This paper was published in
Lecture Notes in Computer Science
(http://www.springer.de/comp/lncs/index.html)

# Interoperability of OCM-based On-line Tools

Marian Bubak[1,2], Włodzimierz Funika[1], Bartosz Baliś[1], and Roland Wismüller[3]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] LRR-TUM – Technische Universität München, D-80290 München, Germany
{bubak,funika}@uci.agh.edu.pl, balis@icsr.agh.edu.pl, wismuell@in.tum.de
phone: (+48 12) 617 39 64, fax: (+48 12) 633 80 54

**Abstract.** In the course of a parallel application development, the use of supporting tools for debugging, performance analysis or visualization is indispensable. Since the services provided by the tools usually complement one another's, it is necessary to enable the tools to cooperate with each other. This cooperation, often referenced as interoperability, is feasible by means of the OCM universal monitoring system. This paper presents some issues of interoperability of two OCM-based tools, the DETOP debugger and the PATOP performance analyzer. An insight into the tool environment based on the OCM is also provided.
**Keywords**: monitoring, on-line tools, interoperability, OMIS.

## 1 Introduction

Tools for parallel programming support are important components of parallel application development. Each type of tools for parallel programming support has a well-defined functionality, therefore, in order to achieve a complex set of services in a tool environment, the *interoperability* of tools is highly desirable, which is meant as the capability to run concurrently and be applied to the same application with possible synergetic effect [5]. Ideally, we would like to enable interoperability between two tools coming from different vendors. However, such tools are likely to be incompatible with each other and it might be even not possible to run them concurrently due to low-level conflicts, or, event if the tools are able to run concurrently, further conflicts may occur at higher levels.

On-line tools need a specialized module for observing and possibly manipulating of a parallel program state, which is called *monitoring system*. Sometimes this module is integrated with a tool but it is much more profitable to have a separate facility to provide information on parallel application processes and mediate in controlling the application. One benefit of this approach is modularity: the tool development is separated from the monitoring system development. The most important benefit, however, is that multiple tools are enabled to use a single monitoring system, which not only reduces the overhead induced by running multiple tools, but also gives prerequisites for tools interoperability.

This paper presents how interoperability of on-line tools is enabled in a tool environment based on the OCM (OMIS-Compliant Monitor) universal monitoring infrastructure.

## 2 Interoperability

The term *interoperability*, in the context of monitoring, refers to on-line tools, and means their capability to run concurrently and be applied to the same application [6]. Moreover, a cooperation between tools is possible to provide additional functionality to the tool environment. For example, if a performance analyzer runs concurrently with a load balancer, and the latter migrates a process , the former should visualize the migration on its displays. And vice versa, a process migration may be forced manually via the peformance analyzer.

The first basic requirement for interoperability concerns the possibility to run different tools concurrently. In case of tools coming from different vendors, supplied with their own monitoring modules, *structural conflicts* between different portions of the monitors may occur, which may even prevent tools from concurrent running. As multiple tools may request an operation on a single object (e.g. writing into a process address space) at the same time, an infrastructure must exist to handle the multiple requests. For these reasons, unless tools form a monolithic, integrated environment being dependent on each others' implementation, interoperability of tools based on distinct monitoring systems is hardly possible due to likely structural conflicts or conflicts on exclusive objects among the monitoring modules [5].

Further problems may occur at the user level and manifest in *logical conflicts*. For example, if a debugger and a visualizer work concurrently, and a process is stopped by the former, the latter might not show it on its displays unless it is notified of the event. This results in inconsistencies in representation of the monitored system state, which we call *consistency problems*.

The next two sections describe a universal monitoring system OCM and provides an insight into the interoperability support within the OCM.

## 3 OCM - A Universal Monitoring System

### 3.1 General Structure

The OCM is an implementation of the OMIS (On-line Monitoring Interface Specification) [3] specification, being a centralized distributed system, composed of a central module, called NDU (Node Distribution Unit), which is interfaced to a tool, and a collection of modules, called local monitors, which are interfaced to the application. The operation of the OCM is thoroughly presented in [8].

In accordance with the OMIS specification, the target parallel system is viewed by the OCM as a hierarchical set of objects. The specification defines 5 types of objects: nodes, processes, threads, message queues and messages, with a collection of services to operate on objects. The services fall into three categories: *information services* to obtain information about an object, *manipulation services* to change the state of an object and *event services* to trigger an action list whenever a specified event occurs.

The OCM is currently adapted to support PVM [2] and MPI [4] applications. In the course of the TOOL-SET [7] project development, several tools were adapted to work on top of the OCM: the DETOP debugger, the PATOP performance analyzer and the VISTOP visualizer.

### 3.2 Interoperability Support in the OCM

The OCM provides some coordination features to address low-level conflicts in accessing shared objects by multiple tools [6]:

- requests referring to a single object are mutually exclusive,
- requests operating on more than one node are distributed to local monitors via an atomic multicast operation, to provide their execution in the same order on each node.
- requests can be locked to prevent any other requests on any node from execution while the locked requests is being executed.

Furthermore, the concept of events, as defined by the OMIS specification, allows to address higher level conflicts. These issues are described in the following sections.

## 4 OCM-based Tool Environment

In this section, we focus on the interoperability of two tools, DETOP and PATOP. An insight into the structure of an OCM-based tool environment and some its components is provided. Also covered is the startup "protocol" of the environment.

### 4.1 General Structure of an OCM-based Tool Environment

General structure of an OCM-based tool environment composed of DETOP and PATOP is shown in Fig. 1. The OCM is a layer between the application and tools. In fact, the tools communicate with the OCM indirectly through a high level routine library, called ULIBS [1].

OCTET is a newly developed tool to provide a management of the environment. It will be described in the next subsection.

### 4.2 The OCTET Tool

The OCTET (OCM-based Tool Environment top-level Tool) tool was created to work on top of the tool environment. OCTET performs two tasks, the first of them being the start-up of the tool environment, which includes spawning the

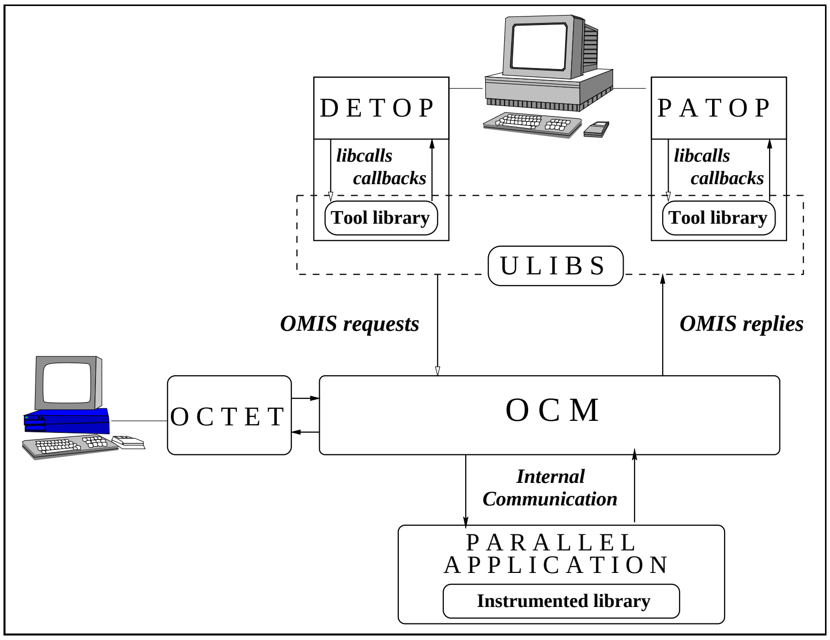


**Fig. 1.** OCTET in the tool environment.

application processes and running the tools, while the second is to provide tools with information to resolve consistency conflicts.

OCTET is a console application that provides a simple interface for setting up a number of parameters like the name of a parallel environment (PVM or MPI), paths to the application and tools' executables, number of processes to be run (in case of MPI only). A sample session with OCTET is shown in Fig. 2. The `set` command is used to set up the environment including the parallel library type (PVM or MPI), path to the application executable and possibly other parameters. The `run` command schedules the specified tool to be run. The tools as well as the application are actually run after the `go` command is invoked. Commands which are not recognized by OCTET are considered as explicit requests to the monitoring system, hence are sent to the OCM and replies to them are printed to the standard output.

### 4.3 Startup Mechanism in the Tool Environment

The startup of the tool environment is managed by OCTET. Time dependencies at the tool environment startup are shown in Fig. 3.

First, OCTET establishes communication with the monitoring system, which typically means start of of the OCM. Next, basing on the information provided by the user, OCTET orders the OCM to start the application. This process may

```
octet> set mode MPI
application type set to MPI
octet> set app-path "$HOME/MPI/cpi"
application path set to /home/balis/MPI/cpi
octet> run patop
PATOP has been scheduled to run
octet> run detop
DETOP has been scheduled to run
octet> go
starting the tool environment...

  [tools and the application are being started]

octet> :mpi_get_proclist()
```

**Fig. 2.** A sample session with OCTET.

vary depending on the parallel environment[1]. In the next step, OCTET starts the tools and provides them with a list of application process tokens. The tools are supposed to attach to each of the application processes. Finally, OCTET provides the tools with information on the environment to enable possible interactions between them (see Subsection 5.2).

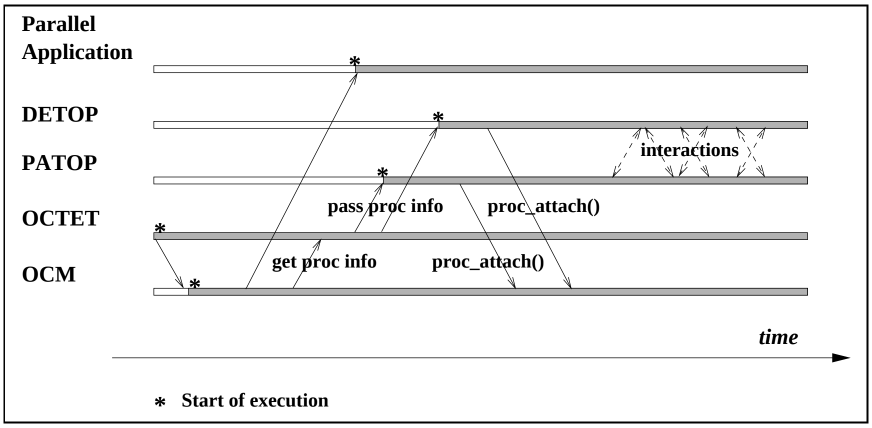


**Fig. 3.** Time dependencies for the tool environment startup.

[1] In the current implementation, this pattern is actually reversed in case of an MPI application: the application itself is started prior to the OCM.

## 5 Interoperability of DETOP and PATOP

### 5.1 Possible Benefits of DETOP and PATOP Cooperation

Let us consider a long-time running parallel application. PATOP, as a performance analysis tool, can be used to monitor and visualize the application execution. Suppose the application reveals an unexpected behaviour, observed via PATOP performance displays. We would like undoubtedly to localize the application's point of execution to find the cause of the behaviour. This is what DETOP helps with, as it works at the source code level. After having suspended the application execution with DETOP and a possible examination of proper variables, DETOP is used to resume the application execution.

### 5.2 Direct Interactions

PATOP and DETOP cooperation reveals consistency problems (section 2). The incorrect behaviour occurs in two cases:

- When the application is started with PATOP, it starts reading the performance data from the OCM and updates performance displays to visualize the execution. However, when the application is suspended by DETOP, PATOP proceeds reading data and updating displays, while the expected behaviour is that, PATOP hangs up monitoring while the application is being suspended. This is only possible if PATOP is notified whenever DETOP suspends the application execution.
- Once the application is started with DETOP, PATOP does not start monitoring. Again, a notification that the application processes has been continued is necessary. Similarly, when the application is resumed by DETOP after having been breaked, the notification is also needed.

Fortunately, the notion of events provided by the OMIS specification help resolve these problems [3]. Basically, PATOP needs to "program" a reaction to each event of thread suspension or continuation. It can be achieved if PATOP issues two following conditional requests to the OCM:

```
thread_has_been_stopped([]): print([$proc])
thread_has_been_continued([]): print([$proc])
```

The semantics of these requests is as follows: *whenever a process to which PATOP is attached has been stopped (continued), the process identifier of the stopped (continued) process is returned.* The events are handled by means of a callback mechanism. The process' identifier is actually passed to the appropriate callback function, which is invoked on every occurrence of the event. This callback function performs actions to stop (or resume) the measurements.

The succeeding questions are:

1. PATOP can program reactions to various scenarios of tools' cooperation. However, how can PATOP learn the actual configuration of the tool environment (which tools are running) so that it can perform appropriate actions?

2. Where should the above requests be implemented? We might decide to insert the appropriate code directly to PATOP, however, this would be an intrusion into the tool implementation, which contradicts the principal ideas of a universal monitoring environment, where tools are independent.

In [6], the second problem is resolved by dynamically inserting and calling the necessary code in the tool via machine-level monitoring techniques like dynamic instrumentation. A drawback of this approach is its complexity and the resulting poor portability. The approach presented in [6] is currently implemented only for PVM on Sparc/Solaris. For our environment, we thus chose a more high-level approach. For each tool, a specific library is provided in which every possible scenario of tools' cooperation would be handled.

One might get the impression that in this approach tools actually know about each other, thus it is arguable whether they remain independent of each other. However, all the interoperability related code is implemented as a new module in ULIBS which might be considered as an independent component of the tool environment (Fig. 1), although it is implemented as a library being linked to the tools' executables. Thereby the tools themselves are not really affected. It should be stressed, that the new module is designed to provide a general support for interoperability of any combination of tools, not only DETOP and PATOP. Although at present only the case of DETOP-PATOP interoperability is implemented, the other scenarios can easily be added.

Note that with the implementation described above, the tools have to be provided with information on which tools are running in the environment. The component which possesses *de facto* knowledge on the whole system, in particular, which tools are running is the OCTET tool. OCTET can pass the information to all the tools, which would cause a part of the interoperability module to be activated, which is appropriate to the given scenario. For example, if OCTET knows that it would run PATOP and DETOP, it can pass to PATOP the information that DETOP is running. This information would actually be processed by the startup module in ULIBS and passed to the interoperability module, in which, as a result, the two requests described earlier would be issued.

## 6 Concluding Remarks

Interoperability of on-line tools for parallel programming support is a key feature to build a powerful, easy-to-adapt tool environment. With the interoperability support lying in the environment infrastructure, not in the tools themselves, the user is enabled to customize its environment by picking tools which best fit his needs.

A system that supports interoperability must meet a number of requirements. First of all, the tools must be able to run concurrently. Next, a way to enable interactions between the tools must be provided. Finally, there must be a control mechanism to coordinate access requests to the target system objects.

The OCM monitoring system provides mechanisms that are enough to meet these requirements. Tools adapted to the OCM are enabled to run concurrently and operate on the same object. Moreover, a definition of tools' interactions, which leads to effective tool cooperation is possible without intrusion into their implementation.

Future work will be concentrated on the problem of direct interactions between tools. The current implementation is just the most basic implementation of the idea presented in Section 5.2. Further development will be focused on extending the role of OCTET in "programming" the interactions. Currently, OCTET just provides the basic information, while the whole rest is up to ULIBS. In future, OCTET can even provide general directives on how to "program" the interactions of tools.

**Acknowledgements.** This work has been carried out within the Polish-German collaboration and supported, in part, by KBN under grant 8 T11C 006 15.

## References

1. Bubak, M., Funika, W., Iskra, K., Maruszewski, R., and Wismüller, R.: Enhancing the Functionality of Performance Measurement Tools for Message Passing Applications. In: Dongarra, J., Luque, E., Margalef, T., (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface. Proceedings of 6th European PVM/MPI Users' Group Meeting, Barcelona, Spain, September 1999, Lecture Notes in Computer Science 1697, Springer-Verlag Berlin-Heidelberg, pp. 67-74 (1999)
2. Geist, A., et al.: PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, Massachusetts (1994)
3. Ludwig, T., Wismüller, R., Sunderam, V., and Bode, A.: OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series, (1997)
`http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz`
4. MPI: A Message Passing Interface Standard. In: Int. Journal of Supercomputer Applications, **8** (1994); Message Passing Interface Forum: MPI-2: Extensions to the Message Passing Interface, July 12, (1997)
`http://www.mpi-forum.org/docs/`
5. Trinitis, J., Sunderam, V., Ludwig, T., and Wismüller, R.: Interoperability Support in Distributed On-line Monitoring Systems. In: M. Bubak, H. Afsarmanesh, R. Williams, and B. Hertzberger, editors, High Performance Computing and Networking, 8th International Conference, HPCN Europe 2000, volume 1823 of Lecture Notes in Computer Science, Amsterdam, The Netherlands, May 2000. Springer.
6. R. Wismüller. Interoperability Support in the Distributed Monitoring System OCM. In R. Wyrzykowski et al., editor, Proc. 3rd International Conference on Parallel Processing and Applied Mathematics - PPAM'99, pages 77-91, Kazimierz Dolny, Poland, September 1999, Technical University of Czestochowa, Poland.
7. Wismüller, R., Oberhuber, M., Krammer, J. and Hansen, O.: Interactive Debugging and Performance Analysis of Massively Parallel Applications. *Parallel Computing*, **22**, (1996), 415-442
8. Wismüller, R., Trinitis, J., and Ludwig T.: OCM - A Monitoring System for Interoperable Tools. In: Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools SPDT'98, Welches, OR, USA, August 1998.