AspectIX
An Aspect-Oriented and CORBA-Compliant ORB Architecture

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

September 1998 TR-I4-98-08

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany

A shorter and earlier version of this paper was accepted as work in progress paper at the Middleware '98 Conference (Windermere, UK, Sep. 14–18, 1998).

AspectIX

AspectIX: An Aspect-Oriented and CORBA-Compliant ORB Architecture

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

{hauck,ubecker,geier,meier,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

Abstract: The CORBA architecture defines the semantics of the interaction of distributed objects. These semantics are fixed and 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, 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 be dumb and act as a stub only, as in the CORBA-compliant *AspectIX* profile.

A client can configure several functional and nonfunctional properties of the object's semantics, so-called aspects, by using a generic configuration interface. A local fragment implementation may be transparently replaced by another one if it cannot fulfill the requirements of a new configuration. We will show how this configuration interface helps to solves problems in wide-area systems (replication, consistency), mobile agents (mobility), and process control systems (real-time constraints, fault tolerance).

1 Introduction

The CORBA architecture defines the semantics of the interaction of distributed objects [OMG98]. These semantics are fixed and hardly extensible. CORBA allows synchronous object invocation with exactly-once semantics in the error-free case, and at-most-once semantics in the case of errors; it also allows asynchronous invocation with best effort at-most-once semantics. CORBA functionality can be extended by CORBA services, which more or less have to be based on these semantics. As one example, it is very hard, if not impossible at all, to integrate a <u>transparent</u> replication service into CORBA.

AspectIX is an open ORB architecture, which is more flexible than CORBA. The *AspectIX* architecture uses CORBA-compliant but extended interfaces between the ORB and applications. Thus, an *AspectIX* implementation is CORBA-compliant and is able to host CORBA applications. On the other hand, *AspectIX* objects are fully interoperable with CORBA objects.

AspectIX adopts a fragmented object model, which means that each client owns a local part of the distributed object. Communication is always made with this local part, called fragment. The local part may communicate with other fragments of the distributed object, but this communication is transparent to the client and internal to the distributed object. A local fragment can be intelligent and carry a part of the distributed object's functionality, e.g. replicate or cache data, or it can be dumb and act as a stub.

With *AspectIX*, a client can configure nonfunctional and functional properties of the semantics of a distributed object. This relates to aspect-oriented programming [KLM+97]; these properties are therefore called aspects. For that configuration, *AspectIX* provides generic interfaces that allow programmers to dynamically introduce aspects and that allow clients to individually and dynamically decide on the current configuration on a per object basis. The configuration interface is completely independent of the normal functional interface of a distributed object. Configuring aspects helps to solve a variety of problems in wide-area systems (replication, consistency), mobile agents (mobility), and process control systems (real-time constraints, fault tolerance). To grasp these problems and their aspects we defined several profiles, which define the requirements of several application classes to the ORB in form of aspect definitions.

This paper is organized as follows: Section 2 will introduce *AspectIX* architecture including the object model and the interface for configuring aspects. In Section 3, we will show four so called profiles of application classes and how they benefit from the flexibility of the *AspectIX* architecture. The current status of the project is given in Section 4. Section 5 will give our conclusion.

2 *AspectIX* Architecture

2.1 Object Model and Aspects

The *AspectIX* architecture adopts a fragmented object model similar to *Fragmented Objects* of INRIA's *SOS* [MGN+94] and *Globe* of 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 one of these fragments in its local address space, but there can exist additional fragments without direct clients. In a CORBA environment, the stub of the CORBA architecture can be considered to be a client-side fragment of the distributed object of the *AspectIX* architecture. The CORBA server object is another fragment, which may have no local client.

Unlike CORBA, fragments at the client side can also be intelligent in *AspectIX*. A fragment may hide the local 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 examples of intelligent fragments. In principle, the programmer of a distributed object has full control over the internal communication semantics used between the fragments, and over the internal semantics of fragments. The semantics are implemented with support by the ORB (e.g., support for real-time communication).

As it is also desirable to give the client of an object control over some functional and nonfunctional properties of a distributed object the programmer of such an object may add some con-

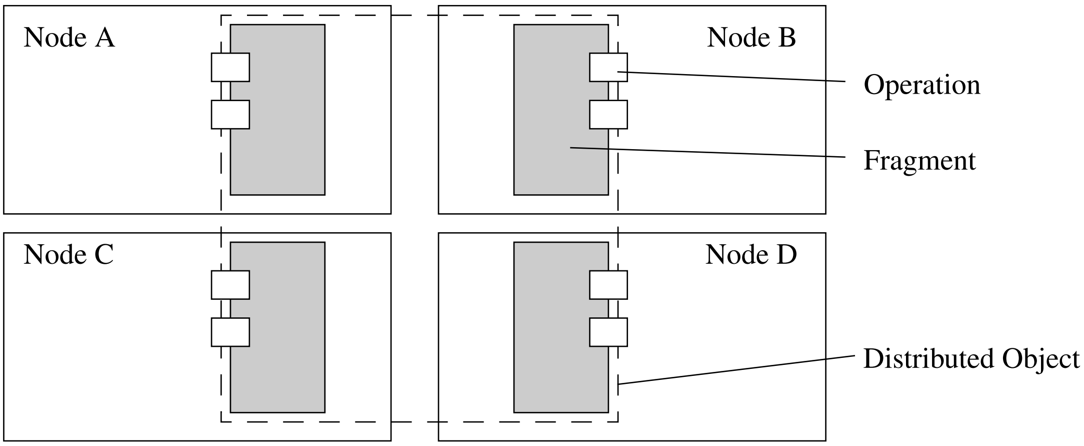

Fig. 2.1 A Distributed Object Fragmented onto Four Nodes

figuration interfaces for these properties to object. A programmer may, for example, want to allow clients to decide whether they want a local fragment that implements just a stub or a fragment that replicates the object's state. This decision may heavily depend on the needs of the client. Another example is to allow the client to configure the maximum time for the object invocation in case of a real-time aspect.

Controlling functional and nonfunctional properties in an orthogonal way is a goal met by aspect-oriented programming [KLM+97]. Therefore, we name these properties aspects. *AspectIX* helps programmers to add configuration interfaces for controlling aspects by providing a generic object interface for aspect control. In general, a distributed object can support several aspects even at the same time. Each aspect has a globally unique name and defined semantics. This allows portable applications within the *AspectIX* architecture as long as the *AspectIX* implementations support these aspects. Newer object implementations can support older aspect definition for the sake of compatibility. In any case the aspect configuration is a matter of the client and its local fragment with respect to a certain distributed object. So, different clients may use different configurations.

Aspects can be activated and deactivated during the lifetime of a fragment. The aspects' parameters may also be changed dynamically (e.g., the maximum time for an object invocation). Changing the aspects' configuration may motivate a fragment to replace itself by another implementation that is more suitable to fulfill the requirements. Thus, the programmer of a distributed object can decide to provide several fragment implementations for specific semantics or one fragment implementation being able to handle all supported aspect configurations.

2.2 Implementation of Fragments

A local fragment of a distributed object provides the normal object interface described in CORBA IDL. This interface can be widened and narrowed as in CORBA. When a fragment is created, e.g., as a return parameter of a method invocation, or by invoking the CORBA-compliant ORB method `string_to_object`[1], the ORB creates two local objects in the corresponding language mapping: a fragment interface and a fragment implementation.

1. In CORBA this method creates a local stub object for a stringified object reference.

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 replacement of the fragment implementation at run-time.

A fragment implementation may be connected to multiple fragment interfaces, which in turn may implement different IDL types (e.g., super types in the inheritance hierarchy of an object). A fragment implementation may load a new fragment implementation using some ORB mechanisms for dynamic code loading. After loading, it can trigger all fragment interfaces to use the new fragment implementation as a delegate.

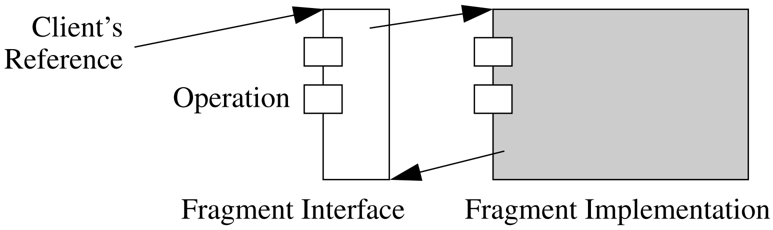

Fig. 2.2 Components of a Fragment

2.3 Aspect Configuration

The fragment interface provides not only the methods described in the object's IDL description, but also some methods defined for all CORBA objects. *AspectIX* extends this set of methods by methods for modifying and exploring the aspects, which are supported by the distributed object, and their configuration.

An aspect definition has a unique name[1] and consists of the specification of its semantics and of the definition of a specific configuration object. This object has to implement a subtype of interface `Aspect` (see Listing 2.3 of the *AspectIX* extensions in PIDL). The configuration object at least implements the methods to activate an aspect or not (e.g., for enabling local caching of data), but it may define additional methods to control aspect-specific parameters (e.g., the maximum time for a method invocation). With `get_current_aspects`, a client can get a collection of all aspect configuration objects, which are supported by the distributed object. Having the collection of configuration objects at hand, the client can retrieve the configuration object of each aspect individually, change it, and finally put it back to the collection. With `set_aspects`, a new set of aspect configurations can be set.

As it is possible that the fragment implementation cannot fulfill the new aspect configuration, the fragment implementation may replace itself. Therefore, it uses an internal interface to the fragment interface to exchange its binding to the fragment implementation. The exchange of fragment implementation is usually completely transparent to clients, as clients deal with fragment interfaces only.

AspectIX ORBs can also handle new aspect definitions, which have not been defined at start-up time. We also imagine that aspects that need additional ORB mechanisms can trigger an ORB

1. We anticipate a similar procedure as it is used to name Java classes.

```
module AspectIX
{
    typedef string AspectName;
    typedef sequence<AspectName> AspectNames;
    interface Aspect                    // object for the configuration of one aspect
    {
        AspectName get_aspect_name();
        void activate();
        void deactivate();
        boolean is_active();
    };
    interface Aspects                   // all aspects of a fragment in one container
    {
        Aspect get_aspect( in AspectName aspect_name );
        void set_aspect( in Aspect aspect );
        void remove_aspect( in AspectName aspect_name );
    };
    exception INV_ASPECT_CONF {};

    // CORBA Object Extensions
    interface Object : CORBA::Object
    {
        Aspects get_aspects( in AspectNames aspects );
        AspectNames get_current_aspect_names();
        Aspects get_current_aspects();       // get current configuration
        void change_aspects( in Aspects ac )
          raises( INV_ASPECT_CONF );         // change the mentioned aspects
        void set_aspects( in Aspects ac )
          raises( INV_ASPECT_CONF );    // set mentioned, discard prev. aspects
        void check_change_aspects( in Aspects ac )
          raises( INV_ASPECT_CONF );
        void check_set_aspects( in Aspects ac )
          raises( INV_ASPECT_CONF );
    };
};
```

Listing 2.3 CORBA Extensions of *AspectIX* in PIDL

to dynamically load new modules for these mechanisms. Thus, an ORB does not need support for all aspects at the same time and can be made very small, e.g., for use in embedded systems.

3 Application Profiles

At the moment, we consider four so called application profiles. A profile contains the requirements of a certain application class in form of several aspect definitions. Note that an aspect may be required in multiple profiles. The four profiles are: CORBA-compliant profile, profile for mobile agents, profile for wide-area distributed applications, and profile for process control systems.

CORBA-compliant profile. If a distributed object supports the CORBA aspect, then the local fragment can be converted into a dumb and CORBA-compliant stub. CORBA applications get this aspect automatically activated so that they cannot see the *AspectIX* extensions of the distributed object, but nevertheless they can access the object. One of the fragments of the object acts as a CORBA server object which is used by the stub to address invocations to. Fig. 3.1 shows a CORBA-stub fragment and its CORBA-server fragment.

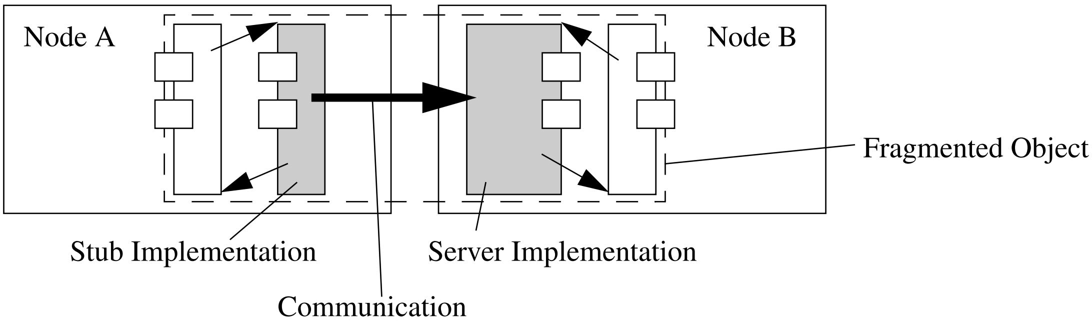

Fig. 3.1 A CORBA-Style Fragmented Object.

Profile for mobile agents. Mobile agents need mobile objects. Mobility can be easily expressed in a fragmented object model by creating a new fragment at another location and deleting the old one at the old location. The identity of the distributed object (e.g., the agent) remains the same. Fig. 3.2 demonstrates this procedure in four steps. The internal communication interface for the transfer of the fragments state can be modeled in IDL so that this approach can easily handle the problem of heterogeneity.

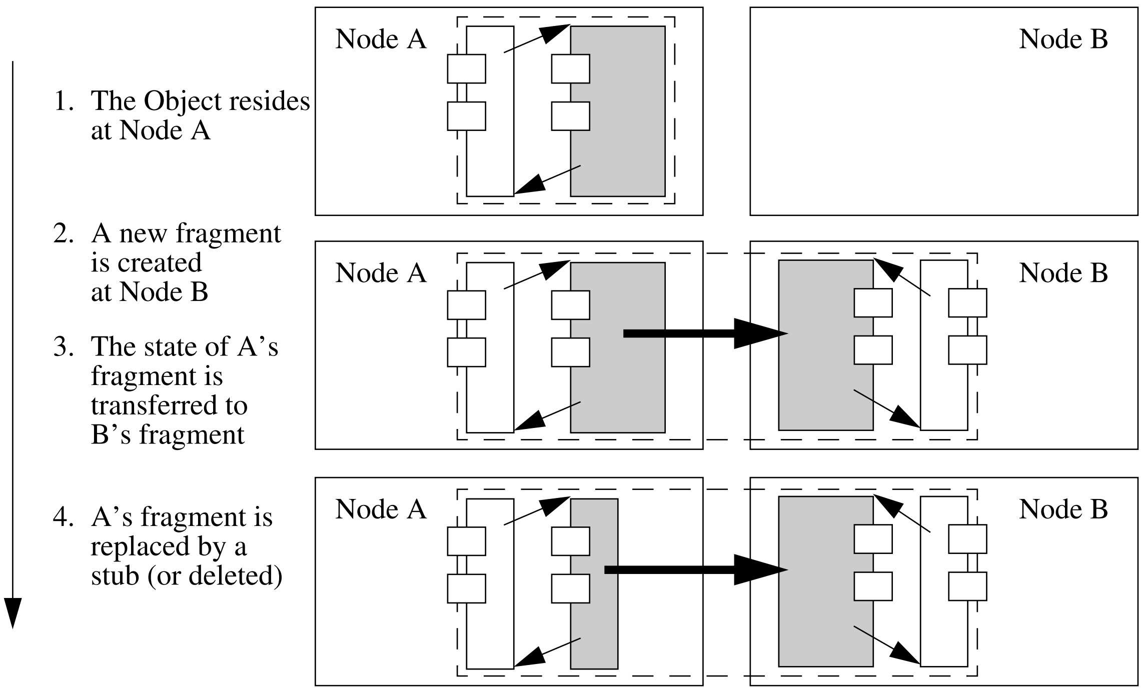

Fig. 3.2 A Mobile Object on the Move.

Profile for wide-area distributed applications. Wide area application cannot scale without replication. In CORBA it is very difficult to hide replication from the client. In the *AspectIX* architecture we can have intelligent stub fragments that choose one replica (a certain fragment) for delegating invocations to. Update operations have to be propagated to all fragments hosting the replicated state. This can be done by ordered multicast communication from the stub to all replicas or by certain update protocols run by the replicas between one another. Fig. 3.3 shows a distributed object with two fragments implementing two replicas and one fragment as client stub.

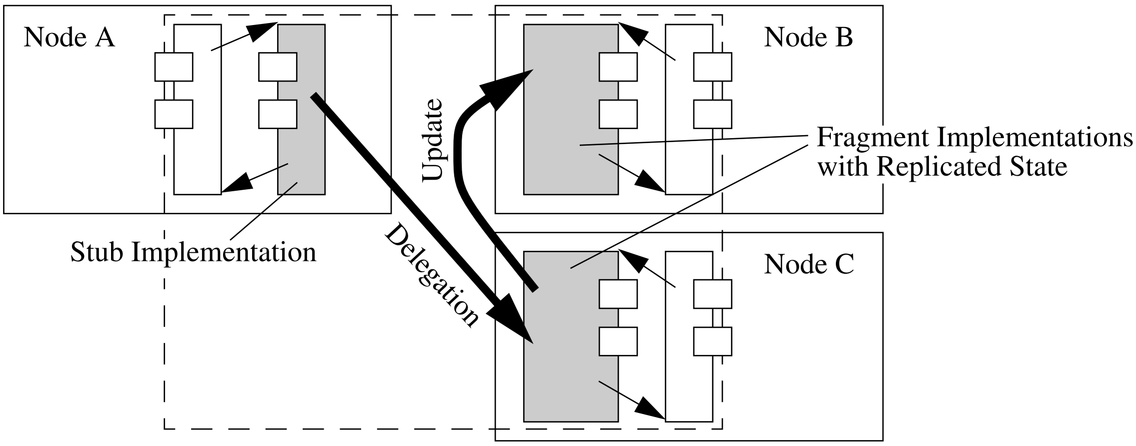

Fig. 3.3 A Fragmented Object with Replicated State and One Client.

Profile for process control systems. Process control systems can benefit from replication as well as wide-area systems (i.e., this is an example of using one aspect in several profiles). Another requirement is the definition of real-time constraints. This can be done by a real-time aspect, which allows the definition of the maximum execution time of an invocation. If the time has expired the invocation is aborted by an exception. In the case that the client needs guarantees about the execution time the fragment may also reserve resources in the ORB.

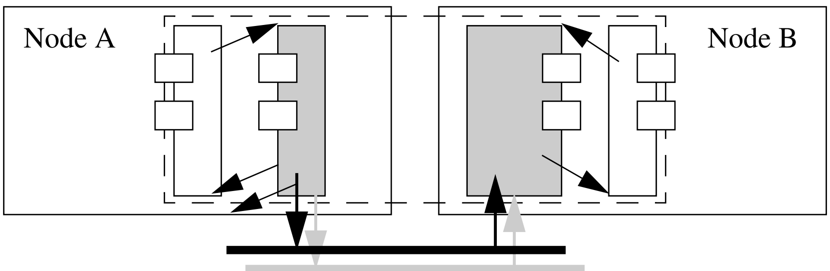

Fig. 3.4 Two Fragments Communicating by Redundant Communication Channels.

Another aspect of process control systems is to use redundant communication hardware to achieve fault tolerance. Intelligent stub fragments can switch the communication channel if the old one cannot get any connection to the server fragment (see Fig. 3.4).

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 *AspectIX*. As soon as the definitions have settled, we will start building a prototype based on public domain CORBA implementations like

ORBacus[1] [LaSe97] or *JacORB* [Bros97]. After that, we plan to continue with the C++ language mapping and implementations of various aspects including tests of applicability.

5 Conclusions

This paper describes the basic ideas of *AspectIX*, a new ORB architecture that extends CORBA. Functional and nonfunctional aspects, e.g., real-time constraints, data consistency, and all kinds of invocation semantics, can be integrated into a distributed object in a generic way. The aspects can be dynamically configured; the object's implementation can adapt dynamically to the client's needs. We imagine that it is even possible to define new aspect semantics and to dynamically download new modules into an ORB implementation if new mechanisms are needed to implement the new aspect.

Because of its generic interface for aspect configuration *AspectIX* is a very open architecture, which can be easily adapted to requirements of different classes of applications.

6 References

Bros97 G. Brose: "JacORB: Implementation and Design of a Java ORB". In: *Procs. of DAIS'97*, IFIP WG 6.1 Int. Working Conf. on Distr. Appl. and Interop. Sys., Sep. 30–Oct. 2, Cottbus, Germany, Chapman & Hall 1997.

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.

1. *ORBacus* was formerly named *OmniBroker*.