

# Promoting efficiency and separation of concerns through a hybrid model based on ontologies for context-aware computing\*

Ricardo C. A. da Rocha<sup>†</sup> and Marco A. Casanova and Markus Endler  
Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio  
Laboratory for Advanced Collaboration  
Department of Informatics, Rio de Janeiro, Brazil  
{rcarocha,casanova,endler}@inf.puc-rio.br

## Abstract

*Several projects in context-aware computing have adopted ontologies for modeling context information, due to its rich constructions for modeling high-dependent concepts and its mechanisms for describing complex inferences. However, ontologies introduce scalability and performance drawbacks on context-aware systems, which usually process high-volume and distributed information. To avoid such disadvantage, some research efforts have proposed the adoption of hybrid context models. This paper presents a hybrid context model for the MoCA, a middleware for developing and deploying context-aware collaborative applications for mobile users.*

## 1. Introduction

Context-awareness is the main programming paradigm for ubiquitous and mobile applications. It enables applications to dynamically adapt according to changes in the state of their environment, i.e. their context. The context-awareness paradigm has stimulated the development of several application prototypes, middleware systems and sensor technologies. However, the initial promise of leveraging the development of a huge array of context-aware services and applications has not yet been achieved. Unfortunately nowadays we can find only a few such services on the market.

One of the main problems to be solved is the ability to specify context models capable of describing precisely the physical environment in which the system is part of. Such models are important because they define how the data sensed/collected in the environment is to be interpreted and

processed by the applications and the context-provisioning infra-structure.

Several models have been proposed in the past, e.g. pair-value models, object-oriented model, logic-based models [10], but all of them turned out to be insufficient for fully describing the environment's complexity in terms of the inter-relationships, potential of ambiguities, degree of precision, lack of accuracy of context information.

Since the last few years, some researchers have used ontologies for modeling context (e.g. [1]). Although ontologies offer huge expressiveness and powerful inference tools, they also impose restrictions on its scalability and efficient implementation. In order to circumvent these problems, more recently researchers have proposed hybrid context models based on ontologies [5, 11], with the goal of taking advantage of both the expressiveness and efficient implementations, specially for resource-constrained devices and networks.

In this article we present a hybrid context model for our middleware architecture named MoCA [9], its purpose, use and its limitations. The article is organized as follows: in section 2 we discuss the use of ontologies for modeling context in ubiquitous computing and its point to their main limitations. Then, section 3 gives an overview of the middleware MoCA and its underlying context model. In section 4 we describe our approach of a hybrid context model for the middleware MoCA, and how the main concepts are mapped to the proposed ontology, as well its benefits and drawbacks. Finally, in section 5 we discuss the current stage of our research and the next steps.

## 2. Ontologies and Hybrid Modeling Approaches in Context-aware Systems

Several context-aware systems have adopted ontologies as the approach to model context information. CoBrA [1] is a well-known example of such context-aware systems,

---

\*This work is being funded by CNPq Research Grants nos. 55.2068/02-2 (ESSMA) e 479824/04-5 (Ed. Universal)

<sup>†</sup>Supported by CNPq

which adopted SOUPA [2] and CoBrA-ONTO [1] ontologies to model smart room context-aware applications. Other systems, such as ACAI [7] and GAIA [8], have shown the power of ontologies to model context information and to separate context information description from inference rules.

However, the adoption of ontology-based modeling has an important drawback: it limitates the efficiency and scalability of dissemination, management and processing of context information [11, 12]. This problem is more critical when the system is managing context information that is highly variable and that demands frequent inferences. Moreover, ontologies do not offer a suitable paradigm para accessing contextual information. For example, ontologies do not allow directly the modeling of contextual changes and adaptations, which are natural abstractions for developing context-aware applications.

In order cope with such limitations, recently some hybrid approaches for context modeling have been developed (e.g. [5, 11]), integrating ontology based modeling with other modeling and processing mechanisms. In this sense, hybrid approaches aim at taking advantage of the strengths of each integrated modeling approach.

### 3. Context Modeling and Processing in MoCA Middleware

MoCA [9] is a service-based middleware for the development of context-aware collaborative applications. In MoCA architecture, we have modeled two base-level context information: a *local context* information, which describes the execution context for a device, including information such as battery level, memory and CPU usage, and the *wireless connectivity* context information about reachable IEEE 802.11 access points and the corresponding signal strengths. The latter context information has been used to infer device's location context, in terms of symbolic locations. (e.g. *Room A, Building B*).

Recently, we proposed an extension of original MoCA's context model [9] to promote extensibility to context modeling and to allow the modeling concepts of relationship, dependency, and association among context information, as well to work efficiently when handling dynamic and static context. In addition, we have introduced a simple approach for modeling quality of context.

In this context model, the base element is the context attribute, which encapsulates either a numeric value or an association to another context information. The user models the runtime behavior of a context information such as: if the information is static or dynamic, how the information accuracy decreases through the time, specific information for helping the deployment of the context type on context service infra-structure, which context provider is respon-

sible for publishing the context information, and the domain in the distributed environment in which the context applies. An user describes these context information properties on a MoCA-specific XML document. To deploy a new context on MoCA infra-structure, the users processes the XML document using a tool called CT (*Context Tool*), which validates the semantic of the XML and the current state of the context type system. In addition, this tool generates a library for accessing the deploying context on a context-aware application, using object-oriented constructions. Both library and context service use the runtime properties of the context information to guarantee a better performance and scalability in the context processing [3].

Each context attribute can be associated with a meta-information of quality-of-context, described in terms of *precision* and *accuracy*. *Context precision* specifies a range of values associated with a context attribute value. For example, for a location information retrieved from a GPS sensor, the precision is usually a constant value, e.g. approximately five meters. *Accuracy* specifies a numerical value to estimate how correct a context data is.

Contextual events represent abstractions of environmental situations and conditions that a context-aware system may be interested in. A contextual event is specified in terms of context attribute values and predicates. Contextual events are the basic elements for building asynchronous notifications and adaptations in context-aware systems. An application typically changes its behavior as a reaction to context changes specified by a contextual event. An example of a context change is the drop of a device's energy level. An application could be designed to react to such a change by subscribing to the corresponding contextual event, and thus avoiding the polling overhead that it would have been required to notice the context change. In some cases, the condition that fires a contextual event is too complex to be described by the model, so it should be implemented by the context provider which is responsible for publishing the context information.

Context queries allow the context modeler to devise specific semantics for queries, and which can be shared among all applications that need to use this context. For example, our location inference service provides queries as "*which users are in Room 4*" and "*what rooms are placed on floor 3*". As for the events, also here the application developer may have not enough knowledge about context semantics in order to describe queries correctly, so its implementation is delegated to the context provider, instead of using explicit queries on the context model. A more detailed discussion about MoCA's context model can be obtained in [4].

## 4. Approach for Model Integration using ONTO-MoCA

The ONTO-MoCA ontology describes the concepts and the abstractions of the MoCA’s context modeling, described in previous section. This ontology has three main goals: (1) to offer an alternative model to describe MoCA’s context model and thus enabling the sharing of MoCA’s concepts with other architectures or systems; (2) to allow the modeling of more complex characteristics of an environment that do not have direct influence on middleware behavior; and (3) to allow model checking and inference mechanisms in addition to the already supported in MoCA’s model.

The ONTO-MoCA describes all the concepts and properties of MoCA’s model, except those properties related to runtime behavior of a context information (e.g. if it is static) or properties used for deployment purposes (e.g. context provider or domain). A user typically does not use such properties to develop a context-aware application. We keep on ONTO-MoCA only the concepts that may interest such user. In this sense, the ONTO-MoCA is a complementary model approach to MoCA, instead of being a substitute. This approach differs from other hybrid modeling approaches such as [5], that prefer to model on ontology all the concepts and attributes of the original modeling approach, although compromising the readability of the ontology model.

The core of ONTO-MoCA comprehends the three main abstractions of MoCA’s context model: context attributes, contextual queries and events. These concepts are directly modeled as classes in ONTO-MoCA.

Context attributes are directly mapped to the ontology through properties and classes. In the same way, queries and contextual events are modeled as classes associated to context class, in order to maintain its meaning of abstraction. However, they are only partially mapped to ONTO-MoCA for two reasons. Firstly, because in an implicit query or event, we cannot associate a class to a specific context provider at system runtime. The second reason is that to describe the expression of an explicit query and event, we would need to adopt an ontology query language (e.g. RDQL) and a more complex description language (e.g. RuleML, SWRL), respectively. An ontology query language does not work to query context information, because context instances are maintained **only** on MoCA’s model.

Figure 1 shows a subset of ONTO-MoCA, describing the main properties of a context. A *Context* contains context attributes (*ContextAttribute*), a *ContextTarget* and the set of queries and contextual events (*ContextualEvent*) that specify the interface for context usage. *ContextTarget* specify the entity, person or activity that a context is associated to. Each *ContextAttribute* has a QoC

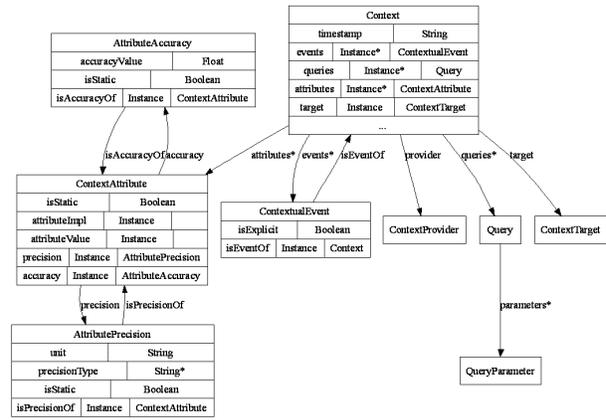


Figure 1. Context structure according to ONTO-MoCA

information, in terms of *AttributePrecision* e *AttributeAccuracy*. Queries and contextual events contain a particular property: they can be specified by a context provider (for a *ProviderDefinedQuery* and a *ProviderDefinedEvent*), depending on the property *isExplicit*.

### 4.1. Mapping Inter-models

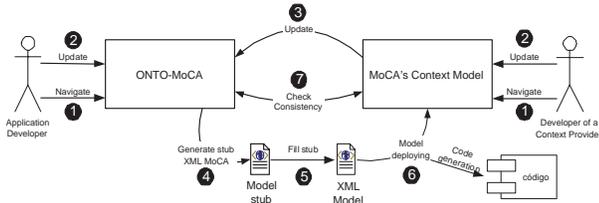
In order to guarantee the consistency between ONTO-MoCA and MoCA’s context model, we have developed a static and dynamic mechanism to update one model when the other is changed. The static mechanism maps concepts created on MoCA’s model to ONTO-MoCA in context deployment time, using the CT and the XML document that describes the context model. Dynamic mapping takes place at runtime, when context information must be updated on ONTO-MoCA, as result of changes on the environmental state. The dynamic maintenance of ontology is the critical point for performance on a context-aware system.

To implement static mapping, we used modules included to CT that access the context model currently available on MoCA. Using Jena [6], a module updates the classes of ONTO-MoCA and verify inconsistencies that a new context type may introduce (see section 4.2).

ONTO-MoCA does not interfere in the paradigm of context access and use. Applications continue to use object-oriented constructions to access context, following the MoCA’s traditional model, in order to avoid compromising application performance.

Figure 2 shows operations that maintain the consistency between the two models. The figure also shows two users that may interact with each model: a context-aware application developer, and a context provider developer, which is responsible for introducing either new sensors or inference

mechanisms that will publish a new context information.



**Figure 2. Operations between MoCA's model and ONTO-MoCA**

Since ONTO-MoCA maintain more high-level concepts, we expect that an application developer use it to understand how the environment state is described in the models in terms of context information, whereas the context provider developer acts on MoCA's model, describing the runtime behavior of a context information being published. Both users perform two operations described in Figure 2: *navigate* (1) and *update* (2). When a model is changed, as a result of a user interference or environmental change, the system infra-structure performs operation from (3) to (6), to maintain them in a consistent state.

A change on ONTO-MoCA is not enough to determine the corresponding change on MoCA's model, because ONTO-MoCA implements only a subset of MoCA's model. Thus, if a change on ONTO-MoCA must produce an updating on MoCA's model, for instance at the inclusion of a new context type, a *stub for the MoCA's XML model* is generated (4), which must be *fulfill* (5) with information that describes the behavior of the context, by either application developer or context provider developer. After this step, the new context type is *deployed* on MoCA's model (6), by using CT tool, which validates the new type in ONTO-MoCA and generates the binding code for accessing the context type on an application. As the last step, the validated model change is updated on ONTO-MoCA. Any conceptual change on ONTO-MoCA that does not interfere on MoCA's model can be introduced without any additional step. For example, a developer can introduce an additional classification of context types or add new concepts of interest restrict to the application.

In the dynamic mapping, the ontology is updated with the new concepts and events added on MoCA's model at runtime. Hence, this mapping is restricted from MoCA's model to ONTO-MoCA. In order to do not compromise system performance and scalability, context instances on MoCA's model are not represented on ONTO-MoCA, but only its changes in runtime iff there is an application interested in such change. When an event is fired, an instance of *AdaptationEvent* (subclass of *ContextualEvent*) is created, representing the contextual event that has occurred.

There is only one instance of *AdaptationEvent* for a same context instance and interested application and, thus, on the creation of an instance, the previous instance for the same event is destroyed.

## 4.2. Reasoning on a Contextual Model

The proposed hybrid approach allows the introduction of new model checking mechanisms, based on ontology, besides the already implemented by the CT. Moreover, an user can make new inferences based on the ontology representation of MoCA's model. For example, the developer of a context provider can use the inference mechanisms of Jena to reasoning about new context information or more complex contextual events, such as activity or situations.

However, inference agents are still limited to use MoCA API to retrieve synchronously context information, directly from MoCA's model, because the current state of all context information is not represented on ONTO-MoCA, for performance reasons. If an inference is based on the current value of a context information, instead of notifications of context changes, it must use the traditional mechanism for accessing context synchronously. MoCA offers a library that converts a context information based on MoCA's object oriented model to an instance of an ontology class.

If an inference is based on a contextual event, it can be done directly on ONTO-MoCA. We decided to allow such inference directly on the ontology, because MoCA's asynchronous notification model may be not suitable for implementing inference mechanisms. However, it is still possible to convert a MoCA contextual event on its correspondent ontology instance.

## 4.3 Advantages

The main advantage for adopting our hybrid mechanism is the sharing of MoCA's model concepts with another systems, and the ability to access from MoCA's model concepts previously defined in other ontologies, such as time (DAML-Time) and people relationship (FOAF).

Another advantage of this hybrid model is the possibility to use ontologies to model complementary concepts and classifications. For example, ubiquitous computing systems such as CONON [12] and GAIA [8], use ontologies to promote an additional classification to context, using categories such as *computational*, *location* and *document* related context. Although such classifications are not decisive to the behavior of the middleware that manages context, they can be important to the application developer to understand how the environment is modeled as context and to select the context information closer to application needs. In MoCA's model, such additional classification cannot be modeled.

In this hybrid model, the user can complement inference mechanisms using inference engines based on ontologies. This capability is very useful in scenarios where the system manipulates complex context information or when the inference mechanism is complex.

Finally, the proposed hybrid model provides a separation of the model according to user interest or task. On one hand, the developer of a context provider usually is interested in information maintained on MoCA's model. On the other hand, an application developer needs to comprehend the high-level characteristics of context information, which are better described in ONTO-MoCA. This separation of concerns eases the tasks of both users.

#### 4.4 Limitations

Our hybrid approach contains some limitations. Although we have a interface to modeling context through ontologies, a context-aware application still need to use the MoCA API, at least to register interest in context or contextual events. This requirement can hinder the access of MoCA context by an application based on another middleware or language.

In order to allow the coexistence among context-aware applications without compromising the system performance, we must establish a mechanism for describing ontologies and rules specific for an application domain. In ONTO-MoCA, although some contextual events can be limited in an application, they are shared among all applications.

Since the mapping of concepts between the two model could not be completely implemented, the XML stub that is generated when a context type is added by the ontology (operation 4 on Figure 2) demands the inclusion of many complementary information, making the task of filling the stub more error-prone.

#### 5. Conclusions

This paper presented a hybrid context model that integrates ONTO-MoCA to MoCA's model. Both models and respective model checking mechanisms and inference are complementary, instead of equivalent. Adopting this mechanism, we can develop more complex context-aware applications without compromising the performance of MoCA middleware.

We aim at validating our ideas implementing an integration prototype using Jena and experimenting an integration between a MoCA-based application and the ontologies and mechanisms offered by CoBrA system. In order to increase the focus on performance and scalability, it is still necessary to develop a mechanism for establishing ontology domains,

allowing to restrict some context information to an application or network domain.

#### References

- [1] H. Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, December 2004.
- [2] H. Chen, F. Perich, T. Finin, and A. Joshi. SOUPA: standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 258–267, 22–26 Aug. 2004.
- [3] R. C. A. da Rocha and M. Endler. Context management in heterogeneous, evolving ubiquitous environments. *IEEE Distributed Systems Online*, 7(4), April 2006. art. no. 0604-o4001.
- [4] R. C. A. da Rocha and M. Endler. Supporting context-aware applications: Scenarios, models and architecture. In *Proc. of the XXIII Simpósio Brasileiro de Redes de Computadores (SBRC)*, volume I, Curitiba, Brazil, May 2006.
- [5] K. Henriksen, S. Livingstone, and J. Indulska. Towards a hybrid approach to context modelling, reasoning and inter-operation. In *1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 54–61, Orlando, Florida, March 2004.
- [6] HP Labs. Jena – A semantic web framework for Java. Available at: <http://jena.sourceforge.net/>, 2006. (Last visited: July, 2006).
- [7] M. Khedr and A. Karmouch. ACAI: agent-based context-aware infrastructure for spontaneous applications. *Journal of Network and Computer Applications*, 28(1):19–44, 2005.
- [8] A. Ranganathan, R. E. McGrath, R. H. Campbell, and M. D. Mickunas. Use of ontologies in a pervasive computing environment. *The Knowledge Engineering Review*, 18(3):209–220, 2003.
- [9] V. Sacramento, M. Endler, H. K. Rubinsztein, L. S. Lima, K. Goncalves, and F. N. do Nascimento. MoCA: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, 5(10), Oct. 2004.
- [10] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, USA, December 1994.
- [11] M. Strimpakou, I. Roussaki, C. Pils, N. Kalatzis, and M. Anagnostou. Hybrid context modeling: A location-based scheme using ontologies. In *3rd International Workshop on Advanced Context Modelling, Reasoning and Management*, Pisa, Italy, March 2006.
- [12] X. Wang, D. Zhang, T. Gu, J. Dong, and H. K. Pung. Ontology based context modeling and reasoning using OWL. In *1st International Workshop on Advanced Context Modelling, Reasoning and Management*, Orlando, Florida, March 2004.