Using Message Reflection in a Management Architecture for CORBA

Maarten Wegdam[1,3], Dirk-Jaap Plas[1], Aart van Halteren[2,3], and Bart Nieuwenhuis[2,3]

[1] Lucent Technologies - Bell Labs Twente, Capitool 5, 7521 PL Enschede, The Netherlands
{wegdam,dplas}@lucent.com
[2] KPN Research, PO Box 96, 7500 AB Enschede, The Netherlands
{A.T.vanHalteren,L.J.M.Nieuwenhuis}@kpn.com
[3] Faculty of Computer Science, University of Twente, Enschede, The Netherlands

Abstract. The availability of object middleware, such as CORBA, is rapidly being accepted as a means for cost effective and fast development for a wide range of distributed applications. Distributed applications that are built using these technologies often comprise many objects and become more and more complex. The deployment of such large distributed applications requires a significant improvement of management methods and tools. In this paper, we present a management architecture for object middleware based systems. We use message reflection to extend the middleware layer with management capabilities, i.e. we monitor the application by observing the messages exchanged between the objects of the distributed application. We argue why management should be transparent to the application developer and show that message reflection supports this management transparency. We have compared different mechanisms to implement message reflection in CORBA, and argue why portable interceptors are the most suitable. Finally, we describe our prototype and the lessons we learned.

1. Introduction

In order to keep a deployed system in an operational and usable state, capabilities for the different areas of management have to be provided. These areas are fault management, configuration management, accounting management, performance management, and security management [9]. Object middleware technology does not offer sufficient management capabilities in any of these areas.

We believe that management should be dealt with in a generic manner, and thus should not be solved by the application developer. The application developer should only be concerned with the functional behavior of the software under development, and not with the management issues. The benefits of solving the management problem in a generic manner are that it only has to be solved once in stead of again and again for every new component or application. This reuse of code reduces, among others, development costs and time-to-market. Ideally, when developing and

A. Ambler, S.B.Calo, and G. Kar (Eds.): DSOM 2000, LNCS 1960, pp. 230 - 242, 2000.

deploying a new application the required management functionality is inherently present.

In this paper, we describe how to add management to an object middleware system in a transparent manner, effectively extending the distribution transparencies with a new management transparency. We propose to do this by using message reflection, i.e. intercepting and reflecting on the messages that are sent between the different components of an application.

In Section 2, we describe what we mean with management of middleware, the different roles we distinguish, and what the different roles require from the management system. Section 3 describes related work in this area. Section 4 compares different mechanisms to implement message reflection in CORBA. Section 5 describes our management architecture, how it uses message reflection, and our prototype. Section 6 describes the lessons learned and Section 7 finally describes our conclusions and future work.

2. Management of CORBA

We divide the management of object middleware in two separate but related issues:

- Management of the ORB itself.
  Examples are the number of threads that are available, how many threads are actually used, what network resources are available, queue sizes, the number of registered objects, used policies and the lifecycle of object adaptors.
- Management of the objects running on top of it.
  This is partly object specific and partly generic. We only consider the generic aspects in this paper. Examples are the availability of an object, i.e. whether an object is alive or not and able to respond to requests, in what stage of its lifecycle it is, what the delay on requests is, detection of user- or system exceptions that occur, the logging of requests, and the uptime.

The requirements for a management architecture for object middleware, like CORBA, can be derived by considering the parties involved in developing and deploying object middleware applications. We distinguish four different roles:

- The application or component developer
- The system administrator
- The management tool vendor
- The ORB vendor

Fig. 1 depicts who develops which part of the object middleware and management system. A different texture indicates a different role. Please note that the instrumentation is collocated with the managed entities.

The *component developer* focuses on the business logic of the component, and should be masked as much as possible from the technology specific details of the object middleware platform. The main purpose of object middleware is to provide distribution transparencies [23] such as location and access transparency. From the application developer's perspective, management of the application is not part of the business logic, and should be dealt with by the middleware. Application management should be transparent to the application developer. We refer to this as management transparency.

A *system administrator* of a multi-vendor middleware environment could be faced with different, and possibly incompatible, ways that ORB and application vendors have made their software manageable. This is not acceptable, because the system administrator can not be expected to learn all the vendor specific features in order to be able to manage a middleware-based application. Therefore, both the ORB and the components running on top of it must be manageable in a uniform manner. No matter who the ORB or component vendor is, the management view should be equal. A second requirement is that the management should demand a minimal effort from the system administrator, both when installing the management tools, and at run-time. In addition, a system administrator may want to create an integrated view covering network management, application management, and middleware management.

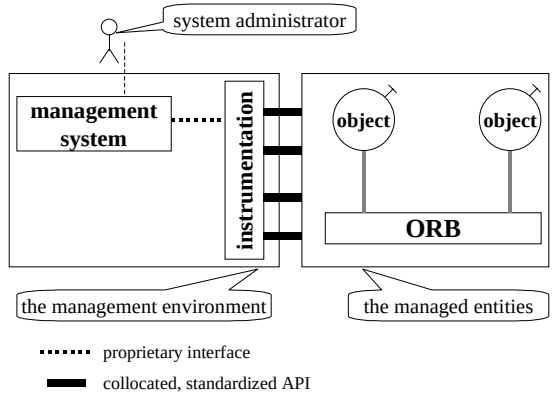

Fig. 1. The Different Roles

A *vendor of management tools* wants to provide his management solution as a third party add-on to ORB implementations of different vendors without adaptation or negotiation of a special management interface to a specific ORB. This is comparable to the requirement of a system administrator for uniform management. It requires a standardized (management) interface to an ORB to add management functionalities.

ORB vendors may not want to get into the business of ORB management tools but still want to provide a manageable product and allow the customer to choose their favorite management tool. This also requires a standardized (management) interface on the ORB.

3. Related Work

The management of networks and network elements has been an active research area for a relatively long time, resulting in mature products and standards. Examples of these are the Telecommunications Management Network (TMN)[25], the Common Management Information Protocol (CMIP)[6] and the Simple Network Management Protocol (SNMP)[24]. The management of distributed applications recently is getting more attention from academia and industry, for example within the Distributed Management Task Force (DMTF)[31]. The management of ORBs and the components running on top of it are a new area of active research. We mention the most important papers, standards and projects in both industry and academia.

In the Fachhochschule Wiesbaden (Germany) work was done on management of distributed systems [15], specifically the monitoring of Orbix based applications with the so-called ObjectMonitor. They use Orbix proprietary filters to intercept in and

outgoing requests, and have an SNMP interface to ObjectMonitor. Current research seems to mostly focus on the automation of management tasks.

A paper from the University of Aachen (Germany) describes the usage of non-co-located proxy servers to intercept in- and outgoing requests [17]. The major benefits of this approach are that it is ORB independent, and does not require recompiling an existing server. However this approach is relatively inflexible, it has to be done manually, and it probably introduces substantial overhead. Other papers, especially [16], focus more on the use of agents for management of CORBA, without explicit consideration on how to instrument the ORB and without providing a design.

In [28] three CORBA management tools for Orbix are compared. These are OrbixManager, CORBA Assistant and Object/Observer. They conclude that these three tools focus on fault and configuration management, and that manageable units are commonly CORBA processes. All three tools do instrumentation by using Orbix filters, which requires adding a few lines of application code to activate them.

The best-known management application for CORBA is probably OrbixManager [20]. OrbixManager is a combination of instrumentation to the Orbix ORB and a management service and console. OrbixManager manages the Orbix-based middleware components of an application. It extends the applications with management functionality by linking them with a management library. Some other ORB vendors have also implemented some proprietary management extensions to their ORBs. For example Inprise's Visigenic ORB [32] and BEA's Weblogic Enterprise ORB [33].

Sun, together with some other companies, made a specification called the Java Management Extensions (JMX)[13] for management of Java based application using Java technology. Main features are a push distribution mechanism, usage of JavaBeans, and remote access through several protocols, including SNMP, RMI and IIOP. JMX is based on a product of Sun called the Java Dynamic Management Kit (JDMK)[12]. A claimed benefit of JMX based products, compared to more static solutions, is that the management intelligence can be easily distributed, and can be located with the managed entity. This can reduce network traffic and increase flexibility.

Marvel [2] is a management environment that is comparable with JMX in that it also is Java-based and allows the uploading of management code to agents, and allows the usage of different management protocols. In Marvel however one can define automatically computed views of management information, which is not possible in JDMK. This can increase the scalability of the management systems. Marvel also includes functionality to visualize the management information.

Fosså and Sloman describe in [10] a management system that can be used for configuration management of a CORBA system. The Darwin configuration language is used to describe the initial configuration. A system administrator can change the configuration of the distributed system by altering the binding that exists between the CORBA objects. The objects have to be altered to allow this third-party binding. The paper focuses on the configuration description, the configuration evolution and the GUI. The implications on the object specific code is not very clear, including if this can be done in a CORBA compliant manner

MIMO, MIddleware MOnitor, is an on-line monitoring tools for distributed object-environments [22]. The distributed environment is separated in different layers, each

layer is monitored, and information from the different layers is mapped to each other. This approach has not yet been implemented, and the issue of how to instrument the monitored objects is for further research.

Research done at the University of Lancaster [3] uses reflection in middleware. They argue that current ORBs have a pre-defined and mostly standardized behavior, and that reflection can be used to easily construct a customized ORB from more or less independent components to get the behavior that is desired for a specific domain or application. An architecture for reflective middleware is described in which the meta-space of an object is divided in three different meta-models; the compositional meta-model, the encapsulation meta-model and the environment meta-model. A description of an initial implementation of this architecture, implemented in Python, can be found in [8].

4. Message Reflection

One of the major requirements stated in Section 2 is to make the management transparent to the application developer. This means among others that we cannot intertwine management functionality with the core functionality of the object. We propose to exploit the fact that distributed objects interact by exchanging messages. By intercepting and inspecting messages, we can deduct information relevant for the management of these objects. This is also known as message reflection.

In this section, we present the mechanisms for implementing message interception for management of CORBA, and we discuss the relative advantages and disadvantages of each mechanism.

Sniffing

A very straightforward method for intercepting messages is network sniffing. This is typically done by intercepting TCP/IP messages. After filtering out non-relevant messages, the IIOP messages are parsed to determine the GIOP message type and parameters, effectively de-marshalling the requests. The obvious advantage of this method is that it is completely non-intrusive and transparent for the client, the server and the ORB. Disadvantages are that only messages actually passing through the network segment will be sniffed, excluding messages sent between clients and servers on the same host. A second problem is that this method is only practical on a network that uses broadcast technology, such as Ethernet. It would otherwise require a sniffer for each host. A third limitation of this method is that does not allow message to be altered.

Instrumented Stubs and Skeletons

In the normal case of static invocations and interfaces all messages will pass through the stubs and skeletons, which the IDL compiler has generated. Since the stubs and skeletons are always available in source, they can be instrumented to read or even change messages that pass through them. The main disadvantage of this method is that it is very ORB and IDL compiler dependent. Another disadvantage is that

messages for dynamic invocations (Dynamic Invocation Interface in CORBA) and dynamic interfaces (Dynamic Skeleton Interface in CORBA) are not intercepted, since dynamic messages do not pass through the stubs or skeletons.

Wrapping

Wrapping is of course a well-known pattern to add functionality to an (existing) object or class. Wrapping can be used to intercept messages going to and from objects in a distributed application. The main advantage of this method is that it is usually transparent to the server object. The problem is that the client has to send requests to the wrapper object instead of the actual object, which is especially difficult when object references are passed between clients. This problem requires a lot of administration and thus introduces a management problem of itself. Also, it introduces a substantial delay. But in a system with a fixed number of objects on fixed locations this can be a solution worth considering.

Inheritance and Delegation

At first glance, it might seem like a good idea to use inheritance to add management intercepting capabilities to an object. One can introduce a new class at the top of the inheritance tree that all other objects inherit from, or one can do the opposite and create a subclass of an object to do the intercepting. The first approach is not suitable for intercepting messages without requiring major changes to the ORB, since the instrumentation will not be in the invocation path. It can be used to intercept lifecycle events on an object. The second approach could be a solution, but introduces so-called inheritance anomalies [1]. It is also quite intrusive to the application object and requires the usage of an object-oriented implementation language. Delegation has similar disadvantages as inheritance, especially since it is intrusive to the application object.

Composition Filters

Composition filters [1] is a modeling concept in which the actual object has explicit incoming and outgoing filters that can manipulate messages, e.g. to delay or to dispatch messages. It allows for a very clean separation of concerns, and solves the problem of inheritance anomalies. Composition filters require support by the implementation language, or even better support by CORBA (for example as an extension to IDL). Unfortunately there is only limited support for this for most implementation languages, and there is certainly no support for it in CORBA/IDL.

CORBA Portable Interceptors

Interceptors were first introduced in CORBA in version 2.2 of the CORBA specification [7]. This specification defines interceptors, which can intercept requests at defined points inside the ORB. This interceptor specification is rather ambiguous, and is about to be superseded by the *portable interceptor* specification. The portable

interceptor specification [21] defines two kinds of interceptors: request and IOR interceptors.
Request interceptors are located in the invocation path of all ORB mediated requests, thus also invocations to co-located objects. They can intercept in- and outgoing requests on both the client and the server-side, resulting in a total of four interception points, see Fig. 2.

A request interceptor can affect the outcome of a request by raising a system exception at any of the interception points, or directing a request to a different location. The target and parameters of a request can be inspected, but not altered. Several interceptor instances can be registered for one interception point, in which case they run in sequence. A request interceptor can inspect and alter the ServiceContext information.

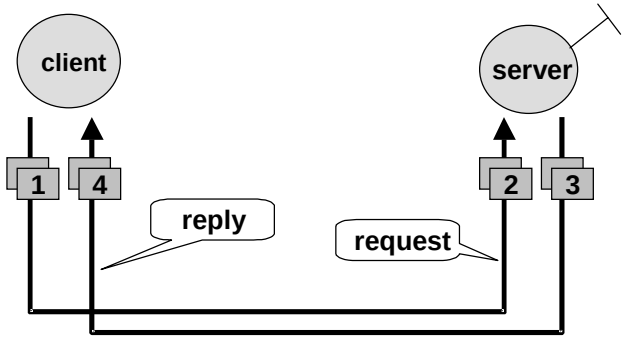

Fig. 2. Request Interceptors and the Invocation Path

IOR interceptors are purely server side, and are called when the ORB is creating an IOR, or to be more precise when it is assembling the list of components that will be included in the IOR. This does not necessarily mean that this interceptor is called for every individual object reference. For example, the Portable Object Adapter (POA) specifies policies at POA granularity and therefore this operation might be called once per POA rather than once per object.

At the time of writing some ORBs, like Orbacus [18], have already implemented the portable interceptors. Most other ORBs have either the 'old' CORBA 2.2 request interceptors, or have a similar mechanism, e.g. Orbix' filters [14] or VisiGenic's interceptors [32].

Interceptors in CORBA are relatively non-intrusive, and can be developed by the provider of a CORBA management system and simply be ‘plugged in’ into any ORB that needs to be managed. It can intercept all the requests going into and out of a CORBA object. The disadvantages are that depending on the programming language it can requires recompilation and a small code change in the application code to activate the interceptors.

Operating System Interceptors

A final mechanism we describe is what we call Operating System (OS) interceptors. These interceptors are positioned between the ORB and OS-level interface to the network. Instead of intercepting a message within the ORB, the messages are intercepted after they leave the ORB, but just before they enter the TCP/IP library. This approach is used in Eternal [19]. The major benefit of this approach is that it is

completely transparent to the component programmer and to the ORB implementation. There are however several disadvantages. A major one is that since the intercepted messages are IIOP messages, the information at this level is quite dense and requires reverse processing (i.e. de-marshalling) to obtain request information. Besides this, the method depends on the usage of dynamically linked libraries, and is dependent on the OS and network. Last but not least, requests between co-located objects cannot be intercepted, since they usually bypass the TCP/IP library.

Summary

Although the possibilities for reflection in general within CORBA are limited [27], there are several ways for extending an ORB if we limit ourselves to message reflection. We consider CORBA interceptors to be the most suitable mechanism for this, because they can intercept messages for co-located and remote invocations, without depending on the network or OS. The new portable interceptors provide for an intercepting mechanism that can be used for monitoring, but it has only limited capabilities for control functionality.

5. Architecture

In this section we describe our management architecture. Our management architecture is based on the Manager-Agent paradigm [11]. We compare how this paradigm is used in different management systems. We use this as input for our management architecture. After this we describe and evaluate our management architecture, and our initial implementation.

Manager-Agent Paradigm

The general usage of the manager-agent paradigm in management systems like SNMP and CMIP already has proven its applicability for management. This paradigm is used in two ways.

The first way is to centralize all management functionality with the manager. This approach is followed by the traditional management systems like SNMP and CMIP, and by OrbixManager. The centralization of management functionality however introduces problems with respect to scalability, information overload at the manager and network delays.

The second way is to distribute the management functionality over the manager and the agent. JMX, Marvel and the University of Aachen [16] follow this approach. By distributing the management functionality it is tried to solve the above mentioned problems.

The Manager-Agent paradigm can be implemented in several ways. The major design choices when implementing the Manager-Agent paradigm are [4]:
1) Which technology is used for communication?
2) Which part initiates the data transfer?
3) How are the parts bundled?

Based on these choices two commonly used approaches can be distinguished; the library based agent and the application based agent.

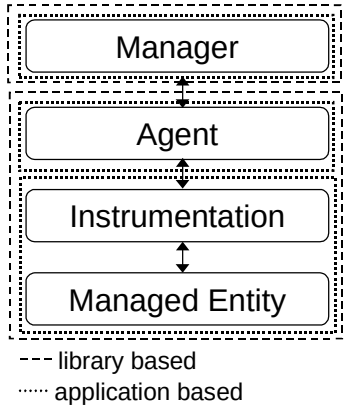

Fig. 3. Library versus Application Based Approach

In the library based agent approach the agent and the instrumentation are implemented as a library that is linked to the managed entity. The manager is the only part of the management system that runs outside the library. OrbixManager uses this approach. In the application-based approach the agent is a separate entity, and only the instrumentation is co-located with the managed entity. Both approaches are depicted in Fig. 3.

As explained in [4] the application based approach allows for more of the management functionalities to be implemented in the agent, and is thus more scalable. We are therefore using the application-based approach.

The Management Architecture

Based on the requirements mentioned in Section 2 and the above-mentioned issues, we have developed a management architecture as depicted in Fig. 4. In the following subsections we describe each part of the architecture.

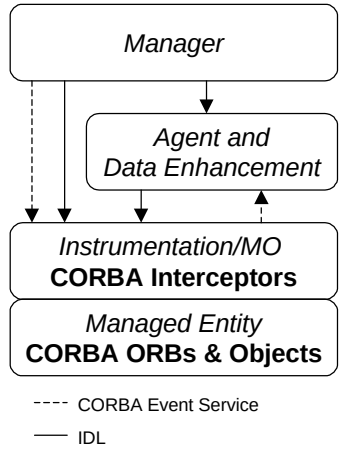

Fig. 4. The Management aArchitecture

The Manager. A management console implements the manager. It provides a Graphical User Interface to the administrator for the whole management system. It provides views for individual managed entities via the IDL interface the managed object offers, and views for groups of management entities.

The Agent. The main responsibility of the agent is to store and enhance management information. The agent provides an IDL interface for the manager to access this information. The managed objects push relevant management events though a

CORBA Event Service to the agent. The agent also uses synchronous communication (IDL) to request specific management information from the managed objects.

The Managed Object / Instrumentation. The managed object provides the agent with a management view on the managed entities. It is implemented using CORBA interceptors and standardized CORBA interfaces.

We use interceptors to monitor in and outgoing requests. The agent uses this data to derive, for example, response times, network failures and client-server relationships.

The ORB internal interfaces are not specified with the intention to be used for management. We do however use the existing POA, Object and ORB interfaces because they do offer some useful management functionalities [26] not available through interceptors.

The Managed Entity. The managed entities are the CORBA ORB core, and the CORBA objects. By managing these, we also manage the applications and services the objects are a part of.

6. Lessons Learned from the Prototype Implementation

We have prototyped our management architecture, and have successfully tested it.

With CORBA interceptors it is possible to develop management functionality independent of the ORB implementation, and thus use our management system for every ORB that implements portable interceptors. We can derive configuration, accounting, performance, and availability information about the application objects. By combining different information items the information processor is able to enhance this information, for example to determine how an application is spread over different hosts.

The control possibilities are limited by the possibilities of the instrumentation, thus the ORB interceptors and ORB internal interfaces. As a consequence our implementation has only limited possibilities for control.

The usage of portable interceptor, and thus the instrumentation, is completely transparent to the component developer because the Java language mapping allows portable interceptors to be added to already compiled code.

The management system can be distributed and, although not discussed in this paper, allows a hierarchical structure. This enhances scalability. The hierarchical structure also provides a way for management of domains of managed entities. We currently use the CORBA Event Service for asynchronous communication due to the lack of a suitable implementation of the CORBA Notification Service. The Notification Service however has a better filter and subscription mechanism, which also will reduce network traffic.

We use a generic event format defined in XML for exchanging management information. This flexible design enables easy integration with existing, e.g. SNMP or CMIP based, or new management systems.

The usage of request level interceptors introduces a significant overhead. Preliminary testing revealed that with JacORB [30] the typical delay overhead is 300 ms per request. This can be minimized by disabling interceptors that produce irrelevant management information.

7. Conclusions

We have described how message reflection can be used to manage object middleware based applications. We have compared different message reflection mechanisms in CORBA, and have selected CORBA interceptors as most suitable. CORBA interceptors can monitor object interactions in a non-intrusive manner, are ORB independent and are transparent to the object developer. We have an initial implementation of a management architecture for the management of CORBA based applications. Besides interceptors, our implementation also uses standardized ORB interfaces for instrumentation.

Based on our experiences we have identified a number of issues for further study. One of the major issues is which resources we should manage and which not. We will evaluate our current choice in different projects that use CORBA, and with experiences gained from these projects we will adjust our current choices.

We did not address the issue of policy based management in this paper. We do believe however that this is the way to go, and are working on using this within our management architecture.

Our architecture assumes one logically centralized manager that controls all the distributed components. For cross-organizational applications this will not be the case, several managers will exist, each independently managing their own domain. This will have consequences for our management architecture.

We believe that the management architecture should allow for application or environment specific management to be integrated with the more generic management functionality it now provides. This could be implemented by facilitating application specific extensions to be plugged into the management system.

We are currently working on extending our management architecture to allow for pluggable management functionalities, possibly using concepts or parts from Marvel or JMX.

Similar mechanisms as we currently use, and the same management architecture is suitable for management of the new generation component models like EJB, COM+ and CORBA Component Model (CCM) [5]. We plan to migrate our current implementation as soon as CCM ORBs become available.

References

1. Mehmet Aksit, Lodewijk Bergmans and Ken Wakita: *An Object-Oriented Model for Extensible Concurrent Systems: The Composition-Filters Approach,* IEEE Transactions on Parallel & Distributed Systems, 1993.
2. Nikolaos Anerousis: *Scalable management services using Java and the World Wide Web,* Ninth Annual IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM '98), 1998.

3. G.S. Blair, G. Coulson, P. Robin, and M. Papathomas: *An Architecture for Next Generation Middleware*, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), Lake District, UK.
4. Hajo Brunne: *Principle Design Patterns for Manageable Object Request Brokers*, Submission to the OMG CORBA Management Workshop "Vendor/System Integrator Views" Session Monday, 22 September 1997.
5. Cobb, E. et all:*CORBA Components – Volume I*, ed. E. Cobb, Joint revised Submission, OMG TC Document orbos/99-07-01, August 1999.
6. ISO: *ISO 9596: Common Management Information Protoco*l. 1991. Geneva.
7. OMG: *The Common Object Request Broker: Architecture and Specification*, formal/98-07-01 (http://www.omg.org).
8. Fabio Costa, Gordon Blair and Geoff Coulson: *Experiments with Reflective Middleware*, ECOOP Workshop on Reflective Object-Oriented Programming and Systems (ROOPS '98), Brussels.
9. ISO: *ISO 10040: Information Technology - Open Systems Interconnection - System Management overview*, 1992.
10. Halldor Fosså and Morris Sloman: *Interactive Configuration Management for Distributed Object Systems*, IEEE Proceedings First International Enterprise Distributed Object Computing Workshop (EDOC '97), Australia, 1997.
11. ISO: *ISO 7498-4: Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management framework*, 1989.
12. Sun, Java Dynamic Management Kit, http://www.sun.com/software/java-dynamic/.
13. Java Management Extensions, http://java.sun.com/jmx.
14. IONA's Orbix ORB, http://www.orbix.com.
15. Reinhold Kroeger, Markus Debusmann, Christoph Weyer, Erik Brossler, Paul Davern, Aiden McDonald: *Automated CORBA-based Application Management*, DAIS 99, 1999, Helsinki, Finland.
16. Steffen Lipperts, Anthony Sang-Bum Park: *Managing CORBA with Agents*, Interworking 98, 1998, Otawa, Canada.
17. Steffen Lipperts, Dirk Thißen, *CORBA Wrappers for A-posteriori Management: An Approach to Integrating Management with Existing Heterogeneous Systems*, DAIS '99, 1999, Helsinki, Finland.
18. ORBacus ORB, Object Oriented Concepts, http://www.ooc.com
19. Priya Narasimhan, Louise E. Moser, P.M. Melliar-Smith: *Using Interceptors to Enhance CORBA*, IEEE Computer 32[7], p. 62-68, 1999.
20. IONA, OrbixManager, part of OrbixOTM (http://www.iona.com/products/orbixenter/orbixotm/index.html).
21. OMG: Portable Interceptors revised submission, orbos/99-12-02.
22. Günther Rackl: *Multi-Layer Monitoring in Distributed Object-Environments*, second International Working Conference on Distributed Applications and Interoperable Systems (DAIS'99), Helsinki, June 1999.
23. ISO/IEC 10746-3: *Open Distributed Processing – Reference Model (RM ODP), Part 3, Architecture*, 1995.
24. IETF: *RFC 1157 – A Simple Network Management Protoco*l. 1990.
25. TeleManagement forum, http://www.tmforum.org/.
26. Maarten Wegdam, Dirk-Jaap Plas, Aart van Halteren, Bart Nieuwenhuis: *ORB instrumentation for the Management of CORBA*, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), June 26-29 2000, Las Vegas, USA.
27. Maarten Wegdam, Aart van Halteren: *Experiences with CORBA interceptors*, position paper for the Workshop on Reflective Middleware, co-located with Middleware 2000, April 2000, New York, USA.

28. Bernd Widmer, Wolfgang Lugmayr: *A comparison of three CORBA Management Tools*, Technical Report, Technical University of Vienna, TUV-1841-99-07.
29. Yasuhiko Yokote: The Apertos Reflective Operating System: The Concept and Its Implementation, OOPSLA'92 Proceedings, October 1992.
30. JacORB is an open source Java ORB developed at the Freie Universität Berlin, http://www.inf.fu-berlin.de/~brose/jacorb/.
31. Distributed Management Task Force, http://www.dmtf.org.
32. Inprise's Visigenic ORB: Visibroker, http://www.visigenic.com/visibroker/.
33. BEA's Weblogic Enterprise, http://edocs.bea.com/wle/index.html.