Integrating Fragmented Objects into a CORBA Environment

Hans P. Reiser[1], Franz J. Hauck[2], Rüdiger Kapitza[1], and Andreas I. Schmied[2]

[1] Dept. of Distributed Systems and Operating System, University of Erlangen-Nürnberg, D-91058 Erlangen, Germany, {reiser,kapitza}@cs.fau.de
[2] Dept. of Distributed Systems, University of Ulm, D-89069 Ulm/Donau, Germany, {hauck,schmied}@informatik.uni-ulm.de
WWW home page: http://www.aspectix.org

Abstract. The design of distributed applications based on a fragmented object model has many benefits. Unlike traditional middleware with a RPC-based client-server interaction, the implementation of a fragmented object may be distributed over an arbitrary number of fragments, without restrictions on internal structure or interaction, while maintaining a transparent, standardized interface on the outside.
In this paper we describe a middleware system that integrates the concept of fragmented objects into a CORBA environment. Our fragmented objects support implicit binding using the ORB's marshalling mechanism by defining customized IOR profiles, while full interoperability with traditional CORBA applications is maintained. Furthermore, we show that a broad range of tasks in distributed systems can be solved elegantly using a fragmented object approach. Our own CORBA middleware AspectIX implements the described functionality.

1 Introduction

Fragmented objects have been proposed in previous research projects [2, 3] as a basic principle for designing distributed applications that is superior to the traditional RPC-based client-server architecture found in most traditional middleware systems. As we will explain in the next section, a wide range of important tasks may be solved by using the fragmented object model. These include the application of custom transport protocols, e.g., for multimedia applications, the transparent support for mobile objects and for fault-tolerant replication, flexible internal partitioning of objects and the development of middleware-based real-time applications. For several of these tasks, using the fragmented object model is a novel design principle.

One drawback of the systems FOG [2] and Globe [3], which directly support fragmented objects, is that they are proprietary systems that are unable to interoperate with current popular middleware platforms like CORBA [4]. In contrast, we aim at providing the advantages of the fragmented object model within a CORBA-compliant environment, where using and interacting with legacy applications is directly supported. At the same time, applications aware of the additional functionality may easily use and benefit from it.

Another essential difference to both systems is the binding mechanism used for getting access to an object. While FOG and Globe require that a client binds explicitly to a fragmented object, our middleware is able to instantiate a local fragment automatically as soon as an object reference is passed through marshalling mechanisms. Such implicit binding is a prerequisite for true object-based programming as object references have to be transparently passed around an application.

The outline of this paper is as follows. The next section discusses in more detail the fragmented object model for designing distributed systems, and illustrates the benefits of this object model for handling various tasks in distributed programming. Section 3 describes existing mechanisms in CORBA, shows how we use them in our AspectIX middleware to integrate the fragmented object model, and addresses further implementation issues. Section 4 compares our developments with related work, and Section 5 concludes.

2 Fragmented Objects in Distributed Systems

2.1 Fragmented Objects

In a traditional, RPC-based client-server structure, the complete functionality of an object resides on a single node. For transparent access to the object, a client instantiates a stub that handles remote invocations (Fig. 1a). The stub code is usually generated automatically from an interface specification. Thus the stub code is equal for all objects with the same interface. Loading the stub is accomplished by the middleware runtime system as soon as the client binds to an object reference. This binding may happen implicitly, if a remote object reference is passed to a client via the ORB's marshalling mechanisms.

In the fragmented object model, the distinction between client stubs and the server object is no longer present. From an abstract point of view, a fragmented

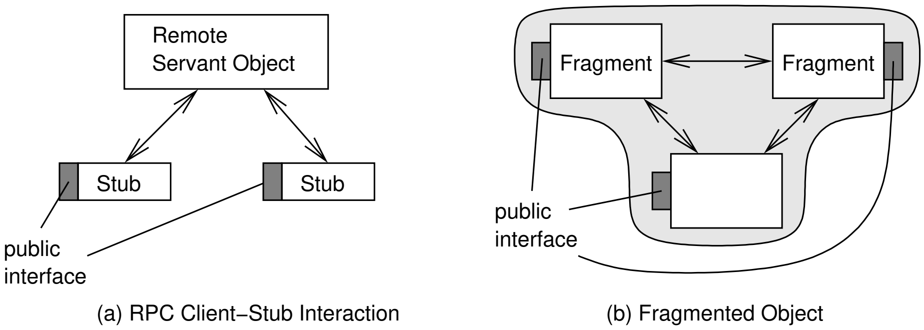

Fig. 1. RPC-based Client-Server Interaction vs. Fragment Objects

object is a unit with unique identity, interface, behavior, and state, like in classic object-oriented design. The implementation of these properties however is not bound to a certain location, but may be distributed arbitrarily on various "fragments" (Fig. 1b). Any client that wants to access the fragmented object needs a local fragment, which provides an interface identical to that of a traditional stub. However the local fragment may be specific for exactly that object. Two objects with the same interface may lead to completely different local fragments. This internal structure allows a high degree of freedom on where the state and functionality is provided, and how the interaction between fragments is done. The internal distribution and interaction is not only transparent on the outer interface of the distributed object, but may even change dynamically at runtime, e.g., in adaptive distributed systems.

There are several ways to take advantage of the high degree of internal freedom. Of course, whenever desired, a traditional RPC-based structure may also be obtained in the fragmented object model. For this purpose, a single fragment has to contain the complete implementation of the object's functionality. All other fragments will simply contain stub code, which uses a remote invocation protocol (like IIOP in the case of CORBA) to interact with the server fragment.

2.2 Fragments Acting as Smart Proxies

The idea of smart proxies [5, 6] is to put more functionality into stubs, with the purpose of, e.g., adding caching mechanisms to reduce communication overhead and latencies, or forwarding method invocations to some member of a replica group for the purpose of load balancing or fault tolerance. Current smart proxy implementations usually either modify the middleware such that compatibility is no longer maintained, or they use means such as portable interceptors (for CORBA, this is an official OMG standard [4]). In the latter case, a significant overhead in remote invocation mechanisms is introduced by adding additional levels of indirection on the client side.

In a fragmented object model, such smart proxies may be obtained in a simple and efficient way: By defining appropriate proxy fragment types, the code for the additional functionality can be loaded directly at the client side. But in addition to being used as smart proxies, the fragmented object model offers the possibility to do much more powerful things, as we will show in the next section.

2.3 Arbitrary Internal Configurations Allowing for Partitioning, Migration, and Replication

As most significant advantage object fragmentation allows the state of an object to be migrated between and distributed on several nodes, instead of keeping it completely in one single location. Again, such internal structure will be kept completely transparent for the client.

As one option, the internal configuration can be defined statically when the fragmented object is developed. The developer may describe explicitly how the functionality and state is partitioned over several locations. Mechanisms for instantiating the right fragments in the correct locations need to be provided by the middleware. We will return to the this issue later in Section 3.4.

Alternatively, one can consider the case that functionality and state of a distributed object gets relocated dynamically at runtime. A simple version might just migrate the "server" part of the object from one location to another (preferably a location where the object is most efficiently accessible) and update all other fragments with the new location information. A more sophisticated service object could allow replicating the complete state in several fragments, determined automatically depending on usage patterns and quality-of-service requirements. The most flexible version might support partitioning the object state dynamically between several nodes.

2.4 Arbitrary Internal Communication

In addition to the flexibility of internal distribution, another significant advantage arises from the fact that also the internal communication may be chosen arbitrarily. This allows to design service objects with a standard CORBA interface in situations, where the standard CORBA communication model is not practicable. For example, a multimedia service (e.g., an Internet radio or a video conference) might be modeled as a distributed service object with a certain interface offered to clients. But it is not desirable to use IIOP to transport the audio/video data, but rather an RSVP-based transport or some other real-time protocol.

In the fragmented object model, a client may simply bind to the service object, and the middleware automatically loads an appropriate fragment for using such protocol to interact with the server. This of course also extends to being able to use peer-to-peer communication mechanism between several fragments. Furthermore, one can consider the case that the middleware implementation evaluates environmental conditions (like network connection speed) and the client's quality-of-service requirements before loading and instantiating a local fragment, initialized with suitable parameters.

Another related area of usage are real-time applications. The disadvantages of most middleware systems, when trying to use them for real-time applications, are that they, firstly, involve a certain overhead, and, secondly, pass all invocations through automatically generated marshalling code and use complex communications protocols that are hard to analyze regarding their timing behavior. In a fragmented object model, the developer can implement a special fragment type that uses custom marshalling mechanisms and accesses, e.g., a CAN-bus with known characteristics for communication. The only computational overhead introduced by the middleware is one virtual method invocation from the client code to the method in the fragment implementation, which is easily bound by worst-case assumptions. The fragment code can be under full control of the application developer, and thus allows completely analyzable code with low overhead. Nevertheless the client may still use the same programming model as in a traditional CORBA environment.

3 Integrating Fragmented Objects into CORBA

We have demonstrated several situations, where the fragmented object model can be used with substantial advantages. In practice however, many distributed systems will be built based on existing systems and will be required to interact and cooperate with such. The usability of the fragmented object model is improved significantly, if it can be used within common traditional middleware systems. We shall present a successful approach how to achieve such integration into a standard CORBA environment.

3.1 Existing CORBA Mechanisms

CORBA uses IORs (Interoperable Object References) to uniquely address objects in a distributed system. An IOR has a stringified external representation and is valid independent of location. Internally, an IOR may be composed of several IOR profiles, each of them specifying a potential way to contact the object. Each type of profile is uniquely identified by a vendor tag assigned by the OMG.

For standard protocols like IIOP (Internet Inter-ORB Protocol), the profile content itself is also standardized. Taking the IIOP example, this data consists of an TCP/IP address and an unique object identifier. Using this information, a stub is able to connect to an object adaptor in a remote ORB, and to perform remote invocations. For other vendor-specific profile types, arbitrary data may be stored in the profile.

When an ORB receives a reference parameter containing an IOR, it automatically tries to instantiate a local stub for the referenced object. For this purpose, it parses the IOR to retrieve all profiles that it is able to interpret. It then tries to contact the object using the profiles in a specific order, until it succeeds. This implicit binding mechanism allows clients to pass object references to remote objects transparently in any remote method invocation.

3.2 Defining a New IOR Profile for Fragmented Objects

In the fragmented object model, the difference to this traditional CORBA behavior is only minimal. As soon as the ORB receives a reference to a fragmented object, it has to bind to this object by instantiating a local fragment of appropriate type. The basic mechanism is thus the same (parsing the IOR and interpreting the profile data to instantiate a certain piece of code), just instead of loading a fixed stub, the ORB may have to choose from a set of possible fragment types.

Consequently, we define a custom profile type (named APX) for fragmented objects in our AspectIX middleware. Whenever our ORB binds to an object containing an APX profile in its IOR, it uses this profile's data to load and initialize some fragment. The mechanisms used to decide which initial fragment is chosen will be discussed in detail in Section 3.4.

Full interoperability with traditional CORBA system is maintained in this scheme. Whenever an ORB capable of interpreting APX profiles has to bind to an object reference containing only an IIOP or other standard profile, it simply instantiates a fragment that acts as a simple stub for this protocol. Access to existing CORBA objects is thus no problem at all. In the other direction, if a fragmented object should be accessible from legacy CORBA systems, it has to implement an appropriate way of access. That is, some fragment of the object has to implement, e.g., an IIOP-compatible access point, and in addition to the APX profile, an IIOP profile is encoded into the IOR. Thus, full access to the object can be allowed when desired, of course with only limited use of the special features like custom marshalling that an ORB supporting the fragmented object model offers.

3.3 The Structure of Local Fragments

Figure 2 shows how a local fragment is structured internally. The client holds a reference to a *Fragment Interface* of the fragmented object that represents the type of the reference (any kind of interface implemented by the fragmented object). The fragment interface delegates any method invocation to the currently active fragment implementation.

The standard Java mapping of CORBA uses a similar delegation mechanism: The client holds a reference to an instance of a stub, which delegates to a low-level class that performs the actual marshalling. The delegation mechanism mainly has to purpose of allowing type casts (which replace the stub part by a new one with the desired interface, but retain the part that is delegated to). This of course means that our delegation mechanism does not introduce any additional overhead in direct comparison to CORBA.

In our fragmented object model, type casting is only one motive for the delegation. A second reason behind it is that our ORB shall be able support dynamic, transparent exchange of the local fragment implementation. Having the delegation mechanism, one may simply create a new fragment implementation and update the delegation reference in the fragment interfaces.

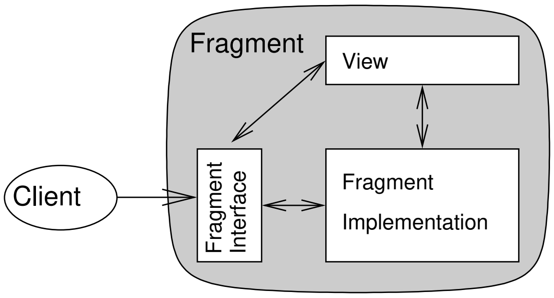

Fig. 2. Structure of Local Fragment

Additionally, we introduce a third component named *View*. It stores central data of the fragment, e.g., the IOR, and it has to maintain the fragment interfaces: It is possible that a client holds several references of different types (corresponding to different interfaces implemented by the fragmented object), all pointing to the same fragment implementation. The View helps in performing updates of the delegation pointers as soon as the fragment implementation is exchanged, inclusive handling all coordination necessary in this step. Furthermore, local quality-of-service requirements may be stored in the View, and evaluated in the process of choosing and parameterizing the local fragment.

3.4 Loading the Appropriate Fragment

Loading a simple stub in CORBA is a process that is uniquely defined by the type of the object and the IIOP contact address. In the fragmented object model, we potentially want to support loading any arbitrary fragment of some object. In our AspectIX ORB implementation, we have implemented three different kinds of mechanisms to load the initial fragment.

In the simplest model, the initial fragment type may be hard-coded into the object reference. This model is nevertheless applicable for fragmented objects: The initial fragment may contain code that takes additional steps to reconfigure or replace itself with a more suitable implementation. The developer of the initial fragment has full control over which fragment will finally be used.

In a homogeneous situation, this mechanism is sufficient. But CORBA allows heterogeneous environments, which means that there is no one-to-one relation ship between an abstract fragment type and its real implementation. The precise piece of code to be loaded may not only depend on the programming language, but also on other system properties like operating system and hardware architecture. For this purpose we implemented a Dynamic Loading Service (DLS) [8] that automatically maps a unique fragment type ID to some specific code.

Furthermore, for additional flexibility, our AspectIX middleware supports applications by means of a policy decision service that allows to evaluate expressive policy rules at runtime [9]. The decision results are based on the rules itself and on additional information like environmental conditions, user preferences and quality-of-service requirements stored in the View object. This service can be used to decide at runtime, which fragment type shall be loaded as initial local fragment. This allows separating the implementation of the object's functionality from code that decides which implementation to use, leading to a cleaner design.

3.5 Development Support

In a traditional middleware, the developer only needs to implement the server code which will run at a single location. Standard tools (like IDL compilers) are available to create stubs and skeletons automatically. For a fragmented object, potentially a larger amount of individual fragments need to be developed. That is, at first sight, the design of a fragmented object may require additional efforts.

However, most of this additional effort may also be automated by appropriate tools. As a first step in this direction, we have implemented a flexible IDL compiler tool, IDLflex [10], which allows an XML-based specification of a code generation process that creates arbitrary code based on CORBA IDL files. Fragments for custom marshalling protocols or smarter stubs are easily created automatically with it.

Upon this basic tool, we have developed a first prototype of a more powerful AspectIX Development Kit (ADK). It allows code generation not only based on IDL specification, but also on fragment source code provided by the developer and additional annotations. Even complex fragments, e.g., for partitioning and fault-tolerant replication, can be supported by this tool. The specification of the code generation process allows easy extension and reuse of existing specifications. Ultimately, even complex designs can be realized efficiently with our fragmented objects. Details will be subject of another publication.

4 Related Work

The fragmented object model was first proposed by Shapiro [1] and used in the FOG project [2]. The authors suggest this model as a promising way to design distributed applications. To a large extent, their work concentrates on tools for supporting the development of fragmented objects.

Globe [3] also uses the fragmented object model, but only for achieving scalability by caching and replication. Explicit binding is used by clients, in contrast to our implicit binding mechanism that allows for automated marshalling of references to fragmented objects and for true object-based programming. Another difference of both FOG and Globe to our AspectIX middleware is, that we provide the fragmented object model within a CORBA-compliant middleware system.

Smart proxies [5, 6] have goals similar to ours, but stay closer at the traditional client-server structure. In our opinion, the fragmented object model is not only more powerful and flexible, but at the same time is not more difficult to implement or use, and thus offers a better solution.

For custom marshalling, the CORBA standards provide for ESIOP (environment specific inter-orb protocols) [4]. These may be implemented in a CORBA ORB for some specific communication mechanisms (e.g., DCE-CIOP for using DCE-RPC for interaction). In contrast to ESIOP, our fragmented object approach allows supporting new protocols without modifying the ORB itself and also allows a dynamic selection of protocols at runtime. In this regard, our work is similar to the framework for pluggable protocols described in [7].

5 Conclusion

We have presented a novel approach for providing a fragmented object model within a CORBA-compliant middleware. The prime benefit is a transparent support for interaction models that do not map directly to the RPC-based client-server model of traditional middleware, offering a high degree of freedom.

With several case studies, we illustrated how various tasks ranging from multimedia and real-time applications to migratable and replicated services may benefit from this extended object model. Moreover, this model supports designing self-adapting applications, which are able to reconfigure themselves transparently depending on use patterns, environmental conditions and explicit quality-of-service requirements.

We have implemented the necessary mechanism in our CORBA-compliant middleware AspectIX. The current prototype allows us to verify the basic concepts presented in this paper. One major application is a fully implemented policy distribution service for our policy architecture [9]. This service is modeled as a fragmented object and supports fault tolerance via active replication, scalability through dynamic caching and hierarchical distribution of policy rules, combined with dynamic adaption to the environment and the usage pattern of the application using the policy service. A later version of our middleware system will be made available under GPL and LGPL licenses. More details can be found on our web site at `http://www.aspectix.org`

References

1. Shapiro, M.: *Structure and Encapsulation in Distributed Systems: the Proxy Principle* Proc. 6th Int. Conf. on Distributed Computing Systems, Cambridge MA, USA (1986) 198–204
2. Makpangou, M., Gourhand, Y., Le Narzul, K.-P., Shapiro, M.: *Fragmented objects for distributed abstractions.* Readings in Distr. Computing Systems, IEEE Comp. Society Press (1994) 170–186
3. Homburg, P., van Doorn, L., van Steen, M., Tanenbaum, A.S., and de Jonge, W.: *An Object Model for Flexible Distributed Systems.* Proc. First Annual ASCI Conference, Heijen, Netherlands (1995) 69–78
4. Object Management Group: *The Common Object Request Broker: Architecture and Specification.* 3.0 edition. OMG Technical Committee Document formal/02-06-01 (2002)
5. Koster, R., Kramp, T.: *Structuring QoS-supporting services with smart proxies.* Proceedings of the IFIP/ACM Middleware Conference (2000)
6. Santos, N., Marques, P., Silva, L.: *A Framework for Smart Proxies and Interceptors in RMI.* ISCA 15th Int. Conf. on Parallel and Distributed Computing Systems (2002)
7. O'Ryan, C., Kuhns, F., Schmidt, D., Othman, O., Parsons, J.: *The Design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware* Proc. IEEE Real Time Technology and Applications Symposium (1999)
8. Kapitza, R., Hauck, F.: *DLS: a CORBA Service for Dynamic Loading of Code.* Tech. Rep. TR-I4-02-06, University of Erlangen-Nürnberg, Germany (2002)
9. Meier, E., Hauck, F.: *Policy enabled applications.* Tech. Rep TR-I4-99-05, University of Erlangen-Nürnberg, Germany (1999)
10. Reiser, H., Steckermeier, M., Hauck, F.: *IDLflex: A flexible and generic compiler for CORBA IDL.* Proc. of the Net.ObjectDays, Erfurt, Germany (2001)