AspectIX
A Middleware for Aspect-Oriented Programming

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

September 1998 TR-I4-98-06

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany

This paper was submitted as a position paper to the
ECOOP '98 Workshop on Aspect-Oriented Programming.

AspectIX: A Middleware for Aspect-Oriented Programming

Franz J. Hauck, Ulrich Becker, Martin Geier,
Erich Meier, Uwe Rastofer, Martin Steckermeier

{hauck,ubecker,geier,ehmeier,rastofer,mstecker}@informatik.uni-erlangen.de
http://www4.informatik.uni-erlangen.de/Projects/AspectIX/

IMMD IV, University of Erlangen-Nürnberg
Martensstr. 1, D-91058 Erlangen, Germany
Tel.: +49.9131.85.7906, Fax: +49.9131.85.8732

Abstract: Object-based middleware systems, like CORBA, provide the basis for object-based distributed applications. For aspects concerning distributed programming an aspect weaver has to generate code which interacts with the middleware system. As parts of these aspects often have to be implemented on top of the middleware, weavers are very dependent on the middleware system and on the implementations on top of it.

This paper introduces a middleware called *AspectIX* that provides aspects as a generic mechanism to implement and control nonfunctional properties of a distributed object (e.g., properties related to the communication mechanisms used for a distributed object or the consistency policies between replicated data parts of the distributed objects). *AspectIX* provides an open and generic interface to all of these nonfunctional aspects. Aspect configuration can be controlled and changed at run-time with an immediate effect on the distributed object.

The *AspectIX* architecture allows implementations of a distributed *AspectIX* object to be dynamically replaced in order to fulfill a new aspect configuration. Thus, an *AspectIX* implementation is also open for new aspects.

1 Introduction

Object-based middleware systems, like CORBA [OMG98], 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 describe various aspects of an application. Thus, better isolation, composition, and reuse of the corresponding aspect code can be achieved [KLM+97]. Aspect code is weaved into the basic program to form a complete piece of software. Therefore, the aspect code is converted to functional code that is executed as defined by the aspect's semantics.

For aspects concerning distributed programming, an aspect weaver has to generate code that interacts with the middleware system to implement the desired semantics. Unfortunately, parts of these semantics are often implemented on top of the middleware which makes weavers very dependent on the middleware system and on the implementations on top of it.

In this paper we will present a middleware architecture called *AspectIX* that provides aspects as a generic mechanism to implement and control nonfunctional properties of distributed objects. Examples of these are properties of the communication mechanisms inside a distributed object or properties of the consistency policies between replicated data parts of the distributed objects, just to name a few. *AspectIX* provides a generic and open interface to all of these nonfunctional aspects. Aspects are typed and types represent a certain and defined semantics. If an object provides a certain type of aspect it has to implement the semantics defined to this type.

Users can control and configure the aspects provided by a distributed object manually. On the other hand, we can imagine a special weaver that can weave certain aspect languages into a final program that interacts with the *AspectIX* configuration interface.

The *AspectIX* architecture allows implementations of a distributed *AspectIX* object to be dynamically replaced in order to fulfill a new aspect configuration. Thus, an *AspectIX* implementation is also open for new aspects. New middleware mechanisms needed by new aspects may be provided by dynamically loadable modules loaded into an *AspectIX* implementation.

This paper is organized as follows: Section 2 introduces the *AspectIX* architecture. Section 3 relates to aspect weaving. The current status of the project is given in Section 4. Section 5 will give our conclusions.

2 The *AspectIX* Architecture

From the outside, an *AspectIX* implementation looks like a CORBA implementation. There are location transparent names for objects (in CORBA: Interoperable Object References, IORs), which are converted to a local object referring to the distributed object. In CORBA the local object is a stub that delegates invocations to a so-called server object.

Unlike CORBA, the *AspectIX* architecture adopts a fragmented object model similar to *Fragmented Objects* from INRIA [MGN+94] and *Globe* from the Vrije Universiteit Amsterdam [SHT97]. A distributed object consists of several so called fragments, which can interact (see Fig. 2.1). 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), which is created on the client side. The stub may connect to another and special 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 view possibilities.

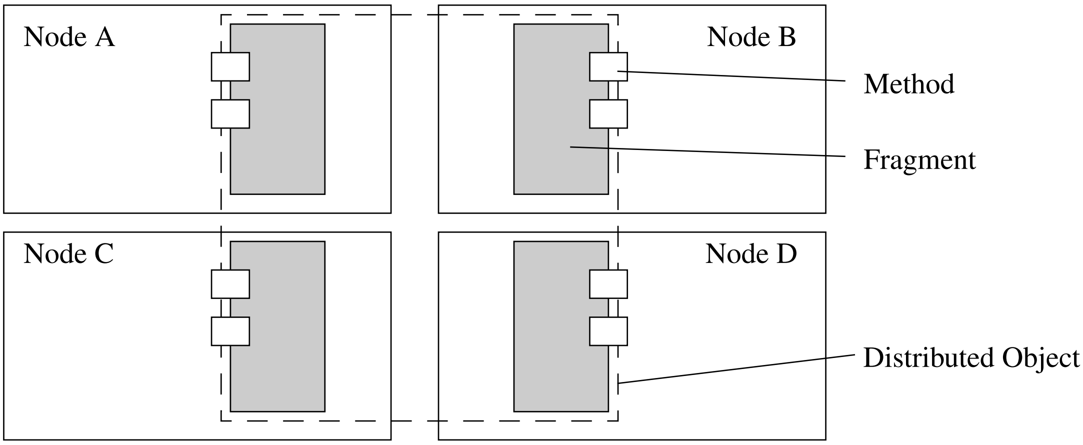

Fig. 2.1 A Fragmented Object Distributed on Four Nodes

Nonfunctional aspects of the interaction with the object can be configured at the object's interface (i.e., at each fragment) in a generic way. Thus, a client can give the maximum time for the object invocation in case of the real-time aspect, or a client can provide the sort of consistency for the local data cache of a caching fragment. In general, a distributed object can support several aspects even at the same time. Each aspect has a unique type and defined semantics. This allows portable applications within the *AspectIX* architecture as long as the *AspectIX* implementations support the same aspect types.

Aspects can be activated and deactivated during the lifetime of a fragment. The aspects' parameters may also be changed dynamically. Changing the aspects configuration may cause a fragment to replace itself by another implementation that is more suitable to fulfill the requirements.

AspectIX extends CORBA in the sense that we adopt CORBA's IDL for multiple language support and CORBA's ORB and object interfaces for CORBA compliance. Thus, the local fragment of a distributed object provides an interface described in CORBA IDL. This interface can be widened and narrowed as in CORBA. When a fragment is created, e.g., as a result parameter of a method invocation or by invoking the CORBA-compliant `string_to_object` ORB method, the ORB creates two local objects in the corresponding language mapping: a fragment interface and a fragment implementation (see Fig. 2.2).

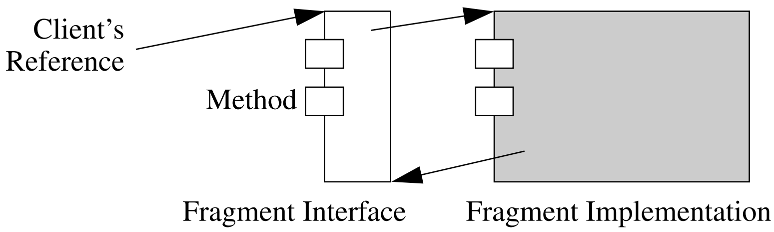

Fig. 2.2 Components of a Fragment

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. It delegates method calls to the fragment implementation, of which it maintains exactly one. The fragment interface can hide the exchange of the fragment implementation at run-time.

Each aspect is represented by a local object that can be retrieved using the generic configuration interface provided by fragment interfaces. An aspect object is typed by IDL or PIDL (pseudo IDL) respectively. The aspect object provides the aspect specific interface (e.g., methods to control the aspect's semantics). Aspect configurations can be changed, set, and checked for validity.

Changing the aspect configuration may cause a replacement of the current fragment implementation by another one that is more suited for the new configuration. Thus, the programmer of a distributed object can adopt many different implementations of the object's fragments depending on different aspect configurations. If the implementation of an aspect needs additional middleware services or mechanisms (e.g., a group communication primitive for a replication aspect) the ORB implementation may load special modules for that. Thus, *AspectIX* is open to new implementations of aspects and even to new aspects, not known in advance.

3 Aspect Weaving

So far, *AspectIX* extends CORBA to provide a generic and open interface to control and implement nonfunctional aspects of distributed objects. A client has to use this interface in order to enable his own aspect configuration for his local fragment of a distributed object.

A more sophisticated approach would be to allow the user to program in different and multiple aspect languages. An aspect weaver has to generate the necessary functional code for the corresponding aspect code. The generation of this code for the *AspectIX* configuration interface is much simpler than for individual add-on mechanisms for current middleware.

4 Status of the *AspectIX* Project

Currently, we are defining the extensions of *AspectIX* to CORBA. At the moment, only a Java language mapping is considered and defined. After that we will start building a prototype based on a public domain CORBA implementation, e.g., *OmniBroker* [LaSe97]. Adopting the C++ language mapping and the implementation of various aspects, including tests of applicability, will follow.

5 Conclusions

This paper describes the basic ideas of *AspectIX*, a new middleware architecture that extends CORBA. Nonfunctional properties of distributed programming (e.g., real-time constraints, data consistency, and all kinds of invocation semantics), can be configured using so-called typed aspect objects. The aspect configuration interface is generic and open to new aspects. This allows for manual client implementations but also for generic weavers that provide multiple aspect languages to the user.

6 References

KLM+97 G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin: *Aspect-oriented programming*. Techn. Report SPL97-008 P9710042, Xerox Palo Alto Research Center, Feb. 1997.

LaSe97 M. Laukien, U. Seimet, Object-Oriented Concepts: *OmniBroker*, Version 2.0.1 Manual. Billerica, Mass., Ettlingen, 1997.

MGN+94 M. Makpangou, Y. Gourhant, J.-P. Le Narzul, M. Shapiro: “Fragmented Objects for distributed abstractions”. In: T. L. Casavant and M. Singhal (eds.), *Readings in Distributed Computing Systems*, IEEE Computer Society Press, 1994 – pp. 170–186.

OMG98 Object Management Group: *The Common Object Request Broker Architecture*, Version 2.2. February, 1998.

SHT97 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, March 1997.