Technische Universität Berlin

Forschungsberichte
der Fakultät IV – Elektrotechnik und Informatik

Object Technology for Ambient Intelligence

Workshop Reader for OT4AmI at ECOOP 2007

Holger Mügge[1], Éric Tanter[2], Pascal Cherrier[3], Jessie Dedecker[4],
Christina Lopes[5] and Michael Cebulla[6]

[1] University of Bonn, Germany
[2] University of Chile, Chile
[3] France Telecom, France
[4] Vrije Universiteit Brussels, Belgium
[5] University of California at Irvine, USA
[6] Technische Universität Berlin, Germany

This reader comprises the submissions to the third workshop on object-technology for Ambient Intelligence and Pervasive Computing held at ECOOP 2007.

Bericht-Nr. 2007 – 14

ISSN 1436-9915

Table of Contents

Proximity is in the Eye of the Beholder

Peter Barron[1] Jessie Dedecker[2] Éric Tanter[3]

[1] Distributed Systems Group,Trinity College, Dublin, Ireland
[2] Programming Technology Lab,Vrije Universiteit Brussel, Belgium
[3] DCC/CWR, University of Chile, Santiago, Chile
`Peter.Barron@cs.tcd.ie - jededeck@vub.ac.be - etanter@dcc.uchile.cl`

Abstract. The notion of proximity is a key to scalable interactions in distributed systems of any kind, both natural and artificial, and in particular in pervasive computing environments. However, proximity as such is a vague notion that can be considered both in a very factual manner (spatial distance) and in a very subjective manner (user affinity). We claim that an adequate system or programming language for ambient intelligence applications ought to support an open notion of proximity, making it possible to rely on different, possibly subjective, understandings of proximity, as well as their combinations.

1 Proximity: A Key to Scalability

Proximity can be defined as a *state of nearest*, the perception of *being close* to something or someone. Proximity naturally plays a significant role in how, as humans, we interact with our environment. This can be seen in the relationships we maintain with others, or in the manner in which we interact with everyday objects around us. Think about the closeness of a friend or relative, or about how books of a particular topic are considered to be close to each other.

The concept of proximity is interesting in the field of pervasive computing and ambient intelligence, where the focus is on unobtrusively managing and assisting in the tasks of users. Introducing proximity in these systems allow for better scalability both at the level of interactions [8] and at the level of demarking content of interest [6]. The scoping strategies enabled by proximity make it possible to tailor system behavior to better match the situations of users, going a step further in the direction of the non-intrusiveness requirement identified by Weiser [9].

As an example, consider how service discovery in a pervasive computing environment can take advantage of proximity. Instead of trying to discover *any* accessible service provider that matches the required service type, embedding a proximity criteria within the discovery process can drastically reduce the set of answers to process at the client side. Service providers that are able to determine that they do not meet the proximity requirement will simply skip the service request, thereby alleviating the burden of the client.

Also, the concept of proximity is indeed prevalent in biological systems, such as shoals of fish and social insects like ants and termites, where interactions are limited to the local environment [2, 4]. To unleash the potential of biologically-inspired communication models in pervasive computing environments, it is hence required to have at hand a notion of locality [1].

From this point of motivation, that *proximity should be supported by pervasive computing environment*, an analysis of possible useful notions of proximity is presented in this paper. The proliferation of potential notions in turn suggests that *proximity should not be hardwired into the infrastructure*, but rather supported in an *open* manner so that application-specific notions can be used and propagated among participants.

2 What Do You Mean, "Proximity"?

> *"Proximity is defined as the state, quality, sense, or fact of being near or next"* – The American Heritage

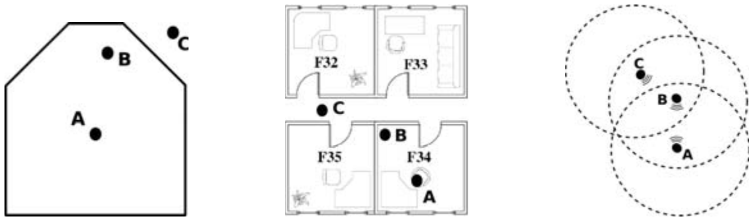
The above definition of proximity leads us to considering two orthogonal dimensions when it comes to analyzing proximity. First, being near or next depends on the notion of distance used; that is, one entity is closed to another *with respect to* a given metric. Such a metric can be based on *physical* properties of the entities (*e.g.* physical location), or on a more *abstract* criteria, not related to the material world (*e.g.* nearness of relatives). We discuss physical vs. abstract proximity in Section 2.1, and then consider the interests of being able to *compose* several proximity metrics in Section 2.2.

Second, the definition mentions the word "sense" in addition to "state" or "fact", which tends to suggest a *subjective* notion of proximity, that depends on the actual perception of the subject entity. This is in contrast to *objective* criteria or metrics, for which all entities share the same understanding of what it means to be close. We elaborate on this dimension in Section 2.3

2.1 Physical vs. Abstract Proximity

Physical Proximity. In current pervasive computing and ambient intelligence systems, the proximity of entities is primarily determined by physical considerations. For example, in YABS [1] interactions are limited to the local environment, where "local" is defined by a geometric parameter (Fig. 1(a)). In Gaia [7], proximity is administratively bound to a physical location which, in this case, is a meeting room (Fig. 1(b)). Taking a different approach, systems such as AmbientTalk [3] implicitly define proximity based on the signal strength of wireless communications (Fig. 1(c)): interactions can only take place when entities are in range of communication.

Abstract Proximity. Physical notions of proximity are very useful in developing pervasive computing systems [6, 8], but it is also possible to extend the benefits of proximity considerations by examining abstract notions of proximity:

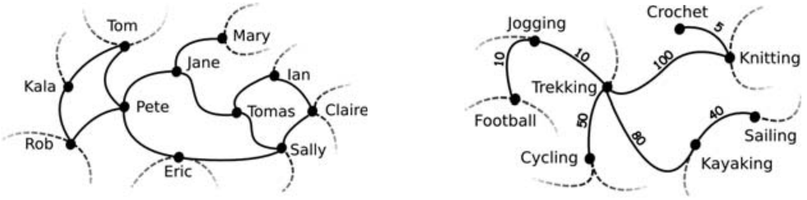


(a) Defining proximity in the form a polygon.

(b) Defining proximity using a physical boundary of a room.

(c) Defining proximity using the signal propagation of a wireless network.

Fig. 1. Different notions of physical proximity (B is "near" A, but C is not).



(a) Defining proximity using the relationships of users. Distance is determine by the degree of separation between two users.

(b) Defining proximity base on the interests or hobbies of users. Weightings on links indicate similarity of hobbies.

Fig. 2. Different notions of abstract proximity

an abstract proximity does not directly map to physical characteristics of the considered entities, but rather relies on logical, domain specific criteria.

First of all, one may consider a virtual rather than physical concept of place: *e.g.* although videoconference participants are geographically at distant places, they all share the same virtual meeting room. On another line, one can define proximity based on the *relationships* of users - friends, acquaintances, or friends of friends. The distance between two users (or entities owned by users) is the degree of separation between them, *i.e.* the length of the path relating them on a relationship graph (Fig. 2(a)). This metric can be used for instance to allow access to your personal devices to yourself, your friends, and friends of friends (that is, a friendship distance of at most 2). One can consider that present instant messenger applications consider the buddy relationship, restricting interactions to a distance of 1. In a different vain, it is possible to define proximity based on the interests or hobbies of users. The distance in this case can be described in terms of the similarity of one hobby or interest to another. For instance, jogging is arguably much more similar to trekking than to knitting (Fig. 2(b)). It is likewise possible to devise a wide number of abstract proximities, related to particular domains or applications.

```
proximity(5);  // circle of radius 5
proximity(-5,-5,-10,5,-10,20,10,20,10,5,5,-5); // polygon as in Fig.1(a)
proximity(F34); // symbolic location as in Fig.1(b)
```

Fig. 3. Proximity definitions in YABS.

2.2 Composite Proximity

Most pervasive computing systems consider proximity as a singular concept: the idea of *composing* different proximities to refine the overall scope of interactions is generally not considered. This is a strong limitation, because considering the potentially wide variety of proximity notions we have discussed above, it is clear that there is a lot to gain in being able to combine different types of proximity to express a more subtle requirement.

For example, composing a proximity base on geometric distance (Fig. 1(a)) and user hobbies (Fig. 2(b)) would first, aid scalability through the scoping of interactions within the local environment, and secondly, highlight content in the local environment that may be of interest. One could also consider spatio-temporal proximity, relating entities that are or have been, within a given time frame, in the same local environment. Another example is to combine spatial locality with network link quality, *e.g.* to aid in the development of an application disseminating multimedia content to local participants

2.3 Objective vs. Subjective Proximity

We now turn to a crucial issue when it comes to considering different notions of proximity in the context of open networks.

Objective Proximity. Existing pervasive computing systems support a notion of proximity that can be defined as *objective* in the sense that the semantics of the proximity function are hardwired in the middleware layer. That is, all entities in the system share the same notion(s). In a system like AmbientTalk, where network connectivity is the only proximity factor, this shared understanding is obvious. In Gaia as well, proximity is defined by physical presence in an active space, *i.e.* a meeting room. In a system like YABS, each entity can define its own proximity requirement using the `proximity` function (Fig. 3). Although the actual parameters of the proximity functions are specific to each entity, the *interpretation* of the proximity function is defined in the infrastructure, and cannot be changed.

Subjective Proximity. The way systems reliant on objective proximity work implies that the different shared interpretations of proximity are installed or configured upfront in the infrastructure. Although this approach is feasible if we consider a limited and fixed number of interpretations (like in YABS), it does not fit our claim that many proximity notions are of interest, both physical and abstract, and that these notions are potentially specific to certain applications or domains. It is necessary that clients are able to define, compose and use new notions of proximity.

In other words, if a new entity joins a certain environment and looks for services of a certain type that are "close" to it, this entity ought to be able to use its *own notion* of what it means to be nearby. This means that the proximity function should be defined by the client itself, not pre-defined by the underlying infrastructure. In this case there is no globally shared understanding of the proximity, rather a *subjective view* of the client, that reflects the particular requirements of the application.

3 Perspective: A Proximity Metaobject Protocol for AmbientTalk

Pervasive computing systems typically fail to support many abstract and composable notions of proximity, as well as to allow subjectivity in proximity definitions. It is clear that such flexibility raises important challenges at the implementation level. It is however, to our understanding, a very important and valuable approach to enable better scalability and usability in open pervasive computing systems.

We are currently exploring a proximity metaobject protocol for the Ambient-Oriented Programming language AmbientTalk [3]. Metaobject protocols are well-defined interfaces to the language implementation that allow the semantics of the language to be customized by programs [5]. In our case, this extends to a distributed language with proximity support.

We plan to first provide this proximity metaobject protocol at the service discovery level. In a second phase, it is necessary to go further, considering that since proximity can change dynamically, a service that was near at the time of discovery may "move away" while interactions are in progress.

Finally, although subjective notions of proximity imply that one client perceives its surrounding in a particular manner, it is important to distribute the *evaluation* of proximity functions among nodes, to limit network traffic. It can also be interesting to actually propagate proximity functions so as to dynamically upgrade the knowledge of involved participants with new proximity notions.

4 Summary

In this position paper, we have drawn attention to the important notion of proximity for building scalable and relevant pervasive computing and ambient intelligence applications. Starting from the different possible notions of proximity that can be useful, both physical and abstract, as well as their user-defined composition, we have argued that proximity should not be hardwired into the infrastructure, but rather supported in a way that makes it possible to use application-specific notions in a subjective manner.

References

1. Peter Barron and Vinny Cahill. YABS: a domain-specific language for pervasive computing based on stigmergy. In *GPCE '06: Proceedings of the 5th international conference on Generative Programming and Component Engineering*, pages 285–294, New York, NY, USA, 2006. ACM Press.
2. Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence From Natural to Artificial Systems.* Oxford University Press, 1999.
3. Jessie Dedecker, Tom Van Cutsem, Stijn Mostinckx, Theo D'Hondt, and Wolfgang De Meuter. Ambient-oriented programming in AmbientTalk. In Dave Thomas, editor, *Proceedings of the 20th European Conference on Object-oriented Programming (ECOOP 2006)*, Lecture Notes in Computer Science, pages 230–254, Nantes, France, July 2006. Springer-Verlag.
4. P.-P. Grass. Le reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.
5. Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol.* MIT Press, 1991.
6. Tim Kindberg and Armando Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1), 2002.
7. Manuel Roman, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):65–67, 2002.
8. M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10 –17, Aug 2001.
9. M. Weiser and J. Brown. The coming age of calm technology. *PowerGrid Journal*, 1.01, July 1996.

Context-Aware Leasing for Mobile Ad hoc Networks

Elisa Gonzalez Boix, Jorge Vallejos, Tom Van Cutsem [⋆], Jessie Dedecker, and Wolfgang De Meuter

Programming Technology Lab
Vrije Universiteit Brussel, Belgium
{egonzale,jvallejo,tvcutsem,jededeck,wdmeuter}@vub.ac.be

Abstract. Distributed memory management is substantially complicated in mobile ad hoc networks due to the fact that nodes in the network only have intermittent connectivity and often lack any kind of centralized coordination facility. *Leasing* provides a robust mechanism to manage reclamation of remote objects in mobile ad hoc networks. However, leasing techniques limits the lifetime of remote objects based on timeouts. In mobile networks, we also observe that devices need to continuously adapt to changes in their *context*. In this position paper, we argue that changes in context not only require adaptation in the behaviour of the application but also permeate to distributed memory management, leading to the concept of *context-aware leasing*.

1 Introduction

The recent advances in the field of Ambient Intelligence (AmI) have set new challenges to the development of a new type of distributed applications with sophisticated characteristics. AmI applications are distributed among mobile devices interconnected by wireless communication media that allow them to interact spontaneously with other devices in the environment forming *mobile ad hoc networks*. Example mobile ad hoc networking applications range from modest, already commonplace applications like collaborative text-editors, to more futuristic pervasive and ubiquitous computing [11] scenarios. Such scenarios also introduce new opportunities to build applications that can sense and deal with changes in their physical and computational *context*. Typically, these context changes require adaptation in the behaviour of the application.

In previous work, we have explored the impact of the hardware phenomena of mobile networks on distributed memory management and proposed the use of leasing [4] as a robust technique to reclaim remote objects in such network topology [3]. This paper focuses on the repercussions of context information on leasing and subsequently proposes the integration of context events directly into distributed memory management resulting in the concept of *context-aware leasing*.

2 Context-awareness and Leasing

Before discussing the repercussions of context-dependent adaptations on distributed memory management, we first introduce some terminology and concepts from the area of context-awareness and leasing.

[⋆] Research Assistant of the Fund for Scientific Research Flanders, Belgium (F.W.O.)

Leasing. Leases were originally introduced as a fault-tolerant approach in the context of distributed file cache consistency [4]. A *lease* denotes the right to access a resource for a limited amount of time. In distributed memory management, remote references play the role of the lease and the objects they refer to play the role of the resource. In other words, client objects from other machines can reference remote objects by means of *leased object references*. Therefore, a leased object reference is a remote object reference that grants access to a remote object only for a limited period of time. When a client first references a remote object, a leased object reference is created and associated to the remote object. From that moment on, the client accesses the remote object transparently via the leased reference until it expires. A leased reference can be renewed or revoked before its lease time expires. When the time interval has elapsed, the access to the remote object is revoked, i.e. the lease expires and thus the client can no longer access the remote object. Once all leases for a remote object have expired, the object can be garbage collected when no local references refer to it.

Leases are a robust technique for distributed garbage collection in mobile ad hoc networks such that remote objects can be reclaimed in face of both transient and permanent disconnections. In previous work, we described the above sketched leased object references as time-decoupled remote object references with built-in leasing semantics [3]. Throughout this paper, we assume such a leasing approach as the basis for our reasoning and examples. In what follows, we introduce specific features necessary for this paper but further details are available in a technical report [3].

As in contemporary leasing approaches (e.g. in Java RMI [9], Jini [10] and .NET Remoting [6])), the lifetime of remote objects is determined by means of timeouts. Our leasing approach also incorporates two variants of leased references which transparently adapt their lease time under certain circumstances. The first variant is a *renew-on-call* leased reference that automatically prolongs the lease upon each method call received by the remote object. The second variant is a *single-call* leased reference that automatically revokes the lease upon performing a method call on the remote object. Such leases are useful for objects which adhere to a *single call* pattern such as the callbacks objects that are often passed along with messages in asynchronous message passing schemes to return computed values. Our leasing approach provides the following language support to create a basic leased reference and the two above mention variations:

```
lease: timeout for: object
renewOnCallLease: timeout for: object
singleCallLease: timeout for: object
```

A leased reference created with the **lease** construct only lasts for the given time unless a renewal or revocation is explicitly issued. The **renewOnCallLease** construct creates a lease that is automatically prolonged on every message invocation with `timeout` value. Finally, **singleCallLease** construct creates a lease that expires either after the remote object receives a single message or when the `timeout` expires if no messages have been received. Other language support is provided to explicitly manipulate the lifetime of a leased reference (i.e. **renew** and **revoke** language constructs).

Context-awareness. We adopt the definition of *context information* as proposed in [5]: context is any piece of information which is computationally accessible. Examples of such information include not only information that can be automatically extracted from the surroundings by means of sensors (e.g. location or temperature), but also information from the computation environment (e.g. when devices disjoin and join the network) or user preferences. For the sake of this paper, we assume the existence of a mechanism to extract meaningful context information, e.g. ContextToolkit [8], and the use of a common standard ontology for context information that allows applications to understand this information [7], i.e. all applications use the same terminology for context events.

In the remainder of this section, we discuss the effect of such context information on leasing and illustrate why we need extensions to leasing to deal with it.

2.1 Leasing based on time is not enough

Leasing allow developers to guide the distributed garbage collector by determining the lifetime of remote objects. However, this places burden on developers. Determining the proper lease period is not straightforward and may even depend on system parameters such as the number of clients. Typically, leasing only allows developers to express the validity of a lease based on timeouts [9, 10, 6]. In mobile ad hoc networks, we observe that the validity of the leased reference can also depend on changing context parameters which monitor different types of events such as hardware events (e.g. disconnections of devices) or physical events (e.g. location of devices). As a concrete example, consider a remote reference which should remain valid only while the battery level of the device hosting the remote object is above an acceptable limit. A first approximation to implement this scenario in our leasing approach is to extend the **lease** language construct as follows:

```
renewOnCallLease: minutes(10) renewalConditions: {if: { batterylevel > 10%}}
  for: object
```

The above code excerpt illustrates the use of a boolean condition in a renew-on-call lease so that the leased reference is automatically renewed in relation to a context event, i.e. battery consumption.

Such renewal conditions can also be applied to other context events. Consider an example of a user attending a conference who wants to print one of their PDA files on a printer located at the conference building. Typically, such users will have restricted access to the resources available during the time that the conference is held, e.g. their internet access is limited to the conference building or they cannot print more than 100 pages. This example could be modelled in our leasing approach by extending the **lease** language construct to express the lease time in terms of boolean conditions. For example, restricting the number of pages that a user can print could be expressed as follows:

```
lease: getTimeLeft(days(3)) revokedOn: {if: { printedPages > MAXIMUM}} for: (
  object: {
     printingQueue : Queue.new();
     def print(doc){
       queue.add(doc);
     }
  }
)
```

As shown in the code above, a leased reference to a remote object offering a printing service will thus expire either when the conference finishes, i.e. in 3 days, or when the user exceeds its printing quota.

Another example of the impact of context information on leasing is related to the connection volatility phenomenon featured in mobile ad hoc networks. Transient disconnections should not affect an application, allowing both parties to continue their collaboration where they left off. Often, a remote reference may only be useful while the devices are in each others communication range. Note that a disconnection event is not related to time: it can happen at any point in time in mobile ad hoc networks due to the mobility of devices together with the limited communication range of devices. As a concrete example, consider an application to visualize the map and request sightseeing information of a site as the user moves about. The application will interact with devices embedded in different locations (e.g. buildings or touristic information points) to visualize the map and get information about a particular location. As the user moves out of range, the leased reference established between devices will become disconnected. However, such references can be immediately revoked since the application will search another suitable service in the environment to rebind the reference.

All these examples demonstrate that applications running on mobile networks need more semantic means to express the validity of leased references than merely timeouts. A leasing mechanism needs to incorporate context information in their semantics to provide developers more expressive language constructs to create and manage leased object references.

2.2 Adaptive Leasing: Changes on the state of a leased reference

In the previous section, we use renew-on-call leases which are automatically prolonged upon each method call. Although in those examples we assumed that the renewal time is the time interval specified when a leased reference is created, our leasing mechanism also allows developers to specify a renewal time. Renewing a lease means to change the state of the leased reference which is extended with a certain time interval.

Determining the proper renewal time is another issue to consider in any leasing mechanism. In particular, questions arise regarding how long this renewal time should be since it may depend on dynamic parameters of the system such as the number of clients. In order to abstract away as much as possible such low-level leasing management details, a leasing approach where the renewal time is dynamically adapted seems much more suitable. We observe that such adaptations are also based on context information. For example, objects exported in the context of a transaction, i.e., banking payment, could be renewed with a longer time interval in order to increase the level of robustness in the presence of transient failures so omnipresent in mobile ad hoc networks. On the other hand, leased references to remote objects that can be reconstructed with persistent data from a database could be renewed with a smaller time interval. Such examples illustrate that changes on the state of a leased reference performed by renewals may depend on context information.

In the context of service-discovery protocols, Bowers et al. have proposed self-adaptive algorithms for varying lease periods in response to the system size [1]. Although the authors specifically focus on restricting the lease time to guarantee minimum average responsiveness in a Jini system, this technique can be interpreted as another instance of how a low-level parameter in the computation context, i.e. responsiveness of the system, influences the frequency of renewal of leased object references. We argue in favour of a generalization of such adaptive techniques based on context information.

2.3 Distributed Garbage Collection Policies: Changes on the behaviour of a leased reference

We observe that changes in context not only affect the state of a leased reference as argued in the previous section, but also its complete behaviour, i.e, they introduce changes on the way how remote objects are collected. Recall the example of a user in a conference who wants to print one of their PDA files. The leased reference between the user and the printer located at conference eventually expired when the conference terminates. Consider now the same interaction at the user's home. In that case, the leased reference can be permanently kept since the user will return home eventually and print other files. This is a naive example but it illustrates that depending on the physical environment where devices are, different distributed garbage collection strategies can be applied. For example, a leased reference that never expires, i.e. a strong reference, can be applied if the user is at home or a leased reference for a concrete time interval if the interaction happens at a conference. We claim that there is no single strategy to reclaim objects in mobile ad hoc networks. However, a leasing mechanism should still provide means to adapt its behaviour in relation to changes on the context of the application.

3 Position Statement

Applications running on such mobile networks must adapt their behavior to different context events such as frequent disconnections of devices or location of mobile devices. Leasing provides a robust mechanism to reclaim remote objects in such sophisticated network topology. However, in current leasing approaches the validity of a leased reference is entirely based on timeouts. We have demonstrated by means of concrete examples that more expressiveness is required to determine the period of validity of a leased reference and how context information permeates to distributed memory management. As a result, we argue that leasing should to be augmented with context information: distribution memory management should be *aware* of the changes on the context of application to properly reclaim objects in mobile ad hoc networks. We thus propose *context-aware leasing* as a generalization of leasing which in response to context information provides automatic adaptation of the renewal time of leased references and dynamic adaptation of different garbage collection strategies.

We are currently implementing the extensions to leasing that we describe in this paper in AmbientTalk[2], a programming language especially designed for pervasive computing. We have already experimented with different combinations of the three types of leases supported and explored the integration of leasing with other language constructs such as futures. In future work, we want to explore a number of open issues such as how to deal with combinations of contexts events or performance considerations.

References

1. BOWERS, K., MILLS, K., AND ROSE, S. Self-adaptive leasing for jini. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications* (Washington, DC, USA, 2003), IEEE Computer Society, p. 539.
2. DEDECKER, J., VAN CUTSEM, T., MOSTINCKX, S., D'HONDT, T., AND DE MEUTER, W. Ambient-oriented Programming in Ambienttalk. In *Proceedings of the 20th European Conference on Object-oriented Programming (ECOOP)* (2006), vol. 4067, Springer, pp. 230–254.
3. GONZALEZ BOIX, E., VAN CUTSEM, T., DEDECKER, J., AND DE MEUTER, W. Language support for leasing in mobile ad hoc networks. Technical Report VUB-PROG-TR-07-08, Vrije Universiteit Brussel.
4. GRAY, C., AND CHERITON, D. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *SOSP '89: Proceedings of the twelfth ACM symposium on Operating systems principles* (New York, NY, USA, 1989), ACM Press, pp. 202–210.
5. HIRSCHFELD, R., COSTANZA, P., AND NIERSTRASZ, O. Context-oriented programming. To appear in the Journal of Object Technology (2007), `http://www.jot.fm`.
6. MCLEAN, S., WILLIAMS, K., AND NAFTEL, J. *Microsoft .Net Remoting*. Microsoft Press, Redmond, WA, USA, 2002.
7. PREUVENEERS, D., VAN den BERGH, J., WAGELAAR, D., GEORGES, A., RIGOLE, P., CLERCKX, T., BERBERS, Y., CONINX, K., JONCKERS, V., AND DE BOSSCHERE, K. Towards an extensible context ontology for ambient intelligence. In *EUSAI* (2004), pp. 148–159.
8. SALBER, D., DEY, A. K., AND ABOWD, G. D. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1999), ACM Press, pp. 434–441.
9. SUN MICROSYSTEMS. Java RMI specification, 1998. `http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html`.
10. WALDO, J. The Jini Architecture for Network-centric Computing. *Commun. ACM 42*, 7 (1999), 76–82.
11. WEISER, M. The computer for the twenty-first century. *SIGMOBILE Mob. Comput. Commun. Rev. 3*, 3 (september 1991), 94–100.

AmI: The Future is Now – A position paper

Johan Fabry and Carlos Noguera

INRIA Futurs - LIFL, ADAM Team
40, Avenue Halley, 59655 Villeneuve d'Ascq, France
`{johan.fabry|noguera}@lifl.fr`

Abstract. Because of the unique nature of the AmI domain, specifically the high amount of industrial involvement in this area, we fear that a classical long-term scenario for the use of academic research is no longer valid. In this paper we argue that the AmI research community should adapt to this context. To do this, we consider a short-term approach, and raise some points for discussion.

1 Introduction: A Pessimistic Scenario for AmI Research

The main goal of research at an academic level is to investigate fundamental problems. Concrete applications of these research results in industry can be typically estimated to be ten years into the future. This estimation however presumes that innovation in the industry proceeds at a steady pace, building on existing research work. The question we wish to raise here is how in the booming area of AmI, which has some radically new constraints, we can still perform research at academic level while being relevant to the industry.

The idealized scenario for research at an academic level is as follows. This research is the search for conceptually clean solutions for fundamental problems faced in the domain at hand. These are, therefore, inherently long-term goals, and the focus is not on creating industrial-strength solutions. The closest this is approached are demonstrators of the, possibly exotic, solutions. Achieving industrial applicability of such work is performed by industry R&D labs. This more short-term endeavor can rely on previously published results. It then provides an industrial-strength implementation of the conceptual solution as envisaged, a decade in the past, by academic research.

How does this idealized scenario apply to the AmI domain? We think it does not. Consider AmI development by the industry and look at the pace at which hardware development has progressed. PDA's have now become machines with enough power to run 3D-games. Run-of the mill cellphones have 16Mb of ram, smartphones have 64Mb of ram. Wireless connectivity, e.g., using Bluetooth, comes as standard on almost all these devices. Linked with this abundance of features is the software that is being developed for these systems by the industry **right now**. But it is not trivial find few research results of, say, seven years ago, that are directly applicable by industry in this context. For example, for the work on supporting connection volatility we shall describe in the next section, we were hard-pressed to find relevant related work. Because AmI has popped up on the software research radar so recently, there is not enough mature work that is usable by the industry. On the other hand, however, the hardware is out there now, and there is an immediate need for the software to exploit these features to the fullest.

Instead of the idealized scenario above, we fear for the following, more pessimistic scenario: In absence of research work that can be reused, the industry will create their own solutions to the problems they face. It is clear that AmI is a domain that promises to be very profitable, so in the extreme case, companies will just throw resources at the problem until a workable solution is found. As a result, the efforts performed by industry will vastly out-pace what is done in the research community. The industry will hack together their solutions **now**. These solutions will not be elegant, may be hard to develop and maintain, and have a significant performance or memory overhead, but they will work, and they will sell. As a result, by the time academia has a clean and elegant conceptual solution that is demonstrated to be applicable, say three years from now, too much time will have passed. The industrial hacks will have been in use for years, and have become entrenched. Furthermore, the industry will have moved on to new opportunities.

The important question that is raised by the pessimistic scenario is: how can academic research stay relevant in such a setting? We see two ways in which this can be achieved. A first way is to set our goals in the far future, so that we have ten years before the industry need is apparent. We think that this would however exclude a number of current-day research topics like, e.g., management of connection volatility or context-adaptation. The existence of these topics is quite well-known outside of research, they are good candidates to be addressed first by the industry. A second solution is to attempt to think in a shorter term: take smaller innovation steps that have the ability to be incorporated by industry quickly, say two to three years.

In our research we have chosen to take the second route. We attempt to take smaller steps that may be picked up by industry more quickly. To illustrate how we are taking this route, we will briefly discuss our current research on abstracting connection volatility, called Spoon-Graffiti. We will then raise a number of questions for discussion, before concluding this position paper.

2 Spoon-Graffiti: Targeting The Day After Tomorrow

To achieve a higher chance of being industrially relevant in a shorter time-frame, we have taken a pragmatic approach. We focus on making smaller conceptual contributions, that are implemented using technology that can be considered less exotic. The problem we are addressing is providing abstractions for connection volatility that are also able to reduce the cross-cutting nature of this concern.

2.1 Tagged Futures

The abstraction we propose is the use of tagged futures. Futures [3], also known as promises, have already been proposed as a mechanism to address issues with connection volatility, by Dedecker et al. [1]. We can use futures as empty placeholders for return values of network operations. When the return value of the operation is known, the future transparently changes to that value. This process is known as future resolution. As long as the contents of a future are not accessed, it can be passed around like any other object. However, when the future itself is accessed, the application blocks until the future is resolved.

Futures have the important advantage that they do not introduce any tangling of the connection volatility concern in the application. Their downside is however that they do not provide support for specifying offline behavior, the application simply blocks. We are investigating the concept of tagging futures with metadata that specifies mock values to be used instead of blocking. This then goes hand in hand with an update mechanism to allow these mock values to be replaced with the real data when the future is resolved, and an invalidation mechanism that reverts these to the mock values upon disconnection.

2.2 Spoon Graffiti: Implementing Tagged Futures in Java

Our experiments are performed using Java and rely on Java annotations for the expression of metadata. The annotation features of Java are known to be used in industry, e.g., they are extensively used in the Enterprise JavaBeans V3 standard, arguably an industrially important middleware standard. In general, to implement the behavior associated with annotations, an annotation processor is added as an extra compile-time phase. For our research, we have chosen to use the Spoon transformation engine [2], and our system is called Spoon Graffiti[1].

To use tagged futures in Spoon Graffiti, a developer adds `@Future` annotations to fields or getter methods, which takes an optional parameter. This indicates that these objects should behave as futures. If no optional parameter is given, read accesses of the fields or calls of the getter methods in an offline state will block. The optional parameter of the annotation specifies an expression to be returned as a mock value, instead of blocking. Future resolution needs to be implemented by the developer, as standard Java does not provide a powerful enough meta-level to perform this fully transparently. Future resolution is implemented in a method that is annotated by the `@Connect` annotation. The method should replace the contents of the different annotated fields with the value to be used when online. While this entails extra work, this does have the upside that the developer has full control over the future resolution mechanism. Similarly, the developer needs to implement methods for the update mechanism in the different classes that need to be aware of such updates. These methods are then annotated with the `@Online` tag. The invalidation mechanism is alike, where the `@Offline` tag is used. At compile-time Spoon Graffiti transforms the code of the application, as directed by the annotations, achieving the semantics associated with the annotations.

---

[1] Because the future is tagged.

2.3 Experimental Results

Experiments with tagged futures, and our implementation, show that the concept works quite well, and is applicable in typical AmI scenarios. A good example is a shopping list application, where, if a connection is made to a shop server when a customer is inside the store, extra information is obtained, such as the price of products. This information is actually a tagged future. In an offline state the default value for the price is specified as to be determined (the String `"TBD"`). The resulting behavior of the application is that items can be added to and removed from the list, both in an online and offline state. Whenever the application is online, i.e., has a connection to the shop, the `"TBD"` values are replaced with the actual values. When the connection is dropped these values revert to `"TBD"`. As a result, the prices always reflect the prices of the store the customer is in.

2.4 Discussion

Using standard Java has allowed us to fairly straightforwardly make a experimental shopping list application complete with user interface, reflecting the minimal behavior of such an application. Creating a more complete application is in line with the goals of our lab to produce fully functional platforms and tools. The rationale for this goal is that it allows issues to be raised which are overlooked in less complete implementations. The benefits of such a holistic approach are also present here. We first made an implementation using blocking futures, inspired from the proposal of Dedecker et al. [1]. Coupled with the user interface behavior this however caused our application to immediately lock up in an offline state. The reason for this is that the user interface tries to display the extra information, e.g., the price. This constitutes reading the future, which blocks. As a result the user interface thread of the application is blocked, and therefore the application locks up. Tagging the future with a default value avoid this erroneous behavior.

Using languages that are less research-oriented, and conforming to standards that are used in industry, Java and annotations in our case, is known to add a certain overhead to the experimentation process. This is as these languages were not designed for flexibility, and adhering to standards automatically adds overhead. However, because of our use of an existing tool for code transformations based on the annotation standard, we were able to perform our first experiments quite quickly. Furthermore, as we are using standard Java, we have full access to its IDE tools and myriad of libraries, allowing us to fairly quickly build the experimental application, complete with user interface and network communications. Contrast this with the other extreme of defining radical new languages or language constructs that are made to provide good support for experimentation. Their implementation can require a significant cost upfront, and they do not automatically benefit from IDE support (e.g., a debugger) and libraries (e.g., user interface support). These downsides of special-purpose languages are already known, of course, an interesting treatment of this theme is performed by Stroustrup in [4]. Furthermore, such radical approaches are not so likely to be adopted by the industry in the short term. As a result, the combination of this with a significant upfront cost negates the goal of making changes that can be incorporated by industry quickly.

3 Questions For Discussion

Our experience with Spoon Graffiti and the concept of having more short-term targets raise a number of interesting points for discussion. We itemize them here and outline our stance.

- Which of the two scenarios proposed in the introduction is the most pertinent? We assume the pessimistic scenario, hence our decision to aim for more short-term innovations that still are conceptually sound.
- What is the overhead of building prototypes due to the use of a more industrial language? The disadvantage is that such languages are not designed for experimentation, but the advantages in use of programming environments, libraries, and rate of industrial acceptance need to be taken into account.
- Are tools and standards that are starting to be accepted in industry, e.g., code transformations based on annotations, becoming powerful enough to use them for research? Our experience with code transformation, both in Java and previously in Smalltalk showed us that the use of a good code transformation tool (in Java) can make up for features of a superior language that are missing.
- Supposing that we take a short-term pragmatic approach, such as we have illustrated above, are we still slower than innovation performed in industry? If so, we still run the risk of becoming irrelevant.
- Should we take small, discrete, steps? Their advantages are not only that they can be adopted faster in industry, research-wise they also can be valorized faster.

4 Conclusions

The position we take in this paper is the following. Because of rapid development by industry in the domain of AmI, the classical long-term scenario for the usability of research results, has a high risk to make these results irrelevant. Instead, we should aim for more short-term results that are still conceptually clean. Furthermore, we should take into account how these can be transferred to industry more rapidly and consider the tradeoffs that this involves.

References

1. J. Dedecker, T. V. Cutsem, S. Mostinckx, T. D'Hondt, and W. D. Meuter. Ambient-oriented programming in ambienttalk. In *ECOOP 2006 - Object-Oriented Programming*, volume 4067 of *LNCS*, pages 230–254. Springer, July 2006.

2. R. Pawlak, C. Noguera, and N. Petitprez. Spoon: Program analysis and transformation in java. Technical Report 5901, INRIA, may 2006.
3. J. R. Halstead. Multilisp: a language for concurrent symbolic computation. *ACM Trans. Program. Lang. Syst.*, 7(4):501–538, 1985.
4. B. Stroustrup. A rationale for semantically enhanced library languages. Library-Centric Software Design workshop at OOPSLA05, 2005.

Introducing Context-Awareness in Applications by Transforming High-Level Rules

Carlos Andrés Parra, Maja D'Hondt, Carlos Noguera, Ellen Van Paesschen

INRIA Futurs - ADAM Team
Parc scientifique de la Haute-Borne
40, Avenue Halley
59650 Villeneuve d'Ascq Cedex
{carlos-andres.parra, maja.d-hondt, noguera, ellen.vanpaesschen}@lifl.fr

Abstract. In the last years, we have witnessed the increase in the popularity and capabilities of mobile technologies. This evolution has enforced the idea of smart environments, in which devices are aware and able to react to changes in their environment. In this position paper we describe a specific approach for the development of context-aware software. We propose to make existing applications context-aware by means of three main components: context models, high-level rules and code-generation processors. We present each component and analyze the issues related to the development of context-aware software following this strategy.

1 Introduction

The proliferation of mobile devices in everyday life is moving from fixed computers to a wide range of smart and mobile devices with wireless networking capabilities. Mobility opens the door for new kinds of challenges in software, one of such is the capability to adapt applications to be *aware* of the context and to react properly to its changes. By context we informally mean a set of situations in a given environment at a given time. In order to achieve this *awareness*, devices and applications should interact with each other to share context information, gathered mostly through physical sensors that can detect different data such as temperature, time or presence of other nearby devices.

Context-aware applications require mechanisms to specify actions when context changes. One way to do this is through the definition of a *language* in which we are able to express specific behavior in the following form: *for some specific context conditions, perform an action.* This structure has two main parts, the conditions on the context and the action that has to be performed when the conditions are met. In this paper we discuss the issues related to guiding the behavior of applications based on the events produced by changes in the context.

Model-driven Architecture (*MDA*) [6] aims for a higher level of abstraction in software development, by separating the business elements from the specific details related to the platform and implementation issues. We intend to follow the same strategy with the definition of high-level *rules* to express the behavior and the preferences of users in a platform and implementation-independent way.

These rules can be used then as an entry point to a process of code transformation.

This paper is organized as follows. In section 2 we present a brief summary of the related work. In section 3 we first introduce our approach to the development of context-aware software. Then we describe the three main components of our proposal: a context model that represents the basic structure of context information, a rule language to express the context-action pairs and a process that adapts the application based on the context model and the rules. Finally, in section 4, we present some conclusions.

2 Related Work

There are several related areas involved in the development of context-aware software. We introduce briefly some of the most relevant, classified in three different categories. Firstly, the middleware and the implementation details. Secondly, the representation of context information, and finally, the use of rules to express behavior in terms of high-level elements.

On the implementation side, the need for new middleware arises. As stated in [13], current middleware technologies are not adequate to handle the restrictions that mobility and smart environmental systems impose. These characteristics include: volatile connections, processing and memory restrictions on mobile devices, narrow communication channels, reduced screens, restricted input mechanisms, and the list goes on. A set of context-aware implementations can be found in the literature, which, represent working prototypes of context-aware applications. Some of these implementations are `Hydrogen` [5] a three layer architecture and a framework to support context-awareness, `Gaia` [11] a middleware infrastructure to support the development of mobile and ubiquitous systems, and `CybreMinder` [2] a prototype application implemented using the `Context Toolkit` [12], to handle reminders associated with context information. We assume as a basis for our approach that a context-aware middleware exists and that is able to propagate updated context information.

An approach to develop adaptive service-oriented is presented in [8]. In this approach, the authors aim for the generation of adaptation points for a given application, based on meta data. Then, at runtime and using these points of adaptation exposed as services, context-sensitive aspects can interact with the original application and adapt properly to the situation. In our approach, we also start from meta data about the applications, but we also use high-level rules that describe the desired behavior of applications to face a context change, and information about other devices and applications. With this elements, we propose to build code-generation processors to generate the parts that the application requires to be context aware. Our approach and all the elements on it, will be described in more detail in section 3.

A different field is concerned with the representation of context information. A generic ontology is presented in [10]. It is based on four main concepts: user, environment, platform and resources. Here, ontologies are mainly employed to enable communication across different devices in the same network. The Unified Modeling Language (UML) can also be used to model context, as proposed by `ContextUML` [1]. This is a language for the model-driven development of context-aware web services. Here the models are used to separate the definition and information related to the context from the specific implementation. There are other characteristics that make context information difficult to model. As stated in [4] and [3], sometimes it is necessary to differentiate between static and dynamic information. Static information refers to data that does not change, for example, the name or date of birth of a user, whereas dynamic information refers to data that may change over time as for example the location of a user.

As we will show in detail in Section 3.1, we borrow some concepts from these and other approaches to construct our own context information model using an UML class diagram.

Finally, rule languages are used in some cases to achieve context-awareness. `CRIME` [7] for example, is a prototypical implementation of the `Fact Space Model`, which is a coordination language that provides the applications a view of their environment. The rules in `CRIME` describe the behavior of the applications according to the context information. `CRIME` also deals with disconnection by invalidating the facts and the conclusions that are drawn from devices that are no longer available in the environment. As we describe and argue in Section 3.3, it is our intention to *compile* context rules, rather than to *interpret* them. A result of our approach is that we will not be able to invalidate inferred facts when the context changes.

3 Context-Aware Applications

In this section, we present our approach to the development of context-aware applications. We define high-level rules to express software adaptation to context. Using a rule language, it is possible to specify the preferred actions to follow changes in context. The rules are then transformed into software assets that are merged with the existing application.

To illustrate this approach, let us consider for example a basic jukebox application. This application has a standard functionality: play a song, surf a list of songs, and turn the volume up and down. If we wanted to introduce context-awareness, we would need to transform the application to add some extra behavior, for example, to detect if there is a cellphone nearby and, subsequently, to turn the volume down automatically. In such a transformation we do not want to create new functionality to turn the volume down, but to use such functionality whenever a cellphone is present in the context. To achieve this functionality we propose a process like the one shown in Figure 1.

In this diagram, there are three main parts. First, we need a model to express context information (`ContextModel`). That way, we are able to represent information related to users, devices, location, time, etc. The rules(`High-level Rules`), on the other side, represent the actions to be followed by the context-aware application when a certain change in the context occurs. Finally, there is

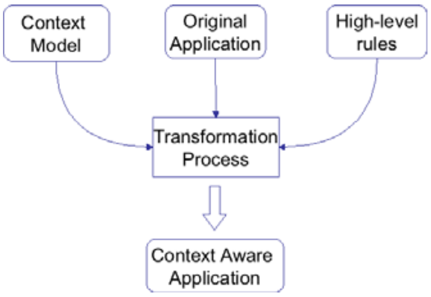


Fig. 1. Process to transform non-context-aware applications

a transformation process in charge of extending the (non-context-aware) original application, taking as input the context model and the rules.

3.1 Context Model

Regardless of the variety of devices, there should be a common understanding between all the participants about the context information they want to share. To achieve this, we first design a context meta-model (left side of Figure 2), to describe the classes and relationships in a context model for a given situation. The first element is the `Context` which stands as the root of the meta-model. A `Context` contains `ContextDescriptor` and `Entity` elements. The `Entity` elements represent all the actors of an environment, for example the users and their devices. `Entity` as well as `ContextDescriptor` elements may have a group of `Attribute` elements. Finally the `Entity` offers functionality represented by the `Service` element. On the right side of the (Figure 2), there is a context model that conforms to the context meta-model. This model represents a particular context in which there are `Jukebox` and `Cellphone` elements, contained in an `Environment`. Here we use the stereotypes to represent the relation between the elements in the model and their corresponding meta-class in the meta-model. The context model should be used as the language to express behavior and actions to follow in the high-level rules.

Aside from having a structure to represent context information, it is necessary to define the responsibilities for populating the context model. In context-aware software development, where a wide variety of devices and applications interact in the same environment, fundamental questions have to be addressed such as, what part is responsible for keeping the context information up to date, and what update strategy is going to be used. For example, when someone holding a cellphone arrives, it is possible that the jukebox application retrieves the updated context every once in a while and detects the new device, but it is also possible that an underlying middleware triggers an event when the cellphone is detected. In our proposal we assume the presence of a middleware that generates an event every time a change in the context is detected.

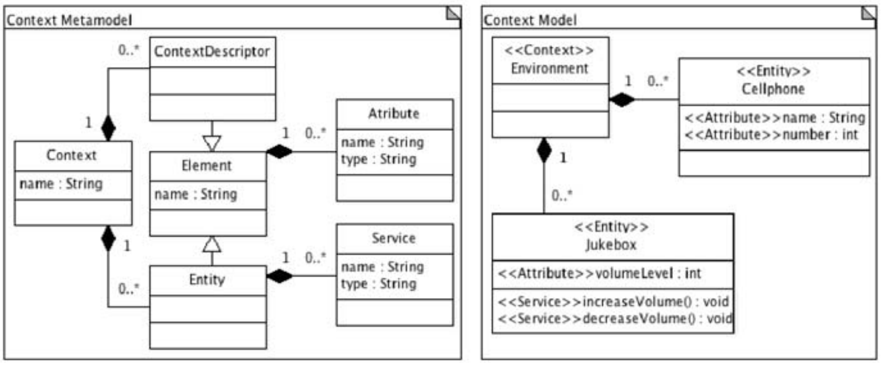


Fig. 2. Metamodel and model of context information

3.2 Rule Language

As we mentioned in Section 1, rules play an important role in our approach. By writing rules in terms of the context model, we think it is possible to express the desired behavior of the applications. We consider that a scheme where rules express conditions of the type `IF (condition) THEN (action)` is appropriate for our purposes. The condition as well as the action can be written in terms of the context model as is shown in the following example:

```
IF(Cellphone.name = MyCellphone)
DO Jukebox.decreaseVolume
```

Here we present a single rule that evaluates a condition and describes an action to follow, if the condition is met. In this case, the jukebox application is asked to decrease the volume if a cellphone with the name *MyCellphone* exists in the context model. One important question to answer about the rules, is when they need to be evaluated. This is directly related to the mechanisms we use to keep the context information updated. As we mentioned en the previous section, we assume the existence of an underlying middleware in charge of populating the context and of notifying the application whenever a change in the context is detected and the model needs to be updated.

3.3 Program Transformation

Given that in our approach we propose to extend existing applications so that they are sensible to changes in the ambient context, a module that transforms these applications is necessary. Such transformation must insert into the application the concepts described in the context model, and it must modify its control-flow to reflect the behavior specified by the rules.

We opt for a compile-time adaptation of the program rather than a dynamic one to better cater for the limitations of current ambient execution environments. Indeed, mobile devices deal with restricted resources in terms of available memory and processing power. By relying on source code transformation, our approach does away with the overhead of additional libraries or custom execution environments (for example, a modified Java VM) that would be needed for dynamic adaptation. In addition to this, by including the context model description and rule set definition at compilation time, it becomes possible to *compile* rather than *interpret* the context rules. This will allow the transformation module to optimize their implementation, although, to what degree remains to be seen.

To perform the transformation, programmer inserted annotations in the base application, as well as the context model and rule set, will be consumed by a source code processor. This processor will be responsible for the weaving of the rules and context entities in the base program, while leaving the application's original function intact. To implement the transformation for programs based on Java, we use the Spoon framework [9], since it provides a fine-grained representation of the program and offers several facilities for the processing of annotations.

Nevertheless, by using a compile-time approach, we cannot offer certain capabilities to users as to create or modify rules at runtime. To minimize the impact of such restrictions, it is imaginable to create means to automatically compile and deploy new rules. However, the development of this kind of features remains as future work and is out of the scope of this paper.

4 Conclusions

Despite the recent wave of attention in context-aware systems, we think that there is still a gap in order to bring context modeling to a real implementation. First, we believe that models ought to be used not only to exchange design ideas, but also as a core of the development of context-aware applications. We envision context models being used in transformation and generation processes. As such, code will be generated in order to adapt applications according to variations on their context.

This paper presents our approach to making existing applications context-aware by means of context models, high-level rules and code-generation processors. This approach is currently mostly in the conceptual phase, although most parts have been implemented or can be largely based on our previous work and experiences. The largest challenge is without doubt determining the locations in the existing application where the transformed high-level rules and context information need to be inserted. Another fundamental question is to what extent we will be able to generalize our approach, independently of the application that we wish to transform.

References

1. *ContextUML: a UML-based modeling language for model-driven development of context-aware Web services*, 2005.
2. A. K. Dey and G. D. Abowd. Cybreminder: A context-aware system for supporting reminders. In *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 172–186, London, UK, 2000. Springer-Verlag.
3. K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems, 2005.
4. K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems, 2002.
5. T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, Washington, DC, USA, 2003. IEEE Computer Society.
6. J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, Object Management Group (OMG), 2003.
7. S. Mostinckx, C. Scholliers, E. Philips, and C. Herzeel. Fact spaces: Coordination in the face of disconnection, 2007.
8. H. Mugge, T. Rho, and A. B. Cremers. Integrating aspect-orientation and structural annotations to support adaptive middleware. In *MAI '07: Proceedings of the 1st workshop on Middleware-application interaction*, pages 9–14, New York, NY, USA, 2007. ACM Press.
9. R. Pawlak, C. Noguera, and N. Petitprez. Spoon: Program analysis and transformation in java. Technical Report 5901, INRIA, may 2006.
10. D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere. *Towards an Extensible Context Ontology for Ambient Intelligence*. 2004.
11. M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mobile Computing and Communications Review*, 6 (4):65–67, 2002.
12. D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM Press.
13. T. Strang and C. L. Popien. A context modeling survey, September 2004.

Reasoning About Past Events in Context-Aware Middleware

Eline Philips, Christophe Scholliers,Charlotte Herzeel and Stijn Mostinckx
{ephilips, cfscholl, caherzee, smostinc}@vub.ac.be

Programming Technology Lab
Vrije Universiteit Brussel

1 Introduction

The hardware advances in networking technology of the past few decades have resulted in novel kinds of distributed systems, commonly referred to as *mobile ad hoc networks*. Such networks are populated by small, mobile handheld computers or cellular phones interconnected by highly volatile wireless communication links. Whereas the resources of these devices tend to be limited, their true strength stems from their ability to seamlessly integrate computing into our everyday life. A crucial factor to preserve this integration is that devices and the applications they host respond to the context in which they are situated.

The development of context-aware applications is widely supported by a large variety of frameworks such as JCAF [1], WildCAT [2], and Lime [6]. A common trait of these frameworks is that they cater for an event-driven programming style, where reactions are triggered as context events are fired. The one-to-one mapping between events and reactions requires that a reaction which is to be triggered when two context events occur simultaneously should install two separate event-handlers and hard-code the combination logic in the event handlers.

In previous work we have presented Crime, a logic coordination language which allows expressing reactions to be triggered upon the simultaneous occurrence of various context events [5]. By capturing these constraints in general-purpose rules, additional conditions can be imposed in a declarative fashion. Hence, Crime alleviates the need for imperative checks to be performed in body of the context event handlers.

On the other hand, we have also built Halo, an aspect-oriented extension to the Common Lisp Object System that supports reasoning about the execution history of a program in its logic-based aspect language, and thus enables expressing context-aware aspects [4].

In previous experiments, we have noted that although Crime allows one to reason about the current context, it can be equally interesting to reason about context events which were fired in the past. The next section provides an example scenario which illustrates a clear use case for this behaviour. This position paper proposes to integrate features of Halo into Crime as a basis to explore reasoning about past events in context-aware middleware.

2 Scenario

In previous work we have presented the following scenario as an example of a complex context-aware system whose behaviour can be expressed in a declarative fashion using CRIME [5].

> Alice, Jim and Bob are students which share an apartment. A great deal of their life is all about music. When one of them is relaxing in the joint living room of their apartment, it is quite common to find their jukebox playing music. Unfortunately, they do not always share one another's taste in music. Whereas this might be a recipe for endless quarrels in any other situation, there is no arguing over who is in charge of choosing the music being played. This is due to the fact that the jukebox is in fact a small computer (a Mac Mini in our setup) which combines information regarding the presence of its users with their respective musical preference to construct a playlist which is acceptable for all present users. Moreover, if Alice, Jim and Bob invite some friends, their musical preferences can be taken into account as well. Finally, the jukebox also stops playing automatically when it detects that no-one is present. [5]

In this paper we impose an additional constraint on the system. In order to avoid playing the same songs over and over again, the jukebox should avoid playing tracks that its users have heard while they were previously in the room. As we will demonstrate, keeping track of such songs in CRIME – such that the reasoning engine can actively use this knowledge – is greatly facilitated by the introduction of HALO's temporal operators. However, before delving into more advanced topics, the next section first presents the core of CRIME and HALO.

3 Contextual and History-based Reasoning

3.1 Contextual Reasoning in a Mobile Environment

CRIME is a coordination language dedicated to reason about context information in a mobile environment [5]. The language consists of two essential building blocks: a data model and a programming model.

CRIME's data model – called the *fact space model* – extends upon the federated tuple space model popularised by LIME [6]. The chief difference between both systems is that tuple spaces correspond to a white board where messages can be published, read and removed, whereas CRIME considers a distributed knowledge base describing the context of all nearby devices. The chief *operational* difference between both models is that in a distributed knowledge base both the *assertion* and the *retraction* of a fact are meaningful events which may trigger reactions. The retraction of facts is automatically triggered when context providers disconnect, allowing programs to respond to the (presumed) invalidity of the information they provided.

Crime's programming model consists of a rule-based formalism with a syntax akin to Prolog to describe the causal relation between (a combination of) context events and the corresponding context event handlers. The Crime rules are interpreted by a Rete network which allows for an event-driven and optimised reasoning engine [3]. Rete is a forward-chained algorithm which actively derives every valid conclusion from given a set of facts. As a consequence, Crime applications do not need to manually query the fact space, as the appropriate context event handlers are triggered automatically when the reasoning engine is notified of changes to the fact space.

3.2 History-based Aspects Using Logic

The need to reason about past program state in order to correctly handle events does not only manifest itself in context-aware systems for mobile ad-hoc networks. An interesting parallel can be made with aspect-oriented programming languages offering support for expressing context-dependent behaviour, in e.g. the domain of business rules. One such business rule could be that in an e-shop application, discounts a customer receives upon checkout should depend on whether a discount was active when the user added the item to the shopping cart. This strategy is to be preferred over taking into account discounts active at the checkout, since customers respond badly when they see items they have selected when a discount was active have become more expensive [7, 4].

Similar to event-driven programming approaches like Crime, aspect-oriented pointcut languages allow responding to events (in this case in the program execution) using, for example, a combination of `execution` and `cflow` predicates. However, they typically fall short when events are considered relevant which are no longer active (i.e. they are no longer on the dynamic call stack).

Context-aware aspects introduce an extensible pointcut language where context information can be aggregated into a *context snapshot*. These snapshots can be used to determine whether a pointcut occurs in a conceptual context which is no longer necessarily tied to the dynamic call stack [7]. Whereas that framework is very general, the need to manually snapshot context at certain points in time imposes an imperative style where programmers are actively considering how reasoning about past events should be facilitated.

Halo is an aspect language, built upon the idea of context-aware aspects, yet introducing them in the form of a logic-based pointcut language, enabling a declarative programming style [5]. In Halo, context is modelled as logic facts. Pointcuts can be restricted to such a context, by linking join point conditions to context facts. To make it possible to describe past join points and the past program context in wich they occured, primitives from temporal logic are integrated in the language. Hence pointcuts are aware of the (past) context in which join points occurred

As both Crime and Halo use the Rete algoritm to implement their reasoning engines, it seems plausible that Crime's support for distribution, and Halo's support to manage the fact history can be combined into a single framework.

4 Implementation of the Scenario

We illustrate how the programming model of CRIME accommodates the development of the jukebox application described in section 2. In this section we focus on the actual rules to script the jukebox, and assume the presence of facts of the form `location(‘‘Alice’’, ‘‘DiningRoom’’)` to represent the current location of users and `prefers(‘‘Alice’’, ‘‘Rock’’)` to describe their musical preferences. These facts are published into the distributed knowledge base such that the jukebox application has access to them.

The rule presented below triggers the context event handler `Toggle`. The rule keeps track of the amount of persons which are currently in the jukebox room. When one person is detected the room, this rule will be activated. This implies that the `activate` method of the context event handler will be called, which in turn will start the music player. Similarly, when no-one is left in the room the `deactivate` method will ensure that the music player is stopped.

```
:Toggle() :-
        location(?person, ‘‘Jukebox Room’’).
```

Listing 1.1. `Toggle` Rule

The rating the jukebox attributes to a particular genre depends on both the current number of people in the room and the number of people who prefer the genre. The following two rules calculate these two values by making use of the findall and bagof constructs borrowed from PROLOG. The `findall` construct used in the `total` rule, accumulates all persons located in the room in the *?persons* variable. Similar to the `findall`, the `bagof` construct used in the `category` rule also accumulates all persons in the room but groups them according to the specific genre they like.

```
total(?quantity) :-
        findall(?person, (
                location(?person, ‘‘Jukebox Room’’)),
                ?persons),
        length(?persons, ?quantity).

category(?genre, ?quantity) :-
        bagof(?person, (
                location(?person, ‘‘Jukebox Room’’),
                prefers(?person, ?genre)),
                ?persons),
        length(?persons, ?quantity).
```

Listing 1.2. `total` and `category` Rule

The quantities calculated by both rules presented above are used to trigger the `UpdateRatings` context event handler provided by the jukebox. This event handler will update the ratings of the songs according to the present users' combined preference. These ratings are used in turn by the music player to compile a playlist where highly rated music is featured more often.

```
:UpdateRating(?genre, ?rating) :-
        category(?genre, ?absolute),
        total(?total),
        rating is ?absolute / ?total.
```

Listing 1.3. `UpdateRatings` Rule

4.1 Introducing HALO's Temporal Operators in CRIME

To the best of our knowledge, contemporary frameworks for the development of distributed context-aware applications do not provide reified support to reason about past contexts. In contrast with HALO, reasoning about the past is done manually by recording and manipulating past events in the code of the context event handlers. The current incarnation of the CRIME coordination language, as described in section 3.1, exhibits the same shortcoming. However, its event-driven reasoning engine makes it a suitable candidate to introduce the temporal operators developed in HALO.

As a starting point, we propose to introduce the following set of temporal operators from HALO. Note that temporal operators are always implicitly parametrised by the fact that precedes them. That is to say, they have implicit access to the timestamp t_1 at which this fact was triggered.

sometime-past This operator takes one explicit argument (timestamped with t_2) and allows only matching facts such that $t_1 > t_2$. In other words, a rule body of the form `f1(), sometime-past f2()` only matches facts `f2` which occurred **before** a matching fact `f1`.

most-recent This operator has similar semantics as `sometime-past` with the explicit restriction that only one matching fact can be returned. In other words, a rule body of the form `f1(), most-recent f2()` only matches **a single** fact `f2` which occurred **before** a matching fact `f1`.

since This operator takes two explicit arguments (respectively timestamped with t_2 and t_3) and matches facts such that $t_1 > t_3 > t_2$. In other words, a rule body of the form `f1(), since (f2(), f3())`, matches events `f3` which occurred **between** `f2` and `f1`.

Fig. 1. HALO's temporal operators.

These three operators provide an expressive set of building blocks to identify relevant past events. To illustrate this, we complete the scenario described in section 2 by automatically removing songs from the playlist which a user has heard when he has last seen in the jukebox room. The code excerpt below is an outline for a possible implementation.

```
1  :DeleteFromPlaylist(?person, ?songs) :-
2          location(?person, ‘‘Jukebox Room’’),
3          most-recent( not location(?person, ‘‘Jukebox Room’’) ),
4          since (
5                  most-recent( location(?person, ‘‘Jukebox Room’’) ),
6                  findall( ?song,
7                          played(?song),
8                          ?songs).
```

Listing 1.4. Implementation using temporal operators

The rule in the code excerpt is triggered whenever a person enters the jukebox room (line 2). At this point in time, the system recalls the last time when the person left the jukebox room (line 3). This timestamp is used as the end of a `since` interval (line 4), the starting point of which is the previous time the user was spotted by the system (line 5). The fact being sought for in this interval is a `findall` which accumulates all songs played in the interval (lines 6-8). These songs are then deleted from the current playlist (using the `DeleteFromPlaylist` context event handler) as they should not be repeated (line 1).

5 Position Statement

This position paper has identified the need for mobile context-aware applications to be able to reason about past events in order to better adapt their behaviour to the current context. Rather than deferring the reasoning to explicit checks in the context event handlers, we advocate the use of a logic coordination language which incorporates temporal operators as basic language constructs. Such temporal operators have already proven their merit for aspect-oriented programming, a setting which is not dissimilar from the one proposed in this paper. We therefore consider them to be a prime candidate for inclusion in context-aware application toolkits and intend to prepare a proof-of-concept implementation which combines features of CRIME and HALO to be presented at the workshop. With this experiment, we intend to contribute a discussion of problems related to integrating temporal reasoning in mobile ad-hoc networks (e.g. volatility and management of distributed historical data etc.).

References

1. J. E. Bardram. *The Java Context Awareness Framework (JCAF) A Service Infrastructure and Programming Framework for Context-Aware Applications.* 2005.
2. P. David and T. Ledoux. Wildcat: a generic framework for context-aware applications. In *Proceeding of MPAC'05, the 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing*, 2005.
3. C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. In J. Mylopoulos and M. L. Brodie, editors, *Artificial Intelligence & Databases*, pages 547–557. Kaufmann Publishers, INC., San Mateo, CA, 1989.
4. C. Herzeel, K. Gybels, P. Costanza, and T. D'Hondt. Modularizing crosscuts in an e-commerce application in lisp using halo. ILC 2007, 2007.
5. S. Mostinckx, C. Scholliers, E. Philips, C. Herzeel, and W. D. Meuter. Fact spaces: Coordination in the face of disconnection. In *Proc. of 9th Int. Conf. on Coordination Models and Languages*, 2007.
6. G. P. Picco, A. L. Murphy, and G.-C. Roman. LIME: Linda meets mobility. In *International Conference on Software Engineering*, 1999.
7. É. Tanter, K. Gybels, M. Denker, and A. Bergel. Context-aware aspects. Proc. of the 5th International Software Composition Symposium, 2006.

Ambient-Oriented Programming in Fractal

Aleš Plšek, Philippe Merle and Lionel Seinturier

Project ADAM LIFL, INRIA-Futurs,
Université des Sciences et Technologies de Lille (USTL), FRANCE,
{ *plsek* | *merle* | *seinturi* }*@lifl.fr*

Abstract. Ambient-Oriented Programming (AmOP) comprises a suite of challenges that are hard to meet by current software development techniques. Although Component-Oriented Programming (COP) represents promising approach, the state-of-the-art component models do not provide sufficient adaptability towards specific constraints of the Ambient field. In this position paper we argue that merging AmOP and COP can be achieved by introducing the Fractal component model and its new feature : Component-Based Controlling Membranes. The proposed solution allows dynamical adaptation of component systems towards the challenges of the Ambient world.

1 Introduction

Ambient-Oriented Programming (AmOP) [1] as a new trend in software development comprises a suite of challenges which are yet to be addressed fully. So far, only a few solutions facing the obstacles of ambient programming have been developed. In this paper we focus on AmbientTalk [1] since in our opinion it represents one of the most sophisticated solutions.

Although AmbientTalk conceptually proposes a way to implement applications for the ambient environment, this is achieved by defining a new programming language. Consequently, AmbientTalk potentially introduces a steep learning curve for the developers. From this point of view, it is reasonable to search for an approach which uses well-known techniques and is powerful enough to face the obstacles of ambient programming. We believe that these requirements can be met by the introduction of Component-Based Software Engineering techniques.

Our goal is therefore to propose a dynamically evolvable middleware system based on the Fractal component model [2] to facilitate development of applications adapted to the ambient environment. To reflect the goal, this position paper is summarized as follows. Section 2 anchors our research into the context of AmOP and Component-Oriented Programming (COP). Section 3 proposes our approach to the challenges of Ambient Programming. Section 4 describes the experiment we have conducted to demonstrate the abilities of the proposal. Section 5 concludes.

2 Context

2.1 Ambient-Oriented Programming

Ambient Intelligence [3] represents a new trend of computing where technology is gracefully integrated into the everyday life of its users. This new field in distributed computing comprises wireless devices which spontaneously communicate with each other.

The specific character of a highly dynamical mobile environment however imposes special constraints (facing the connection volatility, the ambient nature of resources, etc.). These challenges form a new group of programming techniques – Ambient-Oriented Programming. Although the main stress here is laid on facing the so-called Hardware Phenomenon [1], we believe that software engineering aspects supporting more effective development of ambient oriented applications should be more emphasized. Therefore we additionally pose the following requirement:

- **Evolvability.** Since the ambient environment is from its nature highly dynamical, the solution has to keep up with hardware evolution and to addresses specific needs of the target environment. Consequently, the ability to develop systems which can dynamically evolve towards changing conditions and mission goals is essential.

AmbientTalk is a programming language that explicitly incorporates potential constraints of the distributed mobile environment in the very heart of their basic computational steps, thus addressing directly the obstacles of application development for mobile devices. To deal with the ambient environment characteristics, AmbientTalk implements several features. For this discussion we focus on two keystone concepts: Ambient Reference and Non-blocking Futures.

The ***Ambient Reference*** concept represents a powerful solution to referencing objects in ambient environment. Ambient reference operates in two states - unbound and bound. When an ambient reference is *unbound*, it acts as a discovery channel looking for remote service objects in the environment to bind to. Once such a suitable object is found, the ambient reference becomes *bound*. Once bound, an ambient reference is a true remote object reference to the remote service. When the service object to which an ambient reference is bound moves out of communication range, the ambient reference can become unbound again. Then it acts as a peer discovery mechanism again and tries to rebind to the same or another matching service.

Since the concept represents an asynchronous way of communication, it is necessary to face the challenge of returning the result of a client's request. To provide this, the ***Non-blocking Futures*** concept is introduced. It allows to associate a block of code which will be triggered on the client once its request is resolved – the returning value from the server is thus processed. The main motivation for employing this feature is to manage the returning value processing without the introduction of callback methods.

Other solutions to the ambient environment challenges exist. Due to the space limitations, we do not present them here, but refer to [4].

2.2 Fractal Component Model

The Fractal component model [2] is a light-weight component model, focused on programming language concepts. In contrast to other component models, such as EJB, .Net or CCM, it does not require the extra-machinery supporting its functionality. The model is built as a high level model and stresses on modularity and extensibility. Moreover it allows the definition, configuration, dynamic reconfiguration, and clear separation of functional and non-functional concerns.

The component model is hierarchical in the sense that a component may be primitive, or composite. The central role is played by interfaces, which can be either business or control. Whereas business interfaces are external access point to components, control interfaces are in charge of some non-functional properties of the component, for instance its life-cycle management, or the management of its bindings with other components.

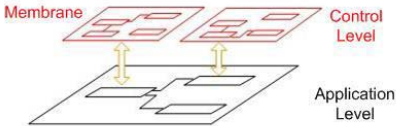


Fig. 1. Component-based Control Membranes

Component-based Control Membranes (CBCM) The abilities of the Fractal component model are even more extended by a new feature introducing the component-based architecture for the control environment surrounding components. Similar to EJB's containers, the Fractal component model features a controlling environment, called *membrane*. This supports before mentioned non-functional properties of components. However, in contrast with fixed structures of EJB containers, the control membrane of a component is implemented as an assembly of so-called control components and can dynamically evolve. The whole idea is depicted in Fig. 1.

Not only does this approach brings effective development in the sense of reusability and transparentness, but the main benefits lay in the ability to introspect and dynamically reconfigure the architecture of the control layers of each component. Moreover, the membranes can be designed individually thus precisely fitting the needs of specific components. This leads to a reflective component model, where both the business layer and the layer which controls it are implemented using components.

3 Ambient-Oriented Programming in Fractal

Considering the challenges of AmOP we believe that binding both Ambient- and Component-Oriented Programming techniques together would bring numerous benefits to the world of Ambient Intelligence.

Our vision is to use COP to develop dynamically evolvable software systems that are able to adapt themselves towards the challenges of AmOP. Additionally, we propose to use COP also to develop a middleware layer that will support the ambient nature of these systems and shield the developer from potential complexities of designing ambient aware systems.

To achieve a higher level of symbiosis between both the system and the middleware, we propose to use the CBCM feature of Fractal to implement the middleware layer. It enables us to precisely deploy ambient functionality only to those components where it is needed. To achieve this, we lay out the following tasks:

- **Component-Oriented Approach.** COP allows to achieve clear separation of concerns, desired granularity and effective management of the life-cycle and concurrency properties. These characteristics extensively support adaptability, which is highly demanded property in ambient-aware systems.
- **Fractal CBCM Application** We believe that the component oriented architecture of the Fractal membrane provides the necessary extendability to host the features supporting the ambient nature of the software applications. Therefore, extending the Fractal membrane is the key design choice.
- **Ambient Middleware.** The ambient middleware emerges from the implementation of previous points. The ambient functionality is spread among the components in the application and implemented through the component-oriented membrane extensions, thus virtually forming a middleware layer that can evolve.

4 Ambient-Oriented Middleware : Experimental Implementation

To demonstrate the potential abilities of our proposed solution, we have conducted an experiment that implements a middleware layer supporting the Ambient Reference and Non-blocking Future concepts - the fundamental features of AmbientTalk.

The experimental implementation involves two actors : a *server* that provides a given service and a *client* that is searching for the service and that spontaneously enters and leaves the communication range of the server. The task is to use Ambient Reference and Non-blocking Futures concepts and thus hide the ambient character of the environment. To focus only on the implementation of these two concepts, we have extended this system by a third actor - a *discovery service*, which manages the service provisions and requirements in the environment. The discovery service operates at the middleware layer, communicating only with ambient-aware parts of actors.

4.1 Membrane Extensions

The adaptability of the membrane, described in Section 2.2, is the key feature we want to employ during the implementation of our solution. As already said, each component membrane can be extended individually thus perfectly fitting the specific needs of particular component. Applied to our experiment, we extend membranes of components implementing the communication between both actors with the ambient functionality. Thus the functionality is deployed only to specific components, they are extended with following units:

Ambient Controller The ambient controller is a new managing unit introduced into the component membrane architecture. The task of the controller lays in managing the ambient functionality of the component. Particularly, the key responsibilities of this unit are the control of the ambient references and the deployment of ambient interceptors.

Ambient Interceptor The interceptors deployed on every component interface allow to trace the component communication and to adjust the communication towards the specific needs of the ambient environment. E.g. either forward the messages to the recipient or buffer them when the recipient is unavailable.

4.2 Ambient Component

Through the membrane adaptation we are able to achieve the ambient functionality, obtaining an *ambient component.* The business code is not affected thus putting no extra burden on the developer. Moreover, ambient-awareness extensions are transparent and can co-exist with the remaining unmodified components – achieving that potentially every Fractal component systems can be extended.

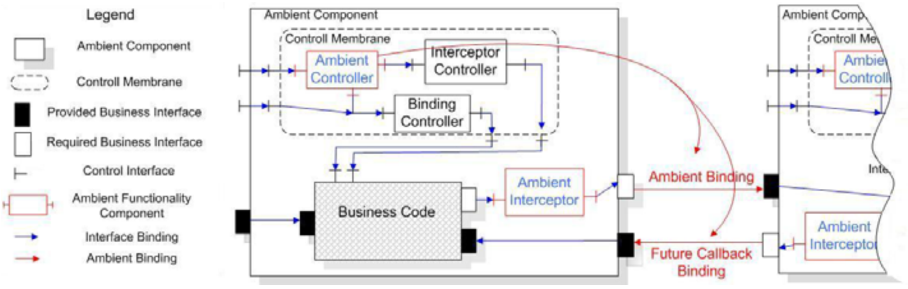


Fig. 2. Ambient Component

When applying the approach to our experiment, the membranes of components participating in the ambient communication are extended by the ambient controllers. An ambient binding, a component-oriented variant of the Ambient Reference concept, is instantiated once the *discovery service* announces that a client's desired service becomes available. Then, the ambient controller creates the binding between ambient components and deploys an ambient interceptor on the interface of the client component. Once the ambient binding is instantiated, the ambient controller keeps this reference updated and notifies the interceptor every time the discovery service announces that an ambient resource is unavailable. The role of the interceptor is to either transmit messages to the server interface or to buffer them when the ambient service is currently unavailable.

To implement the Future concept, the callback technique is used even though the original implementation of the concept in AmbientTalk avoids a callback. Every communication of the component with its environment has to be provided through an interface, it is therefore necessary to define a method for resolving a returning value and to expose this method in an interface definition. However, the callback binding is created automatically with the creation of an ambient reference. Both bindings are managed by the ambient controller and thus no special burden is laid on the shoulders of the developer.

The architecture of the ambient component is depicted in Fig. 2, where we can see membrane extensions, the ambient binding, and the Future callback binding which is created simultaneously and is managed in cooperation of ambient controllers on both client and server components.

5 Conclusion

In this position paper we propose a new approach to the design of Ambient-oriented systems. Our proposal is based on the usage of a new feature of the Fractal component model : Component-based Control Membranes. These allow to dynamically deploy the ambient functionality only to those parts of the system which really need it. Moreover, dynamic adaptability is achieved without putting any special burden on the developer.

The experiment we conducted showed that the Fractal Control Membrane provides sufficient extendability to develop Ambient-oriented components. Furthermore, it indicates that the proposed solution potentially represents an equivalent alternative to the AmbientTalk. In our future work we are further investigating abilities of membranes to extend towards additional AmI services (discovery services, concurrency management, etc.).

References

1. J.Dedecker, T. Van Cutsem, S.Mostinckx, T. D'Hondt, and W. De Meuter. Ambient-Oriented Programming. *In "OOPSLA '05: Companion of the 20th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications"*, 2005.
2. E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The Fractal Component Model and Its Support in Java. *Software Practice and Experience, Special Issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 2006.
3. IST Advisory Group. Ambient Intelligence: From Vision to Reality. 2003.
4. A. Gaddah and T. Kunz. A Survey of Middleware Paradigms for Mobile Computing. *Carleton University Systems and Computing Engineering Technical Report SCE-03-13*, 2003.

Dealing with Ambient Intelligence Requirements

Are Self-adaptive Mobile Processes a feasible Approach?

Holger Schmidt[1], Rüdiger Kapitza[2], and Franz J. Hauck[1]

[1] Institute of Distributed Systems, Ulm University, Germany
{holger.schmidt,franz.hauck}@uni-ulm.de
[2] Dept. of Comp. Sciences, Informatik 4, University of Erlangen-Nürnberg, Germany
rrkapitz@cs.fau.de

Abstract. Ambient Intelligence is characterised by a heterogeneous and highly dynamic infrastructure. In this paper we present requirements that we identified for developing applications for Ambient Intelligence scenarios. We sketch our own approach based on self-adaptive mobile processes that makes application development a manageable task and fulfils parts of these requirements.

1 Introduction

There is a trend to embed computing hardware into everyday physical objects and to allow interactions between these objects. This is even strengthened by the idea of Ubiquitous Computing and Ambient Intelligence (AmI), which envision a future in which people are surrounded and unnoticeably supported by small devices [1].

As AmI is among other things characterised by a highly heterogeneous and dynamic infrastructure, application development is a non-trivial and error-prone task. We think there is a need for application development support.

In this paper we identify and discuss AmI requirements. We think that most of these requirements can be met by an infrastructure that is based on self-adaptive mobile processes (SAMProc). These SAMProcs enable the specification of the life cycle as well as distribution aspects of a mobile application. Additionally, SAMProcs allow the adaptation of the process to the current context (run-time environment). In contrast to other approaches, we think that SAMProcs enable a continuous application development support based on the Model-Driven Architecture (MDA) [2].

The paper is structured as follows. In Section 2, we present the identified requirements for AmI. Then, we sketch our SAMProc infrastructure and give details on how this infrastructure can meet AmI requirements. After briefly discussing related work in Section 4, we conclude in Section 5.

2 Requirements for Ambient Intelligence

AmI characterises a set of requirements. In the following we give an overview of requirements that we identified.

As in AmI every physical object is potentially able to contribute, there is **heterogeneous hardware**. For providing developers more flexibility, support for **heterogeneous programming languages and platforms** is needed.

AmI is characterised by high system **dynamics**. As hardware is embedded into everyday objects, these are able to spontaneously build open networks. Furthermore, by physically moving a device, other devices can not be contacted anymore whereas new devices can be discovered.

For interacting with each other, objects require **communication**. As participating objects are characterised by heterogeneity, interoperability is needed. Additionally, volatile network connections have to be supported as these cannot be assumed stable within AmI networks. Furthermore, an abstraction from the underlying physical infrastructure (e.g., WLAN, Bluetooth) should be provided.

As AmI networks are build spontaneously there should be some **security mechanisms** provided by the infrastructure. Among other things there should be mechanisms for ensuring privacy, data confidentiality and integrity. As applications rely on the infrastructure it is important to provide trust mechanisms.

For supporting the user and the developer, **autonomy** may optionally be required. This should enable autonomous and transparent decisions. In extreme this leads to organic computing (i.e., self-healing/-organization/-. . .).

For handling dynamics and heterogeneity, applications should enable **adaptability**. This allows an adaptation of the application to the current context. This may for example lead to limited functionality on a resource-restricted device.

Because of the diverse set of requirements, application development for AmI scenarios is a complex and error prone task. For enabling a development independent from the underlying infrastructure there should be some automatic code generation support using high-level abstractions for **making requirements controllable** (i.e., development support providing mechanisms for handling presented requirements, e.g., using an MDA-like approach).

For fulfilling their task objects should **support cooperation**. For example, an AmI infrastructure may provide resource-management for an improved resource usage among the participating objects (e.g., memory, CPU, network).

AmI infrastructures have to **care about resource-limited devices** as embedded hardware within the physical devices may be limited (e.g., memory, CPU, energy). Thus, the infrastructure as well as the application should be scalable to the devices' capabilities.

We identified several requirements, which result from the dynamics in AmI networks: For **identification** of the participating objects (i.e., devices, services, applications, etc.) these should have a globally unique identifier (GUID). This can basically be achieved using already existent mechanisms for GUIDs.

An AmI infrastructure should enable the **localisation** of participating objects. A discovery process enables searching for reachable objects on the basis of either metadata (e.g., type, location) or the object identifier.

For an improved reaction to incidents, **context** is needed within the applications and the infrastructure. This includes context about the application's environment (e.g., current resource usage and location) as well as about other object's context (e.g., location, provided services). Context data has to be standardised to ensure the interoperability among participating objects.

As physical objects can be moved around, AmI infrastructures should support **mobility**. Therefore, mobility of the mobile device and mobility of applications (aka migration, for improved usage of network resources) should be enabled.

3 Self-adaptive Mobile Processes

In this section, we sketch our approach using SAMProcs, which we define as:

> *A self-adaptive mobile process (SAMProc) is an ordered execution of services. The SAMProc is able to adapt itself in terms of state, functionality and implementation to the current context (which is represented by the runtime environment) and to migrate either for locally executing services or for accessing particular context.*

3.1 Proposed Prototype Implementation

Recently, we sketched an approach for a mobile-process-based platform for supporting ubiquitous computing applications [3]. There, a SAMProc allows specifying the behaviour and interactions of an application that is able to dynamically change its execution context as well as to adapt itself to the current environment. Within our platform, the execution context is characterised by key-value pair metadata (e.g., location data, locally available resources and interfaces). Conceptually, SAMProcs are independent of the underlying infrastructure. However, in our prototype, SAMProcs are implemented as self-adaptive Web services supporting weak migration in conjunction with dynamic adaptation to the context (i.e., possible change of the mobile Web service's interface, provided state and used implementation code at runtime). Based on previous work [4], we integrated dynamic code deployment, as locally existent code cannot be assumed for migration in AmI scenarios that are characterised by heterogeneity and dynamics. This enables transparently exchanging the implementation language at runtime.

We propose to use BPEL as a basis for describing applications as SAMProcs. As standard BPEL is represented by a Web service, it is intuitive to implement the SAMProc using self-adaptive mobile Web services (SAM-WS, cf. Figure 1). However, other implementations are also possible. We enable the annotation of BPEL for non-functional properties within the SAMProc (e.g., desired invocation location). Additionally, we introduce new BPEL tags for migration, copying and cloning of the SAMProc for a particular process step of the application.

For supporting the developer, we advocate the use of an automatic code generation approach that is able to map the BPEL description to the underlying infrastructure. In case of our prototype implementation it is mapped to concrete self-adaptive mobile web services. As there are BPEL engines, which automatically evaluate standard BPEL processes and generate the execution code (e.g., IBM WebSphere Process Server), we argue that it is possible to generate migration code for our extended BPEL processes as only context-support has to be

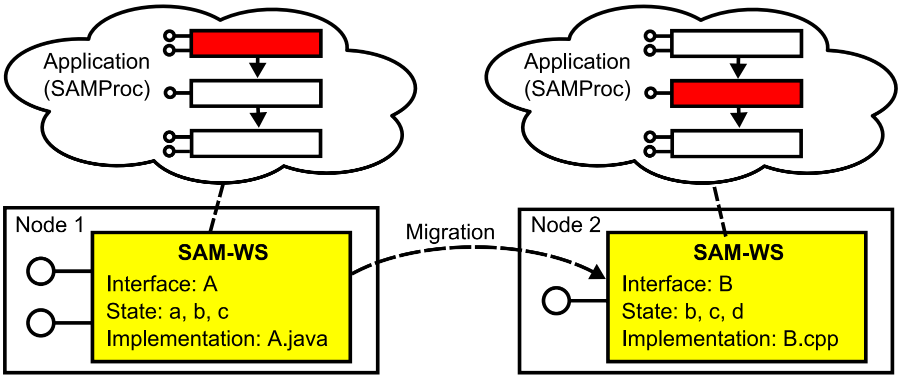


Fig. 1. Implementation of a SAMProc by migration of a SAM-WS

added. For supporting context, we provide a *context decision service* within our infrastructure, which is able to select best-fitting migration targets according to given criteria (e.g., required interface or location). This enables automatic code generation, as only context data (represented by key-value pairs) of calls to this service is dynamic and can be handled as parameters.

More details on our mobile-process-based platform as well as a simple example application can be found in our previous work [3].

3.2 Meeting Ambient Intelligence Requirements

In this section we revisit the identified AmI requirements of Section 2 and show our particular solutions. We fulfil the requirement of **heterogeneity** by building on Web service technology. Like standard Web services, our SAM-WS uses XML-based hardware-, platform- and language-independent data transfer. However, other implementations may use other platform- and language-independent protocols.

For supporting system **dynamics** our proposed infrastructure evaluates context at runtime, e.g., the context decision service is able to select the migration target at runtime. Additionally, we enable dynamic code deployment based on a loading service that is able to load locally unavailable code.

Web services are already the standard interface technology in many environments. We think that Web services will even spread in the future which results in more interoperability by building on this technology. Additionally, our SAM-WS supports volatile network connections for **communication** by offering asynchronous calls as well as migration capability (connection only needed during migration). Our infrastructure provides an abstraction from the underlying infrastructure (Web services, TCP).

Our infrastructure considers **basic security**, details can be found in [3].

Our SAM-WSs can be enhanced supporting a mobile-agent-like behaviour for providing **autonomy**. Therefore, a dynamic entry point has to be introduced that is called after migration to ensure running the autonomous activity.

We support **adaptation of the application** that is represented by a SAMProc and—within our prototype—is implemented by a SAM-WS.

To enable **controllability of requirements** we support an MDA-like automatic code generation: The application is specified as a BPEL process that is mapped on the underlying infrastructure, e.g., a SAM-WS. Developers have to implement pure service logic as migration code is automatically generated. We provide a development independent from the underlying heterogeneous structure using Web services.

For **supporting cooperation** we provide support for copying and cloning SAMProcs. Additional efforts are subject to future work.

For supporting **resource-limited devices** only few parts of the infrastructure have to be locally available. The remainder of the infrastructure can be distributed in the network. However, a precise specification of a minimal infrastructure is still subject to future work.

Identification of services (applications) is realised by a GUID. For Web services, we implemented a reference concept, which is based on the idea that the ID does not change after migration.

Our infrastructure provides a service for **localisation** of the current position of a self-adaptive mobile Web service using the GUID. Additionally, we provide a discovery service for searching for possible migration targets according to provided metadata and interfaces.

In our infrastructure **context** is represented by key-value pairs and collected by a local context service. Standardisation of context is subject to future work.

Providing **mobility** support for physically moving devices, which might result in changing the IP-Address, is not part of our infrastructure. However, this can be solved using Mobile IP. Migration as an implementation of application mobility is supported as detailed before.

4 Related Work

There is a lot of related work for supporting AmI. As an extensive summary of all related work is beyond the scope of this paper, we present basic ideas each with an exemplary realisation.

There are some research groups that try to support AmI by providing an own programming language or at least programming language extensions. An example for such language is *AmbientTalk* that builds on the ambient-oriented programming (AmOP) paradigm [5]. AmbientTalk faces volatile connections and system dynamics by providing an active object model that is based on concurrent distributed prototypes. These prototypes have eight mailboxes that contain the active object's state as well as the context. However, in contrast to our solution, the system is restricted to the AmbientTalk programming language.

Another approach is middleware support for AmI. There, the middleware provides basic functionalities, e.g., *BASE* [6]. BASE follows a micro broker approach, in which only basic functionality is implemented in the middleware core. Transport protocols are implemented as plug-ins, which enables an abstraction from the real protocols for the application developer. Additionally, the BASE system was extended to a component system: *PCOM* [7]. PCOM's focus is the development of a component system that enables a dynamic adaptation of components' dependencies: If a particular application component is not available anymore, the task is taken over automatically by an adequate component without user interaction. However, in contrast to our approach, BASE as well as PCOM do not provide mobility in terms of migration and do not continuously support application development.

In relation to the presented SAMProc approach, there is also work on mobile processes for supporting ubiquitous environments. *DEMAC* [8] aims at the exchange and the distributed execution of processes by means of abstraction from the underlying transport protocols. Therefore, a custom process description language was developed. However, DEMAC does not provide continuous support for application development. In contrast to our approach, evaluation of the process description is done within the participating nodes whereas we pre-evaluate the process descriptions by building on an MDA-like approach. Additionally, DEMAC does not support adaptive migration.

5 Conclusion

We presented requirements, which we identified for AmI applications. These requirements can in most instances be fulfilled by using our sketched SAMProc approach. We think that our SAMProc-based infrastructure especially enables AmI requirements of heterogeneity, dynamics, communication, context, adaptability, mobility and controllability of requirements.

References

1. M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.
2. OMG. MDA Guide Version 1.0.1. OMG Doc. omg/2003-06-01, 2003.
3. H. Schmidt, R. Kapitza, and F. J. Hauck. Mobile-process-based ubiquitous computing platform: A blueprint. In *1st MW-App. Interact. Works.* ACM Press, 2007.
4. R. Kapitza, H. Schmidt, U. Bartlang, and F. J. Hauck. A generic infrastructure for decentralised dynamic loading of platform-specific code. In *DAIS'07*, 2007.
5. J. Dedecker, T. van Cutsem, S. Mostinckx, T. D'Hondt, and W. de Meuter. Ambient-oriented programming. In *OOPSLA'05*, pages 31–40. ACM Press, 2005.
6. C. Becker, G. Schiele, H. Gubbels, and K. Rothermel. BASE - a micro-broker-based middleware for pervasive computing. In *PerCom'03*, page 443. IEEE, 2003.
7. C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM - a component system for pervasive computing. In *PerCom'04*, page 67. IEEE, 2004.
8. C. P. Kunze, S. Zaplata, and W. Lamersdorf. Mobile Process Description and Execution. In *DAIS'06*, 2006.

Pervasive Communication: The Need for Distributed Context Adaptations

Jorge Vallejos, Brecht Desmet, Pascal Costanza, Wolfgang De Meuter

Programming Technology Lab – Vrije Universiteit Brussel
Pleinlaan 2 - 1050 Brussels - Belgium
`{jvallejo,bdesmet,pascal.costanza,wdmeuter} @vub.ac.be`

Abstract. This paper focuses on the effects of pervasive computing on today's software technology. We present an existing software application for communication as a case study and establish a set of requirements this application should accomplish to become a pervasive computing application. We define what a pervasive communication is and identify the need for distributed context adaptation schemes.

1 Introduction

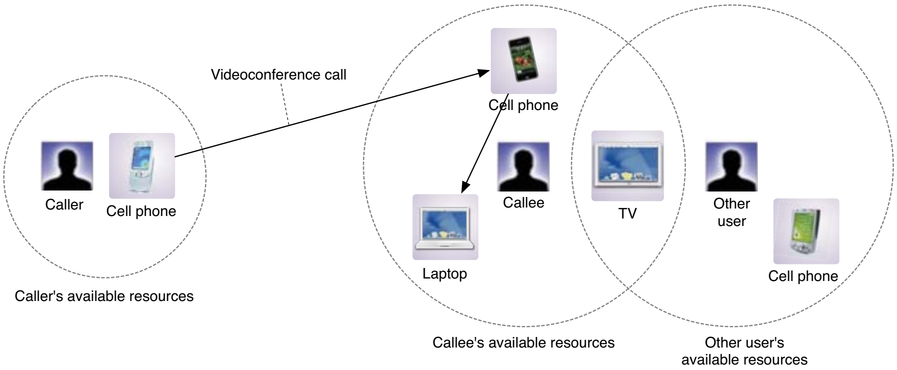
In the pervasive computing paradigm, software applications vanish into their users' surroundings, spreading their functionality across computers integrated into everyday devices [1]. This pervasive condition entails a number of new challenges which we illustrate in the following scenario.

Consider the case of a software application for communication like *Skype* [2], *Google Talk* [3] or *iChat* [4]. The main property of such an application is to offer *multiple* traditional and new communication services to its user (based on text, audio and video), who only requires *one* identity to use all these services. Assume that this application – named *ContextCom* in this paper – runs in a pervasive computing environment composed of a set of devices provided with processing and communication capacity: an interactive TV, a cell phone and a laptop. Although *ContextCom* is available at any of these devices, its user may have some preferences on where to use the different communication services. For instance, the user may prefer the interactive TV for having videoconferences whereas he opts for his cell phone to chat via text messages. Additionally, the user's preferences may be conditioned to the context in which the communication occurs. For example, if somebody else is also watching the TV at the moment the user receives a videoconference call, then he may prefer to use his laptop instead. Finally, the fact that the services of *ContextCom* are available in different devices should not imply for the user to have independent instances of this application in every device. It should be possible that the user has a single account that he can simultaneously use on all devices.

Based on the scenario above, we argue that a pervasive computing application for communication should be able to *dynamically distribute* the communication across the devices available in the user's environment, to *adapt* this dynamic distribution to the context, and to *preserve* the user's identity regardless of the dynamic distribution. In the following sections, we further analyse these requirements and discuss their implications for the development of such an application.

2 Understanding Pervasive Communication

To understand how a pervasive computing application for communication (such as *ContextCom*) differs from existing communication applications, we first need to understand what a pervasive communication is. We generically define a pervasive communication as an interaction that occurs through the environment of its participants, i.e. the devices with processing and communication capacity available in this environment (see Figure 1). This kind of communication has a number of particularities that we describe in this section.



Videoconference call
Cell phone
Caller
Cell phone
Callee
TV
Other user
Laptop
Cell phone
Caller's available resources
Callee's available resources
Other user's available resources

Fig. 1. A pervasive communication.

Distributed Presence of Identity In a pervasive communication, the availability of a user is determined by the set of devices available in his surroundings. This means that the user is capable to communicate with others (e.g. to initiate or to receive a call) as long as he has at least one device available for this purpose. This also implies that in case of having more than one device available, the identity of the user should be *distributed* in all these devices so that he can, for instance, be reached by a call at any of these devices. The constitution of this identity is dynamically reconfigurable as the devices become (un)available in his environment.

In the figure above, we also observe that same device can be used by more than one user (e.g. the TV). The device, in this case, may be part of the identities of several users.

Dynamic Context Adaptations The devices in a pervasive communication may exhibit different behaviours. A device, for instance, may not only hold a communication (e.g. a videoconference call) but also delegate it to another device in the user's environment (e.g. the cell phone of the callee delegates the videoconference call to his laptop). This adaptation of behaviour occurs dynamically and depends on the context of the communication in which the device is involved. By context, we mean any piece of information which is computationally accessible [5]. Some examples of context information we consider in this scenario are the type of communication, the availability and hardware characteristics of the devices (e.g. the TV is the most convenient device for a videoconference with respect to video and audio quality), and the presence and preferences of the users (e.g. the laptop may be a better alternative for the videoconference if there is more than one user watching the TV).

Coordination of Distributed Context Adaptations A pervasive communication may also require the adaptation of several devices. Such a case requires a coordination scheme between the devices involved in the communication. For instance, while having a videoconference at the laptop, a user may want to be notified of all the incoming calls or messages at this device. This implies that if any of the other devices receives such calls or messages during the time of the videoconference, it should send a notification to the laptop.

Devices may be involved in several interactions at the same time. Since presumably these interactions require also different adaptations, there is a high probability that devices end up with adaptations that conflict with each other. In Figure 1, for example, the TV is a device shared between two users and as such it can be involved in two interactions: the user who is having a videoconference in his laptop and for which the TV should act as a notifier of the incoming calls, and another user that may have a conversation via text messages directly in the TV.

Distributed Context Reasoning The context of a pervasive communication is not a monolithic and homogeneous set of information, it can vary with time and from one device to another (e.g. hardware characteristics). Additionally, some context information may not, and probably should not, be known by all the devices (e.g. user preferences). The context-dependent adaptation required for a communication, therefore, cannot be decided in only on device but it requires a distributed reasoning process. In our scenario, for instance, the cell phone of the callee might not have the means to detect that this user is not watching the TV alone. Thus, this cell phone can only decide to delegate the videoconference call to the TV (as the TV has better audio and video attributes) and let the TV decide to delegate the call to the laptop.

A distributed context reasoning scheme also preserves the autonomy of the devices. For example, if the caller could decide the device that the callee should use for the videoconference, the callee would lose the possibility to discern whether and how to receive the calls.

2.1 Summary: The Need for Distributed Context Adaptations

In summary, in a pervasive communication (i) the identity of the users should be distributed in the devices found in their environment, (ii) the devices should dynamically adapt their behaviour to the context of the pervasive communication, (iii) the adaptations of several devices should be coordinated, and (iv) the context-dependent reasoning process should be distributed. We refer to all these conditions as the *need for distributed context adaptations.*

3 Ongoing Work and Discussion

We are currently working on an object-oriented programming model to address the requirements identified in the previous section. This model is a combination of previous works presented in [6] and [7]. Our intention in this model is to provide dedicated language constructs to define local and distributed context adaptations of applications, i.e. the adaptation of the behaviour of one device and the coordination of several adaptations of different devices. In this model, devices decide their adaptation as well as their participation in a distributed coordination scheme. Finally, this model also enables the distribution of user's identities between several devices.

There are still some open issues of pervasive communication and distributed context adaptations which consequences need to be further investigated. Some of these issues are listed below:

Context beyond proximity Different from the notion of context that is commonly described in context-aware systems and which is associated to proximity [8, 9], the context that influences a pervasive communication may be found at completely different physical locations. For instance, in the scenario of the videoconference (Figure 1), the callee may use different configurations based on the caller's identity or location [6].

Different notions of adaptation scope In a pervasive communication, we observe two cases of context adaptations that uses two different notions of scope. The first adaptation is related to the user's identity which scope is the user's physical surrounding. This identity changes according to context events that occur in the environment, e.g. new devices or communication types that become available. The second adaptation is the one required for a communication which scope is related to the execution time and the part of the environment this communication affects. For instance, the adaptation for a videoconference is required only during the time this videoconference occurs and may only affect some devices.

User-assisted context adaptations There might be situations in which the adaptation for a pervasive communication cannot be automatically decided and requires the interaction of the user. The user may also want to create new adaptations or modify the existing ones. For such cases, the application for communication should provide means to involve the user in the context reasoning process.

Changes of context during the communication During a communication, the devices that hold this communication may change their context conditions, e.g. if they become (un)available. Such changes may require different adaptations that the one decided at the beginning of the communication.

References

1. Weiser, M.: The computer for the twenty-first century. Scientific American (1991) 94–100
2. Zennström, N., Friis, J.: Skype. http://www.skype.com (2007)
3. Google Inc.: Google Talk. http://www.google.com/talk/ (2007)
4. Apple Inc.: iChat. http://www.apple.com/macosx/features/ichat/ (2007)
5. Hirschfeld, R., Costanza, P., Nierstrasz, O.: Context-Oriented Programming. Submitted to Journal of Object Technology. http://www.jot.fm (2007)
6. Vallejos, J., Ebraert, P., Desmet, B., Cutsem, T.V., Mostinckx, S., Costanza, P.: The Context-Dependent Role Model. In Indulska, J., Raymond, K., eds.: 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2007), Paphos, Cyprus. LNCS 4531, Springer (2007)
7. Costanza, P., Hirschfeld, R.: Language Constructs for Context-Oriented Programming - An overview of ContextL. In: Dynamic Languages Symposium. (2005)
8. Barron, P., Cahill, V.: Using stigmergy to co-ordinate pervasive computing environments. In: WMCSA '04: Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'04), Washington, DC, USA, IEEE Computer Society (2004) 62–71
9. Sørensen, C.F., Wu, M., Sivaharan, T., Blair, G.S., Okanda, P., Friday, A., Duran-Limon, H.: A context-aware middleware for applications in mobile ad hoc environments. In: MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, New York, NY, USA, ACM Press (2004) 107–110