and specification of model extension.

Figure 2 also illustrates which documents are produced as *output* of some activities, and which documents are demanded as *input* to activities.

An important element in this (1) analysis and specification activity is the ontology-based context model used as input to two subactivities: (1.3) analysis and specification of model reuse and (1.4) analysis and specification of model extension. A high level ontology-based context model, produced by an ontology engineer, should provide a vocabulary of terms used by most context-aware applications. In the subactivities of this stage, a model specific to the application being designed is produced (system ontology model) along with the definition of the elements reused from the high level ontology (model reuse document).

According to POCAp, the four subactivities of the (1) analysis and specification activity are as follows:

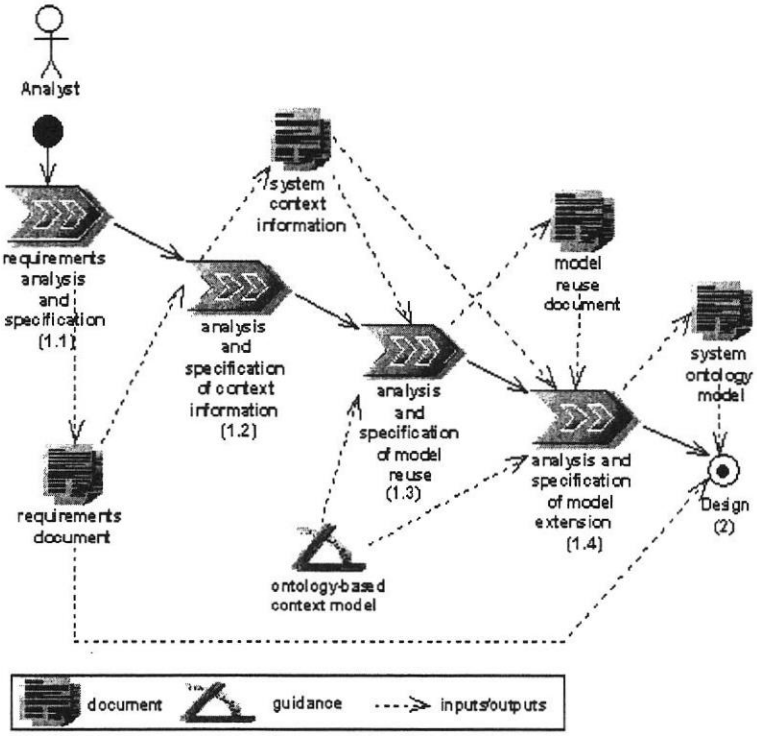

Figure 2. The POCAp (1) analysis and specification activity.

(1.1) Requirements analysis and specification is the activity in which an analyst should collect the concepts of user's and system's requirements in the form of functional and non-functional requirements. As *output*, it is generated a software requirements document, whose structure may follow a standard such as the IEEE/ANSI 830-1998 [ANSI/IEEE 1998]. This requirements document is *input* artifact to the subactivity (1.2) and to the whole (2) Design activity.

(1.2) Analysis and specification of context information is the activity in which an analyst should identify which information is relevant to the system. Such relevance characterizes what we call context information, which can be obtained from the functional requirements specified in the requirements document. The *output* artifact from this activity is a document called system context information.

(1.3) Analysis and specification of model reuse is the activity that aims to delimit the reuse of an existing ontology-based context model, which plays the role of a *guidance* for context modeling. The *input* artifacts for this activity are not only the ontology-based context model, but also the document containing the system context information. An analyst should map the system context information to ontological terminology such as concepts, relations and axioms. In this case, it is recommended that the analyst collaborate with an ontology engineer, who must follow an ontology development methodology such as the one proposed in [Noy and McGuinness 2001]. Once identified those ontological terms, the analyst should match them to the concepts, relations, and axioms of the ontology-based context model. This allows the identification of terms that the system in question can reuse from the context model underlying the ontology-based context model. As a result, the analyst produces the model reuse document.

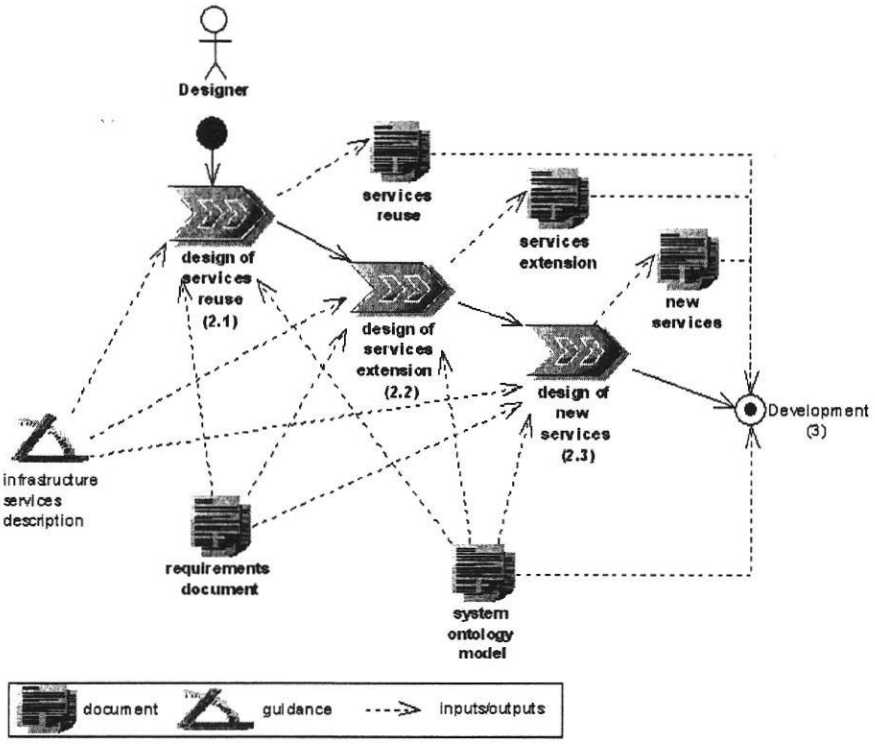

Figure 3. The POCAp (2) design activity.

(1.4) Analysis and specification of model extension is the activity in which the analyst identifies system context information that is not modeled by the existing ontology-based context model. During this activity, the analyst should specify, with support from an ontology engineer, the ontological terms that will comprise an extension to the existing context model to be formalized in the system ontology model *output* document. An important issue the analyst must take into account during the specification of new ontological terms is the expressiveness of the ontology language underlying the ontology-based context model. As depicted in Figure 2, in order to generate the system ontology model, the analyst should use the following input artifacts: the ontology-based context model, the system context information document, and the model reuse document. The resulting system ontology model document is used as input for the design activity.

3.3. The (2) design activity

Figure 3 illustrates the POCAp (2) design activity, which comprises three subactivities: (2.1) design of services reuse, (2.2) design of services extension, and (2.3) design of new services. These subactivities use three input artifacts in common as follows: the infrastructure services description document, which plays the role of a *guidance* for context processing purposes, the requirements document (output generated by activity 1.1 in Figure 2), and the system ontology model (output generated by activity 1.4 in Figure 2).

It is important to observe that the input *guidance* artifact infrastructure services description gives detailed information with respect to a general configurable infrastructure which, on the one hand, is capable of interpreting the semantics of ontology-based context

models and, on the other hand, may provide common services demanded by context-aware applications such as storage, retrieval and reasoning about context information. The subactivities of the (2) Design activity are as follows:

(2.1) Design of services reuse is the activity in which the designer should delimit the reuse of an existing service infrastructure, which is capable of interpreting the semantics of ontology-based context models. Firstly, a designer should match the functional and non-functional requirements (from the requirements document) with the infrastructure services description document to identify the set of services that can be reused. Secondly, a designer needs to verify if those services can handle the ontological terms defined in the system ontology model specific for the application being designed. As a result, it is generated the services reuse document, with may be used as input for the (3) Development activity.

(2.2) Design of services extension is the activity that aims to identify which services operations provided by the existing service infrastructure do not meet the functional and non-functional requirements of the application being designed, as stated by the input requirements document. A designer should then match the system requirements with the service infrastructure description to define the services to be extended. The configurability level of the service infrastructure is an important issue that the designer should take into account — the designer should specify those new services to handle the ontological terms defined in the system ontology model. The outcome of this activity is the services extension document, which may be used as input for the (3) Development activity.

(2.3) Design of new services is the activity in which a designer should point out which services are not provided by the existing services infrastructure (according to the infrastructure services description) to meet the current functional and non-functional requirements (according to the requirements document). As in the previous activity (2.2), a designer should also specify the new demanded services to exploit the semantics of the system ontology-based model. As a result, a document containing the specification of those demanded services is produced, called new services document, which may be also used for the (3) Development activity.

3.4. The (3) development activity

In order to guide developers' tasks, the POCAp development activity gets the following input artifacts: the system ontology model, generated by activity 1.4 (Figure 2), and the services-related documents – services reuse, services extension and new services – generated by activities 2.1, 2.2 and 2.3 (Figure 3). The output of this activity is the context-aware system implementation.

Based on the system ontology model, the developer should choose the appropriate computational support to processing the underlying ontology language. For instance, the Ontology Web Language (OWL) [Bechhofer et al. 2004] is well-supported by the majority of the ontology-enabled toolkits, APIs and frameworks.

According to the services-related input artifacts, developers should take into account the most well-suited computational languages, techniques and tools that meet the design needs of each service to be reused, extended, and implemented from the scratch. For instance, in terms of processing ontology-based context information, three services

must be carefully observed: the storage service, the query service, and the inference service.

A developer should opt for a storage service that provides efficient mechanisms for storage and retrieval of ontology-based context information.

When developing a query service, developers should choose query languages on ontology-based information sources that are compatible with the ontology language underlying the system ontology model. In addition, the query language must also meet the expressiveness required by every query specified in the design activity.

When developing an inference service, developers must identify the appropriate inference engines (or reasoners) that accomplish the design needs specified in the services-related documents and in the system ontology model. For instance, there are reasoners that support multiple types of reasoning as well as different ontologies and rules languages with various levels of expressiveness. Developers may also prefer a reasoner for other reasons, such as if it requires few lines of code to use, or it is written in an portable programming language.

It is important to observe that the POCAp software process is neutral with respect to the technology used to support the development of ontology-based context-aware applications.

3.5. The (4) verification and validation activity

The POCAp verification and validation activity involves executing a context-aware system with test cases that are derived from the specification of the real context information to be processed by that system. The input artifacts of this activity are then the requirements document, the system ontology model and the model reuse document generated by the (1) analysis and specification activity, and the context-aware system implementation, generated by the (3) development activity.

Functional requirements are used in this activity for checking if the system meets the corresponding specifications. In the case of ontology-based context-aware systems, validators must be also be concerned with non-functional requirements.

Given that the analyst has extended the ontology-based context model that the POCAp process assumes to generate the system ontology model and the model reuse document, during this activity it is important to evaluate if those specifications are properly cared for in the implementation. This is relevant so as to better accommodate future extensions on the current context-aware system. The support of an ontology engineer may be useful during this activity.

For context-aware environments, interoperability between heterogeneous software systems is one of the challenges to be addressed. Validators should evaluate the suitability of standards towards promoting the interoperability not only between different systems, but also between the services of software infrastructures.

Another important non-functional requirement is the storage space. Validators should evaluate the implementation strategies used to promote the adequate storage of ontology-based context information.

The performance of context-aware systems may be affected depending on the fea-

tures of the inference engines used by developers. Validators should evaluate the reasoning response time for ontology-based context information to meet the performance system's requirements. In addition, reasoning response time may be subdivided into other processes that can be used for checking if certain reasoning operations are taking longer than they are supposed to. This can provide validators with valuable information about how ontology-based reasoning time can be improved [Bulcão Neto and Pimentel 2006].

4. Instantiating POCAp

In this section, we present an instantiation of the POCAp process to illustrate its use. Given that the POCAp software process takes into account that the building of context-aware applications is supported by an ontology-based context model (in activity (1)), and a semantic-enabled service infrastructure described by a infrastructure services description (in activity (2)), we first present the SeCoM context model [Bulcão Neto and Pimentel 2005] and the SCK service infrastructure [Bulcão Neto et al. 2005], which we use as input *guidance* artifacts corresponding to those needs.

4.1. Semantic approach for context modeling and reasoning

This section gives an overview of both the SeCoM context model (detailed in [Bulcão Neto and Pimentel 2005]) and the SCK service infrastructure (detailed in [Bulcão Neto et al. 2005]), which are independent from each other.

The Semantic Context Model (SeCoM)

The SeCoM context model represents the basic concepts of actor, location, time, device, events, and activity (dark ovals in Figure 4), as well as the relationships between these concepts (an arrow indicates when a lower ontology imports concepts from an upper ontology). Figure 4 depicts an overview of the ontological context model [Bulcão Neto and Pimentel 2005]. SeCoM provides a set of generic classes, properties, relations, and restrictions that applications can import and/or extend for particular domains. SeCoM ontologies have been built using the OWL language [Bechhofer et al. 2004], which provides a rich vocabulary for describing classes, properties, relations between classes, (in)equality of concepts, and cardinality restrictions.

The *Actor* ontology models the profile of entities that can perform actions in a pervasive environment such as people, groups and organizations. This ontology is imported by other ontologies that were built to deal with actors' profiles including social role, social relationship, knowledge, contact information, document and project.

The *Spatial* ontology describes the whereabouts of actors. It models virtual and real-world indoor and outdoor places, containment and spatial relations between places, geographic coordinates, among others. Spatial events are those that have spatial extensions and can be classified as virtual and physical events. Such kind of events are modeled in the *Spatial Event* ontology.

The *Time* ontology represents temporal information in terms of temporal instants and intervals. It models relations between instants and intervals, relations between intervals, and calendar and clock information. Events with temporal extensions are modeled in the *Temporal Event* ontology, and can be classified as instant or interval events.

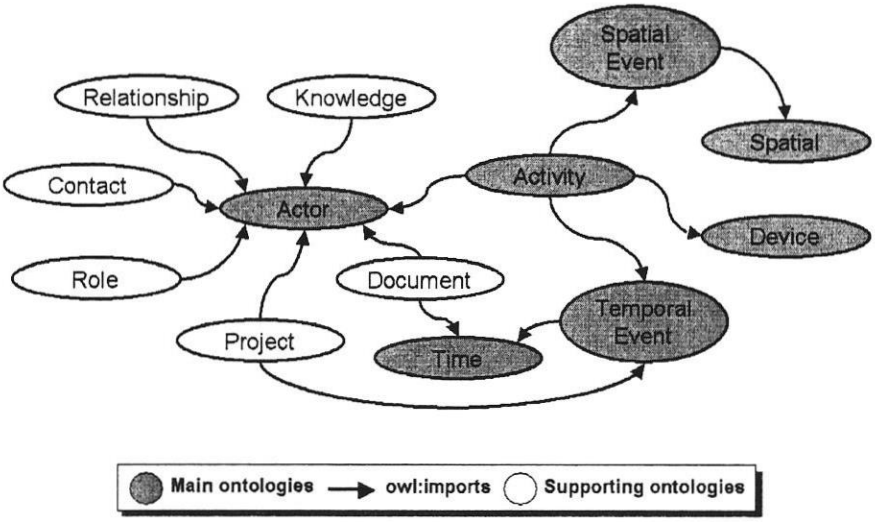

Figure 4. An overview of the Semantic Context Model.

The *Device* ontology describes devices by means of their hardware and software platforms. The former includes information about storage and battery capacity, multimedia support, and network connectivity, whereas the latter allows to describe operating systems and browsers supported, virtual machines installed, among others.

The *Activity* ontology describes actions that actors do or cause to happen. Activity is modeled as relevant spatiotemporal events that characterizes it including the corresponding actors and devices involved in. Hence, it imports the following SeCoM ontologies: *Actor*, *Spatial Event*, *Temporal Event*, and *Device*. Activities are modeled as impromptu (or informal) and scheduled (i.e. planned in terms of time and place).

The SCK architecture

Figure 5 depicts the Semantic Context Kernel architecture [Bulcão Neto et al. 2005]. Context information is provided by *Context Sources*, which may be represented by applications, Web services, and physical sensors. *Context Transducers* convert the information captured from context sources into a common semantic representation: the RDF triple model [Klyne and Carroll 2004], which provides a graph-based data model with nodes (resources) connected by labeled arcs (properties or relations describing resources).

Context Consumers make use of context information stored by context sources so that the former can adapt themselves according to the current situation (e.g. applications). The *Discovery Service* provides context transducers and every service layer with an advertising mechanism so as to allow context consumers to locate services. The *Context Query Service* allows context consumers to query context via declarative languages for RDF models based on simple conjunctive triple patterns. In general, query expressions are represented as a matching of a triple pattern against an input RDF graph.

The *Context Persistence Service* allows developers to choose the type of persistent storage. It allows context sources to store context information in relational databases or on a context file. The *Context Inference Service* provides context consumers with a configurable inference support over context. It allows developers to exploit two types of

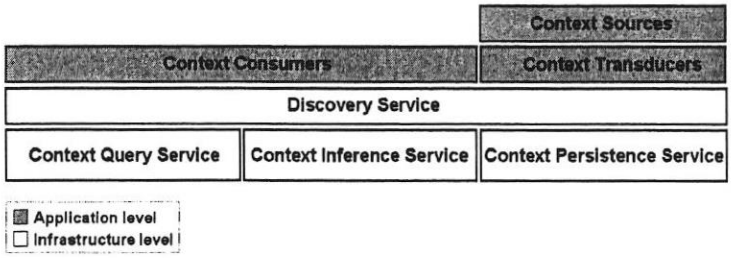

Figure 5. The Semantic Context Kernel architecture.

reasoning which include ontology-based and rules-based reasoning. The former exploits the semantics of ontology languages for the inference making process, whereas the latter uses rules chained in such way that predicates in a rule are represented in the current ontology and instances are asserted in the input RDF graph.

4.2. Instantiating POCAp with SeCoM and SCK

We present an instance for each of the four main activities of POCAp using SeCoM and SCK.

An instance of the analysis and specification activity

As the POCAp activity (1.1) requirements analysis and specification is independent from the use of SeCoM, the analyst should first produce the requirements document. Similarly, during the activity (1.2) analysis and specification of context information, the analyst should generate the system context information document independently from SeCoM.

The first activity dependent on the SeCoM context model is the (1.3) analysis and specification of model reuse. In this case, SeCoM is used as a *guidance* so that the analyst can map and match the system context information document in accordance with the ontological terms provided in the form of actors, devices, location, time, and activities provided by SeCoM (as shown in Figure 4). The analyst then generates the model reuse document, which contains the ontological terms that SeCoM can represent.

During the activity (1.4) analysis and specification of model extension, the terms that the SeCoM model does not represent are modeled by the analyst as an extension of SeCoM. For instance, if a context-aware system demands representing physical attributes of people (e.g. height and weight), the analyst must create a particular ontology for representing such type of context information. The SeCoM model is extended when the new physical attributes ontology imports the *Actor* ontology in which people are already represented, i.e., PERSON is subclass of ACTOR.

An instance of the design activity

Based on both the requirements document (generated by activity (1.1)) and the system ontology model (generated by activity (1.4)), the designer should match the system's demands in terms of storage, query, and reasoning about ontology-based context information with the corresponding SCK services. The configurability characteristic of SCK aims to accommodate a broad variety of storage, query and reasoning requirements.

When the operations available in existing SCK services do not meet system's de-

mands, the services must be extended to implement the required operation — this demands the designer to specify the new operations according to the extension mechanism supported by SCK.

If a context-aware system requires a service not provided by SCK, the designer should then specify the structure of that service to be implemented. If the new service needs to handle the semantics of context information, it has to exploit the semantics provided by the system ontology model.

An instance of the development activity
In this activity, developers can benefit from the use of the SeCoM context model and the SCK service infrastructure.

Regarding SeCoM, all knowledge related to a context-aware system is apart from the system logic. This makes it easier the maintainability, the portability and the evolution of the system code. Once SeCoM is based on the OWL ontology language, it leverages the ability of a context-aware system regarding context reasoning.

The set of services provided by SCK also facilitates the development of ontology-based context-aware systems due to the following reasons: (a) such systems do not need to implement the whole context storage, query and reasoning infrastructure; (b) SCK provides a uniform format for context information interchange (the RDF triple model); and (c) systems can take advantage of the configurability aspect of the SCK services.

An instance of the verification and validation activity
As previously observed, validators should generally be concerned about non-functional requirements during this activity.

Regarding the expressiveness of context models, some SeCoM ontologies have the higher level of expressiveness that an OWL-based ontology can achieve, for instance, the *Activity* and the *Spatial* ontologies. This has implications with respect to reasoning time complexity, which is in turn a concern related to systems performance.

SeCoM can be considered as a viable context model due to its two-layer architecture by means of which SeCoM provides a set of general knowledge so that context-aware systems can reuse and/or extend it for their own purposes. This has been proved by measurements with respect to the time of importing each SeCoM ontology, which is less than 2% of the total reasoning response time, on average [Bulcão Neto and Pimentel 2006].

Regarding non-functional requirements, we have mostly concerned about how much time the SCK inference service takes for context reasoning. Validators should take into account the reasoning capabilities and the range of services provided by each inference engine supported by the SCK inference service. The current SCK implementation provides full reasoning support about OWL-encoded context information as well as the ability of measure different timing information with respect to context reasoning [Bulcão Neto and Pimentel 2006].

5. Concluding Remarks

In order to address the increasing need for engineering techniques applied for developing ubiquitous computing systems, this paper proposed a software process called POCAp,

which comprises a structured set of activities for specifying, designing, implementing and testing ontology-based context-aware systems, in particular.

For every POCAp activity, we defined the concerns a system development team should tackle with assuming the support of an ontology-based context model and a semantic-enabled service infrastructure.

In order to clarify the POCAp process, we have shown its instantiation using a ontology-based context model and a context-based service infrastructure.

The POCAp process has been refined towards accommodating aspects related not only to ontology-based context-aware systems but also related to context-aware systems in general.

Although we are currently using the POCAp process in the development of a real-world context-aware system, we understand that its evolution demands its use by independent developers. In order to both evaluate and refine the POCAp process, we have planned an experiment with a group of developers who will exploit common scenarios of context-aware computing.

We also envision developing computational tools for supporting developers during every POCAp activity.

Acknowledgments

Renato Bulcão Neto is a PhD candidate supported by the FAPEMA funding agency (n.03/345). Maria da Graça Pimentel is supported by FAPESP and CNPq. Taciana Kudo is also supported by FAPESP (n.05/53058-0).

References

Abowd, G. D. (1999). Software engineering issues for ubiquitous computing. In *Proceedings of the International Conference on Software Engineering (ICSE'99)*, pages 75–84, Los Angeles, CA, USA.

Abowd, G. D. and Mynatt, E. D. (2000). Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58.

Abowd, G. D., Mynatt, E. D., and Rodden, T. (2002). The human experience. *IEEE Pervasive Computing*, 1(1):48–57.

ANSI/IEEE (1998). IEEE recommended practice for software requirements specifications.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). Web Ontology Language (OWL), W3C recommendation. `http://www.w3.org/TR/owl-ref/`. Visited on 10/03/2006.

Bulcão Neto, R. F. and Pimentel, M. G. C. (2005). Toward a domain-independent semantic model for context-aware computing. In *Proceedings of the 3rd Latin American Web Congress (LA-Web'05)*, pages 61–70. IEEE Computer Society.

Bulcão Neto, R. F. and Pimentel, M. G. C. (2006). Performance evaluation of ubiquitous inference services: reasoning-related issues. Technical Report 272, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo. 16p.

Bulcão Neto, R. F., Teixeira, C. A. C., and Pimentel, M. G. C. (2005). A Semantic Web-based infrastructure for supporting context-aware applications. In *Proceedings of the IFIP International Conference on Embedded and Ubiquitous Computing (EUC'05)*, volume 3824 of *Lecture Notes on Computer Science*, pages 900–909.

Chen, H., Finin, T., Joshi, A., Perich, F., Chakraborty, D., and Kagal, L. (2004a). Intelligent agents meet the Semantic Web in smart spaces. *IEEE Internet Computing*, 8(6):69–79.

Chen, W., Wang, C.-L., and Lau, F. C. M. (2004b). A collaborative and semantic data management framework for ubiquitous computing environment. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing (EUC'04)*, pages 962–971, Aizu-Wakamatsu, Japan.

Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction Journal*, 16(2-4):97–166.

Fuggetta, A. (2000). Software process: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE'00)*, pages 25–34, Limerick, Ireland.

Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220.

Gu, T., Pung, H. K., and Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18.

Helal, S. (2005). Programming pervasive spaces. *IEEE Pervasive Computing*, 4(1):84–87.

Henricksen, K., Indulska, J., and Rakotonirainy, A. (2002). Modeling context information in pervasive computing systems. In *Proceedings of the International Conference on Pervasive Computing (Pervasive'02)*, pages 169–180, Zurich, Switzerland.

Klyne, G. and Carroll, J. J. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C recommendation. `http://www.w3.org/TR/rdf-concepts/`. Visited on 10/03/2006.

Kortuem, G., Garcia, A., Marshall, I., Mascolo, C., Morrison, R., and Sloman, M. (2006). Workshop on Software Engineering Challenges for Ubiquitous Computing. `http://ubicomp.lancs.ac.uk/workshops/seuc2006/`. Visited on 24/03/2006.

Noy, N. F. and McGuinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology. TR KSL-01-05, Stanford Knowledge Systems Laboratory.

OMG (2006). Software Process Engineering Metamodel, version 1.1. `http://www.omg.org/technology/documents/formal/spem.htm`. Visited on 23/03/2006.

Ranganathan, A., Al-Muhtadi, J., and Campbell, R. H. (2004). Reasoning about uncertain contexts in pervasive computing environments. *IEEE Pervasive Computing*, 3(2):62–70.

Sommerville, I. (2004). *Software Engineering*. Addison Wesley, 7th edition.

Tan, J. G., Zhang, D., Wang, X., and Cheng, H. S. (2005). Enhancing semantic spaces with event-driven context interpretation. In *Proceedings of the Third International Conference on Pervasive Computing*, pages 80–97.

Van den Bergh, J. and Coninx, K. (2005). Towards integrated design of context-sensitive interactive systems. In *Proceedings of the Workshop on Context Modelling and Reasoning (CoMoRea'05) held in conjunction with the 3rd IEEE Conference on Pervasive Computing and Communications (PerCom'05)*, pages 30–34, Kauai Island, Hawaii, USA.

Weis, T., Handte, M., Knoll, M., and Becker, C. (2006). Customizable pervasive applications. In *Proceedings of 4th IEEE Conference on Pervasive Computing and Communications (PerCom'06)*, pages 239–244, Pisa, Italy.

Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84.