Complementarity of Process-oriented and Ontology-based Context Managers to Identify Situations

Amel Bouzeghoub, Chantal Taconet, Amina Jarraya, Ngoc Kien Do and Denis Conan
Institut Télécom, Télécom SudParis; CNRS UMR SAMOVAR
9 Rue Charles Fourier, 91011, Évry Cedex, FRANCE
Chantal.Taconet,Amel.Bouzeghoub,Amina.Jarraya,Ngoc_Kien.Do,Denis.Conan@telecom-sudparis.eu

Abstract

One issue for context-aware applications is to identify without delay situations requiring reactions. The identification of these situations is computed from both dynamic context information and domain specific knowledge. This identification is the output of a process involving context interpretation, aggregation and deduction. In smart environments, these treatments have to be efficient since they may be partly performed on constrained mobile devices. Two main approaches exist in the literature: process-oriented and ontology-based context management. In this paper, we claim that they are complementary and we propose an architecture which integrates the two approaches. We show in a scenario how context-aware applications can benefit from this architecture both to scale to numerous mobile users and to identify complex situations.

1. Introduction

One issue for mobile applications is to identify without delay new situations requiring appropriate adaptation actions from context-aware applications or notifications of recommendations to end-users. As stated in [14], "situations are external semantic interpretations of low-level context, permitting a higher-level specification of human behaviour in the scene and the corresponding system services. Situations inject meaning into the application and are more stable, and easier to define and maintain than basic contextual cues. Adaptations in context-aware applications are then caused by the change of situations (i.e., a change of a context value triggers adaptation if the context update changes the situation)." Two main approaches exist in the literature to identify situations from applications' context: process-oriented context management (PCM) and ontology-based context management (OCM). They bring into play context processors and inference engines with ontologies, respectively.

PCM involves context processors that collect, filter, aggregate, and interpret input context data to compute higher-level observations. These context processors are organised into a graph, more particularly a forest, with leaf nodes collecting raw data from context sources, non-leave nodes infering more abstract context data, up to root nodes delivering situation identifications. Since smart environments may comprise numerous mobile and constrained devices, context processors should be deployable on small devices. *Context Toolkit* [5], Contextor [4], COSMOS [3], and the MUSIC context manager [12] are representative of the PCM approach.

For several years, ontologies have been used for the development of ubiquitous computing applications with the goal of modelling the information managed by such applications. OCMs are characterised by the use of one or more ontologies as central elements of the system. This is a knowledge-based system whose architecture consists mainly of a knowledge repository (formed by an ontology) and an inference engine. It makes it easy to automatically deduce further implicit high-level contexts or situations (like user's activities) from low-level context data (like location, temperature or noise). In addition, they offer a support for interoperability and heterogeneity since they can be shared with other domain ontologies. CoBrA [2], SOCAM (conon ontology) [7], GAIA [13] and MUSE [1] are examples of OCMs.

In this paper, we claim that the PCM and OCM approaches are complementary and we propose an architecture which integrates them. We show in a scenario how context-aware applications can benefit from this architecture both to scale to numerous mobile users and to identify complex situations.

The outline of the paper is as follows. In Section 2, we motivate and give our objectives through an illustrative scenario. We present an overview of our proposition in Section 3 and detail the implementation of the campus scenario with the proposition in Section 4. Section 5 compares PCMs and OCMs and highlights their complementarity. Then, we compare our contribution with regard to related work in Section 6. Finally, we conclude and present some perspectives in Section 7.

2. Motivations and objectives

In this section, we present the illustrative example applications of a campus scenario and some situations to detect. We then introduce the requirements on context management.

Campus scenario Marie is one of the thousand students at TSP school. She enters the campus at 08:00 am. She is automatically detected and authenticated. An application installed on her mobile phone proposes two kinds of functionalities: *i)* a “query mode” to search contextual entities such as electronic resources (*e.g.*, printers, computers) or persons (*e.g.*, friends, teachers) and *ii)* a “recommendation mode” in which the system pro-actively notifies contextual recommendations. For example, the application displays the necessary documents for her next class programmed in her agenda at 8:30am as well as the room number and location. Marie wants to print one of these documents. The application uses the query mode to find the nearest available printers which are in unengaged rooms (*i.e.,* rooms not reserved for a lesson). John is Marie’s next class teacher. At 8:15am, the recommendation mode of an application installed on John’s phone notifies him for his class scheduled within fifteen minutes. At lunchtime, John is close to the self-service restaurant. A lot of people are waiting in the line. Since John has a meeting scheduled within one hour, another application proposes him the delivery of a tray lunch in his office adapted to his culinary habits. Yesterday, the line was nearly empty, the application displayed instead the lunch menu, thus John chose his lunch while waiting in the line.

Examples of situations For this scenario, we identify two abstraction levels for classifying situations. The first level may be computed from the observations obtained from Marie’s and John’s devices and from sensors spread over the campus. The second level may involve a knowledge base. For instance, the system detects and identifies a visitor entering the restaurant and proposes the delivery of a tray lunch in the visitor’s office. Firstly, the situation “a visitor enters the restaurant” is detected from dynamic context data. Afterwards, the recommendation “delivery of a tray lunch in the visitor’s office” requires the use of both context observations and a knowledge base: the visitor’s profile, agenda and habits, and the estimation of the waiting time in the line.

Requirements for context management From the scenario and the examples of situations, we identify the following requirements for the context manager with a criterion name for each of them. Context data such as visitor’s location evolve constantly (*Handling constantly evolving context* criterion). Context data collected or computed on a device shall be consumed elsewhere (*Distributed context management* criterion). Complex situations shall be computed from dynamic context data and from application-specific knowledge, both unforeseenable at design time (*Handling unforeseen situation* criterion). Situation changes shall be both observed by and notified to applications (*Observation/Notification* criterion). When permitted by the freshness requirement, context data requiring resource intensive processing shall be prepared in order to be consumed rapidly later on (*Observation preparation* criterion). The solution proposed shall scale up to thousands of visitors (*Scalability* criterion), all of them being equipped with mobile devices (*Constrained device* criterion). Recommendations shall be presented to end-users with appropriate delays according to end-users’ perception (*Situation detection latency* criterion). Application designers shall have a means to express relevant situations to be detected by the system (*Expression of situations* criterion). In Section 5, we will discuss each of these criteria with respect to our proposition.

3. Hybrid OCM and PCM proposition

This section presents our proposition, an hybrid architecture combining a PCM (COSMOS, COntext entitieS coMpositiOn and Sharing) and an OCM (MUSE, Multi-ontology based User Situation awarenEss). Then, we provide a rapid overview of COSMOS and MUSE.

3.1. Hybrid Architecture

Figure 1 presents the architecture combining a PCM and an OCM between the context sources and the application. The *COSMOS* PCM collects the raw context data from the different context sources of the smart environment: *e.g.*, locations, temperatures, images of the restaurant line. As depicted in the COSMOS PCM box, the graph represents context processing paths from raw context data to higher-level context data. Either COSMOS is responsible for infering high-level context data and situations (left part of the box) or it supplies the MUSE inference engine with low-level context data (right part of the box). The *MUSE* OCM is divided into three entities. The *Context Ontology* describes a generic context knowledge. The *Application Ontology* describes applications specific knowledge. These ontologies are used by the *Inference Engine* when processing context data provided by COSMOS and application-specific data.

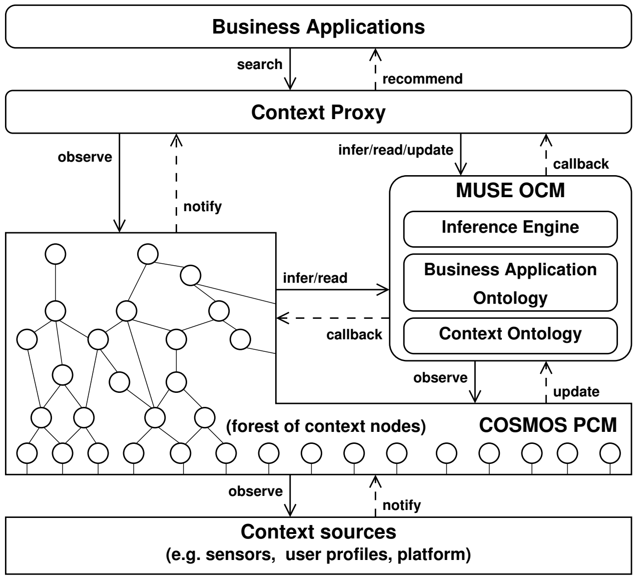


Figure 1. Hybrid architecture

Since this paper focuses on context management, the figure does not show how the inference engine obtains the applications-specific data. The context data produced by COSMOS and MUSE are transmitted by the *Context Proxy* to the *Business Applications* according two modes: an application can browse its context by calling for a search or can subscribe to receive recommendations.

In addition, Figure 1 shows multiple interactions between COSMOS and MUSE. COSMOS provides two interaction modes to observe (pull) context data and to notify (push) newly computed context data or situation changes. MUSE also provides two interaction modes, the first to receive an inference or a read request, and the second to trigger call-backs following situation changes. Notice that, in addition to interactions involving COSMOS supplying MUSE with context data, COSMOS processing nodes can benefit from MUSE ontologies as context sources. For example, in the illustrative scenario of Section 2 (which implementation is described in Section 4), the search performed by COSMOS for the nearest available printers in unengaged rooms implies to get classroom locations from MUSE.

3.2. COSMOS

COSMOS is a framework for the principled specification and composition of context policies. With COSMOS, these policies are decomposed into fine-grained units called "context nodes" implemented as software components and organised as hierarchies with sharing. These units perform basic context-related operations (*e.g.*, gathering data from a system or network probe, computing threshold or average values) and are assembled with a set of well-identified architectural design patterns. A library of context operators allows designers to define new COSMOS nodes by composition: existing context nodes are connected to a context operator (also a component) which takes their provided context data as inputs; all these components are gathered into a new composite component to build a new context node. Some of the context nodes have a specific meaning because they provide context data corresponding to the situations to be detected. These nodes constitute the nodes that are accessed by COSMOS clients (in our architecture, the Context Proxy) and correspond to the context policies.

Every context node of a context policy can also be finely tuned in order to control the flow of context data and to control the operating system resources consumed for context processing treatments, more especially threads and memory space. COSMOS is implemented as an open source framework and is available on a large number of mobile devices including J2ME phones and Android phones[1].

3.3. MUSE

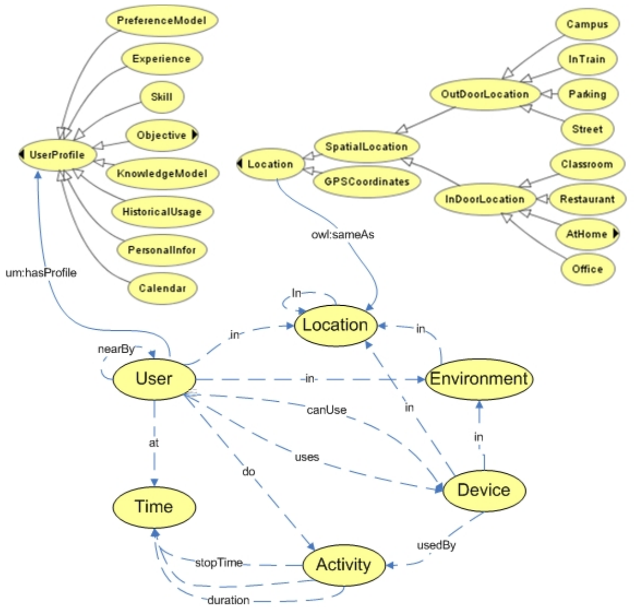


Figure 2. MUSE campus ontology (partial view)

Figure 2 presents the broad lines of the chosen context ontology. It is a multi-ontology representation where each ontology corresponds to an element or a context dimension. Each dimension models and manages a context information

[1]http://picolibre.int-evry.fr/projects/cosmos.

type. We focus on the following six dimensions since they cover all needed information for our scenarios: User, Activity, Environment (Technical Computing and Physical Environment), Device, Location and Time. This approach proposes to bridge over these dimensions with semantic relations (*e.g.*, in, use, nearBy, do) to express facts like: where is the learner? in which environment is he/she? which activity is he/she doing? which device is he/she using? Some of these ontologies are based on existing standards like CC/PP [9] for Device ontology and W3C [8] for Time ontology while the others are specific to MUSE. The User ontology contains all information about the user profile such as competency, preferences, schedule, historical usage. Finally, the Activity ontology describes all possible activities a user may practice. All these ontologies are described with the OWL language and rule-based reasoning is performed with the F-Logic language based on first-order logic.

4. Implementation of the campus scenario

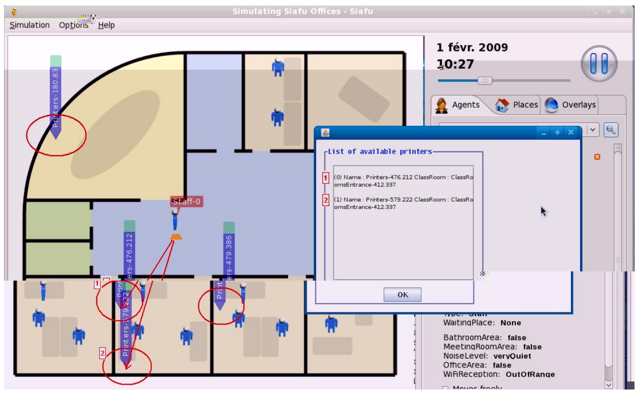
In this section, we present how the implementation of the campus scenario takes benefit from an architecture combining PCM and OCM. We introduce the way we simulate the context using the Siafu context simulator. Then, we present the implementation of two use cases: a search and a recommendation illustrating the two modes of interactions and the combination of PCM and OCM. In these two use cases, we develop the role and the functioning of the COSMOS PCM and the MUSE OCM, respectively.

4.1. Simulating the scenario with Siafu

To exercise our solution, we simulate the context using the Siafu context simulator [11]. The campus is depicted in a map, campus places are tagged with context data (*e.g.*, GPS location, name of the place), overlays are drawn on the map (*e.g.*, WiFi network overlay, gradient temperature overlay), agents simulating visitors are added with their profiles (*e.g.*, role in the scenario, language). Siafu allows the scripting of visitors' movements in the map with different mobility patterns (*e.g.*, random way-point, predefined paths). Siafu provides client applications, here COSMOS and MUSE, with a Web Service to get the context data of the simulation. Therefore, for our experiments, mobile phones execute COSMOS context policies and a fixed host runs the MUSE server, and mobile phones interact with the MUSE server and the Siafu server through a WIFI network.

4.2. "Query mode" for "Nearest printers" use case

We illustrate the query mode of our proposition with the search of printers which are the nearest to Marie. The result of the search is the presentation on Marie's mobile device of a list of ordered printers beginning with the nearest printer. Figure 3 is a Siafu representation of a building of the campus showing in circles Marie and existing printers. The selected printers are linked to Marie's avatar on the map and the result is superimposed. The two selected printers are in a classroom not booked for a class (even if some visitors are present in it; perhaps, they are also currently using the printers). The two other printers are not selected because they are either in the meeting room to be occupied in a few minutes for a meeting (even if nobody is present yet) or in another classroom that is busy (*e.g.*, engaged for a lesson).



List of available printers
1 (0) Name : Printers-476.212 ClassRoom : ClassRoomsEntrance-412.337
2 (1) Name : Printers-579.222 ClassRoom : ClassRoomsEntrance-412.337

WaitingPlace: None
BathroomArea: false
MeetingRoomArea: false
NoiseLevel: veryQuiet
OfficeArea: false
WiFiReception: OutOfRange

Figure 3. Siafu map of "Nearest printers"

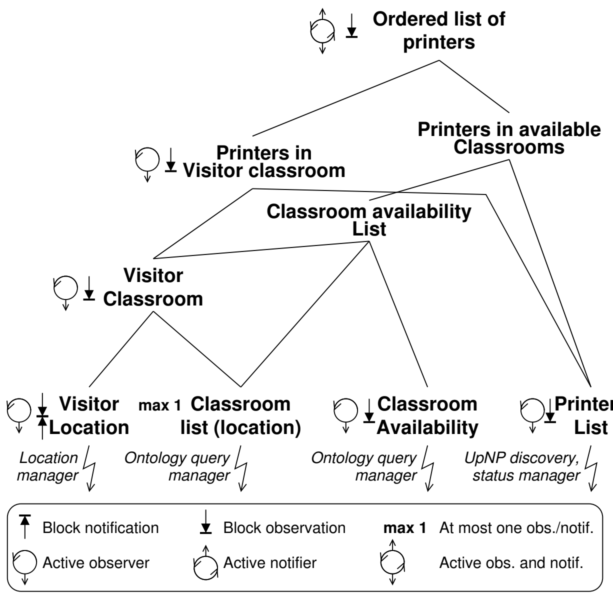


Figure 4. Context policy of "Nearest printers"

Figure 4 presents the COSMOS context policy for this use case. We introduce the configuration capabilities of

COSMOS context nodes when presenting the hierarchy of context nodes from the leaves to the root. Visitor location provides the geographical coordinates of Marie's location. Classroom list outputs the geographical coordinates of the rooms that are obtained from the MUSE ontology. Annotation "max 1" means that the observation is done only once since the campus map does not change. Classroom availability computes rooms availability from data obtained from the MUSE ontology. Printers list is an encapsulation of an UPnP service which determines accessible printers. Except for the node Classroom list which is annotated "max 1", the leave nodes control the collect of the raw context data by blocking the observations: The nodes return the most up-to-date previously collected context data without polling context sources, then avoiding a latency that can be important. For that purpose, those leave nodes are active observers (depending on system load average, the activity adapts its periodicity for polling context sources).

Visitor Classroom deduces Marie's classroom identification from Visitor location and from Classroom list. Classroom availability list produces an ordered list of unengaged classrooms with the nearest first. Printers in visitor classroom gives the list of printers in the visitor classroom if any. Printers in available classroom provides an ordered list of printers in the other classrooms. Ordered list of printers is the root of the hierarchy and outputs the list of printers, with the printers in visitor's classroom first. Notice that the root node is active in observation and in notification: It proactively observes the context to update the list of printers; It can both receive observation requests and notify clients when this list changes (meaning that this is a new situation).

4.3. "Recommendation mode" for "Restaurant" use case

We illustrate the recommendation mode of our proposition with the recommendation notified when John enters the restaurant. The recommendation presents either a selection from the meals of the day provided by the university restaurant or a proposition to deliver a tray lunch in his office. This recommendation is built into two steps. Firstly, COSMOS identifies the situation of John entering the restaurant to have a lunch. This situation is detected when John enters the restaurant at lunch time. This situation is then complemented with information on the estimated time to be served with the evaluated waiting time.

Secondly, following COSMOS notification, MUSE inference engine is called by the Context proxy in order to make an adapted recommendation. More generally, MUSE measures the similarity between the current situation and already known prototypical situations. If an exact similarity is found, the corresponding recommendation is triggered; otherwise, an inference calculus is processed in order to determine the most similar situation among the known ones. In our example, MUSE identifies John's situation as follows:

```
WaitingTimeForRestaurantSituation =
{in(O_User.User, O_Location.Restaurant),
hasWaitingTimeForMeal(O_User.User, @WT),
hasBreakTime(O_User.User, @BT)}
```

The following recommendation rule is then triggered: "If the waiting time is equal or higher than the user break time, the recommendation process proposes to deliver a tray lunch":

```
RecommendationRule = {
ON VisitorEntersRestaurantEvent
IF (WaitingTimeForRestaurantSituation.@WT >=
     WaitingTimeForMealRestaurantSituation.@BT)
THEN Action(deliver(O_Resource.Tray_Lunch,
          EntersRestaurantEvent.User))
}.
```

5. Discussion

In this section, we discuss the proposed hybrid PCM/OCM architecture with regards to the criteria introduced in Section 2. We illustrate this discussion with examples taken from the campus scenario. This discussion highlights the differences and complementarity of PCM and OCM.

Handling constantly evolving context This criterion characterises the ability of the context manager (CM) to handle context data that evolves constantly such as the visitors' location. PCMs have been designed to integrate context collectors. For example, with COSMOS, the leave nodes of a hierarchy wrap a context collector. Interfacing OCMs with PCMs as proposed in our architecture has the advantage of limiting the number of calls to the heavy-weight inference engines of OCMs: PCMs act as filters of situations and are by definition less heavy-weight and more efficient for detecting low-level situations. For instance, in the restaurant use case, John's location updates are transmitted to MUSE *i)* when he enters the restaurant line and *ii)* when he leaves the restaurant line.

Distributed context management In the campus scenario, the number of observed entities is large. Furthermore, the entities are distributed on the whole campus. For performance reasons, a centralised context management would become a bottleneck. With the MUSE/COSMOS architecture, context management is distributed. Context management is done partly on mobile devices by the PCM and partly on a central server by the OCM. For the restaurant use case, the situation "visitor enters in the restaurant line" is autonomously managed by each visitor's mobile device. MUSE is invoked for a second level of deduction. Distribution is also a requirement for disconnection tolerance to avoid single point of failure and disconnection: The PCM is the right place to detect disconnections on mobile devices.

Handling unforeseen situations This criterion defines the ability of the context manager to identify complex situations computed both from raw context data and application-specific knowledge, and also to infer situations unforeseen at design time. With PCMs, all the context processors on a processing path from context collection up to situation detection are precisely organised to detect well-defined situations. On the contrary, the OCMs reasoning process is by definition able to deal with unforeseen situations. These situations were not expressed at design time, thus meaning that no inference rule is triggerable. In such a case, the current situation is compared to the set of historical situations using a case-based reasoning to find the most similar situation and to propose the most accurate recommendation. Furthermore, in order to take into account a larger number of situations, the designer can add new rules.

Observation/notification It is a requirement of context managers to be effective. The PCM terminology is observation and notification whereas the OCM terminology is search and recommendation.

Observation preparation Context inference requiring resource-intensive processing shall be prepared beforehand in order to be ready when required. With COSMOS, each node may be configured as a non-blocking node or not. During an observation, a non-blocking node propagates the observation and requests observations from each of its child nodes. On the contrary, a blocking node provides the client or parent node with the last values computed during the last reception of children inputs. For instance, in Figure 4, the root node which prepares the ordered list of printers is a blocking node: It allows Business applications and Context proxy to consume with no delay a pre-computed/prepared list. When a node is a blocking observer, it must possess an activity for periodically observing child nodes; such a node is said to be active. Node activities are matched to threads: all the activities can be managed by the same thread or each activity can lead to a separate thread, etc. Another advantage of the "blocking node" and "active node" configuration parameters is the possibility to tune the frequency of the calls to context sources. This is important since for instance the collect of context data from a local user profile is much less expensive (processor usage, time) than the collection of context data from a remote UPnP server.

Scalability For the campus scenario, the proposed architecture has to scale potentially to a large number of visitors. For scalability purpose, we claim that the cooperation between PCM and OCM is essential. Indeed, PCM computation may be handled by distributed mobile devices. This characteristic is an opportunity to distribute the CM load. If the OCM were interfaced directly with context collectors, this would not be possible since most interpretations and deductions should be performed by the OCM server. For the restaurant use case, the OCM is called only when the "visitor enters the restaurant" situation is detected by the PCM executing on the mobile device. In consequence, we greatly reduce the load of the OCM server.

Constrained device In smart environments, some context collectors and a part of the context management must be executed on mobile devices. COSMOS PCM is operational on mobile devices if those devices provide the J2ME or Android API. To the best of our knowledge, OCM cannot be deployed on constrained devices. Therefore, the collaboration of PCM and OCM is required.

Situation identification latency In smart environments, situation identification with a reasonably short latency (in the order of the second) improves the degree of acceptance of context-sensitive applications by end-users. In addition, some critical situations automatically managed by proactive services of middleware platforms require a shorter latency (in the order of the hundred of milliseconds). This should be one of the goals of PCMs to control situation identification latencies. For instance, COSMOS provides activity management facilities with blocking facilities to finely tune every context node of a context policy. On the contrary, OCM-based solutions cannot ensure that an inference computation involving artificial intelligence methods is bounded in time. In addition, output context data cannot be computed beforehand and provided when required.

Expression of situations This criterion explains the means proposed to application designers to express the events which have to be detected by the CM. In most of PCMs, interpretation mechanisms are hidden in the code. With COSMOS, situations are expressed by an assembly of existing context sources and operator components. In OCMs, several languages can be used for context modelling and context reasoning (DAML+OIL, DL, Ontobroker, OWL, Prolog, Jena, etc.). MUSE uses Ontobroker reasoner and F-logic rules. A PCM/OCM architecture offers application designers a wide range of possibilities to express how to identify situations. The cost is that application designers shall have multi-language skills.

This section has highlighted that some of the criteria are well handled by both PCMs and OCMs (observation/notification), others are better handled by PCMs (handling constantly evolving context, distributed context management, observation preparation, scalability, constrained device, situation identification latency) or OCMs (handling unforeseen situations, expression of situations). An hybrid

OCM/PCM solution allows designers architecting their system in order to maximise advantages.

6. Related works

As stated before, context management can be achieved using either PCMs or OCMs. This section presents works related to those two approaches and more particularly to studies mixing PCMs and OCMs.

PCMs Context Toolkit [5] acquires, interprets and delivers context data to applications. Developers design and implement widgets (to collect data), interpreters (to compute higher abstractions) and aggregators (to aggregate data from several widgets). Application/widgets interactions are made through queries and call-back notifications. In the spirit of Context Toolkit, interpreters and aggregators are heavy-weight software entities. Nothing precludes encapsulating an OCM in one of them. Nevertheless, Context Toolkit is by design a PCM and the contribution of this framework is not in using context data obtained from an inference engine working with ontologies. The Contextor builds a context manager as a network of contextors [4]. Contextors are software entities equivalent to components with meta-data describing context data quality and controllers modifying their configuration. The target of this framework is the support of ad-hoc networking and dynamic construction of the architecture. These solutions have the same goal and are built using the same approach as COSMOS: the PCM approach. Therefore, they are very powerful in building chains, trees or even forests of efficient context processors going from elementary observables to situation identification. However, they lack the support of a broad context representation which would enable to reason easily on a large knowledge base.

OCMs CONON [7] and CoBrA-ONT [2] are ontologies describing knowledge of the ubiquitous computing domain (or some sub-domain of this discipline) and helping to disambiguate different contexts that may have different meanings. Those ontologies should be integrated with widely accepted ontologies (such as the FIPA Device Ontology [6], and CC/PP [9]) with the goal of knowledge reuse. Some of these ontologies are used in ontology-driven ubiquitous computing applications and during execution for context management purpose. This category of solutions proposes architectures which consist mainly of *i)* a knowledge base, *ii)* an inference engine and *iii)* context acquisition and interpretation mechanisms. CoBrA [2], SOCAM (CONON ontology) [7] and GAIA [13] are representatives of this category. OCMs have advantages regarding their capabilities for reasoning and their level of interoperability and heterogeneity. They are well suited for the recognition of high-level context abstractions. The main problem with OCMs is that reasoning is computationally expensive. Online execution of ontological reasoning is technically impossible on constrained mobile devices and raises scalability issues, especially when the ontology is populated by a large number of individuals. An OCM needs to be wrapped with a context provider in order to reason on updated observations for constantly evolving contexts. OCMs generally do not detail how they are interfaced with a context provider as their contributions are rather at higher levels.

PCM and OCM combination MUSIC [12] includes pluggable context sensors (to collect data) and context reasoners (to interpret data). Each context plug-in defines its required and provided context types. The contribution of MUSIC is the configuration capabilities of the context manager at runtime through the concept of context plug-ins, that is dynamic activation and deactivation of context plug-ins. However, the authors do not discuss the complementary of the context reasoner in an OCM approach with the plug-ins of the context sensors that can in a sense be viewed as the basic components of a PCM approach. In [10], the authors present a framework for the recognition of situations in "real-time" in the presence of uncertain, noisy and rapidly changing contexts. Mobile devices run a resource server similar to a PCM with the following functionalities: sensor measurement, preprocessing, feature extraction and quantisation. These resource servers send their data to a blackboard used by the Context Recognition Service and the Change Detection Service. These latter services encapsulate an ontology, thus implementing the OCM approach. According to the authors, the rationale for this combination of PCM and OCM are the situation detection latency and the scalability. However, they do not view the PCM as a possible client to the OCM. In this paper, we mix an OCM (MUSE) with a PCM (COSMOS) rather than with a simple context provider. The COSMOS PCM is available on mobile devices. This allows us to deploy a great part of the context collection and reasoning on mobile devices. Compared to a direct context provider solution, this drastically reduces the amount of events sent to the MUSE OCM and thus improves the scalability of the solution. MUSE is well suited for the identification of high-level situations. It offers reasoning capability and can deal with unforeseen situations. The mix of COSMOS PCM and MUSE OCM provides better scalability, efficiency, flexibility and enable designers to reason on high-level knowledge such as user activity and user intentions.

7. Conclusion

Identifying situations to trigger without delay appropriate reactions is an important issue for mobile applications. Many context management solutions have been proposed so far for that purpose. Context management design must meet mobile devices constraints, interoperability and scalability features. In this paper, we have presented an hybrid architecture which integrates two types of approaches: process-oriented context management (PCM) illustrated by COSMOS and ontology-based context management (OCM) illustrated by MUSE. The situation identification is performed at two abstraction levels. The first level may be computed from the observations obtained from devices and sensors. The second level may involve a knowledge base. We have identified six criteria to highlight the differences and complementarity of PCMs and OCMs for the two modes of interactions: query or search, and notification or recommendation.

We have demonstrated that the combination of a PCM and an OCM for context management can lead to applications having a good level of scalability and efficiency while allowing mobile application designers to reason on high-level knowledge. Furthermore, it enables the distribution of context interpretation even on constrained devices. The drawback is that it requires mobile application designers to master several complex technologies. In a future work, we intend to measure the scalability of our hybrid solution. We are also following a model driven approach and plan to develop models from which to generate context management code. This will release the application developers from manipulating context management technologies.

References

[1] A. Bouzeghoub and K. Do Ngoc. A situation based metadata for describing pervasive learning objects. In *Proceedings of mLearn 2008 : 1st International Conference on Mobile Learning*, University of Wolverhampton, Ironbridge, UK, October 8-10 2008.

[2] H. Chen, T. Finin, and A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197–207, May 2004.

[3] D. Conan, R. Rouvoy, and L. Seinturier. Scalable Processing of Context Information with COSMOS. In Springer-Verlag, editor, *7th IFIP International Conference on Distributed Applications and Interoperable Systems*, volume 4531 of *Lecture Notes in Computer Science*, pages 210–224, Paphos, Cyprus, june 2007.

[4] J. Coutaz and G. Rey. Foundations for a Theory of Contextors. In C. Kolski and J. Vanderdonckt, editors, *Proc. 4th International Conference on Computer-Aided Design of User Interfaces*, pages 13–34, Valenciennes (France), May 2002. Kluwer.

[5] A. Dey, D. Salber, and G. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Special Issue on Context-Aware Computing in the Human-Computer Interaction Journal*, 16(2–4):97–166, 2001.

[6] FIPA. *FIPA Device Ontology Specification*. Foundation for Intelligent Physical Agents (FIPA), Geneva, Switzerland, http://www.fipa.org/specs/fipa00091/xc00091c.pdf edition, 2001.

[7] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang. An Ontology-based Context Model in Intelligent Environments. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 270–275, San Diego, California, USA, January 2004.

[8] J. Hobbs and F. Pan. W3C Time-ontology. Technical report, W3C recommandation, september 2006.

[9] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran. Composite Capability/Preference Profile (CC/PP): Structure and vocabularies 2.0. Technical report, W3C recommandation, april 2007.

[10] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen, and E. Malm. Managing Context Information in Mobile Devices. *IEEE Pervasive Computing*, 2(3):42–51, July 2003.

[11] M. Martin and P. Nurmi. A Generic Large Scale Simulator for Ubiquitous Computing. In *Proceedings of Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services, 2006 (MobiQuitous 2006)*, pages 1–3, San Jose, California, USA, July 2006. IEEE Computer Society.

[12] N. Paspallis, R. Rouvoy, P. Barone, G. Papadopoulos, F. Eliassen, and A. Mamelli. A Pluggable and Reconfigurable Architecture for a Context-aware Enabling Middleware System. In *Proceedings of the 10th International Symposium on Distributed Objects, Middleware, and Applications (DOA'08)*, volume 5331 of *Lecture Notes in Computer Science*, pages 553–570, Monterrey, Mexico, Nov. 2008. Springer-Verlag.

[13] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *Pervasive Computing, IEEE*, 1(4):74–83, Oct-Dec 2002.

[14] J. Ye, L. Coyle, S. Dobson, and P. Nixon. Using Situation Lattices in Sensor Analysis. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications, PerCom'09*, pages 1–11, Galveston, TX, USA, March 2009.