A Monitoring System for Software-Heterogeneous Distributed Environments*

Aleksander Laurentowski, Jakub Szymaszek, Andrzej Uszok, Krzysztof Zieliński

Institute of Computer Science
University of Mining and Metallurgy (AGH)
Al. Mickiewicza 30, Cracow, Poland
e-mails: {pinio,jasz,uszok,kz}@ics.agh.edu.pl

1 Introduction

Emerging trends in high performance computing tend to exploit massively parallel systems (like SPP 1000) and distributed memory architectures (e.g. SP1, SP2), as well as workstation clusters. The most appropriate computational model for this kind of systems is built of a set of autonomous objects, communicating via message-passing mechanism. Advanced applications within this model may be engineered of components implemented in various programming languages. Hence, particular parts of such applications can be executed in those programming environments which inherently express the nature of computation. This allows mixing various programming and execution paradigms (e.g. imperative, functional, object-oriented or logic programming) in one software-heterogeneous system. Taking advantage of software components reuse and easier implementation process, this technique can speed-up the development of new, modern applications – unfortunately, much increasing their complexity at the same time.

Tools for monitoring, debugging and maintaining such applications could substantially help in that case. Our Managed Object-based Distributed Monitoring System (MODIMOS) aims at monitoring of large distributed software-heterogeneous applications, organized according to an object wrapping technique. The area of its application covers visualization of the systems' structure and activity, management of their logical configuration, thus enabling appropriate tuning. This influenced characteristic features of MODIMOS, like e.g. configurability of its modules, information management and filtering mechanisms, expandability of monitored environments set.

2 MODIMOS Architecture

The functionality of MODIMOS is determined by an abstraction we call the Uniform Model of Computation, pertinent to the object-based environments used

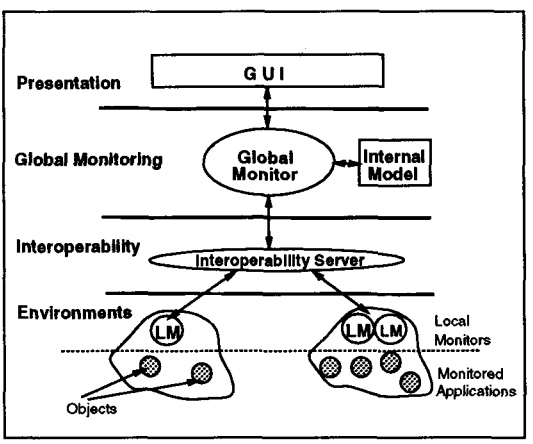
* This work was sponsored by the Polish State Committee of Scientific Research (KBN) under grant no.: 8 S503 015 06.

for wrapping of programming modules. Well-known distributed systems used for wrapping technology show some similarity of their computational models. We have analyzed a set of them and decided that the abstract model should be a superset (union) of their models of computation. Hence, in our Uniform Model the following levels of abstraction have been recognized: environment, application, container, object, interface and method. The detailed mapping of them into a couple of popular programming environments, e.g. ANSA[1], SR[2], and a CORBA-compliant[3] (Orbix) is shown in the table below.

Unit of abstraction	Environment		
	ANSA	SR	Orbix
application	—	makefile	—
container	capsule	virtual machine	process
object	object	resource, global	object
interface	interface	resource spec.	interface
method	operation	operation	operation

A set of relevant events is associated with each abstraction unit. These events are reported to the monitoring subsystem. Events concern the aspects of the items' behavior, such as initiation, creation, destroy, change of state, operation call/reply, etc.

MODIMOS has a multi-layer architecture shown below. Functionality of these layers may be described as follows. The Environments Layer consists of an expandable set of popular distributed programming environments. Any new object-wrapping environment can join the Environments Layer, provided it fits at least partially the Uniform Model and supports basic mechanisms of communication with the outside world. The Environments Layer consists of Monitored Applications sublayer and Local Monitoring sublayer.



The Monitored Application Sublayer represents original application code, instrumented by special preprocessors with addition of notification functions. The events reported by notification functions are collected in the Local Monitors sublayer. Each local monitor is a managed object written in a language provided by the given environment. It has three interfaces: Monitored Events, Management and Reported Events. Management Interface is used for monitoring policy setting, that determines which events received via Monitored Events Interface are forwarded through Reported Events Interface. Information sent via Reported Events Interface is structured according to the Uniform Model. Therefore, above the first layer only the abstract semantics is recognized.

The second layer deals with interoperability aspects. The aim of the Interoperability Layer is to ensure a universal and general platform for operations dispatching between local monitors and Global Monitoring Layer. The dispatch-

ing mechanisms used for this purpose are transparent to the local monitors. This layer dispatches invocations concerning reported events notification to the Global Monitoring module and, respectively, the management decisions from the global monitor (i.e. the user) down to local monitors.

Global Monitoring Layer collects reported events in a database called Internal Model (reflecting the architecture, configuration and current state of the monitored environment), cooperates with Graphical User Interface in the process of information visualization, implements the selective monitoring policy, and ensures consistency of collected data.

3 Basic Features of the System

As MODIMOS is itself a software heterogeneous application, the general interoperability mechanisms proposed and employed during its construction can be used in further multi-paradigm systems. We have analyzed and compared several variants of those mechanisms during design and construction of Interoperability Layer. The first choice is its base platform: a primitive communication interface (such as sockets) or an integrating *glue* environment, which offers more abstract notion of communication providing localization, binding, dispatching, name service, etc. We have chosen the latter solution and implemented Interoperability Layer in Orbix CORBA-compliant system. Messages are sent by invocations of functions from remote servers' interfaces, what frees the programmer from buffers construction. This also ensures well structuring and encapsulation of IL functionality into environments' objects. The *glue* environment must be somehow integrated with the monitored environments. The perfect solution would be construction of an object being a full member of the *glue* and a given monitored environment at the same time. It means that this object should be able to call and receive invocations from both these environments, so it may act as a *gateway* between them. However, there is a lot of environments which do not fulfill this assumption. A solution for this problem is an idea of a *plug* [7]. It is a regular *glue* environment object, which, however, represents the given monitored environment in the *glue* system. It receives remote operations' invocations from the given environment's local monitors, translates them into *glue* system's remote operation calls format, and invokes them on behalf of the local monitors. Invocations from all *plugs* are in an ordinary *glue* system form. The *glue* environment serves as a communication framework for *plugs*, it localizes the global monitor, binds *plugs* to its interface and dispatches their invocations to it.

Applications running in the environments to be visualized consist of items composing a hierarchical structure (a tree), described in terms of the computational model specific for the particular environment, and in terms of the Uniform Model. In order to visualize the items' hierarchy, we have implemented methods of "box-like", two-dimensional trees visualization, in which "child" level figures are placed into the figures representing "parent" levels of the hierarchy.

High performance computing imposes substantial requirements on monitoring tools, e.g. ability to cope with large amount of data, high speed and intensity

of interactions. In contrast to systems like IPS-2 [4], Paragraph [5] or Falcon [6], MODIMOS is not intended to present the whole program execution history. Introduction of filtering and selection mechanisms enables the user to significantly limit the amount of collected data and focus on the the most interesting and "hot" areas of the monitored system's activity. Filtering can take place at Environments Layer, where selection rules can be set up at preprocessing time and through the management interfaces of local monitors, as well as at the Global Monitoring level. Moreover, selection options at GUI enable the user to eliminate a group of objects or levels in the Uniform Model hierarchy during a particular visualization process. We employ two kinds of the visualized tree nodes' selection: *horizontal* (eliminating items from one logical level, e.g. all containers) and *vertical* (eliminating particular tree nodes with its subtrees).

To assure high flexibility of the system, we employed advanced software engineering techniques during MODIMOS's design and construction. As an object-oriented framework [8], the system consists of a set of cooperating software components, thus enabling easy extensions (new monitored environments, multiple monitors, GUI's, etc.). We use design patterns [9] to describe architecture, functionality and interfaces of those software-heterogeneous components.

The system is currently under development. Having implemented the Environments, Interoperability and prototype Global Monitoring layers, we are currently investigating methods of ensuring consistent global state of the monitoring trace. We are also designing the management mechanisms for global and local monitors.

References

1. *ANSAware 4.0 — Application Programmer's Manual*, APM Ltd. Cambridge, 1992.
2. G.R. Andrews, R.A. Olsson, *The SR Programming Language: Concurrency in Practice*, Benjamin/Cummings Publishing Company, 1992.
3. *Draft Common Object Request Broker Architecture Revision 1.1*, OMG Report 91-12-1, OMG Inc., 1991.
4. B. P. Miller, M. Clark, J. Hollingsworth, S. Kierstead, S. Lim and T. Torzewski, "IPS-2: The Second Generation of a Parallel Program Measurment System", *IEEE Trans. on Parallel and Distributed Systems*, 1,2, April 1990.
5. M. T. Heath and J. A. Etheridge, "Visualizing Performance of Parallel Programs", *IEEE Software*, 8(5), September 1991.
6. W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, J. Vetter, N. Mallavarupu, Falcon: On-line Monitoring and Steering of Large-Scale Parallel Programs, Georgia Institute of Technology, Technical Report No. GIT-CC-94-21, April 1994.
7. A. Uszok, G. Czajkowski, K. Zieliński, Interoperability Gateway Construction for Object-Oriented Distributed Systems, *Proceedings of the 6th Nordic Workshop on Programming Environment Research*, Lund, Sweden, June 1994.
8. R. Campbell, N. Islam, A Technique for Documenting the Framework of an Object-Oriented System, *Proc. IWOOS'92*, IEEE Computer Society Press, Sep. 1992.
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Object-Oriented Software Architecture*, Addison-Wesley, 1994.