Object Technology for Ambient Intelligence and Pervasive Computing: Language Constructs and Infrastructures

Holger Mügge[1], Pascal Cherrier[2], Wolfgang De Meuter[3], and Éric Tanter[4]

[1] University of Bonn, Germany
[2] France Telecom, France
[3] Vrije Universiteit Brussels, Belgium
[4] University of Chile, Chile

Abstract. This report summarizes the main activities held during the second workshop on object-technology for Ambient Intelligence and Pervasive Computing held at ECOOP 2006. The workshop covered topics varying from low-level considerations such as garbage collection and object migration, via programming language infrastructure such as reflection and context-oriented programming, to engineering applications using maturing techniques such as aspects.

1 Introduction

In the near future, a new level of dynamicity will be required in order to construct software that is highly context-dependent due to the mobility of both the software itself and its users. This peculiar setting has been given different names like Ambient Intelligence (AmI), Pervasive Computing or Ubiquitous Computing with slightly different meanings. We use the term Ambient Intelligence but address equally all kinds of mobile, distributed software from the software engineering point of view. The idea of Ambient Intelligence is that everybody will be surrounded by a dynamically-defined processor cloud, of which the applications are expected to cooperate smoothly.

Currently, Ambient Intelligence seems to incorporate aspects from previously unrelated fields such as ubiquitous computing, mobility, intelligent user interaction, context dependency, domotics, etc. Early experiments in these fields, as conducted for example by Philips and MIT, already indicate that a full realization of their potential will need a new generation of concepts. These concepts need to support software which is able to operate in extremely dynamic hardware and software configurations. Ambient Intelligence is put forward as one of the major strategic research themes by the EUs IST Advisory Group for the financing structure of the 6th Framework of the EU. "The focus of IST in FP6 is on the future generation of technologies in which computers and networks will be integrated into the everyday environment [...]. This vision of 'ambient intelligence' places the user, the individual, at the centre of future developments for an inclusive knowledge-based society for all." (from the overall vision of the working programme of IST in FP6).

M. Südholt and C. Consel (Eds.): ECOOP 2006 Ws, LNCS 4379, pp. 130–140, 2007.

2 Scope of the Workshop

2.1 Goals

Important goals of the workshop were to identify and discuss the impact of Ambient Intelligence on object-oriented technologies and vice versa, and to outline some fruitful paths for future research concerning the connection between Ambient Intelligence and object-oriented programming languages and systems. In this context, we understand the term object technology to cover the whole range of topics that have evolved around the notion of object-orientation in the past decades, starting from programming language design and implementation, ranging over software architectures, frameworks and components, up to design approaches and software development processes.

We expect a special emphasis on the (seemingly?) conflicting forces of high dynamicity as offered, for example, by delegation- and reflection-based object-oriented systems that provide a high level of adaptability on the one hand, and peoples needs for security, safety and stability on the other hand. How can these forces be resolved, and does the notion of Ambient Intelligence with its concept of high availability of services even lead to new opportunities in this regard?

2.2 Topics

In the call for participation, the following non-exhaustive list of potential topics was included:

- Programming languages: Concepts for coping with new levels of dynamicity and security.
- Reflection: Why could it still be interesting to imagine a better reflective virtual machine and what about security and reflection: Can a reflective (structural / behavioural) object-oriented virtual machine be secured?
- Software evolution: Mobile software must continually adapt itself to potentially unanticipated environments. How can this be tackled?
- Context Modelling: What are the most promising ways to model context and integrate it into the software architecture?
- Adaptivity: What are the requirements for adaptive software? What do current methods and tools provide for building adaptive systems and what is missing?
- Quality of Service: What software engineering techniques support dependable, reliable and safe systems? How to bridge the gap between (too) low-level and (too) high-level rules? How to take structural constraints into account?
- Software development processes: Are the current approaches to analysis, modelling and development able to cope with the specific demands of mobile software?
- Human-device interactions: What is new in comparison to the good old model view controller? What are good new ways for GUI design and implementation? What about extending to 3D graphics and sounds? What about multimodalities?

– Device-device interaction: What are the requirements for embedded virtual machines? How do the existing models (Java, Smalltalk, Scheme, Python) differ from each other in handling events and communicating with people and other devices? Do they enable the software to exploit spontaneous collaborations between multiple devices and people?
– Constrained resources: Specificities of small mobile equipments, impact on the software in terms of object oriented concepts.

This topic list led to the submission of seven papers with topics varying from low level considerations such as garbage collection and object migration, via programming language infrastructure such as reflection and context-oriented programming, to engineering applications using maturing techniques such as aspects.

3 Workshop Organization

The workshop organisation was centred around two invited talks that were scheduled in the morning and in the afternoon. Apart from the invited talks, authors have presented their position papers according to the following schedule. In the schedule, the actual presenter is indicated in a boldfaced fashion.

Time	Content
9:00	Start: Welcome and Introduction
9:30	Invited Talk by **Jacques Malenfant** "Programming for Adaptability in Ambient Computing: Towards a Systemic Approach"
10:30	"Using Mixin Layers for Context-Aware and Self-Adaptable Systems" (**B. Desmet**, J.V. Vargas, S. Mostinckx and P. Costanza)
11:00	"Prototypes with Multimethods for Context-Awareness" (**S. Gonzáles**, K. Mens and S. Mostinckx)
11:30	"Semi-Automatic Garbage Collection for Mobile Networks" (**E.G. Boix**, T. Van Cutsem and S. Mostinckx)
12:00	"Design of a Multi-level Reflective Architecture for Ambient Actors" (D. Standaert, **É. Tanter** and T. Van Cutsem)
14:00	Invited talk by **Bill Griswold**: "Software Architectures for Context-Aware Computing - Experience and Emerging Challenges"
15:00	"Towards an Integration Platform for AmI: A Case Study" **A. Fortier**, J. Munoz, V. Pelechano, G. Rossi and S. Gordillo
15:30	"Towards Context-Sensitive Service Aspects" **T. Rho** and M. Schmatz
16:00	"Context-Aware Adaptive Object Migration" R. Kapitza, **H. Schmidt**, F.J. Hauck
16:30	Summary and roundup discussions
17:00	End

4 Summary of Contributions

This section summarizes the main points of the submitted position papers. These papers can be downloaded from the workshop's home page at
`http://sam.iai.uni-bonn.de/ot4ami/Wiki.jsp?page=Program`

Using Mixin Layers for Context-Aware and Self-Adaptable Systems by B. Desmet. This talk was about technology that allows an application to dynamically adapt its behaviour according to changes in the context in which the application operates. Current-day technology typically consists of a series of programming patterns to achieve such dynamic behaviour adaptation. As a consequence, combining different contexts in such systems has proven to be far from trivial. The talk proposed the use of mixin layers to modularize the context-dependent adaptations separate from the application core logic, together with a composition mechanism that deals with runtime context interactions. Since the classes in mixin layers have no fixed superclasses, they can be combined easily to reflect different combinations of context. The relationships between the different mixin-layers was proposed to be programmed in a declarative way. This enables a dynamic composition mechanism to construct and apply valid compositions of mixin layers according to context changes. The combination of using mixin layers and a declarative language to describe relationships between mixin layers was argued to be a powerful mechanism to deal with the continuously varying behaviour of context-aware systems.

Prototypes with Multimethods for Context-Awareness by S. Gonzales. The talk argued that the incorporation of context information into running mobile applications is currently often achieved using ad hoc mechanisms. To allow for an application to behave differently in a given context, this context-specific behaviour is typically hard-wired in the application under the form of if -statements scattered in method bodies or by using design patterns (e.g. Visitor, State, Strategy) and best-practice patterns (e.g. Double Dispatch). Therefore, the talk explores the Prototypes with Multiple Dispatch (PMD) object model in the light of context-aware mobile applications. The proposal provides a structured mechanism to deal with contextual information in a flexible and fine-grained manner. Context-aware mobile applications rely on a context architecture that aggregates the input from sensors (and possibly other applications) in a way that is accessible to the application. Using multiple dispatch the aggregated context directly influences the dispatch of methods, thereby avoiding hard-wiring context-related behaviour in the application. In other words, the programming model directly supports Context-Oriented Programming as recently proposed by P. Costanza.

Semi-Automatic Garbage Collection for Mobile Networks by E.G. Boix. In recent years remarkable progress has been made in the fields of mobile hardware and wireless network technologies. Devices communicate by means of such wireless infrastructure with other devices in their environment in ad hoc way spontaneously creating networks. However, developing applications for such devices is very complex due to the lack of support in current programming languages to

deal with the specific properties that distinguish mobile networks from the traditional distributed systems. The research presented in this talk focusses on providing programming language support to alleviate the complexity encountered when developing such applications. The talk identified the following phenomena intrinsic to mobile networks: connection volatility, ambient resources, autonomy.

The Ambient-Oriented Programming paradigm was postulated as a new computing paradigm which incorporates these hardware phenomena at the heart of its programming model in order to ease the development of applications for mobile networks. The talk gave an overview of this paradigm and then focussed on the issues of distributed garbage collection for mobile networks. Subsequently a new family of distributed garbage collection mechanisms to cope with them was introduced. It requires annotations from programmers to steer the garbage collection mechanism and is therefore called semi-automatic garbage collection.

Design of a Multi-level Reflective Architecture for Ambient Actors by É. Tanter. This work describes a multi-level reflective architecture for a language that automatically supports open network topologies where devices may join and leave at any point in time, where reliance on central servers is usually impractical and where the connection between two communicating devices is often volatile due to the limited wireless communication range. Rather than requiring the developer to manually deal with the difficult issues engendered by the ambient hardware at the base level, where this code would be severely tangled with and scattered throughout the functional code, the research proposes to offer the programmer a means to express a solution for the issues in a generic manner, at the metalevel. This metalevel is structured according to different levels of abstraction, which gives rise to what is known as a multi-model reflection framework. This structure is simply derived from the fact that not all distribution-related issues are expressible at the same level of abstraction.

The talk proposed a multi-level reflective architecture for ambient actors and its instantiation in the AmOP language AmbientTalk. The architecture combines (a) the engineering benefits of multi-model reflection by structuring meta-level facilities according to different levels of abstraction (passive ob jects, active ob jects, virtual machine), (b) the extreme encapsulation properties of mirror methods by ensuring that objects that are reflected upon can themselves restrict access to their meta-level facilities depending on the client, and this at all levels, and (c) the power offered by an extensible virtual machine in which facilities are made accessible to actors so that they can customize their execution environment, as well as adapting their own behavior according to it. The resulting MOP respects the extreme encapsulation principle thanks to its systematic use of mirror methods.

Towards an Integration Platform for AmI: A Case Study by A. Fortier. Creating intelligent environments requires knowledge from different disciplines such as HCI, artificial intelligence, software engineering and sensing hardware to be combined to produce an application. Therefore, the integration of independently components developed will be necessary. This talk argues that what is needed

to support this is an integration framework (comprising formalisms, tools and a software platform), which allows different components to seamlessly interact, to provide pervasive services and ambient intelligence. In such framework one should be able to specify, in an abstract way, the contextual information that a certain software module needs to perform his task, so that the integration platform can dynamically discover which providers can fit the module needs.

As a contribution towards the development of an integration platform, the talk presents a concrete example of systems cooperation. This example involves two different projects developed at different universities. Both projects address the problem of building ubiquitous software, but they do so using somewhat different approaches. The Software Engineering And Pervasive Systems (SEAPS) project, being developed in the OO-Method research group from the UPV, focuses on the development of a model driven method for the automatic generation of pervasive systems from abstract specifications. The systems that are generated by this method are used to provide services in a particular environment, generally smart homes. To implement the systems, the OSGi middleware, which is based on Java, was used. On the other hand, the Context-Aware (CA) pro ject being developed at LIFIA, in the UNLP, focuses on the user as a service consumer. In this view of a pervasive system, the user carries a mobile device most of the time, which is used to present services according to his context, which can vary dynamically. This framework is implemented in Smalltalk.

By integrating both systems the authors expect to improve the SEAPS project with dynamically-varying context information, to extend the CA project so that it can remotely manipulate SEAPS services, and to build a context model based on the information sensed by the SEAPS. They also expect to gain knowledge about more generic integration needs, to be able to effectively build the integration platform mentioned before. As a result of the work carried out, the talk presents as its contributions: the presentation of a concrete case of independent systems integration, the identification of a set of problems encountered during the integration process and the presentation of the lessons learned for others.

Towards Context-Sensitive Service Aspects by T. Rho. The talk argued that context-aware behavior is usually hard-coded into the application itself using the deployed libraries on the device. Since these have to be known at development time context-processing is limited to the known libraries on the corresponding target device. Besides that, not all context-sensitivity can be anticipated. Being bound to one device neither composition nor sharing of context information is possible. To build flexible applications that adapt themselves to the current situation, the underlying architecture must provide the means to dynamically reconfigure the application based on context information.

Service-oriented architectures (SOA) help to support the dynamic reconfiguration of applications. They modularize applications by decomposing them into distributed components, so called services. Ap- plications are build by composing these services and configuring them at runtime. Therefore the architecture is based on service-orientation.

Aspects help to improve the software design by encapsulating the unstable composition concern. AOP frameworks like have been proposed which realize this concept. Ambient intelligence introduces an even more unstable element —the varying context—, which influences the runtime adaptation of the service composition. Using AOP to encapsulate the service composition the services stay compact and stable because they are independent of adaptation strategies and context information. However, common aspect languages only consider the event flow of programs. The AOP terminology for program flow events is join point. Typical join points are method calls, field accesses or thrown exceptions. In an ambient intelligence setting these join points are not sufficient. The properties of the environment must also be taken into account. To combine contexts and join points a powerful pointcut language is needed.

One cannot apply AOP techniques in full extend on the SOA level, because the concrete implementation of services is, at most times, not accessible to a local system. And even if the service implementation is available there may still be different views onto the same service from the local or a remote system which are in a different contexts. The authors therefore restrict the join point model to calls on the service level. This paper introduces the Ditrios architecture and the service aspect language CSLogicAJ, which provide context-sensitive services aspects for the service-oriented architecture OSGi.

Context-Aware Adaptive Object Migration by H. Schmidt. There is an ongoing trend to integrate and link the physical world to the virtual world of computing resources and applications. Techniques like RFID (radio frequency identifications), sensors (e.g. BTNodes 1) and mobile devices, but also positioning systems like GPS (global positioning system) and wireless communication of all kinds, push this trend. Accompanied with this evolution and the rising diversity of systems, new concepts and techniques to provide adaptable and context-aware applications are required. Often, these applications will migrate between different platforms during their lifetime. As a typical example, a follow me application (e.g. personal information manager application) can have a different interface and state on a laptop, a cellular phone or a publicly accessible web-terminal. In other words, we expect that a mobile application has to adapt its state, the provided functionality and the implementation basis to its execution context, the target system and application-dependent restrictions.

Most recent object-based middleware and agent platforms restrict migration support to a certain programming language and environment. In this talk, the concept of adaptable mobile objects was proposed. These objects are capable of adapting their state, functionality and underlying code basis during migration to the requirements of the target platform and the needs of the object itself. We focus on weak migration. This means that only the state of an object is migrated but no execution-dependent state like, e.g., values on the stack. The proposed research builds on our recent platform- and ORB-independent implementation of the CORBA Life-Cycle Service that is based on CORBA value types, a standard CORBA mechanism for passing objects by value. This service combined with a logical separation of the mobile objects state, functionality and code

enables support for adaptive ob ject migration in heterogeneous environments. In fact, our current prototype of a dynamic adaptation service for mobile objects, the adaptive ob ject migration (AOM) service, supports the migration of ob jects between Java and C++. Supporting other CORBA-supported languages requires only moderate implementation effort. To assist the developer during the implementation process, the AOM tool was presented. Additionally, mobile objects acting as mobile agents (an object having an own thread that executes autonomously on behalf of a user) is supported.

5 Summary of Invited Talks

We were happy to welcome two invited talks:

- "Programming for adaptability in ambient computing: towards a systemic approach" by Jacques Malenfant, Universit Pierre et Marie Curie, Paris, France (`http://www-poleia.lip6.fr/~malenfant/`)
- "Software Architectures for Context-Aware Computing - Experience and Emerging Challenges" by Bill Griswold, University of California, San Diego, USA (`http://www.cs.ucsd.edu/~wgg/`)

This section briefly summarizes these talks. The slides shown by the presenters during the talks can be downloaded from the workshop's website:
`http://sam.iai.uni-bonn.de/ot4ami/Wiki.jsp?page=Program`.

Programming for adaptability in Ambient Computing: Towards a systemic approach by Jacques Malenfant. The talk starts by motivating the point of view that ambient systems are systems that have to survive in a constantly evolving context or environment and that disconnection is no longer a fault but a feature of ambient systems. This challenges all aspects of software deployment. It has repercussions on such things as dynamic installations, quality of service and software reconfigurations. The answer to this problem is "dynamic adaptability": the application should react to changes in the execution context. A very fundamental problem here is that — during the application adaptation — time goes by. This means that the environment is changing *while* the application is adapting itself. This is a very well-known problem since the advent of radar-controlled antiaircraft guns in world war two: the fact that the target moves after having shot has to be taken into account when aiming.

Looking at the architectural considerations of ambient systems, it is probably worthwhile taking into account IBM's vision on the Automatic Computing Blueprint. The main idea is to allow computers to configure themselves and manage their own adaptations. Keywords are self-gouvernance, self-configuration, self-optimization, self-healing, self-protection and self-maintenance. This requires biologically inspired computing which we will call "systemic programming". Systemics is the science that studies systems that are too complex to be tackled through the traditional reductionist approach. It has applications in domains as diverse as biology, sociology and economy. The main idea of systemic programming is to have a large distributed self-control with a higher goal (i.e. evolve over

time by learning from interactions with the environment). Challenges of systemic programming include modelling techniques (how do adequate local and globally emerging control models look like) and decision techniques (e.g., control theory, markovian decision processes, AI-heuristics). Furthermore, we need languages, methodologies and tools to program such systems.

The talk then continues by presenting the presenter's view on how to arrive at systemic programming. Two models are required: a reflection model and a decision model. The reflection model is the answer to the need for obtaining a static and dynamic model of the managed elements. It is used to reify representation of the managed elements. The presenter's view is that classical computational reflection is insufficient to tackle systemic needs. Current models for reflective computation usually take some metacircular form where the meta level implements the base level in a causally connected way. The major drawbacks of these models is that they are based on a strong and strict single threaded coupling between the base level and the meta level. Furthermore, they cannot tackle distribution because no global representation of the system can effectively be built. Finally, they cannot take into account he environment of a computing device because this is not necessarily causally connected.

The presenter moves on to present a new form of computational reflection, to wit asynchronous reflection. In this form of reflection, the meta level is no longer the language processor but takes the form of a general processor that controls the base level, in a concurrent way. It has many drawbacks over the current "synchronous" model of reflection: the base level and meta level keep each other informed about their evolution in terms of notifications and therefore, the meta level can sense events in the environment.

The second requirement for systemic programming is the presence of a decision model that decided on when the system should adapt itself. In classical computational reflection it is the task of the base level to decide when to "escape" to the meta level. In the model presented here adaptation is triggered based on occurrences of events in the environment. The presenter subsequently moves on to present a number of potential decision policies. They all reflect a sequence of decisions to take given a certain state of the system.

Software Architectures for Context-Aware Computing: Experience and Emerging Challenges by William Griswold. The presenter begins the talk by listing a number of facts. He states that there are many real needs and opportunities, yet very little progress. It appears as if the evolution is to have ever more nomadic systems that are increasingly *unaware* of their environment surrounding them: "I still can't find the printer when I need one". Furthermore, we are connected with a reasonable sensor platform (85 percent of Europe carries a mobile phone). Last, he states that mobile phones, as a commodity technology, must be part of the solution.

Next, a big experiment — called ActiveCampus — was presented. It is an ambient system on a university campus that gives students and professors an integrated access to all kinds of material such as librarie, courseware and administration. Furthermore, it offers a few innovative services such as place-its

which can be considered as context-aware post-its: they pop up on your mobile phone whenever you arrive in a certain context. E.g., in a supermarket that has a good selection of wines, a "buy wine" place-it might pop up. The main goal of ActiveCampus is to increase social awareness of its users.

The presenter presents his major claims which boil down to the following:

- First, context-aware computing is governed by "the three laws of context-aware computing", namely the Ubiquity Law, the Commodity Law and the Systems Law. These are further explained in detail (see below).
- A number of complex phenomena arise from these laws. One of them is that failure becomes a normal mode of operation.
- The presenter acknowledges that most academic work acknowledges the same issues. These issues are not to be dealt with as an afterthought: they actually shape systems.

The three laws of context-aware computing in detail:

1. **The Ubiquity Law:** *A context-aware system is useful to the degree that a person can use it everywhere and that everyone can use it.* An example of this law are the omnipresence of the ability to place the place-its described above. All they need is a phone that is location-aware.
2. **The Commoditization Law:** *The cost pressures of ubiquity lead to commoditization, thereby increasing heterogeneity, interoperability, and fragility.* Again the place-its example is taken to explain the law.
3. **The Systems Law:** *Successfully designing a component of a context-aware system requires understanding key aspects of the whole.* As an example used to explain this, ActiveClass was used. This is a web-enabled "backchannel" to enable students to anonymously ask questions in class at any time. The presenter uses the example to show that adding a small feature (or component) can destroy the entire idea behind a system.

These three laws engender a number of key consequences and the presenter has spent a lot of time describing how his projects as UCB have tried to cope with them. The consequences are:

- Commoditization makes failure a normal mode of operation. The question then becomes how to design for this and how to make progress given this fact. Failures cannot be abstracted away. They are manifested in objects and interfaces. Furthermore, applications should remain useful and profoundly rich instead of confusing or hobbled.
- The commoditization and the systems law can be tackled somewhat using object technologies. However, the systems law implies the presence of system-wide interactions that complicate a design. Issues such as failure interweave and crosscut other components. Much of failure has to be designed into the application metaphor. This needs a holistic design. The presenter explains the usefulness of architecture-governed design patterns, AOP, reflection, publish/subscribe architectures and remote objects with rollback/replay facilities in this light.

The presenter finishes by stating that Ubiquity + Failure means that it "is better to operate 20% of the time in 100% of the world rather than 100% of the time in 20% of the world."

6 Discussions and Upcoming Issues

Although the workshop schedule was tight, there were vivid discussions among the 26 participants. We list some of the most prominent open questions here and hope that they will be addressed in later workshops or joint work.

- *Adaptivity and Autonomy:* The invited talk by Jacques Malenfant raised the question whether techniques of autonomous and self-organized systems should be taken into account. In particular when thinking of unanticipated situations, emergent behavior becomes an issue. The relation between programming techniques for adaptivity on the one side and for autonomous systems on the other might be fruitful area for further research.
- *Dynamically vs. statically typed languages and reflection vs. aspects:* On the language level two competing approaches are status quo: dynamically typed languages with reflection on the one hand and statically typed languages enhanced with aspects on the other. The question under which circumstances which of these alternatives is more adequate is still not fully answered.
- *Frameworks, component systems, and programming languages:* Another question involves the language level as well as the component level: Do we need AmI-specific languages, or can we cope the requirements on the component or service level? How do they depend on each other or are open to be combined?
- *Systematic comparison of different techniques:* A basis for a systematic comparison between different programming instruments is missing. A start in this direction could be to create a concise catalogue of technical requirements. An example are typical security and privacy issues and how they can be achieved using reflection or aspects.
- *Context Representation:* How should context be represented? In which should we model is as data and in which as an activity?
- *Limits of Ontology-based approaches:* In order to allow context-driven adaptations to be performed without full anticipation, ontologies will play an important role (e.g. for detecting adequate services in a given application situation). It has to be expected that ontology-based work for service selection will face basically the same problems that were encountered in artificial intelligence (i.e. you cannot compare mathematic functions).