MULTI-LAYER MONITORING IN DISTRIBUTED OBJECT-ENVIRONMENTS

Günther Rackl

LRR-TUM
Lehrstuhl für Rechnertechnik und Rechnerorganisation
Institut für Informatik
Technische Universität München
D-80290 München
Germany

rackl@in.tum.de

Abstract: This paper presents an on-line monitoring concept for distributed object-environments. On-line monitoring systems provide a powerful facility for realizing tools for the development and deployment of distributed applications. A major aspect in this context is the construction of interoperable tools capable of handling the complexity of large heterogeneous environments. Based on the requirements of on-line tools for distributed object-systems, this paper derives a multi-layer monitoring system and outlines an exemplary realization for CORBA environments.

Keywords: On-line monitoring, client/server, middleware, interoperability, CORBA

1 INTRODUCTION

Developing and maintaining large distributed object-oriented applications represents one of the major challenges in computer science at the time. Up to now, to guarantee a high quality of software products, several development methodologies and tools have been proposed.

A class of tools that are not covered by all the development methods are *on-line tools* applied during the run-time of distributed applications. On-line tools are important

L. Kutvonen et al. (eds.), *Distributed Applications and Interoperable Systems II*

during the development phase for debugging and performance tuning purposes, as well as during the deployment phase for supervision, management, or load balancing tasks.

The approach for realizing on-line tools is to provide a common on-line monitoring interface which enables tools to gather required information from the observed application, or to manipulate it. The usage of a separate on-line monitoring system is advantageous because different kinds of tools (both development and deployment tools) can rely on a single interface to observe or modify the distributed application. This also includes aspects like synchronization or interoperability of different tools acting simultaneously. Moreover, in heterogeneous environments the monitoring system can abstract from the underlying run-time environment, with the consequence that tools are not concerned with system-dependent issues. This paper presents an on-line monitoring approach for distributed middleware environments that takes into account the requirements of large object-oriented client/server applications.

Related Work. The approach for defining a monitoring layer lying between a distributed application and the tools has been proposed by the OMIS project [2] at LRR-TUM. Other approaches for monitoring distributed object-systems, include Object/Observer by Black&White Software, a commercial product for monitoring and viewing method invocations, and CORBA-Assistant from Fraunhofer Institut [1], who's aim is to provide a management facility for CORBA environments. Moreover, MODIMOS [5] is an approach for visualizing heterogeneous distributed applications.

2 REQUIREMENTS IN DISTRIBUTED OBJECT-ENVIRONMENTS

2.1 Tool-Requirements

Basically, modern distributed applications are often based on the client/server computing paradigm; their components are implemented using object-oriented languages, and interact using advanced communication middleware. The environments executing these applications are very heterogeneous concerning hard- and software, and several applications are being run simultaneously within a single environment.

Taking into account these system characteristics leads to a set of general tools requirements: At first, *interoperability* between tools is an important aspect, because several tools must be applicable to the same application at a time without interfering with each other. For example, a load balancing and a performance analysis tool might be running simultaneously; or, during deployment, an interactive management tool and a load balancer could be running continuously.

Next, tools need to be able to handle *multi-language applications*, i.e. applications realized in more than one programming language, and it has to be possible to support multiple applications simultaneously, as in large environments various applications could interfere with each other. Finally, as distributed applications are often complex systems, tools need to allow to work on different abstraction levels by concentrating on relevant aspects of the overall system.

Concerning functionality, for development tools tasks like visualization, performance analysis, or debugging are most important. During the deployment phase,

supervising tools observing the system behavior and detecting failures are required. Also, for management and maintenance purposes, interactive tools can be applied to manually manipulate the environment, e.g. for manually initiating object migrations in case of system adaptions.

2.2 Monitoring-Requirements

Based on the functionality tools might realize, requirements for the monitoring systems can be derived. First, the monitoring system must be able to deal with aspects concerning client/server applications. This includes monitoring on a request-level, i.e. it must be possible to observe interactions between clients and servers on the level of remote (or also local, in simple cases) method invocations. Secondly, the monitoring system needs to deal with aspects of object-oriented applications. Monitoring has to be possible on an object-level, i.e. objects from the corresponding object-oriented programming languages used for the implementation have to be accessible by the respective tools. These aspects allow to build tools addressing important issues like synchronization or relationships between distributed objects. Furthermore, the collection of specific system-dependent low-level information needs to be addressed by the monitoring system. The result is a step towards integrated and interoperable tool environments for distributed object-environments [4].

3 MULTI-LAYER MONITORING CONCEPT

3.1 System Model

Figure 1 shows an illustration of the distributed environment under consideration. The system to be monitored consists of six abstraction layers, from which the monitor collects information and provides it to tools only by means of the tool-monitor interface.

The highest abstraction level within the system is the application level. Here, only complete applications are of interest for the monitoring system. Within an application, the whole functionality exported by the components is described by interfaces. These interfaces are defined in an abstract way in the interface layer. The realization of the behavior described by these interfaces is done by objects within the distributed O-O layer. These objects may still be considered as abstract entities residing in a global object space. In order to enable communication between the distributed objects, some type of middleware is required. Especially, a mechanism to define and uniquely identify objects within the object space is needed. All commonly used middleware standards use some kind of globally unique object references. For example, CORBA uses Interoperable Object References (IORs) to identify CORBA objects [3], Sun's Java Remote Method Invocation RMI uses Uniform Resource Locators (URLs), and Microsoft DCOM generates so-called Globally Unique Identifiers (GUIDs) or monikers. As objects on the distributed O-O level are still abstract entities, they need to be implemented in a concrete programming language. This implementation of the objects is considered in the subsequent implementation layer. Obviously, objects may

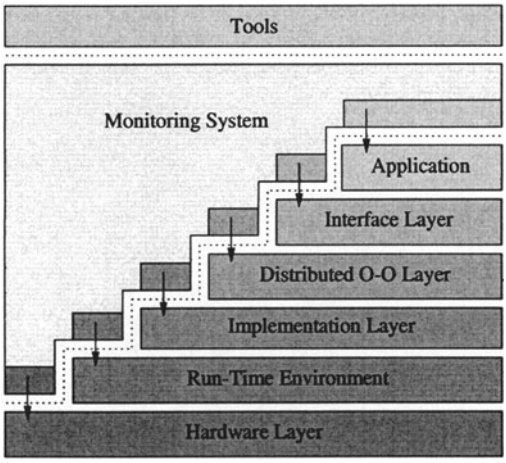

Figure 1. Layer Model of the Distributed Environment.

be implemented using some O-O language, but also non-O-O languages may be used, e.g. for integrating legacy code. Finally, the implementation objects are executed within a run-time environment which can be an operating system or a virtual machine on top of an operating system that is being executed by the underlying hardware nodes.

3.2 Multi-Layer Monitoring

For the monitoring system, two aspects are important: First, it has to be possible to gather data on all abstraction levels in order to serve as an information source for all kinds of on-line tools. And secondly, the mappings between the different layers are of great importance. As all entities within a specific layer are mapped onto appropriate entities within the layer on the next lower level until the hardware layer is reached, keeping track of these mappings is essential because the relationships between entities in two adjacent layers are not necessarily one-to-one relationships.

Tools making use of the monitoring system may be very diverse and therefore operate only on specific abstraction levels (e.g. a visualizer might be interested in interfaces and CORBA objects). For other tools, mappings between layers can be of special interest (e.g. for performance analysis, the process distribution on the nodes can be decisive).

As a consequence, a *multi-layer monitoring* approach which closely reflects the structure of distributed object-environment is well suited for the class of on-line tools described before. For obtaining information from all abstraction layers, specialized modules adapted to requirements of the layer to be observed can be inserted into the monitoring system. Thus, the monitor is kept very modular and flexible and can easily be adjusted to changes of the distributed environment.

4 REALIZATION FOR CORBA ENVIRONMENTS

4.1 CORBA System Model

In order to realize a monitoring system, the general application layer model described above has to be applied to CORBA systems. Figure 2 shows the concrete layers and relationships between the layers. Most relationships are n-to-m because of the complex

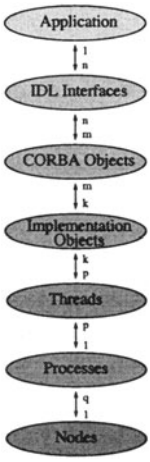

Figure 2. CORBA Layer Model.

mappings between the layers. Starting from the application layer, the mapping contains CORBA IDL interfaces, CORBA objects, implementation objects, threads, processes, and computing nodes.

4.2 Architecture of the Monitoring System

The most relevant aspects of the monitoring system architecture are the tool-monitor interface, the internal monitor architecture, and the monitor-application interaction.

Tool-Monitor Interface. As the monitor itself is a distributed application, the tool-monitor interface can easily be described using CORBA IDL. Services can be classified according to the layer the refer to, and to one of the categories information, manipulation, and event service. In our monitoring system, information services for all layers are foreseen. They provide data about the entities present in each layer, e.g. running applications, exported interfaces, instances of CORBA objects, implementation objects and processes, and nodes executing the respective entities. Manipulation services include starting and stopping entire applications, or the creation and deletion of single objects within the system. Event services finally support tracking of operation calls on the interface and CORBA object level as well as monitoring system data like e.g. current machine load values.

Internal Monitor Architecture. The monitoring system basically consists of a set of local monitor components being executed on every participating node. For each tool, there is a single entrance point to the monitor which provides the tool-monitor interface. It analyzes and distributes the incoming requests to the concerned local monitor components and assembles the results if necessary. Furthermore, the architecture of the monitoring system is kept open to integrate different middleware platforms.

Monitor-Application Interaction. In order to gather information from the observed system or to influence it, some kind of application instrumentation is necessary.

Currently, several instrumentation strategies are investigated concerning their usability for CORBA applications. Obviously, the portability of chosen instrumentation approach causes problems in this context. Our monitor prototype tries to be applicable to various ORB implementations, and to cause only small application modifications.

5 CONCLUSION AND FUTURE WORK

In this paper, a novel concept for on-line monitoring systems for distributed object-environments is presented. The approach for handling the heterogeneity and complexity of large distributed applications is a multi-layer monitoring concept which closely reflects the structure of the distributed environment under consideration. This concept allows to build tools that are coupled tightly to specific abstraction levels of the monitored applications. Moreover, the approach is applicable to different kinds of middleware platforms and therefore contributes to a simplified construction and management process for distributed interoperable systems.

Future work includes a refinement of the instrumentation strategies, integrating a prototypical implementation of the CORBA monitoring system, and extending it to other middleware platforms.

References

[1] FRAUNHOFER-IITB. The CORBA-Assistant — Monitoring of CORBA-based Applications. White Paper, June 1997. http://tes.iitb.fhg.de/corba-assistant/.

[2] LUDWIG, T., WISMÜLLER, R., SUNDERAM, V., AND BODE, A. OMIS — On-Line Monitoring Interface Specification (Version 2.0), vol. 9 of *Research Report Series, LRR-TUM, Technische Universität München.* Shaker, Aachen, 1997.

[3] OMG (OBJECT MANAGEMENT GROUP). *The Common Object Request Broker: Architecture and Specification — Revision 2.2.* Tech. rep., February 1998.

[4] ROVER, D. T., WAHEED, A., MUTKA, M. W., AND BAKIC, A. M. Software Tools for Complex Distributed Systems: Toward Integrated Tool Environments. *IEEE Concurrency* (April-June 1998), 40–54.

[5] ZIELINSKI, K., AND LAURENTOWSKI, A. A Tool for Monitoring Software-Heterogeneous Distributed Object Applications. *Proceedings of the 15th Int. Conference on Distributed Computing Systems* (Vancouver, Canada, May 1995).

Biography

Günther Rackl is a member of the Tools research group at LRR-TUM in Munich. His major research interests are on-line monitoring and management tools for distributed middleware environments.