Stable, Time-Bound Object References in Context of Dynamically Changing Environments

Rüdiger Kapitza[1], Hans P. Reiser[2], Franz J. Hauck[2]
[1]*Dept. of Comp. Sciences, Informatik 4, University of Erlangen-Nürnberg, Germany*
rrkapitz@cs.fau.de
[2]*Distributed Systems Laboratory, University of Ulm, Germany*
{reiser,hauck}@informatik.uni-ulm.de

Abstract

Most middleware platforms lack sufficient support to provide reliable references for accessing distributed mobile objects in the context of dynamic environments, or they heavily depend on infrastructure mechanisms like a globally available location service. Based on the fragmented-object model and on our own middleware implementation AspectIX, this paper presents concepts and mechanism that avoid outdated references for dynamically distributed objects without additional infrastructure services. Our approach is based on time-bound guarantees. Even in the context of malicious nodes and a partial invalidation of references a safe binding of the distributed object is guaranteed if ever possible.

1. Introduction

Common available middleware platforms basically offer transparent remote method invocations on objects deployed over a distributed system. To achieve this they support the classical stub/skeleton-based partitioning for accessing a remote object. While this is sufficient to implement basic applications it lacks appropriate support for so called non-functional requirements like fault-tolerance, scalability and reliability. This is especially the case in context of highly dynamic environments where servers only temporarily support the provision of a service like in peer-to-peer systems or in the area of mobile computing. These environments essentially require mechanisms to replicate and migrate objects to support fault-tolerant and reliable services at the object level and mechanisms to reliably and safely reference these objects. This paper targets solutions for the latter requirement.

CORBA [3] is a distributed Middleware based on a standardized architecture that allows the programmer to create and access objects deployed over a distributed system. CORBA achieves full interoperability in heterogeneous environments by providing location, platform and language transparency. In CORBA, the common middleware tasks like object location, message transmission, and marshalling are undertaken by an Object Request Broker (ORB).

To simplify and standardize the development of highly-available, fault-tolerant distributed applications the Object Management Group has released the Fault Tolerant (FT) CORBA specification. FT CORBA provides mechanisms for entity redundancy, fault detection, and fault recovery. Each object of a fault-tolerant application is replicated via a Replication Manager that manages the replication group, and creates and destroys objects. A replicated object is realized as a group of individual CORBA objects each having the same interface and a unique reference. By aggregating the object references (IORs) of each replica of the group an Interoperable Object Group Reference (IOGR) is formed. The replication of an object is transparent to the client application. A client uses an IOGR in the same manner as it would use an IOR.

If an object group is extended or shrunk, the IOGR will be updated. To distinguish object-group changes the IOGR contains a version number, which is increased with every change. To make clients and servers aware of group changes the version information is passed with every request. If a lower version number is detected at server side, an updated version will be forwarded to the client. If a higher version number is detected, the new IOGR is accepted or the replication manager will be asked for advice.

This approach to reference replicated objects has certain weaknesses in the context of dynamically changing environments. If all hosting peers of a replicated object group change over time, previously generated IORs referencing these peers become invalid and the object can no longer be accessed. Only if the originally hosting peers still belong to the same fault-

tolerant domain hosting the object, the replication manager can provide the current contact information.

In dynamic environments like peer-to-peer systems or in the area of mobile computing this assumption cannot be kept. Such environments require that not only the members of object group can change over time but also the hosts of a fault-tolerant domain or at least their address information. Even worse, a client which has already bound to a replicated object could loose the connection to the object if it does not frequently enough call methods of the object and thereby update the IOR. Furthermore there is no information available if and when an IOR invalidates or the contact between a client and a server breaks away. The consequences are outdated and invalid IOGRs, lost connections or in the worst-case scenario the binding of wrong or even malicious objects.

In the AspectIX [11] project we implemented a CORBA-compliant ORB that provides a fragmented-object model. Fragmented objects have been proposed by other research projects [6, 10] 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. Unlike those proprietary systems, AspectIX is able to interoperate with current popular CORBA-based middleware platforms. Based on our ORB implementation an Environment for Decentralized Adaptive Services (EDAS) [9] is being developed. This project targets the provisioning of services at the object level that support various non-functional requirements in dynamic environments.

In this paper we present mechanisms to provide time-bound object references that assure a safe and reliable object binding and mechanism to keep contact information up-to-date based on the fragmented-object model. To prevent outdated references we introduce an extended IOR profile with a timestamp and an object-specific validity-period schema. Additionally we propose public-private key signing of references to prevent the binding of malicious objects. Finally we provide a simple location-service interface and an extension of the reference to free a fragmented object from managing their reference.

The remainder of the paper is organized as follows: The next section describes the fragmented-object model and relevant features of the AspectIX middleware. Section 3 describes the components of the AspectIX profile to provide stable time-bound object references. In Section 4 the profile is extended to enable a secure binding process in context of malicious nodes. Section 5 introduces infrastructure mechanisms and extends the profile. Finally, Section 6 provides a brief related work overview, and Section 7 concludes the paper.

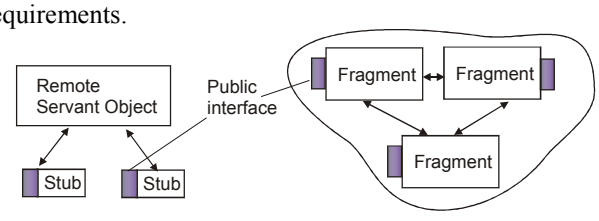
2. AspectIX Middleware

At its core the Aspectix middleware provides a CORBA-compliant ORB and, as such, supports heterogeneous distributed systems. There are extensions that allow direct interoperation with Java RMI and Jini applications. These extensions may be encapsulated in a transparent way for any client or service implementation. Our fragmented-object model, which we will explain in the next subsection, provides a generic abstraction for true distributed objects with arbitrary internal structure. Furthermore, AspectIX provides a dynamic loading service (DLS) [8] that allows loading of platform-dependent code at the client side.

2.1 The Fragmented-Object Model

In a traditional RPC-based client-server system, the complete functionality of an object resides on a single node. For transparent access to an object, a client will get an instance of a stub that handles remote invocations (Fig. 2.1 A). The stub code is usually generated automatically from an interface specification.

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 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 specific location, but may be distributed arbitrarily over various *fragments* (Fig. 2.1 B). 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 from outside of the object interface, but may even change dynamically at runtime. This allows the fragmented-object model to adapt to changing environment conditions or quality of service requirements.



2.1 Client/Server Interaction vs. Fragmented Object

In the next section we introduce the AspectIX profile and the mechanisms that enable a stable and secure binding process of fragmented objects.

3. Time-Bound Interoperable References

Similar to the FT-CORBA extension the AspectIX middleware provides an own profile to integrate fragmented objects into the CORBA world. A new fragment is created by resolving a so called Creation Service provided by the ORB as an initial service. The service allows the creation of an empty IOR. This IOR can be filled with various profiles that are also provided by the ORB. Currently the AspectIX middleware provides profiles for IIOP, Jini and fragmented objects.

3.1 Basic AspectIX Profile

The basic AspectIX profile for fragmented objects consists of the following components:

- Code reference
- Peers
- Version

The "code reference" allows specifying the default fragment implementation that is instantiated during the binding process. It could either be a direct code reference like a Java class or a symbolic name that is resolved via the dynamic loading service. The "peers" component identifies a set of arbitrary communication endpoints that are handed over to the instantiated fragment implementation during the binding process. The version number is used to distinguish the most recent IOR. Each time the peers change, the version number is increased.

3.2 Time-Bound References

So far the AspectIX profile is quite similar to the FT-CORBA reference except for the code reference. Additionally the AspectIX profile provides optionally a *validity period* component that defines how long at least one of the peers is accessible. The validity period is realized as a Universal Timestamp (UTC). Every time the peer set is changed a new timestamp could be assigned. However one has to assure that no previous version is invalidated before the corresponding validity period expires. Hence a new version may have the same validity period as previous version but never an earlier one.

The introduction of a timestamp has various consequences. First of all if a fragmented object resides in a dynamic environment like a peer-to-peer system or an ad-hoc network where no static set of peers is available (as would be required an FT-CORBA-Domain), it will offer the possibility to change the distribution without accidentally invalidating an IOR. However the redistribution of a fragmented object is bound by the guarantees given by the currently valid versions of the depending IOR.

Similar advantages apply for clients that do not possess a permanent Internet connection. This could have various reasons, for example a mobile device that can't access the Internet because the appropriate environment is not available or the network device is stopped for power management or monetary reasons. In these cases the validity period offers the clients a possibility to check whether all needed references will be still valid on reconnection and additionally take appropriate actions if this is not the case. In a power management scenario the client could wake up the network device earlier than originally planned in a cost scenario the client could use an expensive connection medium if no other is available.

3.3 Updating Time-Bound References

On the technical side time-bound IORs require that ever party has a weakly synchronized clock. We expect that a peer is either synchronized via the Network Time Protocol (NTP) or a similar accurate time source. In fact the need for accuracy of the time source strongly depends on the length of the validity period. If the distribution of fragmented object changes very quickly the lifetime of an IOR can only be very short and the internal extension of the validity period to cope with unsynchronized clocks can also only be very small. On the other side in a quite stable system the validity period of an IOR normally will be longer and the same applies to an additional span of time to tolerate badly synchronized time sources.

The length of the validity period mainly depends on the redistribution rate of the fragmented object but also on the number of peers. This is why a fragmented object has to decide how long an IOR should be valid, and with every change all previous still valid versions have to be checked. We envision two general update policies. The easier one could be called *on-update-or-expiration* policy. Each time the distribution changes (a fragment migrates to another location; a new location is added; a peer crashes or simply drops support) a new IOR is generated. This IOR is passed whenever it is needed either explicitly or implicitly (e.g., marshalling). If the most recent IOR version runs the risk of expiration a new version with a new validity period has to be generated. This should typically happen before at least one of the parties holding the IOR expects the invalidation. As a rule of thumb one would suggest half of the lifetime of an IOR.

To clarify the possible realization of the concepts we describe a brief example. A distributed, fault tolerant source code repository should be provided as fragmented object. The functionality is split into two fragment implementation, one client side which provides more or less an intelligent stub and a server implementation that realizes replicated repository storage. An instance of the fragmented object is composed for fault tolerance reasons by at least three serving fragments and depending on the demand by an arbitrary number of client fragments. State changes to the repository are propagated from a client to one of the serving fragments and than via active replication by a multicast framework between the serving fragments. If a new serving fragment is added, an existing one is removed, or a serving host crashes a multicast framework typically exchanges a view-change message. Thereby the leaving of serving peers is controlled to be in tune with guarantees given with pervious still valid IOR versions. If a valid IOR version runs the risk to invalidate before the validity period is over further leaving and changing of serving fragments is inhibited. After each view change each serving fragment creates a new version of the IOR. If there are no view changes for a long time and the most recent IOR version is at risk to invalidate one of the serving fragment detects this and announces a new validity period via the multicast framework. Each serving fragment creates a new version of the IOR with the extended validity period. The client fragment simply stores the IOR and will ask one of the serving fragments if the IOR is requested. This way the client always provides the most recent version to the outside. The client will check the validity period of the locally stored IOR. If the client is on risk to loose the connection to any serving fragment as the IOR expires, it will ask one of the serving fragments for a more recent version of the IOR.

The second update policy may best be named *continuous* policy. In contrast to the former case every time the IOR is requested either explicitly or implicitly a new validity period is assigned. This has the advantage that clients with non-permanent Internet connections know the maximal duration they can leave the network before the IOR invalidates. In the average case this period will be much longer than with the first policy. On the other hand the policy has the drawback that the object has a higher overhead to provide and maintain the corresponding information and it restricts the redistribution because at any point in time a new guarantee is given. So it is more or less anticipated that the distribution changes continuously.

Neither of the both policies fits for all use cases. A developer has to decide which policy fits best and we expect that often intermediate actualization schemes will be chosen.

4. Secure Object Binding

So far the AspectIX profile provides time-bound IORs that assure a reliable binding of a fragmented object. However in dynamic environments where fragmented objects are distributed over a set of changing peers that may belong to various independent parties, techniques for a secure object binding are necessary to cope with malicious attackers.

4.1 Basic Extension for a Secure Binding

In dynamic environments an attacker could take an address of a former peer that simply dropped the support for the fragmented object and provide an IOR that only includes peers that belong to the attacker. This might easily be the case in context of dynamically assigned addresses via a public internet service provider (ISP). A fragment-hosting peer disconnects from the network and the ISP assigns the freed IP to the attacker. This attack can be easily prevented by public key authentication. Apart from the address of each node additionally its public key is included in the IOR.

But even the authentication of nodes is not sufficient to prevent attacks at binding time because former members of the fragmented object could turn into attackers and inhibit the binding process by pretending still being a member of the object. A solution offers the introduction of a public-key-signed checksum. On creation of a fragmented object a public-key pair is generated. Then a SHA-1 checksum is built over all profile components and some general components of the IOR. This hash is signed with the private key. The checksum and the public key are added as new components to the profile. These enhanced IOR now enables a secure binding process.

In a first step the IOR has to be passed via a trusted party to the client because the passing party provides the initial contact to the fragmented object and therefore always has the chance to exchange the whole IOR. This is known as the first contact problem in the context of peer-to-peer systems like Freenet [4]. A signed public key and a well-known certificate authority might solve this problem but introduces additional complexity and overhead. If the IOR could be acquired via trusted party and the fragmented objects holds the guaranties given by the validity period the following protocol allows a reliable and safe binding of a fragmented object.

As long as the validity period of an IOR is not expired at least one of the peers still belongs to the fragmented object. So every accessible peer referenced in the IOR is asked to provide the current version of the IOR. It may appear that one or more peers provide a

newer version of the IOR. Each time the checksum and the signature are verified. If the checksum and the signature are valid all accessible peers referenced by the new IOR are contacted. It is assumed that an attacker cannot provide a new version of the IOR because she does not possess the private key. This process continues until a stable state is reached where all accessible peers provide the same version of the IOR. Then this actual version of the IOR is used to connect to the fragmented object.

Depending on the structure of the fragmented object and the communication mechanisms there is still a possibility to become a victim of a former object member that attacks the binding process. Between the search of the most recent IOR and the actual binding the fragmented object could change. Therefore, after connecting the fragmented object and before submitting requests, the most recent IOR is again evaluated. If the IOR is identically all will be fine otherwise the connection will be aborted and the fragmented object will be reconnected based on the new IOR version. Afterwards the IOR is checked again. This goes on until a former version of the IOR is identically to the version after connecting the fragmented object.

A problem might be if there are many IOR changes in a short time because the search for the current version of the IOR possibly never ends. In practice this will not be an issue because if all peers participate only for very short periods of time such a high fluctuation rate requires very short validity periods. Such small periods reduce the benefit of the overall concept and also reduce the usability of the fragmented object and therefore should be generally avoided. If the fragmented object is hosted by a quiet stable set of peer and there are additional peers that change rapidly these additional peers should not be included into the IOR if possible. Finally the usage of the location-service extension described in the following section also provides a solution for frequently changing fragmented objects with a short validity period.

4.2 Extension for a Secure Binding in Context of Malicious Attackers

The proposed additions are almost enough in context of malicious attackers belonging to the peers of the fragmented object. If a fragmented object addresses this problem for example via a Byzantine agreement protocol this has also to be considered for the binding process. Such protocols guarantee the agreement in context of m attackers as long as the overall number of participating nodes equals or exceeds 3m+1. So it is not enough that one peer is accessible for the whole validity period but it has to be a non-faulty one. Having m+1 hosts permanently available until expiration of an IOR implies that at least one non-faulty node is accessible. The actual binding process remains the same and especially the verification of the most recent IOR after connecting the fragmented object. Of course there may be up to m hosts that incorrectly answer or will not reply at all during the binding process, even if they are included in the most recent IOR, but this can be ignored.

4.3 Updating Secure References

In the pervious section two possible update schemes where proposed. These schemes also apply for the public-key-based extension of the IOR. In the first scenario where only vacated addresses could be taken by attackers one or more peers could hold the private key and sign and propagate the new version of the IOR as needed. In the second and third scenario where former member of the fragmented object or even peers of the fragmented object could be malicious attackers none of the peers should possess the private key and therefore have the possibility to provide arbitrary versions of the IOR. Instead an outstanding party holds the private key on a smart-card. This might be the manager of the fragmented object. Depending on the policy she signs a new version of the IOR and publishes it via the agreement protocol. This procedure has the minor drawback that the key holder represents a single point of failure outside of the fragmented object. The fragment object can not decide on its own for example based on a manager supplied policy when to change the IOR and instead is dependent on availability of the fragment manager or a process run on behalf of her.

A possible solution to the problem offer algorithms which generate private key shares [5]. To encrypt or decrypt a message k out of n key shares are necessary. However these parameters are fixed at creation time of a public private key pair. We currently inspect which additional actions have to be taken to address a dynamically changing set of peers.

5. Location Service

Although the validity period provides good support to keep track of the distribution changes of a fragmented object their might be situations where an IOR simply invalidates. For example if all peers of a fragmented object change rapidly and so the validity period is comparative short or in cases where a node leaves the Internet longer than expected. Aside from these situations there might be fragment implementations that do not provide the necessary mechanisms to refresh the IOR information. A fragment implementation

may only support protocols for the transmission of multimedia streams to connect to streaming servers. Even if the IOR is valid at binding time such protocols normally do not support the transmission of arbitrary, additional control information like a forthcoming address change. If the addresses of the stream provide changes the client fragments simply loose their connection. Thus, additional mechanisms have to be provided. One way would be to provide an additional communication channel is appropriate to announce such address changes. Instead of putting the burden on the fragment developer we introduce the opportunity to leave this to a location service which also offers a solution for the aforementioned problem of invalidated IORs.

To achieve this, we introduce an additional location service component to the IOR profile, which is of course also signed as it is part of profile. This optional component is simply another (embedded) IOR that references the location service of the depending fragmented object. If the IOR of an object contains a reference to a location service the service is bound first. The location service fragment is handed over to the fragmented object. The fragment implementation has now the choice to use the peers provided by the original IOR or to ask the location service for a current set.

The location service could have any interface because the fragment implementation receives a normal object from the ORB during the binding process. We propose the following API but of course a developer is not limited to use it:

```
interface LocationService {
  void registerIOR(in string ior,
                   in boolean observe)
       throws AlreadyRegistered,
                   InvalidFormat;

  void updateIOR(in string ior)
       throws InvalidFormat, NotAuthorized;

  string   getCurrentIOR(in   string   ior)
       throws  UnkownObject,  InvalidFormat;
};
```

This API supports a push- and a pull-based update of IORs. If the default fragment implementation is not able to keep track of IOR changes like in the initial mentioned streaming example the push-based update is suitable. On creation of the fragmented object it is registered at the location service via the `registerIOR()`-method and a false observe-flag. If the fragmented object was prior unknown to the location service and the IOR string is well formatted the object is registered otherwise an appropriate exception is thrown. At any point in time the IOR of a registered object can be updated under the precondition that the provided version is newer than the already registered one and has a valid signature.

If the default implementation of a fragmented object keeps track of all changes of a fragmented object there is no need for the fragmented object to announce these changes to the location service. Instead the fragmented object advises the location service to observe distribution changes via the observe flag handed over to the `registerIOR()`-method. The location service simply binds the fragmented object and now can always determine the actual IOR.

A fragment can acquire the current version of an IOR from the location service via `getCurrentIOR()` by handing over a previous version.

7. Related Work

The concept of time-bound references could be compared to leases. Middleware system like Jini [1] or .Net [12] use leases to de-allocate resources after a certain period of time. If a lease times out and is not renewed the server can safely reassign the resources to other clients. It is in the interest of a server to provide lease that are only valid as long as the client needs the resources. In contrary, time-bound references are not used to assign resources but to keep contact information up-to-date. Moreover it is in the interest of all parties to provide references that are valid as long as possible.

Various middleware systems provide mechanisms for object mobility or support the mobile agent paradigm. Emerald [7] and SSP-Chains [13] try to prevent the invalidation of references by leaving forwarding information behind at previous locations of the object. If this fails Emerald contacts registry services and if this does not help, an exhausting search over all nodes is started. Of course such a search does not scale at a global scope and static always-available registries are assumed. These assumptions cannot be hold in dynamic environments.

The FIPA [2] has specified the agent naming reference model identifier to access mobile agents. Such references consist of a set of physical addresses at which the agent could be located. If this is not the case the reference provides additionally contact information of resolving services. The specification assumes static available resolving services and provides no support for a secure binding of distributed object because a FIPA-compliant agent is always located at one physical site.

6. Conclusions

We have presented an enhanced AspectIX IOR-profile to safely bind truly distributed objects in context of dynamic unstable environments. This is

achieved by time-bound signed IORs. If such an object is actively replicated via a Byzantine agreement protocol to be aware of malicious attackers, this is taken into account by additional profile components. This way the binding process is even safe in the context of malicious attacks.

To provide long-living references and free the developer from implementing mechanisms to update the IOR, we introduced a flexible solution to provide a location service which manages the current IOR of a fragmented object.

Currently we evaluate the proposed mechanisms in context of the EDAS project that aims to provide decentralized, adaptive services. These services target dynamic environments and can change their internal service structure from a traditional client/server scenario to peer-to-peer based model where peers only temporary support service provision. Long-living references that enable a secure binding are essential for this kind of services.

Further steps will be to provide a better support for fragment developer to update an IOR either at the ORB level or as a library and supply a fault tolerant implementation of the proposed location service.

References

[1]Jini Technology Core Platform Specification.

[2]FIPA Agent Management Specifiaction. 2004,

[3]*The Common Object Request Broker Architecture and Specifications. Revision 3.0.2*, Object Management Group (OMG), Dec. 2002.

[4]I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System (2000)," in *Proc. of Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[5]Y. Frankel, P.D. MacKenzie, and M. Yung, "Robust efficient distributed RSA-key generation," in *Proc. of the Annual ACM Symposium on Theory of Comp.*, 1998, ACM Press, pp. 663 - 672.

[6]P. Homburg, L. v. Doorn, M. v. Steen, A.S. Tanenbaum, and W.d. Jonge, "An object model for flexible distributed systems," *Proc. of the 1st Annual ASCI Conf.*, 1995.

[7]E. Jul, H. Levy, N. Hutchinson, and A. Black, "Fine-Grained Mobility in the Emerald System," *ACM Trans. on Comp. Sys.*, 6(1), 1988, pp. 109-133.

[8]R. Kapitza and F.J. Hauck, "DLS: a CORBA service for dynamic loading of code," in *Proc. of the OTM Confederated Int. Conf.*, Sicily, Italy, 2003.

[9]R. Kapitza, F.J. Hauck, and H. Reiser, "Decentralized, Adaptive Services: The AspectIX Approach for a Flexible and Secure Grid Environment," in *Proc. of Grid Services Engineering and Management (GSEM 2004)*, Erfurt, Germany, 2004, Springer.

[10]M. Makpangou, Y. Gourhant, J.-P.L. Narzul, and M. Shapiro, "Fragmented Objects for Distributed Abstractions," in T. L. Casavant and M. Singhal, ed., *Readings in Distributed Computing Systems*, IEEE Computer Society Press, 1994, pp. 170-186.

[11]H. Reiser, F.J. Hauck, and R. Kapitza, "Integrating Fragmented Objects into a CORBA Environment," in *Proc. of the Net.Object Days*, Erfurt, 2003.

[12]J. Richter, *Applied Microsoft .NET Framework Programming*, Microsoft Press, 2002.

[13]M. Shapiro, P. Dickman, and D. Plainfossé. " Robust, Distributed References and Acyclic Garbage Collection," in *Proc. of Symposium on Principles of Distributed Computing*, 1992.