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

AspectIX: A quality-aware, object-based middleware architecture

Technical Report TR-I4-01-04
2001-05-17

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany

Informatik 4 (Distributed Systems and Operating Systems)
Prof. Dr. Fridolin Hofmann

This report was also published as:

F. J. Hauck, U. Becker, M. Geier, E. Meier, U. Rastofer, M. Steckermeier: AspectIX: a quality-aware, object-based middleware architecture. Work in Progress. *Proc. of the 3rd IFIP Int. Conf. on Distrib. Appl. and Interoperable Sys.*—DAIS (Krakow, Poland, Sep. 17–19, 2001). Kluver, 2001.

ASPECTIX: A QUALITY-AWARE, OBJECT-BASED MIDDLEWARE ARCHITECTURE

Franz J. Hauck, Ulrich Becker, Martin Geier,
Erich Meier, Uwe Rastofer, Martin Steckermeier
Informatik 4, University of Erlangen-Nürnberg, Germany

Abstract: Quality of service is becoming more and more important in distributed systems. Current middleware systems lack quality-of-service support on the application and on the system level. *AspectIX* is a CORBA-compliant middleware platform that defines generic interfaces to control quality-of-service and an infrastructure for quality implementations. *AspectIX* is based on a fragmented object model that can provide transparent client-side quality implementations. Quality implementations can be weaved into functional fragments using a hierarchy of Weavelets which are modular code-transforming software components. A distributed policy decision engine allows administrators to influence object-internal decisions, e.g., decisions about how to implement the current quality-of-service requirements.

Keywords: Quality of Service, Middleware, Distributed Objects, Programming Models for Distributed Systems, CORBA, Policy-Enabled Application

1. INTRODUCTION

Quality of service (QoS) becomes more and more relevant for distributed applications. Not only do multimedia applications need a certain bandwidth and a well-defined delivery time, but also a broad variety of traditional applications asks for some quality in terms of accuracy, security, scalability, fault tolerance, and many more. Most middleware platforms today do not address quality of service: neither do they support applications in expressing their requirements on services or application components, nor do they provide mechanisms to integrate quality implementations into the system. Furthermore, distributed applications should adapt to domain-local policies that may prescribe certain quality levels and implementation mechanisms, e.g., a certain encryption algorithm for security reasons.

This paper introduces the ongoing research project *AspectIX* that is about the design and implementation of a quality-aware middleware platform on the basis of distributed objects.

2. THE ASPECTIX MIDDLEWARE

AspectIX is a CORBA-compliant middleware system [2]. Thus *AspectIX* supports distributed objects that can be transparently invoked in a distributed system. The interface of an object is described in CORBA IDL [6]. *AspectIX* integrates quality-of-service awareness on the basis of distributed objects. So, the clients of an object and the administrators of domains and applications may want to configure their requirements on an object's behavior. The object implementation in turn will consider those requirements and use various quality implementations to not only provide its functional behavior but also the requested quality.

2.1 Quality-of-Service Interface

The client interface of an *AspectIX* object has two additional methods that can be used to configure quality-of-service requirements. This QoS interface is generic, i.e., it is the same for every quality-aware object regardless which quality requirements and implementations are supported by the object.

For historical reasons, we name every category of quality an aspect of the functional object implementation. This relates to the term aspect of aspect-oriented programming [3]. On the basis of object references, a client can provide aspect-configuration objects that describe the quality-of-service requirements of the client with respect to certain aspects (e.g., one configuration object for configuring security, another one for fault tolerance). The client can investigate which aspect configurations are supported by the object. An object can immediately refuse to accept requirements if it cannot fulfill them.

If the object accepts the aspect-configuration objects of a client, it will implicitly promise to provide the corresponding quality of service. If an object implementation can no longer fulfill those requirements (e.g., because the network is currently congested) the client will be informed via a callback interface and an exception. In such a case, the client gets not only a list of the failing aspect configuration objects but also a set of alternative configurations. The latter can be influenced by the client by assigning priorities to the different aspect configuration objects. Configuration objects with higher priority will preferably not be changed compared to the current configuration whereas others with lower priority might be changed to compute an alternative configuration that the object implementation is able to fulfill.

Administrators can influence the QoS behavior of objects by providing so-called policy rules. Such rules contain small decision programs that provide a decision for a certain decision type, e.g., shall a communication link send encrypted messages and what encryption algorithms shall be used. As we will see later, the object will request those policy decisions and thus consider the administrators wishes, especially their demands on the QoS behavior of an object.

2.2 Application Programming Model

AspectIX adopts a partitioned or fragmented object model for programming applications. A distributed object is partitioned over multiple hosts. Every client that has bound to a distributed object gets a fragment of the object in its local address space. This fragment serves as a local access point to the object. The fragment will communicate with other fragments of the same object in order to locally implement the object's functionality. In case of modelling a standard CORBA object, most fragments have simply CORBA-stub behavior whereas there is one designated server fragment that is contacted by all the stub fragments.

When it comes to quality-of-service requirements by the client, a simple stub may be not enough, as it can only communicate with a single server using CORBA's remote invocation protocol. For several quality requirements there is a need for other protocols (e.g., real-time protocols) or for some communication with multiple other fragments (e.g., for implementing fault tolerance and scalability by replication).

The local fragment is dynamically loaded by the ORB when a client binds to an object the first time. This binding process is completely transparent to the client. A client just uses CORBA's standard binding techniques (`string_to_object` and reference passing via method calls). The local fragment implementation is also able to transparently replace itself by another implementation. Repositories and location services help in loading fragment implementations and in locating the other fragments of a particular object.

Client-side quality-of-service requirements can be expressed on a per-fragment basis. For the client, all object references to the same local fragment own the same set of aspect configuration objects. There may be multiple fragments of the same distributed object in a local address space, so that different requirements to the same object can be expressed by having an own fragment for each of them.

For all kinds of strategic decisions inside of the object's fragment implementations, a policy decision engine is consulted. Instead of hard-wiring decisions into the fragment code, they are strictly separated from the corresponding mechanisms and expressed as policy rules. For every necessary decision type, the fragment and object developers provide a dedicated policy rule that can decide eventually by considering system conditions and the result of queries to external services. The decision of those developer-provided rules can be delegated to policy rules from administrators. Thus, administrators are allowed to influence not only the quality-of-service but also the object's strategic behavior. We call this concept *policy-enabled application* [5].

2.3 Object-Based Quality Implementations

We assume that fragment implementations contain the quality implementations they need, e.g., consistency protocols for replication and encryption algorithms for security. By replacing the local fragment implementation, the distributed object may switch to alternative quality implementations according to current system conditions and client requirements.

However, interlocking the functional code with different quality implementations is intricate and requires deep knowledge of the quality implementation from the developer of a service. The *AspectIX* approach to that problem is to allow quality implementors to describe a code transformation process that converts quality-unaware functional code into a fragment implementation including the required quality implementations. The object developer just has to write the functional code which is then automatically converted to a fragment implementation. Of course, there is some need for additional parameters to be given by the object developer. Those can be used to control and influence the conversion process. Examples are the tagging of methods as read or write methods, the tagging of variables as transient or persistent, etc.

The code conversion is similar to the weaving process of aspect-oriented programming (AOP) [3]. With AOP, an aspect weaver generates code from both, a functional program and an aspect program. An aspect program contains the concise code for describing an aspect of the functional program which otherwise would need code scattered over the whole functional program. Thus, the aspect program compares to the additional parameters an object developer has to provide for generating fragment implementations. As weaving is a complex process, *AspectIX* supports the quality implementor in defining it. The weaving process is modularized in a hierarchical way. The units of composition are called Weavelets which are internally represented as objects. Elementary Weavelets are provided by the *AspectIX* code generator. They can add new code at the beginning or end of a method, add new variables, change parameters and exception declarations, etc. Complex Weavelets are built by using elementary and other composite Weavelets. A top-level Weavelet finally describes the complete process of integrating a certain quality implementation with functional code. Weavelets create not only code but also skeletons for the policy rules that are necessary to decide on the decision requests inserted into the quality implementations. The skeletons have to be filled with decision code by application developers. As an alternative, the decision code can be provided as additional information to the weaving process.

Still, the interlocking of the functional code with the quality implementations is an intricate process. However, instead of scattering quality implementations over the functional code, the application developers are asked to define the necessary weaving process. This process is expressed by Weavelets. Thus, the knowledge about the weaving process is collected, preserved, and can be reused for other applications. The definition of a weaving process will become the easier the more suitable Weavelet implementations already exist.

Inside of a fragment, the quality implementation has to provide means to monitor and react on changes of the current quality characteristics. This process is supported by *AspectIX* in form of so-called QoSlets. QoSlets are code sections that can be activated by internal events, e.g., on communication failures, incoming and outbound messages, time-triggered events, etc. A QoSlet manager takes care about the correct execution of QoSlets. An activated QoSlet implements certain reactions with respect to the required quality of service, e.g., re-establishing a communication link, metering and monitoring timing and usage behavior, etc. Thus, the code of many QoSlets can be reused in different environments and forms a building block for quality implementations. Special Weavelets can insert QoSlets and QoSlet managers into a fragment implementation and thus automate the integration process.

2.4 Middleware-Based Quality Implementations

Some quality-of-service implementations have to be put into the middleware or the operating system as they touch inherent system behavior like memory management, thread scheduling and communication protocols. So far, *AspectIX* only supports protocol modules that can be dynamically loaded into the ORB in order to adapt it to varying application demands.

To make protocol modules accessible from the application, AspectIX introduces the notion of communication end points (CEPs) that form a well-defined system-independent interface for communication via arbitrary protocols. So far three different kinds of CEPs are supported: message-based, connection-based and invocation-based CEPs.

The distributed policy decision service introduced in Section 2.2 provides the evaluation of policy rules on request of a fragment implementation or even of the system itself. The core of this service consists of a distributed rule base that distributes the necessary policy rules to every location on which a decision request may be necessary. The rule base maintains rules of administrators for all locations that belong to the administrators domain. Thus, domain-dependent decisions are supported.

3. CONCLUSION

We introduced *AspectIX*, a CORBA-compliant middleware system that supports quality-of-service on a per object basis. *AspectIX* compares to some related systems: *MAQS* [1] and *QuO* [7] also integrate quality-of-service implementations into CORBA, but do not have transparent object binding, if the object should immediately use quality implementations. *AspectIX* has transparent binding, as does the *Squirrel* system [4]. Unlike *Squirrel* and *QuO*, *AspectIX* has well-defined interfaces to negotiate quality-of-service requirements. Unlike any of the other systems, *AspectIX* especially supports automatic interlocking between functional and QoS implementation by a modular weaving process.

A complete prototype of *AspectIX* is still under construction. However, a first part, the *AspectIX* IDL compiler IDL*flex* will be released in May 2001. The prototype components have been developed entirely in Java. The current status of the project can be looked up at: `http://www.aspectix.org`.

REFERENCES

1. C. Becker, K. Geihs: Generic QoS specifications for CORBA. *Proc. of Kommunikation in Verteilten Systemen*—KiVS. Informatik aktuell. Springer, 1999.
2. F. J. Hauck, E. Meier, et.al.: A middleware architecture for scalable, QoS-aware and self-organizing global services. *Proc. of the USM Conf. 2000*. LNCS 1890, Springer, 2000.
3. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin: Aspect-oriented programming. *Proc. of the ECOOP Conf.* LNCS 1241, Springer, 1997.
4. R. Koster, T. Kramp: Structuring QoS-supporting services with smart proxies. *Proc. of the Middleware 2000 Conf.*. LNCS 1795, Springer, 2000.
5. E. Meier, F. J. Hauck: *Policy-enabled applications*. Tech. Report TR-I4-99-05, IMMD IV, Univ. Erlangen-Nürnberg, July 1999.
6. Object Management Group, OMG: *The Common Object Request Broker: architecture and specification.* Rev. 2.4.2, OMG Doc. formal/01-02-33, Feb. 2001.
7. P. Pal, J. Loyall, R. Schantz, J. Zinky, R. Shapiro, J. Megquier: Using QDL to specify QoS-aware distributed (QuO) application configuration. *Proc. of the 3rd ISORC Symp.*, 2000.