Monitoring and Visualizing Software-Heterogeneous Distributed Object Applications *

Jakub Szymaszek
e-mail: jasz@ics.agh.edu.pl
Institute of Computer Science
University of Mining & Metallurgy
Al. Mickiewicza 30, Cracow, Poland

Abstract

Distributed applications composed of many cooperating components, written in different computing environments, that will become more common in the near future, increase substantially demands for monitoring and visualization tools. Complexity and the non-deterministic nature of such programs cause that information about them is difficult to manage, store, and visualize. In this paper solutions to these problems and methods of dynamic behaviour visualization of software-heterogeneous applications employed in MODIMOS monitoring system are presented.

1 Introduction

The next decade will bring radical changes to the way we do information processing, as applications composed of many cooperating distributed components, exploiting different computational paradigms become more common. Software technology based on composition techniques will be one of the dominating ones [11]. 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. This is a great challenge to software tools for application management, behavior visualization and debugging to support a wide range of systems with various service demands.

In this paper, we discuss selected issues concerning monitoring and visualization of heterogeneous object-based distributed applications. We try to answer the following questions: 1) How to effectively manage the collected data concerning usually complex applications; 2) How to ensure the consistency of collected information; 3) How to visualize the applications' behaviour. Solutions to these problems proposed in the paper are employed in an experimental monitoring system called MODIMOS (Managed Object-based Distributed Monitoring System).

A typical complex program's execution produces a great amount of events that are difficult to store, manage, interpret, and visualize. Hence, in monitoring systems, particular stress should be put on strong selectivity. In MODIMOS it is ensured by filtering mechanisms of monitoring subsystem as well as selection and navigation facilities provided by visualization subsystem.

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

Monitoring of a distributed application rises a well known problem of out-of-order events. Such events may cause inconsistency of collected and displayed data. We foresee usage of vector clocks technique [6] to solve this problem.

Unlike many past works, oriented towards measurement of parallel or distributed programs' performance (such as IPS [9]), we are investigating methods of on-line visualization of heterogeneous applications' dynamic behaviour, i.e. graphical presentation of the applications' structure and their components' interactions.

The rest of this paper is structured as follows. Since our considerations concern the MODIMOS system, its multi-layered architecture is shortly described in Section 2. Monitored applications' complexity problem and filtering mechanisms are the subject of Section 3. Section 4 addresses the problem of consistency of the collected data. In Section 5, we describe how dynamic behaviour of applications may be visualized and how visualized programs' components may be selected.

2 MODIMOS Architecture Overview

As stated above, MODIMOS allows monitoring and visualization of a heterogeneous family of object-based environments. To achieve that, the system has a multi-layer architecture shown in Fig. 1. Functionality of these layers may be described as follows.

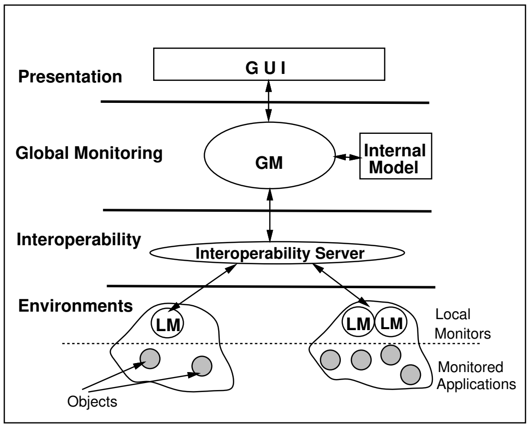

Figure 1: MODIMOS architecture

The Environments Layer consists of an expandable set of popular distributed object-based programming environments, such as: ANSA [1], SR [2] or a CORBA-compliant [4]. The Environments Layer consists of Monitored Applications Sublayer and Local Monitoring Sublayer. The Monitored Applications Sublayer represents the original application code, instrumented with notification functions by a special preprocessor. The events reported by notification functions are collected in the Local Monitoring 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 described in the next Section.

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 dispatching mechanisms used for this purpose are transparent to the local monitors. This layer dispatches invocations corresponding to 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. Interoperability Layer can be based on various integration engines, such as, e.g. CORBA or socket gateways.

Global Monitoring Layer collects reported events in a database called Internal Model (reflecting the architecture and current state of the monitored environments), cooperates with Graphical User Interface in the process of information visualization, implements the selective monitoring policy, and manages domains of local monitors.

Presentation Layer is implemented by Graphical User Interface (GUI), that performs two independent functions: management interface and visualization. The first of them enables management and configuration of parameters of lower layers of the system: Global Monitoring, Interoperability and Local Monitoring layers (for example configuration of monitoring policies for global and local monitors). The second allows selection and graphical presentation of data concerning monitored applications collected by the global monitor and stored in the Internal Model.

3 Complexity of Monitored Applications

The execution of an instrumented distributed application consisting of many components, may produce a great amount of events difficult to store, manage, interpret, and visualize. Therefore, the monitoring system should provide mechanisms allowing filtering of events and selection of monitored components. In MODIMOS, functionality of these mechanisms is based on a hierarchical model of an application and specification of execution event space described below.

3.1 Hierarchical Model of Application

Object distributed environments, that monitoring system is assumed to apply to, show some similarity of their computational models. From logical point of view, a typical application in these environments has a hierarchical structure that offers a sound, natural semantic basis for navigating the execution space and for filtering mechanisms [12]. To provide a common platform for the purpose of monitoring heterogeneous applications, a uniform abstract model of the environments' architecture and activity should be designed, with which they could comply. This model is a superset for models of computation existing in concrete environments. Hence, in our Uniform Model, the following levels of abstraction have been recognized: environment, application, container, object, interface and method. With each abstraction unit a set of relevant events, such as creation or destruction of the unit, is associated.

The Uniform Model leads to the concept of the three-dimensional, execution event space. The first dimension corresponds to the choice of an abstraction level of Uniform Model, on which the monitored systems' activity may be observed. The next dimension correlates with a mechanism enabling the user to select only a set of object instances for the monitoring purposes. Finally, the third dimension is relevant to an observation strategy, making it possible to monitor only events of specific types.

3.2 Filtering Mechanisms

MODIMOS provides mechanisms to reduce event stream either during code instrumentation or data collection in local and global monitors.

To reduce the amount of monitored events generated by notification functions, each preprocessor (dedicated for the particular environment) is parameterized. The available options enable specification of a monitored events' set, i.e. a set of inserted notification functions. The users can require monitoring of specific types of events, concerning only interesting application's levels or interesting application's items.

On Local Monitors Sublayer, monitoring policy can be set individually for each local monitor via Management Interface. It determines a set of Reported Events forwarded to the global monitor.

In the global monitor, global monitoring policy determines a set of stored events that cause changes of the Internal Model database. Global filtering can be used for reduction of the size of the Internal Model database and the number of its modification, whereas filtering in code instrumentation and local monitoring phase influence both the amount of information stored and the amount of data transmitted from monitored application to the global monitor. To express and store policy rules in any MODIMOS layer we plan to design a special language based on regular expressions.

4 Consistency of the Internal Model Database

Monitoring of a distributed application rises the well known problem of achieving events ordering in the presence of no global synchronized time. In MODIMOS, this is necessary to ensure consistency of the Internal Model database and consistent application's visualization.

In order to solve this problem we employ a delivery mechanism in global monitor, that for each incoming event checks if it satisfies a set of rules. These rules, called delivery rules, are responsible for ensuring causal order of events. If all rules are satisfied, the event is delivered to the Internal Model. If the event violates any rule, an appropriate action is performed depending on the type of violated rule. In particular, the event may be placed in a buffer of out-of-ordered events. The delivery mechanism then continues to examine new events, and periodically checks events in the buffer, to see if it is now possible to deliver any of them.

The regular, hierarchical structure of the monitored applications and semantics of events introduce a partial order relation between some of the monitored events. This allowed us on the first stage to implement the filter based on semantic analysis of incoming events and a set of simple delivery rules described below.

We have distinguished three types of events: 1) *create* – creation of the monitored item; 2) *update* – change of the item's state; 3) *destroy* – destruction of the item. Let's denote e_x the event of any type, c_x – the event of type *create*, d_x – the event of type *destroy*, each of them concerning the monitored item x. For each item x and each event e_x the following delivery rules are checked:

1. if e_x is the *create* event and x has a parent item in the application's structure, denoted by y, then e_x can be delivered if the event c_y has already been delivered.
2. if e_x is the *destroy* event and x has any child items, then e_x can be delivered if for each child item y the event d_y has already been delivered.
3. if e_x is the *update* or *destroy* event, it can be delivered if the event c_x has already been delivered.
4. if e_x is the *update*, it cannot be delivered after the event d_y has already been delivered.

In order to check the rules for any event, the filter must search the Internal Model to determine which events have already been delivered. If one of the first three rules is violated, the event is placed in the buffer of out-of-ordered events. If the fourth rule is violated, the event is discarded and neither causes any change of the Internal Model nor is visualized.

We are aware that the filter based on these ad-hoc delivery rules cannot ensure the correct order of all incoming events. Causal relations between *update* events concerning the same item and between events concerning items for which the relation ancestor-descendant in the hierarchy is not determined may be lost. Presented rules are insufficient in general case, although they may be sufficient to monitor the changes of the applications' structure. However, the order of these changes may be lost.

Hence, we are using such filter only for testing the Internal Model database and some visualization algorithms. In the future, in order to recover correct causal relations between all incoming events, we want to implement the delivery rules based on vector clocks [6]. Since we cannot predict the number of monitored processes we plan to use set rather than vector representation of timestamps [6, 3]. The major drawback of vector clocks is the intrusion induced by piggybacking of vectors of size equal to the number of monitored processes [8]. Therefore, we want to implement and test one of the variant of vector clocks' technique, called *direct dependencies coding* [7], that piggybacks scalar value.

5 Visualization of Applications' Dynamic Behaviour

As was assumed, the main objective of the visualization process in MODIMOS system is presenting the current state and structure of the monitored applications and enabling investigation of interaction between their items.

5.1 Presentation of the application's state and structure

Applications running in the environments to be visualized consist of items composing a hierarchical structure (a tree), described in terms of a model specific for the particular environment, and in terms of the Uniform Model, common for all environments.

In order to visualize items' hierarchy, we have implemented methods of two-dimensional trees visualization. A typical distributed application consists of a great amount of items and a tree describing a structure of such application may be very large. Authors of the work [13] prove, that 2-D visualization of large trees is not efficient method and suggest using 3-D visualization in such cases, e.g. cone trees. However, because of strong selectivity provided by lower layers of MODIMOS system and described below selection mechanisms provided by Presentation Layer, a number of visualized items may be significantly reduced. In this case two-dimensional visualization seems to be efficient enough. However, in the near future we would like to implement 3-D visualization techniques as well, as more smart and attractive for the user.

Trees depicting particular parts of items' hierarchy are displayed in, possibly many, windows. Each window reflects the current structure and location of the item, called the *root item*. The user can move over the items' hierarchy, similarly like in a directory tree, selecting interesting nodes (application items), and this way changing the current *root item*.

5.2 Selection Facilities of the Presentation Layer

Selection facilities of the Presentation Layer, described below, are natural completion of the filtering mechanisms provided by monitoring subsystem. They allow selection and visualization of information important for the user stored in the Internal Model. We employ two kinds of tree nodes' selection: *horizontal* and *vertical*.

The *horizontal* selection (Fig. 2) enables specification of the Uniform Model's levels to be visualized in a window, i.e. selection of the first coordinate in three-dimensional data space.

Sometimes, the user may want to focus on selected items of the program. Thus, the visualization subsystem should enable elimination of particular visualized items. The *vertical* selection (Fig. 3) is based on choosing and removing items from the window. It enables specification of the second coordinate in the data space.

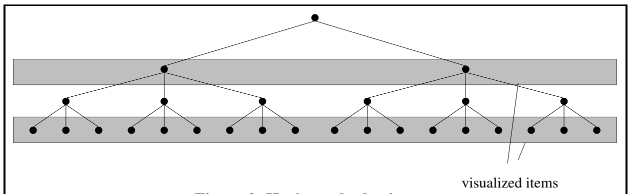

Figure 2: Horizontal selection

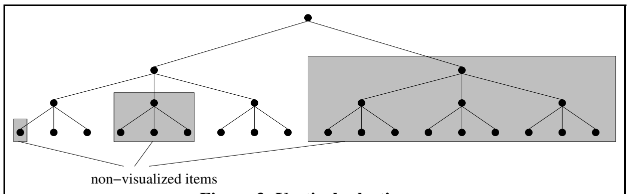

Figure 3: Vertical selection

Apart from items' selection, Presentation Layer also enables selection of a set of the events non-interesting for the user, that will not be represented in the window. This operation allows specification of the third coordinate in the investigated data space. Options causing selection of most useful and sensible sets of eliminated events are: 1) ignoring of all new events, i.e. freezing of an image currently displayed in the window, and 2) ignoring of events concerning new items' creation, i.e. restriction of visualized items to those currently visible in the window (event create cut-off).

5.3 Visualization of Components' Interactions

Visualization of logical structure of the applications is a basis for visualization of other aspects of programs' behaviour. Interactions between applications' components are the example of more interesting ones.

In many performance measurement monitoring tools for parallel or distributed applications [9, 5], interactions are visualized on the level of processes or threads. In object-oriented programs' monitoring systems [12] communication between objects is investigated. In MODIMOS, thanks to the filtering performed by monitoring subsystem, navigation and selection mechanisms provided by Presentation Layer, enabling limitation of a set of monitoring items to any layers of the Uniform Model, interactions may be considered on any level of abstractions: among methods, objects, containers (processes), environments. Because of a great number of visualized objects, in many object programs' monitoring tools only the performed information, which is an abstract description of objects' interactions, is presented, e.g. inter-class call matrix or inter-class call cluster [12]. Since in our system the number of visualized items may be significantly reduced, we plan to design first methods of individual invocations' visualization.

A special case of communication, that we are investigating, are interactions between heterogeneous components. Interoperability in heterogeneous applications is provided by special modules, called gateways. Gateways may be monitored and visualized in two different ways, depending on the user interest. The first one may be used for investigation of behaviour of gateways and treats these components as ordinary application's items. This approach facilitates interoperability components construction. But most programmers using interoperability modules want to hide technical aspects of interoperability. For them, a call of a method in another environment should be represented like an ordinary call. In such cases the second method, ensuring transparency of heterogeneous components' interactions should be employed.

6 Summary

In this paper selected problems and proposed solutions employed in MODIMOS system, referring to monitoring and visualizing of distributed heterogeneous object applications, have been presented. Described filtering and selection mechanisms allow dynamic monitoring policy changes and enable the user to significantly limit the amount of collected data and to focus on the the most interesting and "hot" areas of the monitored system's activity. We try to solve the problem of out-of-order events using delivery mechanism based on semantic analysis of incoming events. But in the future we plan to use one of the variant of vector clocks' technique. We have also presented the idea of dynamic behaviour visualization, that may facilitate investigation of object mobility, dynamic programs configuration methods, interoperability supporting components behaviour, etc. In the future we plan to focus on visualization techniques of heterogenity aspects of distributed systems.

The prototype of MODIMOS is still until development. Up to date we have implemented local monitors for SR, ANSA and Orbix (a CORBA-compliant environment), preprocessors instrumenting applications written in these environments, and the prototype global monitor. We are working on implementation of Interoperability Layer and we are developing various visualization techniques.

References

[1] *ANSAware 4.0 — Application Programmer's Manual.* APM Ltd. Cambridge, 1992.

[2] G.R. Andrews, R.A. Olsson, M. Coffin, I. Elshoff, K. Nilsen, T. Purdin, and G. Townsend. *An overview of the SR language and implementation.* ACM Transactions on Programming Languages and Systems, 10(1):51–86, January 1988.

[3] T. Basten, T. Kunz, J.P. Black, M.H. Coffin, D.J. Taylor. *Time and the Order of Abstract Events in Distributed Computations.* Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands. Computer Science Note. Number 94/06. Feburary 1994.

[4] *Draft Common Object Request Broker Architecture Revision 1.1.* OMG Report 91-12-1, OMG Inc., 1991.

[5] 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.

[6] C.J. Fidge. *Logical Time in Distributed Computing Systems.* IEEE Computer, 24(8):28-33, August 1991.

[7] J. Fowler, W. Zwaenepoel. *Causal Distributed Breakpoints.* In Proceedings of the 10th International Conference on Distributed Computing Systems, pp. 134-141, 1990.

[8] Claude Jard, Guy-Vincent Jourdan, Thierry Jéron, Jaen-Xavier Rampon. *A General Approach to Trace-Checking in Distributed Computing Systems.* In Proceedings of the 14th International Conference on Distributed Computing Systems, pp 396-403, Poznań, Poland, June 1994.

[9] B. P. Miller, M. Clark, J. Hollingsworth, S. Kierstead, S. Lim and T. Torzewski. *IPS-2: The Second Generation of a Parallel Program Measurement System.* IEEE Transactions on Parallel and Distributed Systems, 1,2, April 1990.

[10] F. Nayeri, B. Hurwitz, F. Manola. *Generalizing Dispatching in a Distributed Object System.* Proceedings of ECOOP'94, LNCS 821 pp. 450-473, Springer-Verlag 1994.

[11] O. Nierstrasz, S. Gibbs, and D. Tsichritzis. *Component-Oriented Software Development.* Communications of the ACM, 35(9):160–165, September 1992.

[12] W. De Pauw, R. Helm, D. Kimelman and J. Vlissides. *Visualizing the Behavior of Object-Oriented Systems.* Proceedings of OOPSLA '93, ACM Press pp. 326-337, SIGPLAN, Washington D.C, October 1993.

[13] Jean-Yves Vion-Dury and Miguel Santana. *Virtual Images: Interactive Visualization of Distributed Object-Oriented Systems.* In Proceedings of OOPSLA'94, Portland USA, October 1994.