

AOP Support for Mobile Systems

A. Popovici, G. Alonso, T. Gross

Department of Computer Science
Swiss Federal Institute of Technology (ETHZ)
ETH Zentrum, CH-8092 Zürich, Switzerland

{popovici, alonso, trg}@inf.ethz.ch

CONTACT PERSON: Andrei Popovici

CATEGORY: F

August 4, 2001

1 Introduction

Novel forms of networking that either avoid a fixed infrastructure or allow direct interactions between peers are rapidly becoming available for practical use. To support these new computing environments, it is critical to have an adequate software infrastructure. However, existing software architectures for distributed systems are tied too closely to traditional computing paradigms. To address the constraints imposed by mobile systems (and to take advantage of the opportunities provided by these systems), it is necessary to abandon the current paradigm in software design where software capabilities are determined at build time. Indeed, when the ability to react to changes or new environments is fixed in an application before it is deployed, the range of flexibility of that application is limited to what could be foreseen and was encoded by the designers. The result is that applications are capable of operating in different modes depending on external stimuli but cannot react to changes beyond those that have been anticipated.

Challenging computing environments such as spontaneous networks (e.g., Jini [Wal99]) make this limitation of current architectures critical. When a computing node does not know in advance with which other nodes it will interact and under what conditions, it becomes difficult to facilitate this interaction. One way to overcome these problems is to impose strict standards, and all nodes contain the preconfigured software. But Java-based environments like Jini services demand a more flexible software architecture, where functionality is a dynamic property and applications can be adapted to the environment at run-time. For instance, if several persons carrying mobile computing devices want to exchange information with specific degree of privacy, the applications should be able to agree on a common encryption and access control policy and proceed accordingly. Moreover, we should not assume that all devices provide the necessary functionality.

Aspect oriented programming (AOP) [KLM⁺97] seems to be a promising approach to implement this type of adaptability as a special form of concern composition [TOHS99, ATB97]. For example, AOP-based frameworks have been proposed to perform non-functional adaptation of the CORBA service layer in response to changes in the run-time environment (e.g., network resources) like [ZBS97, TJJ00]. More generally, AOP has been used in areas such distribution and synchronization concerns [Lop97] or real-time concerns [AB96].

AOP allows to integrate a specific concern (e.g., access control) into an existing application even if it affects many decomposition units of the original application. From the self-organization perspective, the *binding time* of the concerns is crucial. In general, AOP can be implemented using compile-time [Kic00, TO00], load-time [KH98, CCK98] or run-time [Bol99, KMM⁺] binding techniques. To enhance a whole community of Jini services with context-specific concerns, two key problems need to be solved. First, an AOP platform that allows run-time composition of services needs to be present in all nodes. Second, a way to encapsulate and distribute aspect instances over a network is needed to remotely control the composition of concerns.

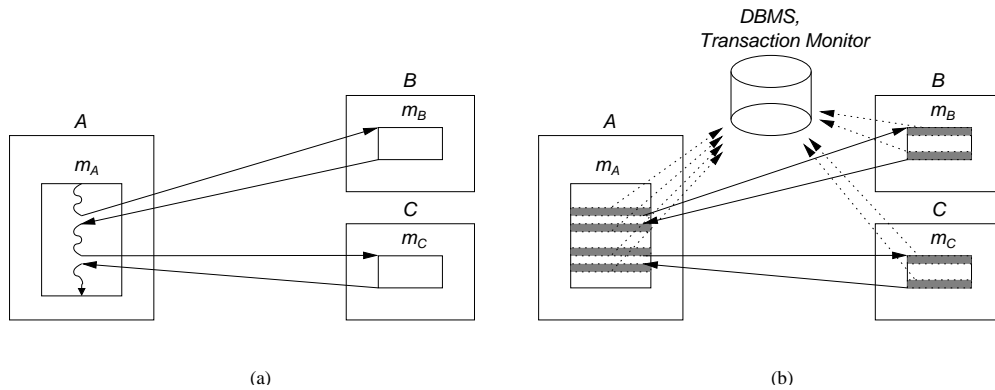


Figure 1: (a) A distributed computation in a Jini community. (b) Transactional processing throughout a distributed computation.

In this paper we show PROSE (*PROgrammable extenSions of sErVICES*, an AOP implementation that allows run-time weaving and permits the adaptation of services in a spontaneous network.

2 Service Adaptation with PROSE

2.1 PROSE Summary

PROSE provides a simple way to describe aspects; it consists of a run-time environment, the Java Virtual Machine(JVM), and a set of libraries. PROSE does not define a new aspect language. All AOP-related constructs are expressed using the base language, Java. To exemplify the use of PROSE in Jini, consider a distributed computation starting at service *A* and calling remote methods of the services *B* and *C* (Figure 1.a). Normally, each of the nodes can fail independently, thereby leaving the system in a globally inconsistent state. To prevent this, incoming and outgoing remote calls have to be bracketed by transactional code, depicted in Figure 1.b as gray rectangles. The transactional code coordinates the commitment of the persistent resources changed in the methods m_A , m_B , and m_C . In an adaptive AOP-based environment, the transactional functionality can be woven into the services *A*, *B*, and *C* as soon as they join the community. Figure 2 shows a PROSE aspect that describes transaction termination at the end of all methods in the service *B* (some minor details omitted, see [PGA01] for complete examples).

```

1 class ExampleAspect extends Aspect {
2
3   Crosscut doTxEnd = new FunctionalCrosscut() {
4     public void ANYMETHOD(ANY anyThis, String cb) {
5       // end transaction
6     }
7     { setSpecializer(
8       (MethodS.AFTER).AND
9       (MethodS.named("m.*")) .AND
10      (ClassS.extending(ServiceB.class)));
11   }
12 };
13 }
```

Figure 2: A PROSE aspect for transaction termination at the end of methods defined in `ServiceB`.

The first observation is that all aspects extend the `Aspect` base class (line 1). An aspect object contains one or several crosscut objects. A crosscut object defines an advice and describes the join-points where the advice should be executed. In Figure 2, there is just one crosscut, corresponding to the `doTxEnd` instance field (line 3); the advice action is defined on the lines 4-6. The number and

types of join-points defined by `doTxEnd` depend on the signature of the advice method and on a *specializer* object attached to the crosscut (lines 7-9). The signature (line 4) restricts the execution of the advice to methods that have a `String` as a second argument. The specializer further restricts the set of join-points to those defined in methods of classes extending `ServiceB` and whose name matches the regular expression `"m.*"`. Specializers are composable by means of `AND`, respectively `OR`, methods. They are used in a way that is similar to that of *pointcut designators* [Kic00] in AspectJ.

To allow expressing basic AOP concerns, a number of predefined classes allow intercepting join-points at method boundaries, field access and modification, as well as exception throwing [PGA01]. Since PROSE aspects are defined in pure Java, the adding of extra constructs (new specializer types, new wildcards, new crosscut implementations) is possible. Thus, the adaptation of PROSE for a specific problem field is easy to achieve.

2.2 Aspect Weaving

Existing approaches to run-time AOP are based on a reflective interface of the run-time environment (e.g., AOP/ST for Smalltalk [Bol99]). Similarly, PROSE's core is located directly in the JVM. PROSE is implemented using the debugger interface of the JVM, the Java Virtual Machine Debugger Interface (JVMDI) [SM00]. PROSE uses the JVMDI to stop the execution of the JVM at join-points (e.g., method boundaries or field modifications) and then calls the advice behavior defined in the crosscut objects. When all advices corresponding to the current join-point are executed, the control is returned to the application. PROSE allows run-time weaving and unweaving of aspects and intercepts the execution of join-points even for classes loaded in the JVM after the aspect weaving. For a more complete description of the implementation, see [PGA01].

PROSE is a prototype implementation of a JVM interface for weaving aspects, a JVM-Aspect-Interface (JVMAI). We envision JVMs with a native JVMAI. In the JVMAI model, the aspects belong to the state of the virtual machine. On top of the JVMAI, a *weaver interface* allows the weaving and un-weaving of aspect instances at run-time. Aspects and objects can be instantiated independently, on different nodes of the network and composed at arbitrary points in time, by calling the methods of the weaver interface.

An central issue is to combine concern composition represented by the weaver interface, with the dynamic character of a spontaneous network. To exemplify this, Figure 3.a illustrates a typical interaction in a Jini community. A service (*B*) joins the community by registering a proxy *pB* at a nearby lookup service (LUS) (1). Clients can query the lookup service (2), obtain *pB* and use the service *B* (3).

For weaving aspects through all services of a community, the weaver interface must be treated as a regular Jini service too. Figure 3.b illustrates service *B* running together with a weaver service on the same JVM. Like any other service, the weaver joins the Jini community (1) and allows remote clients to use its interface. A central repository of aspects, the *extension base*, acts as a client and discovers all active weaver services in the network (2). Depending on the properties of the node, a customized set of aspects instances is remotely sent to the weaver (3). The immediate effect is that a crosscutting concern modifies the behavior of the service methods *m*₁, *m*₂ and *m*₃ (4). The following code fragment illustrates in a simplified manner a possible way to the distribute aspect instances.

```

1 Aspect asp = new ExampleAspect();
2 RemoteWeaver[] xMgr = registry.lookup(RemoteWeaver.class);
3 for(int i=0; i<xMgr.length; i++)
4     xMgr[i].weave(asp);

```

A new instance of the class `ExampleAspect` is created on line 1. Line 2 looks up Jini services that implement the `RemoteWeaver` interface. The `RemoteWeaver` interface allows the weaving of aspects through the application co-located with the weaver service. All applications willing to benefit from run-time adaptation must initialize a `RemoteWeaver` service in their virtual machine.

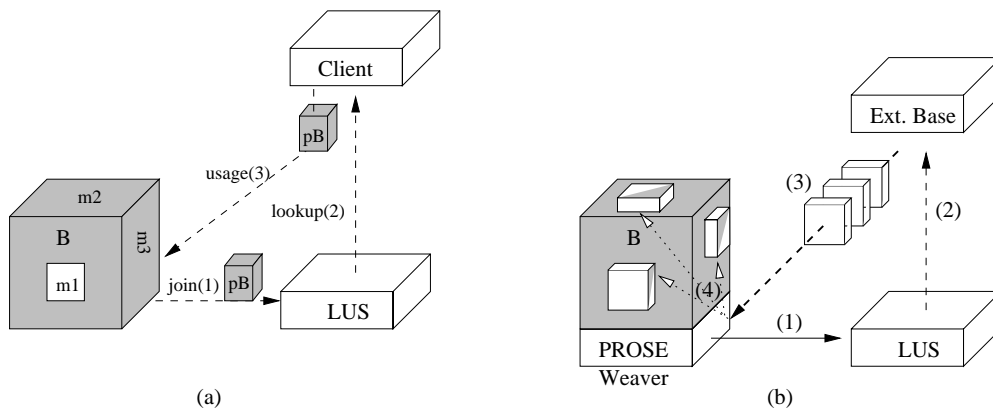


Figure 3: (a) Typical interactions in Jini. (b) Run-time adaptation of application through a Weaver Jini service.

Line 4 contains a remote call to each weaver service that composes the aspect `asp` into the AOP-enabled virtual machine of the participant. After line 4, PROSE starts trapping execution events and dispatching them to the crosscuts of the aspect-instance `asp` in all nodes of the network.

This example pinpoints two problems raised by aspect composition in dynamic communities of services. For one, weaving depends on the identity and preferences of each node and not just on the static properties of the component to be extended. For second, the weaving process must be secure: aspect instances are sent by third-party nodes and their origin must be checked to prevent misuse of the adaptation mechanism. A solution is that aspects have to be signed. For example, PROSE uses the standard Java mechanism that allows signing of Java byte-code. A higher degree of security can be achieved by signing the *state* of an aspect instance. On each node, the weaver checks whether the aspect code and state are signed by a trusted entity and whether sufficient permissions exist for weaving. As a refinement of secure weaving, fine-grained access to local resources is helpful. This requires that the aspect code is not woven through the component code but actually called from the component code. Thereby, aspect and component code are cleanly separated, and the Java security framework can selectively allow or deny access to resources based on the origin of the aspect class.

2.3 Applications of PROSE

A context-specific adaptation of services borrows concepts already used in modern information systems, such as Enterprise Java Beans [Inc01] (EJB). In the EJB model, reusable business objects are deployed in a run-time environment called bean container. The deployment process creates a container-specific layer of software taking care of encryption, authentication, access control, transactions and persistence. A clear distinction is made between the business logic of the service and the context it is running in. The same ideas could be applied to a spontaneous community of services with the appropriate AOP support. Then, context-sensitivity can be achieved for each node. Unlike in EJB, where a container is a host-based run-time environment, one can see the network as a container that composes its own concerns in every service that joins it. Currently, we have used PROSE to design modules that perform network-managed persistence, security, and transactions.

Network Managed Persistence Transparent persistence, similar to the container managed persistence in EJB, is possible using AOP techniques. For example, object - database integration has been already proposed by the use of composition filters [ABV92]. To make the state of a Jini service persistent, an aspect that cuts across accesses to all fields of the objects of interest must be woven into the corresponding node. For each field modification, the changed data is stored in a database connected to the network. This solution has the advantage that an application is truly decoupled from the access to a database, since the database access is

hidden in the aspect code. The aspect code, is woven only during the presence of the database node in the Jini community.

Network Managed Transactions To make the object persistence more reliable, data consistency must be maintained in a distributed computation. For this purpose, an aspect (e.g., the one in Figure 2) can be woven into each node of a Jini community. Upon insertion, the aspect installs a mini-transaction manager[PA00] on that node. Additionally, it defines the join-points of the methods of interest as transaction boundaries and ensures proper propagation of transactional contexts.

Network Managed Security A third type of aspect instance woven into every joining will create an identity of each node. From the context-adaptation point of view, the network decides the level of security. The functionality for authorization and access control can be woven at the beginning of incoming and outgoing method calls, thus enabling a secure interaction of nodes.

For each middleware adaptation described above, we have implemented a prototype using PROSE. Our current efforts are directed toward integrating the three components into a coherent, complete system, the MIDAS (MIddleware ADaptive Services) network space. MIDAS will provide a secure Jini environment based on run-time AOP that will be functionally equivalent to that of an EJB container.

3 Summary

As in any type of platform, it is always possible to develop independent solutions for spontaneous networks that are tailored to each application. However, this approach does not take into account that many non-functional requirements can be composed at run-time to achieve context-specific behavior. To meet this challenge, we are exploring an AOP platform suited for adaptation of nodes in spontaneous networks. The results of our first attempt is PROSE, a system that allows applications to acquire and discard cross-cutting concerns at run time. Of course, this is only a preliminary effort but our experiences so far are very promising. If successful, the ideas behind AOP, or variations on them, can play a significant role in developing applications for mobile ad-hoc systems.

References

- [AB96] M. Aksit and M. Bergmans. Composing Synchronisation and Real-Time Constraints. *Journal of Parallel and Distributed Computing* 36, pp. 32-52, 1996.
- [ABV92] M. Askit, L. Bergmans, and S. Vural. An ObjectOriented Language-Database Integration Model: The Composition-Filters Approach. In *Proceedings ECOOP'92*, volume 615 of *LNC3*, pages 372–395. Springer Verlag, June 1992.
- [ATB97] M. Aksit, B. Tekinerdogan, and L. Bergmans. Achieving Adaptability through Separation and Composition of Concerns. In Max Muhlhauser editor, *Special Issues in Object-Oriented Programming, Workshop Reader of the 10th. European Conference on Object-Oriented Programming, ECOOP'96*, Dpunkt-Verlag, 1997.
- [Bol99] K. Bollert. On Weaving Aspects. Position paper at the ECOOP'99 workshop on Aspect-Oriented Programming, June 1999.
- [CCK98] Geoff A. Cohen, Jeffrey S. Chase, and David L. Kaminsky. Automatic program transformation with JOIE. In *Proceedings of the USENIX 1998 Annual Technical Conference*, pages 167–178, Berkeley, USA, June 15–19 1998. USENIX Association.
- [Inc01] Sun Microsystems Inc. Enterprise Java Beans Technology. <http://java.sun.com/products/ejb/>, 2001.
- [KH98] Ralph Keller and Urs Hölzle. Binary Component Adaptation. In Eric Jul, editor, *ECOOP '98—Object-Oriented Programming*, volume 1445 of *Lecture Notes in Computer Science*, pages 307–329. Springer, 1998.
- [Kic00] G. Kiczales. AspectJ: Aspect-Oriented Programming using Java Technology. JavaOne Conference, June 2000.

- [KLM⁺97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *ECOOP '97 — Object-Oriented Programming 11th European Conference, Jyväskylä, Finland*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, New York, NY, June 1997.
- [KMM⁺] P. Kenens, S. Michiels, F. Matthijs, B. Robben, E. Truyen, B. Vanhaute, W. Joosen, and P. Verbaeten. An AOP Case with Static and Dynamic Aspects. In *ECOOP'98 Workshop on Aspect-Oriented Programming*, Brussel (Belgium) , July 1998.
- [Lop97] C. Lopes. D: A Language Framework for Distributed Computing. Ph.D. Dissertation, College of Computer Science, Northeastern University, Boston, 1997.
- [PA00] G. Pardon and G. Alonso. Cheetah: a lightweight transaction server for plug-and-play internet data management. In *Proceedings of VLDB 2000*, Cayro, Egypt, September 2000.
- [PGA01] A. Popovici, T. Gross, and G. Alonso. Dynamic Homogenous AOP with PROSE. Technical report, ETH Zürich, Department of Computer Science, March 2001.
- [SM00] Sun Microsystems. Java Virtual Machine Debugger Interface Specification. <http://java.sun.com/products/jdk/1.2/docs/guide/jvmdi>, 2000.
- [TJJ00] E. Truyen, B. Jrgensen, and W. Joosen. Customization of Component-Based Object Request Brokers through Dynamic Configuration. In *Proceedings of TOOLS Europe'2000*, IEEE press, June 2000.
- [TO00] P. Tarr and H. Ossher. Hyper/J User and Installation Manual. IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, <http://www.research.ibm.com/hyperspace>., 2000.
- [TOHS99] P. Tarr, H. Ossher, W. Harrison, and S. Sutton. N Degrees of Separation: Multi-dimensional Separation of Concerns. In ACM, editor, *Proceedings of the 1999 International Conference on Software Engineering*, pages 107–119, Los Angeles, CA, USA, 1999.
- [Wal99] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.
- [ZBS97] John A. Zinky, David E. Bakken, and Richard E. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1), 1997.