Scientific Programming 10 (2002) 67–74
IOS Press

Efficiently building on-line tools for distributed heterogeneous environments

Günther Rackl*, Thomas Ludwig, Markus Lindermeier and Alexandros Stamatakis
Lehrstuhl für Rechnertechnik und Rechnerorganisation (*LRR*), *Institut für Informatik, Technische Universität München* (*TUM*), *80290 München, Germany*
E-mail: {*rackl,ludwig,linderme,stamatak*}*@in.tum.de*

Abstract: Software development is getting more and more complex, especially within distributed middleware-based environments. A major drawback during the overall software development process is the lack of on-line tools, i.e. tools applied as soon as there is a running prototype of an application. The MIMO MIddleware MOnitor provides a solution to this problem by implementing a framework for an efficient development of on-line tools.
This paper presents a methodology for developing on-line tools with MIMO. As an example scenario, we choose a distributed medical image reconstruction application, which represents a test case with high performance requirements. Our distributed, CORBA-based application is instrumented for being observed with MIMO and related tools. Additionally, load balancing mechanisms are integrated for further performance improvements.
As a result, we obtain an integrated tool environment for observing and steering the image reconstruction application. By using our rapid tool development process, the integration of on-line tools shows to be very convenient and enables an efficient tool deployment.

1. Introduction

Supporting the software development process within complex distributed environments is still a problem due to the lacking on-line tool infrastructures. Especially, application-oriented approaches for developing integrated and tool-supported applications are rare.

In this paper, we present the MIMO approach, which proposes an infrastructure and methodology for developing on-line tools for distributed heterogeneous environments. As the MIMO approach is application-oriented, we illustrate it by means of a real-world application scenario: We show how to integrate on-line tool functionality into a medical image-processing application. The characteristics of this application are the high performance requirements that make it necessary to build a parallel and distributed version in order to limit processing times. Furthermore, an automatic load balancer is applied to the application for performance reasons, such that the application can be executed in parallel within a cluster of distributed workstations.

As the on-line tool support should not be limited to the pure development of applications, we provide a tool environment that supports both the development and the subsequent deployment of the application. During development, the observation of the distributed application can be used for debugging and performance tuning purposes, while during deployment the steering facilities can be used for maintenance tasks. For example, for management purposes, all computation objects might have to be migrated away from a specific node, what can easily be carried out using our tool environment, without interfering the running computation.

In the following, we first give an introduction into the components participating in our environment, which are the medical image-processing application, the load-balancer, and the MIMO system. Subsequently, we describe the composition of these components into an integrated, tool-supported real-world application. The

*Present address: Günther Rackl, Holzstraße 20, 80469 München, Germany. E-mail: guenther@rackls.de.

ISSN 1058-9244/02/$8.00

evaluation with a genuine test case proves the applicability of our approach.

Altogether, this paper therefore contributes to an enhanced tool development and usage process for complex distributed applications. The MIMO system provides the basis for this approach, while the tool development methodology shows to be an appropriate procedure for its efficient deployment.

2. The medical image-processing application

We explore our concepts by means of the parallel medical image-processing application described in [1, 2]. In this application, a realignment process forms part of the Statistical Parametric Mapping (SPM) application developed by the Wellcome Department of Cognitive Neurology [3]. SPM is used for processing and analysing tomograph image sequences, as obtained for example by functional Magnetic Resonance Imaging (fMRI) or Positron Emission Tomography (PET). Such image sequences are used in the field of neuro-science, for the analysis of activities in different regions of the human brain during cognitive and motoric exercises.

Realignment is a cost intensive computation performed during the preparation of raw image data for the forthcoming statistical evaluation. It computes a 4×4 transformation matrix for each image of the sequence, for compensating the effect of small movements of the patient, caused e.g. by his breath. The images are realigned relatively to the first image of the sequence.

The realignment algorithm for image sequences as obtained by fMRI will briefly be presented. One has to distinguish two cases:

1. *Realignment of one sequence of images:* The reference data set and the first matrix is obtained by performing a number of preparatory computations using the image data of the first image. The matrices for all remaining images are calculated using the reference data set.
2. *Realignment of multiple sequences of images:* The reference data set and the first matrix of the first sequence are calculated. Thereafter, the first images of all remaining sequences are realigned relatively to the first image of the first sequence and its reference data set. Finally, the realignment algorithm as described in the first case is applied to all sequences independently.

At this point the only precondition for the calculation of the transformation matrix is the availability of the reference data set, which is calculated only once for each sequence. Once the reference data set(s) is(are) available, the matrices of the sequence(s) can be computed independently.

As the realignment process has high performance requirements, it is parallelised using Java and CORBA as a programming language and communication platform. Computational parts written in C++ are integrated by using the Java Native Interface (JNI).

The following sections describe the load management system used for tuning the performance of the realignment application, and the monitoring approach used to observe and steer the running application.

3. The load management system

In order to improve the performance of the parallel realignment application, the load management approach described in [4] is applied. In general, load management systems can be split into three components: The load monitoring, the load distribution, and the load evaluation component. They fulfil different tasks at different abstraction levels. This eases the design and the implementation of the overall system. Figure 1 shows the components of a load management system and a runtime environment containing application objects.

The load monitoring component provides both information on available computing resources and their utilisation, and information on application objects and their resource usage. This information has to be provided dynamically, i.e. at runtime, in order to obtain knowledge about the current state of the runtime environment.

Load distribution provides the functionality for distributing workload. Load distribution mechanisms for system level load management are initial placement, migration, and replication.

Initial placement stands for the creation of an object on a host that has enough computing resources in order to efficiently execute an object.

Migration means the movement of an existing object to another host that promises a more efficient execution. As migration is applied to existing objects, the object state has to be considered. The object's communication has to be stopped and its state has to be transferred to the new object, and all communication has to be redirected to the new object.

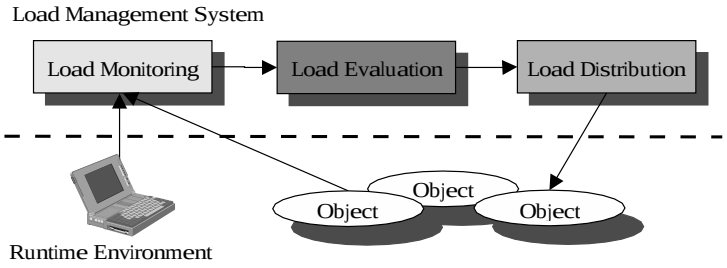

Fig. 1. The components of a load management system.

Replication is similar to migration but the original object is not removed, so that identical objects called replicas are created. Further requests to the object are divided among its replicas in order to distribute workload (requests) among them. Replication is restricted to replication safe objects. This means that an object can be replicated without applying a consistency protocol to the replicas.

Finally, the load evaluation component makes decisions about load distribution based on the information provided by load monitoring. The decisions can be reached by a variety of strategies. The aim of the diverse strategies is to improve the overall performance of the distributed application by compensating load imbalance. There are two main reasons for load imbalance in distributed systems. First, background load can substantially decrease the performance of a distributed application. Second, request overload that is caused by too many simultaneously requesting clients increases the request processing time and thus, decreases the performance of the overall application. Both sources of load imbalance have to be considered by a load manager.

Distributed object oriented environments like CORBA [5] or DCOM [6] are based on some kind of object model. CORBA objects are connected to the middleware by the POA (Portable Object Adapter). The object adapter provides functionality for creating and destroying objects, and for assigning requests to them. The POA is configured by the developer via policies. The ORB (Object Request Broker) provides functionality for creating object adapters and for request handling. A request to an object arrives at the ORB that transmits it to the appropriate POA. Subsequently, the object adapter starts the processing of the request by an implementation of the object (Servant).

The load management functionality, especially load monitoring and load distribution, have to be integrated into the ORB and the POA because we decided to make a system level implementation. Therefore, we added some policies and interfaces to the POA in order to enable state transfer and the creation of replicas.

The migration and replication of objects is realised using new policies that determine the creation of objects by means of factories, and a persistence policy that allows to migrate the state of an object. Request redirection is performed by the CORBA Location Forward mechanism [7]. It enables to hand over object references to clients by raising an `ForwardRequest` exception. The client runtime transparently reconnects to the forwarded reference. This guarantees migration and replication transparency.

4. MIMO

This section introduces the MIMO MIddleware MOnitoring system, an infrastructure for monitoring and managing distributed, heterogeneous middleware [8].

To handle heterogeneity, MIMO is based on a multi-layer-monitoring approach which classifies collected information using several abstraction levels and therefore serves as a foundation for integrating diverse middleware. In order to enable a rapid and flexible tool design and construction, MIMO's structure relies on a three-tier model separating tools, the monitoring system, and the instrumented application through generic interfaces. These interfaces make it possible to use the core MIMO infrastructure for building tools and application instrumentation in an appropriate fashion for the monitored middleware. Finally, in order to design GUI tools advantageously, the component-based MIVIS tool framework can be used to integrate new tool functionality easily by means of Java beans.

4.1. Multi-layer-monitoring

Figure 2 shows an illustration of a typical distributed middleware environment that we consider. The ob-

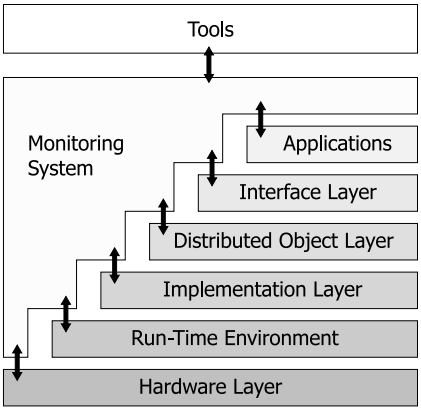

Fig. 2. Layer model of the distributed environment.

served system consists of six abstraction layers from which the monitor collects information and provides it to tools.

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 implementation of the behaviour described by these interfaces is done by objects within the distributed object 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, and especially, a mechanism to define and uniquely identify objects within the object space is needed.

As objects on the distributed object 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. 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.

For various middleware platforms or applications, this abstract model can be mapped to concrete entity types related to the respective environments. We will show the mapping of the realignment application to the MLM model in the following section.

4.2. MIMO design and architecture

The MIMO MIddleware MOnitor provides a framework for online monitoring and management tools

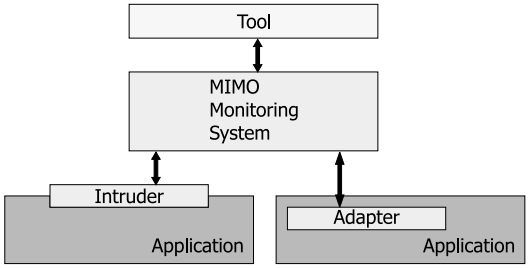

Fig. 3. 3-Tier model of the monitoring architecture.

which is compliant to the multi-layer-monitoring approach. The fundamental architecture relies on the separation of the tools from the monitoring system and the observed applications [9]. Figure 3 illustrates the resulting 3-tier model, which shows tools making use of MIMO by means of a tool-monitor-interface, while MIMO collects information from the monitored applications by means of intruders or adapters which communicate with MIMO through a intruder-monitor-interface. The difference between intruders and adapters is that intruders are transparently integrated into the application, while adapters might be built by inserting code into the application.

Tools interact with the monitor system by means of a standardised tool-monitor interface, while instrumented components make use of a generic intruder-monitor interface that allows to exchange any kind of event-based information.

4.3. MIVIS tool framework

MIVIS (MImo VISualizer) represents a general purpose framework for GUI-based tools. It contains basic visualisation functionality needed for sophisticated observation of any middleware, and allows for easy extension to include other advanced tool functionality. The generic MIVIS framework cares for the interaction with MIMO and presents data it receives in an advantageous way to the user.

To fulfil the requirement of uncomplicated extensibility of the visualisation tool, it is split into a main program and several JavaBeans software components. The main program takes care of the communication with MIMO and the processing of the data, and the JavaBeans do the graphical display. Hereby, all JavaBeans are discovered by MIVIS at startup time, and get dynamically integrated into the GUI. If a different type of display is needed, a user can program that display type using Java and turn it into a JavaBean. This component is placed into a specific directory so that MIVIS

can find and use it. The main program does not have to be changed at all, the only requirement is that the JavaBean implements a minimal interface that enables the main program to communicate with the bean.

The bean-specific properties can be set by the user. MIVIS knows about these properties by means of the introspection mechanism and provides editors to change the settings of these properties. Additional editors for properties of a special data type can be placed inside the JavaBean and used instead of the standard editors. Hence, this approach offers a very dynamic and flexible way to configure the behaviour of various display types. More details about MIVIS can be found in [10].

In the following, we make use of the MIMO capabilities to monitor and steer our load-balanced realignment application.

5. Integrating the load-balanced realignment environment

In this section, we now describe how to monitor and steer the load-balanced realignment application. First, we begin with a general overview of the tool development process proposed by MIMO, before we subsequently show the integration of the realignment application. As a result, we obtain a visualisation tool showing the activities of the realignment application, and a steering possibility that allows to migrate or replicate objects using interactive drag-and-drop mechanisms.

5.1. Tool development with MIMO

To enable monitoring of a new middleware with MIMO, a general methodology consisting of three major steps exists:

- *Define relevant middleware entities and map them to the MLM model:* The first step is to define entities within the middleware which are relevant for being monitored with MIMO. This can either comprise application-specific entities like business-objects, or middleware-specific entities like e.g. CORBA objects. The choice of these entities depends on the focus of interest and strongly influences the further activities.
  After defining the relevant entities, they need to be mapped to MIMO's multi-layer-monitoring model described before. Here, a certain degree of freedom exists and be exploited for the respective goals. The result is a middleware-specific layer-model with a mapping to the general MIMO MLM.
- *Define relevant events:* After defining the entities, relevant events that may occur within the middleware or application have to be determined. For example, these events may include generation or deletion of entities, or interactions between entities. As before, the definition of relevant events highly depends on the focus of the monitoring goals and can be completely user-defined. In any case, the result is a list of event names and their corresponding parameters that have to be passed with them. Also, it is possible to pass events from the application to the tools, as well as passing back commands from tools to the intruders or adapters residing in the application in order to manipulate the running application.
- *Implement instrumentation code and tool:* The last step in our methodology is the actual implementation of intruders/adapters and the tools based on the previous entity and event definitions. Here, MIMO serves as a common monitoring framework, and the MIMO core can be used as an intelligent communication infrastructure between tools and intruders/adapters. Moreover, the MIVIS tool framework can be used for easy development of GUI tools.

Hence, with this procedure new platforms can easily be integrated to the MIMO system by following a fixed set of rules. This general approach therefore enables a rapid and easy tool development which is highly application- and middleware-oriented, such that developers can concentrate on tool development without worrying about general monitoring issues. The steps of the tool development methodology are summarised in Fig. 4.

5.2. Integrating the realignment application

Based on the general tool development process, we now show the integration of the realignment application into the tool environment. Figure 5 depicts the structure of the realignment application. The service offered by the server object is the `compute`() service, which calculates the transformation matrix for an image. The state of a server object consists of a reference data queue (cache). Therefore it is replication safe since it can be replicated without applying a consistency protocol to its replicas, i.e. the required cache data can easily be reestablished. A `getReferenceData`() service is offered by each client and provides the specific reference data to the server if it is not already cached.

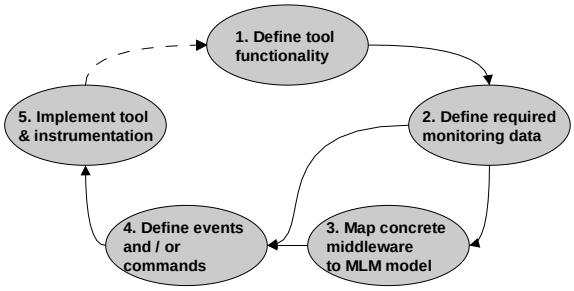

Fig. 4. MIMO tool development methodology.

Table 1
Mapping of the realignment application

Realignment application	MIMO MLM entity
Realignment application	Application
Compute-interface	Interface
Compute-object IOR	Distributed object
Java realignment class	Implementation
Realignment process ID	Runtime
Node	Hardware

For integrating these components within MIMO, the relevant entities have to be determined at first. The resulting mapping of realignment entities to the MIMO MLM model is illustrated in Table 1.

The next step is to define appropriate events of interest. These include both events for observing the progress of the realignment process, as well as steering events allowing to manipulate the running application. The main events being tracked by MIMO are as follows:

- Creation and deletion events for new client and servant objects
- Replication of objects
- Migration of objects
- Information events reporting the load of nodes and objects

For steering the realignment process, two events are used:

- Migrate object
- Replicate object

The parameters for those events include the target object and the respective destination node. More exact details about the events and their parameters can be found in [1].

5.3. Example scenario

The final step of our integration is the development of a visualisation bean making use of the MIVIS framework. We developed a new display that is used for the visualisation of the processes described before. Figure 6 presents the basic layout of the graphical on-line tool. Client and server objects are located within the respective rectangles representing the client and server hosts. In addition, server object load (numerical representation) and server host load values are depicted (numerical and graphical representation). The CORBA method `compute`() is represented as black arrow with a counter and `getReferenceData`() as offset turquoise arrow. Replications and Migrations are represented as white and red arrows respectively. Replication and Migration actions can be initiated manually with a drag-and-drop functionality; the user can therefore easily migrate objects to other nodes, or replicate them, if adequate.

Consequently, the combination of MIMO and MIVIS provides a flexible and extensible infrastructure for the development and the maintenance of large scale distributed applications.

6. Evaluation

In order to prove the efficiency of the presented load management concept and its implementation, we present a test case for our integrated realignment application [11].

The hardware consists of three machines with equal configuration. There is no background load on the machines. The examined CORBA application is the medical image-processing application described in Section 2 with two simultaneously requesting clients. The application is replication safe as already mentioned in Section 3. Thus, migration and replication can be applied to this application.

Figure 7 shows the processing time per image against the number of the processed image for both clients. At the beginning, one server object is created and placed on a machine (initial placement) and the clients start

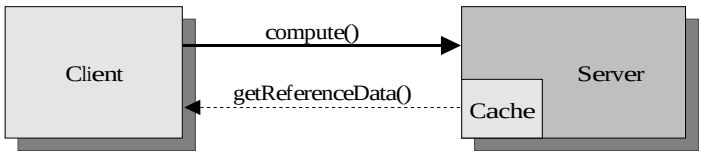

Fig. 5. The structure of the medical image-processing application.

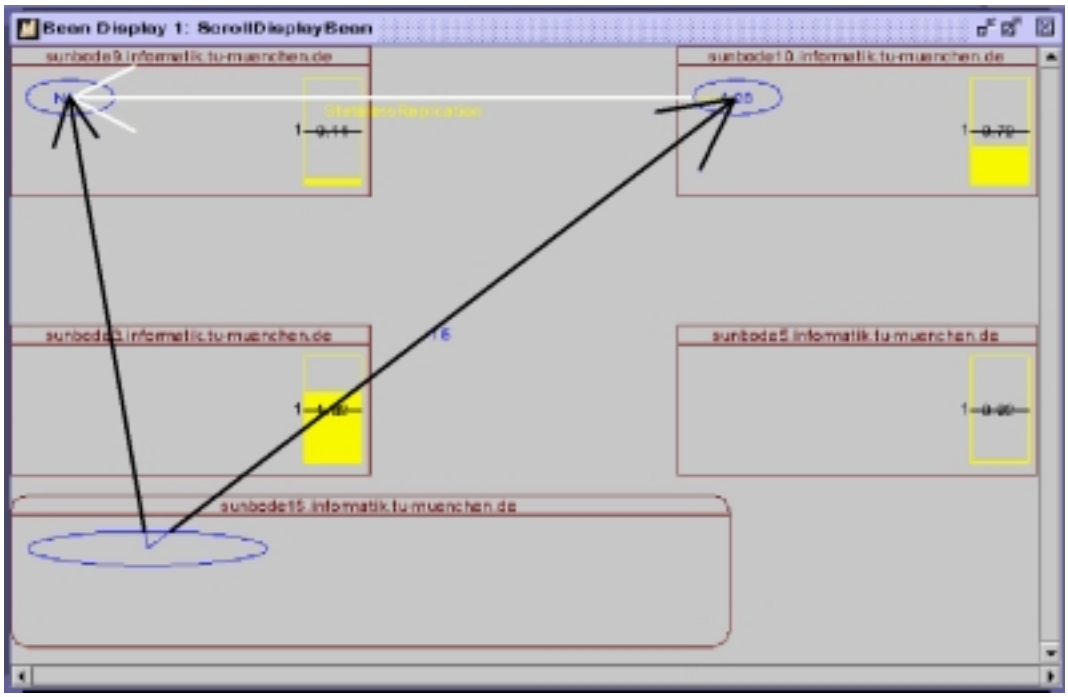

Fig. 6. Visualization of a replication and of object interactions.

requesting the server. The image processing time is equivalent for both clients now because the server alternately processes their requests. After a while the load management system recognises that the server is overloaded because both clients permanently request the server. Accordingly, replication is performed, i.e. a second server object (replica) is created and each client gets a replica on its own. In consequence of the replication, the image processing time of each client decreases about 50%. Some time later background processor load is generated on the machine that is used by the second client's replica. Hence, the image processing time of the second client substantially increases. Again, the load management system recognizes the processor overload and migrates the affected replica to the third machine which was not used so far. The consequence is that the image processing time returns to its normal level.

The test case shows how the load management system is able to deal with different kinds of overload. Request overload is compensated by replication, whereas background load is compensated by migrating an object to a less loaded host. Consequently, the load management systems improves the performance and the scalability of the medical image-processing application.

Additionally, the integrated tool environment makes it possible to visualise and manually steer the load-balanced application, what is helpful for performance tuning as well as for maintenance tasks.

7. Conclusion and future work

In this paper, we have presented an approach for developing on-line tools for distributed middleware-based applications. By means of our load-balanced medical realignment application, we have shown how to integrate monitoring functions and to implement a visualisation and steering tool, which can be used for both development and deployment tasks.

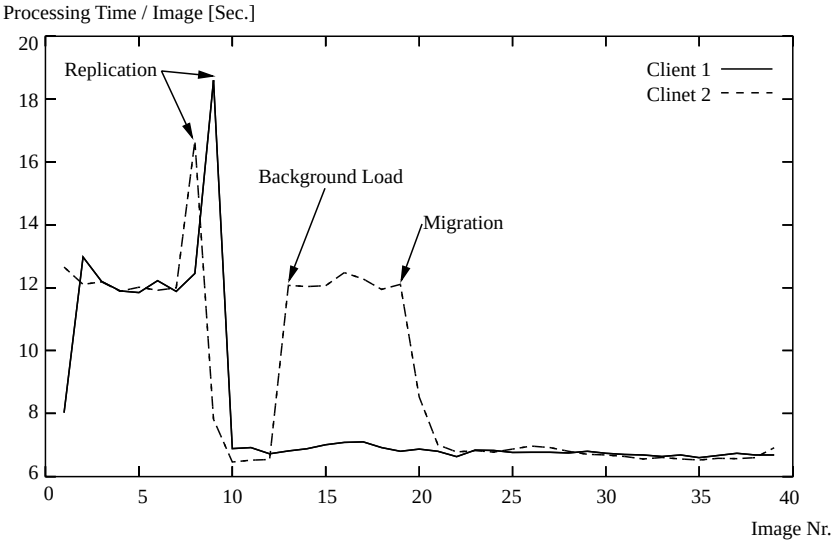

Fig. 7. The load managed medical image-processing application.

The MIMO infrastructure provides an infrastructure for implementing integrated tool environments. The tool development methodology cares for a systematic and concerted development process for tools and instrumentation components. Finally, we have presented a real-world application scenario that proves the applicability of our approach.

Future work includes the elaboration of further tool interoperability concepts, which enable a coordinated and simultaneous application of several tools to a single application. The integration of our load-balancer and the visualisation and steering tool is a first step towards this direction.

The global goal of our endeavours is to contribute to an enhanced overall software development and deployment process by improving the tool support for the "on-line phases" of the software lifecycle. We aim at an increased acceptance of on-line tools by showing an approach for their rapid development and efficient usage.

References

[1] A. Stamatakis, *Interoperability of Tools for Distributed Object-Oriented Environments*, Diploma thesis, Technische Universität München, 2001, (in German).

[2] M. May, *Vergleich von PVM und CORBA bei der verteilten Berechnung medizinischer Bilddaten*, Master's thesis, Technische Universität München, 2000.

[3] K. Friston, SPM, Technical report, The Wellcome Department of Cognitive Neurology, University College London, 1999.

[4] M. Lindermeier, Load Management for Distributed Object-Oriented Environments, in: *International Symposium on Distributed Objects and Applications* (*DOA'2000*), IEEE Press, Antwerp, Belgium, 2000,

[5] OMG (Object Management Group), The Common Object Request Broker: Architecture and Specification – Revision 2.3.1, Technical report, http://www.omg.org, 1999.

[6] G. Eddon and H. Eddon, *Inside Distributed COM*, Microsoft Press, 1998.

[7] M. Henning, Binding, Migration, and Scalability in CORBA, *Communications of the ACM*, 1998.

[8] G. Rackl, *Monitoring and Managing Heterogeneous Middleware*, Dissertation, Technische Universität München, February 2001, http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2001/rackl.html.

[9] T. Ludwig, R. Wismüller, V. Sunderam and A. Bode, *OMIS — On-Line Monitoring Interface Specification* (*Version 2.0*), Vol. 9 of *Research Report Series, Lehrstuhl für Rechnertechnik und Rechnerorganisation* (*LRR-TUM*), *Technische Universität München*, Shaker, Aachen, 1997.

[10] M. Rudorfer, *Visualisierung des dynamischen Verhaltens verteilter objektorientierter Anwendungen*, Master's thesis, Technische Universität München, 1999.

[11] T. Ludwig, M. Lindermeier, A. Stamatakis and G. Rackl, Tool Environments in CORBA-based Medical High Performance Computing, in: *Proc. of the PACT 2001 Conference*, Novosibirsk, Russia, September 2001, To appear.