AspectIX contributions to the ECOOP'98 conference

Franz J. Hauck (Ed.)

September 1999 TR-I4-99-08

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany

AspectIX Contributions to ECOOP ’98 Workshops and Poster Session

Published in:
S. Demeyer, J. Bosch (Eds.): *Object-oriented technology, ECOOP ’98 workshop reader*. LNCS 1543, Springer, 1998.

3rd Workshop on Mobility and Replication

Workshop on Aspect-Oriented Programming

Posters

Support for Mobility and Replication in the *AspectIX* Architecture

M. Geier, M. Steckermeier, U. Becker, F. J. Hauck, E. Meier, U. Rastofer

University of Erlangen-Nürnberg, Germany,
{geier, mstecker, ubecker, hauck, meier, rastofer}@informatik.uni-erlangen.de,
http://www4.informatik.uni-erlangen.de/Projects/AspectIX/

Abstract. Unlike CORBA with its static client-server relationship, *AspectIX* uses the concept of distributed objects. Distributed objects consist of so called fragments, that communicate with other fragments to synthesize the desired behaviour. The local fragment can carry more semantics than a simple CORBA stub and can be replaced at runtime by another fragment to fulfill the application's requirements. *AspectIX* therefore provides a single mechanism that is especially suited to realize both: mobility and replication.

1 The *AspectIX* Architecture

From the outside, an *AspectIX* implementation looks like a standard CORBA implementation [3]. There are location transparent names for objects, which are converted to a local object referring to the distributed object.

Unlike CORBA, the *AspectIX* architecture adopts a fragmented object model similar to Fragmented Objects from INRIA [2] and Globe from the Vrije Universiteit Amsterdam [4]. A distributed object consists of several so called fragments, which can interact with each other. A client of the object needs at least one of these fragments in its local address space.

A fragment could be a simple stub (as in CORBA), which is created on the client side and connects to another server-fragment. On the other hand, fragments at the client side can be more intelligent, e.g. by realizing real-time constraints on the communication channel or replication strategies.

The local fragment of a distributed object provides an interface described in CORBA IDL. When a fragment is created, e.g. as a result parameter of a method invocation, the ORB creates two local objects in the desired target language: a fragment interface and a fragment implementation.

The fragment interface is a generic object that is automatically generated during the development process. It only depends on the IDL description of the distributed object's interface and its main purpose is to delegate method calls to the fragment implementation. In the simplest case, the implementation does nothing more than a remote method invocation, thus the combination of interface and implementation object realizes the same semantics as a traditional CORBA stub.

By the separation into fragment implementation and fragment interface, the actual structure of the distributed object is hidden to the client application. Implementation objects can be replaced and the distributed object can be extended by additional fragments dynamically at runtime. Moreover, there can be different interface objects sharing the same implementation object similar to [1], which allows to distinguish between internal interfaces needed for interfragment communication and external interfaces offered to the client application.

2 Support for Mobility and Replication

Realizing mobility and replication with this architecture is straight forward by extending and shrinking the distributed object.

In case of replication the distributed object is simply extended by an additional fragment which acts as a replica. Nevertheless this replica represents still the same distributed object, i.e. it is transparent to the client application whether it accesses a remote object or a local replica. It is the task of the replica to implement the specified consistency model communicating with the other fragments using standard *AspectIX* communication mechanisms.

For mobility, we first use the same mechanism as for replication, i.e., we extend the distributed object with a new fragment at the destination site. After transferring the whole state from the original fragment to the new one, the old fragment can be replaced by a simple stub acting as a forwarding entity. If no further communication to the distributed object is required from the original site, the fragment on this site can be deleted; this results in a migration of the distributed object. To the client application, this migration is atomic as it only sees the distributed object not its interior fragments that might be in an intermediate state.

References

1. Peter Dickman and Mesaac Makpangou: A Refinement of the Fragmented Object Model, Third International Workshop non Object-Orientation in Operating Systems (1992).
2. Mesaac Makpangou and Yvon Gourhant and Jean-Pierre Le Narzul and Marc Shapiro: Fragmented objects for distributed abstractions, In: T. L. Casavant and M. Singhal (eds.), Readings in Distributed Computing Systems, IEEE Computer Society Press (1994), 170–186.
3. Object Management Group: The Common Object Request Broker Architecture, Version 2.2. (1998).
4. M. van Steen, P. Homburg, and A.S. Tanenbaum: The architectural design of Globe: a wide-area distributed system, Technical Report IR-422, Vrije Universiteit Amsterdam (1997).

Tradeoffs of Distributed Object Models

Summary of Working Group A

Franz J. Hauck[1] and Francisco J. Ballesteros[2]

[1] IMMD IV, Univ. of Erlangen-Nürnberg, D-91058 Erlangen, Germany,
hauck@informatik.uni-erlangen.de
[2] Universidad Carlos III de Madrid, E-28911 Leganes (Madrid) Spain,
nemo@gsyc.inf.uc3m.es

Abstract. Starting with three questions concerning distributed object models working group A of the ECOOP '98 Workshop on Mobility and Replication discussed several tradeoffs of distributed object systems and models. This summary is an attempt to state the things said and lessons learned in this group.

1 Initial questions

The group started with three different but related questions concerning distributed object models:

1. Can fine-grained objects be distributed?
2. Can objects be grouped for scalability reasons?
3. How can dynamic scheduling of replication be realized?

Since a significant percentage of the working group came from the same research group at Univ. of Erlangen-Nürnberg, and being all of us so impressed by their white T-shirts with graphics from their article, most of our discussion was biased towards their research topic, the *AspectIX* architecture [1].

In *AspectIX*, the system is made of a bunch of distributed shared objects. Each distributed object is perceived as a single object by its users. However, the object is actually distributed and made of separate fragments. Such fragments cooperate through the network to implement the object as perceived by its user. The object model of *AspectIX* is very similar to the model used in *Globe* [2] and *SOS/Fragmented Objects* [3].

2 Fine-grained objects

In fragmented object models we face two different object models: one for programming distributed objects and one for local objects (fragments). This makes distributed programming more complex. It also leads to coarse-grained distributed objects and probably more fine-grained local objects. With this in mind we approached the first questions: can fine-grained objects be distributed?

The *Emerald* system [4] showed that it is possible to have a uniform object model, in which every object can be either local or distributed. So, even very fine-grained object can be distributed if necessary. Even more important is the possibility that one object of the same class can be distributed whereas another may remain local. The *Emerald* people claim to have evidence that in nonuniform models a class for local objects needs to be rewritten because an object of that class is to be distributed. This problem does not arise with a uniform object model.

As at least two participants of the working group have belonged to the *Emerald* community, the group discussed the tradeoffs between uniform and fragmented object models. The uniform model has clear advantages in complexity, and code is probably more reusable.

The main problem in large-scale systems is **heterogeneity**. This does not only mean different brands of hard- and software, but also different programming models and languages, especially as it is crucial to integrate legacy code into a new system. Thus, we have a problem to use languages like *Emerald* for this kind of systems, especially because legacy code usually does not use a uniform object model: we would have to rewrite legacy code in *Emerald*.

The focus of nonuniform models like CORBA [5] is on **interoperability**, not only between different languages but also with legacy objects. In CORBA programmers have to deal with two different models one for distribution and one for local (language-based) objects. However, even homogenious systems like Java RMI have slightly nonuniform object models for remote and local objects [6].

In *AspectIX* we also have two different models. On one hand, people programming a distributed object will perceive the object as a (distributed) set of fragments. On the other hand, looking from outside, such fragments appear to be a single distributed object. However, we could imagine some tool support that allows programmers to convert a local fragment into a distributed object and automatically generate different fragment implementations from an abstract object definition, which in turn could use a more uniform object model.

3 Grouping of objects

A uniform object model also introduces fine-grained distributed objects (e.g. integer objects). The administrative overhead of fine-grained distributed objects is very high. Migration and replication of such an object induces tremendous overhead compared to the object's initial functionality. Object grouping can be used as an aid for administrative scalability. It is the same concept used in the real world were several fine-grained objects can be clustered so they could be handled as a single one.

If objects are grouped together the per-object overhead decreases because a whole bunch of objects is migrated or replicated at the same time at almost the same cost. We can distinguish administrative overhead (e.g., costs for addressing and naming) from communication overhead (e.g., costs for data transfer).

The tradeoff arising when objects are grouped together is that of "False Sharing". The objects within a group stay together even if they are not needed together. If we like to have one of the objects locally available we have to migrate them all to our local host. If we want one of the objects replicated they all get replicated. Consider as another example a file being used by several persons, who happen to work on different sections of the file. There is no sharing, but grouping the data will make the system think there is.

4 Scheduling of replication

The third question of how we can realize dynamic scheduling of replication was only briefly discussed. First, we identified that it is not a binary question but has many intermediate approaches. We could have simple stubs talking to a server object on the one end, and we could have fully replicated state on the other end. Inbetween, we could imagine various possibilities in form of different caching techniques and consistency schemes, perhaps even applied to a part of the object's state only.

We discussed how an *AspectIX* distributed shared object could adopt different implementations (ranging for full replication to simple stubs). The fragment programmer would employ utility libraries according to the chosen semantics. Outside the object, the interface would stay the same; within the object, the replication semantics would change.

The appropriate degree of replication and the adoption of stubs or replicas was promptly identified as an open question as well as a good topic for future work.

5 Conclusion

We identified future work in the area of supporting programmers of distributed objects. If there are different object models to deal with at the same time they should be as similar as possible and objects from one model should be able to be converted to objects of the other model.

Another open question is how to adopt and dynamically change replication and caching strategies in systems like *AspectIX*. After all, this kind of systems only provides mechanisms and we have to look at the policies as a separate issue.

References

1. M. Geier, M. Steckermeier, U. Becker, M. Geier, F. Hauck, E. Meier and U. Rastofer: "Support for mobility and replication in the AspectIX architecture." In *ECOOP'98 Workshop Reader*, LNCS, Springer (1998).
2. M. van Steen, P. Homburg, and A.S. Tanenbaum: *The architectural design of Globe: a wide-area distributed system.* Technical Report IR-422, Vrije Universiteit Amsterdam (1997).
3. M. Makpangou, Y. Gourhant, J.-P. Le Narzul and M. Shapiro: "Fragmented objects for distributed abstractions." In: T. L. Casavant and M. Singhal (eds.), *Readings in Distr. Computing Systems*, IEEE Comp. Society Press (1994), 170–186.
4. R.K. Raj, E. Tempero, H.M. Levy, A.P. Black, N.C. Hutchison and E. Jul: "Emerald: A general-purpose programming language." *Software—Practice and Experience*, 21:91–118 (Jan. 1991).
5. Object Management Group: *The Common Object Request Broker Architecture*, Version 2.2. (1998).
6. Sun Microsystems Inc.: *Java Remote Method Invocatin Specification.* Mountain View, CA (Feb. 1997).

AspectIX
A Middleware for Aspect-Oriented Programming

F. Hauck, U. Becker, M. Geier, E. Meier, U. Rastofer, and M. Steckermeier

IMMD IV, Univ. of Erlangen-Nürnberg, D-91058 Erlangen, Germany,
{hauck,ubecker,geier,meier,rastofer,mstecker}
@informatik.uni-erlangen.de
http://www4.informatik.uni-erlangen.de/Projects/AspectIX/

Abstract. The *AspectIX* middleware architecture allows to specify aspects, and to program and configure these aspects via well-defined configuration interfaces. Aspects can be added and controlled dynamically. Distributed objects support aspects and clients only need to know the configuration interfaces to control aspects.

Currently we have considered aspects concerning the internal communication semantics of the object, consistency, replication, and mobility. For combining *AspectIX* with aspect-oriented programming, we envision aspect weavers that generate code for the *AspectIX* configuration interface.

1 Motivation

Object-based middleware systems, like CORBA [1], provide the basis for object-based distributed applications. Nonfunctional properties are usually addressed as add-on mechanisms of the system (e.g., CORBA services) or not addressed at all. In the latter case the user has to build his own concepts on top of the available programming model (e.g., replicated objects on top of nonreplicated objects).

Aspect-oriented programming uses specialized aspect languages to program nonfunctional properties (aspects) of an application. Thus, better isolation, composition, and reuse of the corresponding aspect code can be achieved [2]. Aspect code is typically weaved into the basic program. Therefore, the aspect code is converted to functional code that is executed as defined by the aspect's semantics.

In distributed systems, applications may consist of several parts of which the source code is not available (e.g., server code). Thus, weaving-in of aspects is not always possible. Additionally, aspects need to be integrated and controlled without recompiling the application, which is not possible with available aspect-oriented systems. The *AspectIX* middleware architecture allows to dynamically specify new aspects and to program these aspects via well-defined interfaces.

2 *AspectIX* architecture

Unlike CORBA, the *AspectIX* architecture adopts a fragmented object model similar to *Fragmented Objects* from INRIA [3] and *Globe* from the VU Amsterdam [4]. A distributed object consists of several so called fragments, which can interact with one another. A client of the object always has at least one of these fragments in its local address space and there can exist additional fragments without direct clients.

A fragment could be a simple stub (as in CORBA). The stub may connect to another fragment (in CORBA: the server object) that holds the object's functionality. On the other hand, fragments at the client side can be intelligent. An intelligent fragment may hide the replication of the distributed object's state, it may realize real-time constraints on the communication channel to the server fragment, it may cache some of the object's data, and it may locally implement some of the object's functionality, just to name a few possibilities. In general, different nonfunctional properties can be implemented by special fragments implementing the desired semantics.

Distributed objects in *AspectIX* can support multiple aspects. Aspects have specified semantics and a defined configuration object, which allows for activating an aspect (i.e., the object starts behaving as specified by the aspect) and for tuning parameters of the aspect (e.g., maximum time allowed for a method invocation). Clients can control the aspect configuration of their local fragment. Therefore, *AspectIX* provides a generic interface to retrieve and change the current aspect configurations. If a fragment cannot implement the current configuration it may load a fragment that is capable to do so. New fragment implementations supporting new aspects may be introduced at run-time.

3 Conlusion

The *AspectIX* architecture offers a generic interface for controlling nonfunctional properties of distributed objects on a per-object basis. Currently we have considered aspects concerning the internal communication semantics of the object, consistency, replication and mobility (see [5] for more details).

For combining *AspectIX* with aspect-oriented programming, we envision aspect weavers that generate code for the *AspectIX* configuration interface. As *AspectIX* is open to new aspects and as the join points are usually the same (method invocations and object creation), we can even imagine a generic and configurable aspect language that can be used for new aspects. However this is subject to future research.

References

1. Object Management Group: *The Common Object Request Broker Architecture*, V2.2. (1998).
2. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin: *Aspect-oriented programming.* Techn. Report SPL97-008P9710042, Xerox Palo Alto Res. Center, Feb. 1997.
3. M. Makpangou, Y. Gourhant, J.-P. Le Narzul and M. Shapiro: "Fragmented objects for distributed abstractions." In: T. L. Casavant and M. Singhal (eds.), *Readings in Distr. Computing Systems*, IEEE Comp. Society Press (1994), 170–186.
4. M. van Steen, P. Homburg, and A.S. Tanenbaum: *The architectural design of Globe: a wide-area distributed system.* Technical Report IR-422, Vrije Universiteit Amsterdam (1997).
5. M. Geier, M. Steckermeier, U. Becker, F. Hauck, E. Meier and U. Rastofer: "Support for mobility and replication in the AspectIX architecture." In *ECOOP'98 Workshop Reader*, LNCS, Springer (1998).

The *AspectIX* ORB Architecture

F. Hauck, U. Becker, M. Geier, E. Meier, U. Rastofer, M. Steckermeier

University of Erlangen-Nürnberg, Germany,
{hauck, ubecker, geier, meier, rastofer,
mstecker}@informatik.uni-erlangen.de,
http://www4.informatik.uni-erlangen.de/Projects/AspectIX/

The CORBA architecture defines the semantics of the interaction of distributed objects [1]. These semantics are hardly extensible. CORBA services can extend the basic functionality of an ORB, but they are based on those fixed semantics.

AspectIX is an open and more flexible architecture than CORBA, but an *AspectIX* implementation can also host CORBA-compliant applications. *AspectIX* adopts a fragmented object model similar to the *Globe* system [2], which means that each client owns a local part of the distributed object and that these local parts (called fragments) can interact with one another. A local fragment can be intelligent and carry a part of the distributed object's functionality, or it can act as a dumb stub as in the CORBA-compliant *AspectIX* profile.

With fragments the internal communication semantics of a distributed object is entirely hidden to the client. For example, the object can decide on using replication and caching of data, and on different communication semantics (e.g., fast real-time communication). Often it is also desirable to let the client influence some of these properties. Controlling nonfunctional and functional properties in an orthogonal way is the goal of aspect-oriented programming. Therefore, a set of closely related properties is called an aspect [3].

AspectIX provides generic configuration interfaces for each distributed object that allow clients to activate and control the aspects supported by the object. The object may use a different local fragment if it is more suited to fulfill the configuration. The replacement of fragments is transparent to the client.

Within the *AspectIX* project we investigate various application classes (profiles) and their requirements in form of aspect definitions (see also [4]): CORBA, wide-area systems (replication, consistency), mobile agents (mobility), and process control systems (real-time constraints, fault tolerance).

References

1. Object Management Group: *The Common Object Request Broker Architecture*, Version 2.2. (1998).
2. M. van Steen, P. Homburg, A. Tanenbaum: *The architectural design of Globe: a wide-area distributed system*, Technical Report IR-422, Vrije Univ. Amsterdam (1997).
3. F. Hauck, et al.: "AspectIX: A middleware for aspect-oriented programming." In *ECOOP'98 Workshop Reader*, LNCS, Springer (1998).
4. M. Geier, M. Steckermeier, et al.: "Support for mobility and replication in the AspectIX architecture." In *ECOOP'98 Workshop Reader*, LNCS, Springer (1998).