WIDENING TRADITIONAL MANAGEMENT PLATFORMS FOR MANAGING CORBA APPLICATIONS

Markus Debusmann, Reinhold Kroeger
Fachhochschule Wiesbaden - University of Applied Sciences
Department of Computer Science, Distributed Systems Laboratory
Kurt-Schumacher-Ring 18
D-65197 Wiesbaden, Germany
{debusman | kroeger} @ informatik.fh-wiesbaden.de
http://wwwvs.informatik.fh-wiesbaden.de

Abstract During the past years, enterprises became more and more dependent on the business processes that are often implemented using CORBA middleware. Todays management platforms ignore the dependency between applications and their middleware, and thus give up valuable information sources that are required for proactive management.

In this paper an Integration System is presented by which traditional management platforms are widened for managing CORBA applications. Thus, the investments for an existing management platform are protected. The Integration System consist of a flexible and highly available Integration Agent and the Generic Management Interface for querying management-relevant information from CORBA applications.

Keywords: CORBA, application management, management platforms

1. INTRODUCTION

Today, IT-supported business processes are the vital spots of all enterprises. To cover heterogeneity in hardware and software and to support adaptability, these business processes are frequently implemented on the basis of multi-tier architectures using a middleware layer as glue for integrating the various application components. Due to its rich functionality, CORBA [OMG, 1998b] has received broad acceptance within industry and is thus used for implementing business-critical applications more and more.

In addition, novel applications in the area of Web-based electronic business, require high availability (24x7), short response times and other service-oriented

criteria in order to achieve customer satisfaction and their long-term binding. These requirements can only be met by establishing an application management strategy that permanently monitors the status of the running application environment. Furthermore, so-called proactive management is necessary which, based on extracted information from all layers of the system, detects creeping problems and leads to corrective actions before real failures occur.

Traditional management platforms, like HP IT/Operations [HP, 1997] or Tivoli Management Environment (TME) [Tivoli, 1998] were more oriented towards element management and often only provide a very limited type of application management. Usually, existence of application processes is checked at the operating system level, and application log files may be parsed locally, checking for relevant events actively notified by the application.

In many companies large data processing centers operate the business-critical applications in three shifts. The operation requires an administration center ("Leitstand") that reflects a considerable investment also regarding the specific skills of the administrators. An individual administrator may be responsible for managing a series of different applications or infrastructure components. For him, the view upon a managed system must be as simple as possible and the frequency of events appearing on his console must be kept low in order to ensure their correct and timely processing. Especially when running a distributed CORBA application, the multiplicity of components involved and their static and dynamic relationships are hardly to supervise without any computer-based assistance. To protect the investment, operating concepts for new types of applications, like modern multi-tier CORBA-based flight information systems, have to be invented which ensure a seamless integration into the existing management environment.

As described in [Hegering et al., 1999] CORBA gains increasing significance as a management architecture and thus will serve as a basis for building management platforms in the future. Approaches like JIDM [OG, 1997] provide a simple mapping of SNMP and OSI management information models onto CORBA and vice versa and thus provide interoperability. But for short-term to mid-term solutions an integration of CORBA applications into existing management platforms beyond simple interoperability is required.

This paper presents a solution to this dilemma. The focus of the paper is on a solution, the Integration System and its central component the Integration Agent, for integrating CORBA-based applications into a traditional management environment. The approach presented here was developed in a joint research and development project between the Distributed Systems Laboratory of Fachhochschule Wiesbaden - University of Applied Sciences, and Lufthansa Systems GmbH, Kelsterbach, Germany. The project focussed on integrating CORBA-based flight information applications into traditional management platforms, HP IT/Operations in this case. To achieve this, a so-called

Integration System has been developed which is placed between the management platform and the CORBA application. The integration system hides the complexity of the managed distributed application from the administrator and allows a seamless integration into the management platform. The integration system is itself also based on CORBA.

The paper is organised as follows. Section 2 identifies the requirements for managing CORBA applications from the perspective of a data processing center in more detail. The overall architecture of the proposed solution is outlined in section 3. It mainly identifies General Management Interfaces and the Integration Agent as constituent components of the integration system. The Generic Management Interface is presented in section 4 while the concept of the Integration Agent is described in section 5 with special emphasis on its modularity and reliability. In section 6 the cascaded configuration of Integration Agents is described which solves the distributed case in general. The paper closes with a summary in section 7.

2. REQUIREMENTS FOR MANAGING CORBA APPLICATIONS

In large scale data processing centers any viable solution for managing CORBA applications has to be integratable into the existing management platform. Typical simple interfacing procedures are based on processing various application and system logs, on checking the existence of processes and on executing commands/scripts in order to launch some actions in the managed system. To achieve this, the management station of an administrator communicates with its management agents running locally on the managed nodes. Furthermore, any acceptable solution has to be as generic as possible, i.e. it should be easily adaptable to other upcoming CORBA applications, and should be highly independent of the ORB used, taking into account that no OMG management interfaces have been standardised so far.

Managing highly available applications (99,98%) as stated for modern e-business applications requires a management system that is also highly available. Generally it is expected that the management system itself is even more reliable and robust than the managed system.

In contrast to classical centralised mainframe applications, for CORBA applications the management system has to control a possibly large number of nodes and components involved. Furthermore, a CORBA application is essentially dependent upon the CORBA middleware components and the used CORBA services. Thus, all the underlying components and services have to be taken into account as well. But even modern management platforms generally have no specific build-in model which reflects the structure and the capabilities of a CORBA-based environment. While this simplifies and unifies the view

of all the managed systems, it also gives up a number of valuable information sources especially needed for proactive management which aims at early detection and removal of latent faults. In this case knowledge of the internal state of the application is generally required. This is especially true when a dynamic reconfiguration of the application during runtime has to be supported in order to meet the availability requirements.

As a result, in order to advice corrective actions the management system has to determine a global picture of the managed CORBA application by aggregating and correlating status information from the individual application components and all the underlying middleware, system and network components the application is relying on.

3. GENERAL ARCHITECTURE

Figure 1 depicts the high-level architecture of the proposed Integration System (IS) for managing CORBA applications using existing management platforms. The IS consists of the Integration Agent (IA) and the Generic Managing Interface (GMI). The IA collects management-relevant information from all layers of the managed environment: from the application layer, the CORBA layer, the system layer, and the network layer.

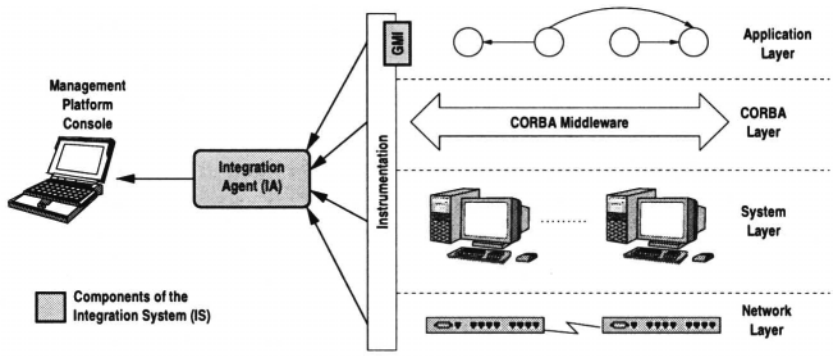

Figure 1. Overall architecture of a solution for managing CORBA applications

From each layer the information is extracted via a layer-specific instrumentation, respectively. In the application layer generic Portable Interceptors [OMG, 1999] in combination with an Application Response Measurement (ARM) instrumentation [TOG, 1998] have been successfully evaluated for performance management [Kloos, 2000] and will be the method of choice in the future. Currently hand-coded instrumentation is often used. Due to missing standardisation, the extraction of information from the CORBA layer depends on the facilities provided by the middleware platform used. If management support is provided, the CORBA layer typically appears as an SNMP private MIB ex-

tension, e.g. for IONA OrbixManager [IONA, 1998] or BEA Manager [BEA, 1998]. For the system layer and the network layer a number of information sources exist, like UNIX syslog and utilities, NT event log and system monitor, or SNMP MIBs [Stallings, 1999].

The main focus of the Integration Agent is the monitoring of the upper two layers. To provide high-quality metrics, information collected from the application layer and the CORBA layer is validated against information collected from the two lower layers.

In order to check the correct functioning of the application logic, to ensure progress and to compute service-oriented performance metrics the IA issues probing test transactions against all critical application objects and middleware services. In analogy to the network 'ping' utility for checking reachability these probing transactions have been called 'application pings'. The corresponding functions have been realised as part of the GMI which is a CORBA interface attached to all relevant CORBA application components. The GMI is covered in detail in section 4. As described above, the CORBA services which cannot be enhanced by a GMI have to be monitored as well in an appropriate manner. For them, service requests are issued which are used for error detection and measuring response times thus excluding stuck-at problems or overload situations. As an example, the CORBA Naming Service [OMG, 1998a] is queried to assure that all necessary CORBA application objects are registered. Subsequently, these can be checked via test transactions supplied through their management interfaces.

All the described checking and validating logic is encapsulated in the IA which constitutes the second type of components of the Integration System. An Integration Agent collects information from different sources, processes incoming notifications, executes validating logical functions, computes metrics and communicates with the traditional management platform, if so desired. The needed high level of adaptability of an IA and its basic fault-tolerance are its main non-functional properties. As will be seen, an IA can exhibit CORBA client and CORBA server functionality. Thus, IAs may be cascaded for covering distributed environments and for supporting aggregation and abstraction (see section 6).

The computed metrics and events generated by the IA are offered to traditional management platforms. The mechanism for providing these types of information is an implementation issue. A possible solution is using the API provided by the management platform directly from the Integration Agent.

4. A GENERIC MANAGEMENT INTERFACE FOR CORBA APPLICATIONS

The Generic Management Interface extends the CORBA application via a static CORBA IDL interface and enables management applications to access management-relevant information from the application objects. The GMI supports the following functions defined in CORBA IDL: *list* (query the information model offered by the application object), *get* (read a variable), *set* (write to a variable), *action* (invoke an operation through the GMI), and *ping* (execute a test transaction, a so-called *application ping*).

The information exchanged via the methods of the GMI is encoded using the Extensible Markup Language (XML) [W3C, 2000]. An XML Document Type Definition (DTD) defines the formats of the exchanged XML documents. This solution has the advantage that the methods of the management interface are always the same, i.e. all application objects that support the GMI can be managed by the same manager. The concrete management information and functionality is individually defined for each managed object type by its associated information model.

Compared to the CORBA `any` type, the use of XML has several advantages. XML supports the definition of grammars that determine the format of the interchanged information which provides great flexibility. In addition, XML is in a human readable format and is supported by numerous tools which enormously support the development and test phase. From a performance perspective, the integration of an XML parser is not a lightweight solution. But on the other hand, the use of CORBA `any` and their dynamic decomposition using the CORBA `DynAny` is also a very expensive approach.

Figure 2 depicts the integration of the GMI into an application object. Of course, the taken approach requires application-specific code for defining and supporting the offered information model. Requests to the GMI are passed to an XML parser that analyses the XML documents given as parameters to the requests. After parsing the XML documents, an access controller handles the interaction with the application logic, e.g. the value of some application variables may be read or a reconfiguration action may be executed. The result is transformed back into an XML document and returned to the caller of the GMI.

Using the application ping the business logic of the application object can be checked. During an application ping all relevant parts of the application object should be involved. To the outside world the result generally is a simple boolean value representing the status of the application object (red/green decision). But it is also possible to have a more fine-grained result represented as a numerical value or an enumeration. Furthermore, the caller of the application ping may take a timestamp immediately before the call and when it returns.

From measuring this response time of the probing transaction the caller may also deduce some load information.

The GMI enables a flexible monitoring and customizing of application objects during runtime as needed by the IA. Furthermore, not discussed here in detail, a CORBA client for the GMI with a graphical user interface has been developed which allows for visualising internal variables and metrics of the running application according to the exposed information model.

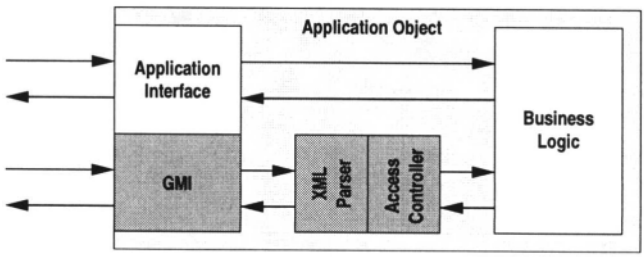

Figure 2. Integration of the Generic Management Interface into an application object

5. CONCEPT OF THE INTEGRATION AGENT

This section describes the Integration Agent with special emphasis on how its main design goals, namely modularity and high availability, have been achieved. Due to these features, it is well-suited to extend existing management platforms for the ability to manage CORBA applications.

Modularity is required in order to integrate a large number of different information sources and to adapt the management logic to the specific project needs. The IA provides a framework for dynamically loading modules at runtime that provide the actual functionality of the IA. This framework is also the basis for defining a hierarchy of operating modes to cope with failure situations.

The principal structure of the IA is depicted in Figure 3. The central component is the Module Manager. It provides a generic mechanism for dynamically loading modules into the IA and activating them during runtime. Loaded modules are under control of the Module Manager.

Furthermore, the Module Manager provides a framework for concurrent communication between modules. Communication is based on asynchronous messaging, and thus enables concurrency within the IA. To achieve this, the Module Manager has an internal pool of threads that handle incoming messages. The number of threads in the thread pool is limited and all threads are created during initialisation. This prevents performance losses because of thread creation and deletion. In addition, an inflation of the process is prevented.

Modules are divided in so-called action modules, event modules, and logic modules respectively. Action modules are used for carrying out desired actions

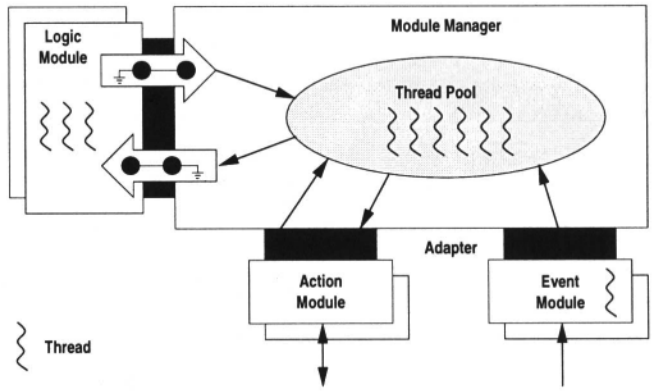

Figure 3. Internal architecture of the Integration Agent

in the managed system, e.g. querying an information source or starting/stopping a process. An action module has to implement a simple generic interface that is used by the Module Manager to communicate with the module. All functionality required to communicate with the outside world is encapsulated by an action module and thus transparent for the Module Manager. In principle, action modules are of relatively small complexity. Compared to the Module Manager, action modules are passive. They are activated if a thread of the Module Manager executes methods of an action module.

Event modules are used for integrating event sources into the IA. In contrast to action modules, event modules are active and wait for the monitored system to submit an event, e.g. sending an SNMP trap or writing a log entry. In case an external event is detected, the event module generates an internal event that is handled by the logic modules.

The logic modules are the driving components of the IA and implement the intrinsic management algorithms. Logic modules are started by the Module Manager. Subsequently, they run concurrently to the Module Manager, i.e. they normally create their own internal threads that handle the necessary tasks. Logic modules are able to send commands to action modules and process their results. Furthermore, logic modules can handle events that were sent from event modules. In addition, logic modules can also sent their own events, e.g. if one logic module detects a critical situation it may advertise this by sending events to other logic modules.

The adapter implements a common interface between the Module Manager and the modules. It abstracts from the various module types and is responsible for the dynamic loading and unloading of a module.

So far, a number of concrete modules are available. Action modules exist for sending SNMP requests, for executing shell scripts and programs, and for

performing CORBA method invocations (CORBA client). An event module is implemented that analyses a proprietary log format. Furthermore, logic modules exist for checking system configurations, for writing a logfile to the IT/O management platform, and for offering a Generic Management Interface (CORBA server) as described in section 4. The latter module enables the distribution of Integration Agents as described in the following section.

High availability is an essential characteristic of a management system when managing business-critical applications. To assure high availability the IA applies a self-checking process pair, continuous self-diagnosis, and hierarchical operating modes.

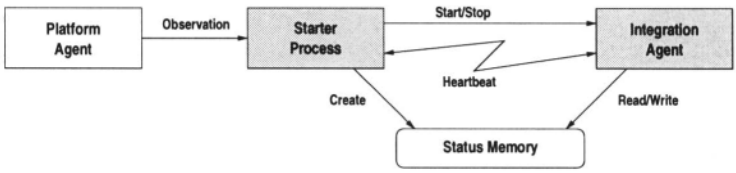

Figure 4. Relationship between the Starter Process and the Integration Agent

The operating system's view upon the IA is illustrated in Figure 4. The IA is a process under the control of a so-called starter process which starts and stops the IA. The complexity of the starter is relatively small thus ensuring a high level of correctness. The IA and the starter process regularly exchange heartbeat signals to indicate each other that they are working correctly. If the starter does not receive a signal it assumes that the IA is no longer running or hangs and restarts it again. In case the IA does not receive a heartbeat signal from the starter it shuts down itself. To complete a supervising chain, the existence of the starter is ensured by the traditional management platform.

The status memory, a shared memory segment that is created by the starter during initialisation and loaded from a file, is used for storing relevant status information of the IA which is intended to survive crashes of the IA. After a restart the IA reads from the status memory and thus can immediately resume from its saved state. Each module has its own section within the shared memory segment and is responsible to assure the consistency of its part of the shared memory segment by setting a consistency flag after all write operations to its section have been processed successfully. The last consistent state of the module is kept in the shared memory segmentt itself and is used after a crash.

By self-diagnosis of the loaded modules and the Module Manager error detection takes place. If loaded modules are considered faulty or if their operating preconditions are no longer valid they are swapped out of the IA. To provide an orderly service, so-called operating modes have been introduced. Each mode is characterised by a set of successfully running modules. The modes depend successively on each other, i.e. the higher the operating mode the more func-

tionality is integrated into the IA by the loaded modules. The Module Manager will always try to be in the highest possible operating mode thus automatically adapting to the current problem situation.

Within the cooperation with Lufthansa Systems the following three modes were identified and realised. *Mode 0 (Hardcore)* comprises the basic functionality. Within this mode the IA only depends on services provided by the operating system, i.e checking the existence of processes through a system module, log processing and SNMP. In *Mode 1 (CORBA Client)* the IA has the ability to invoke methods of CORBA server objects. Within *Mode 2 (CORBA Server)* the IA exports its internal information through a logic module with Generic Management Interface (see section 4), i.e. the information may be queried by other IAs that are at least running in Mode 1.

6. CASCADING INTEGRATION AGENTS

In general, the CORBA client functionality of the Integration Agent enables the supervision of a distributed CORBA application by a single IA. If the information collected through the CORBA client module has to be validated against node-specific constraints, local views of other IAs are required. For example, the overall application may be in a correct state if a certain number of application objects run on the available server machines. This requires a two-level checking: first, the application objects have to be reached via an application ping, and second, the number of object instances on the different nodes have to be checked. This problem can be solved by cascading IAs.

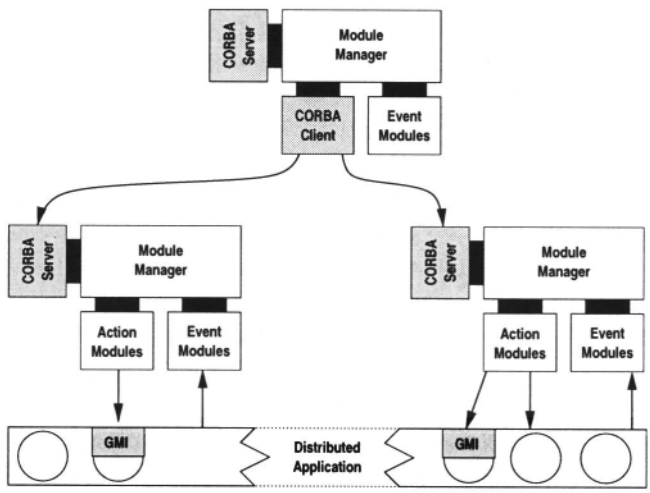

Figure 5. Hierarchical arrangement of Integration Agents

By establishing a hierarchical configuration of IAs as illustrated in Figure 5 several views of individual IAs can be aggregated into a more complex view. The local IAs export their information model through their CORBA server module (logic module). A superior IA uses its CORBA client module (action module) to query the CORBA server modules of the secondary lAs. The information queried by the superior IA through its CORBA client module can be reexported through its CORBA server module so that a hierarchy of any depth can be configured.

The aggregation of several local IA views by one superior IA simplifies the view upon the managed system. For management tools querying information from the superior IA the distribution of the managed components is transparent. This simplifies the integration of the IA into an existing management platform.

7. SUMMARY

Traditional management platforms have deficiencies in managing CORBA applications. Application-internal information that is a prerequisite for proactive management can often not be extracted because processes are regarded as black boxes. This paper presents an Integration System consisting of the Integration Agents and the Generic Management Interface for managing CORBA applications using existing management platforms. The central component is the Integration Agent that collects and aggregates metrics from the managed system and interacts with traditional management platforms.

The Generic Management Interface was developed for extracting management-relevant information from CORBA applications. The interface consists of a static CORBA IDL interface that is integrated into the application. The exchanged messages are flexibly defined using XML.

The Integration Agent is characterised by two main concepts: modularity and high availability. Flexibility is achieved by modules that can be dynamically loaded into and swapped out of the Integration Agent. High availability is achieved by implementing the Integration Agent as a self-checking process pair. In addition, the Integration Agent runs in various operation modes whereas the current mode is determined by continous self-diagnosis. Several Integration Agents may be configured in a cascading manner. This hides the distribution of the managed application to the administrator and thus simplifies management.

The Integration System is implemented and was easily integrated into an HP IT/Operations management environment. The Generic Management Interface was well accepted by application developers and administrators. Currently, the Integration System is in its test phase.

Acknowledgments

The authors like to thank Christoph Weyer from Fachhochschule Wiesbaden for his ideas and implementation work concerning the module manager framework. We also thank Dirk Lindner, Arno Schaefer, Thomas Kullmann, Pascal Mougnon, and Thomas Piwek from Lufthansa Systems for their support and numerous valuable discussions.

References

[BEA, 1998] BEA (1998). *BEA Manager Reference Manual 2.0.* BEA Systems.

[Hegering et al., 1999] Hegering, H.-G., Abeck, S., and Neumair, B. (1999). *Integrated Management of Networked Systems: Goals, Architectures, and their Operational Application.* Morgan Kaufmann Publishers.

[HP, 1997] HP (1997). *HP Open View IT/Operations Concepts Guide.* Hewlett Packard. B4249-90011, Version A.04.00.

[IONA, 1998] IONA (1998). *OrbixManager User's Guide.* IONA.

[Kloos, 2000] Kloos, D. (2000). Performance Management of CORBA Applications Using a Generic Instrumentation. Diploma thesis, Distributed Systems Laboratory, Fachhochschule Wiesbaden - University of Applied Sciences, Germany, (in German).

[OG, 1997] OG (1997). *Inter-Domain Management: Specification Translation.* The Open Group. Document No. 509.

[OMG, 1998a] OMG(1998a). *CORBAServices: Common Object Services Specification.* Object Management Group.

[OMG, 1998b] OMG (1998b). *The Common Object Request Broker: Architecture and Specification.* Object Management Group. Revision 2.3.

[OMG, 1999] OMG (1999). *Portable Interceptors.* Object Management Group. Document no.: orbos/99-12-02.

[Stallings, 1999] Stallings, W. (1999). *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* Addison-Wesley, 3rd edition.

[Tivoli, 1998] Tivoli (1998). *TME 10 Framework User's Guide.* Tivoli Systems. Version 3.6.

[TOG, 1998] TOG (1998). *Systems Management: Application Response Measurement (ARM) API.* The Open Group. Document no.: C807.

[W3C, 2000] W3C (2000). *Extensible Markup Language (XML) 1.0.* World Wide Web Consortium. Second Edition, W3C Recommendation.