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.

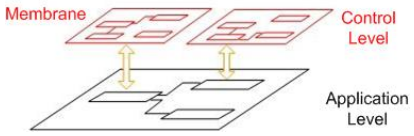

Fig. 1. Component-based Control Membranes

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.

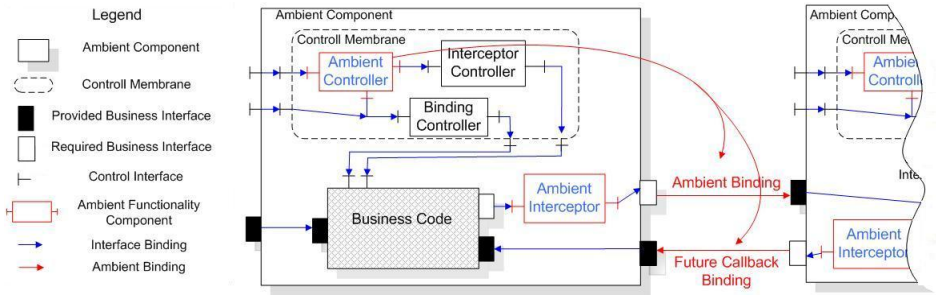

Fig. 2. Ambient Component

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.