



PROTOTYPE DOCUMENTATION

WP 3.3.1

WP3

Document Filename:	CG-3.3.1-CYF-D3.3-v0.1-OCM-G.doc
Work package:	WP3
Partner(s):	CYFRONET
Lead Partner:	CYFRONET
Config ID:	CG3.3.1-CYF-D3.3-v1.0-OCM-G
Document classification:	PUBLIC

Abstract: This document presents the implementation of the first prototype of the OCM-G – Grid-enabled OMIS-compliant Monitoring System. The OCM-G's purpose is to be an intermediate layer between tools for application development support and applications running on the Grid. The OCM-G enables tools to collect information about and manipulate the applications. This document describes the implementation and functionality of the first prototype of the OCM-G. We also provide a description of the installation procedure, and a user manual in which we explain how to use this version of the OCM-G.



Delivery Slip

	Name	Partner	Date	Signature
From				
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Author
0.1	16/01/2003	Draft version	Bartosz Baliś, and Marcin Radecki
1.0	21/02/2003	Pre-Final	Bartosz Baliś, and Marcin Radecki

CONTENTS

EXECUTIVE SUMMARY	4
1. INTRODUCTION	5
1.1. PURPOSE	5
1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS	5
2. REFERENCES	6
2.1. SOURCE CODE	6
2.2. CONTACT INFORMATION	6
3. IMPLEMENTATION STRUCTURE	7
4. PROTOTYPE FUNCTIONALITY	9
5. USER MANUAL	10
5.1. DEPENDENCIES	11
5.2. INSTALLATION	11
5.3. RUNNING.....	11
5.4. INTERFACE DESCRIPTION.....	14
6. INTERNAL TESTS.....	15
7. ISSUES	16

EXECUTIVE SUMMARY

This document describes the implementation of the first prototype of the the OCM-G – Grid-enabled OMIS-compliant Monitoring System.

Sections 1-2 provide introduction and references.

Section 3 describes what includes the implementation of the first prototype.

Section 4 presents the functionality of the first prototype implementation of the OCM-G.

Section 5 contains the user manual of the software.

Finally, section 6 proposes test procedures to verify the correct working of the OCM-G.

1. INTRODUCTION

1.1. PURPOSE

The OCM-G is a monitoring infrastructure which provides monitoring services for tools, such as debuggers or performance analyzers, which are used by developers to support the application development.

This document describes the implementation of the first prototype of the OCM-G. The functionality of the first prototype is limited to work on one site only, and supports only one user. Apart from these limitations, the OCM-G is a fully functional monitoring infrastructure providing a variety of monitoring services.

1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS

CrossGrid	The EU CrossGrid Project IST-2001-32243
LM	Local Monitor
OMIS	On-line Monitoring Interface Specification
OCM-G	Grid-enabled OMIS-Compliant Monitor
SM	Service Manager
FOMP	codename for the first OCM-G prototype used in this document.

2. REFERENCES

2.1. SOURCE CODE

http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_3-moninfr/wp3_3_1-ocm-g/FOMP codename for the first OCM-G prototype used in this document.

2.2. CONTACT INFORMATION

Bartosz Baliś balis@agh.edu.pl
Marcin Radecki m.radecki@cyf-kr.edu.pl
Tomasz Szepieniec t.szepieniec@cyf-kr.edu.pl

2.3. REFERENCED DOCUMENTS

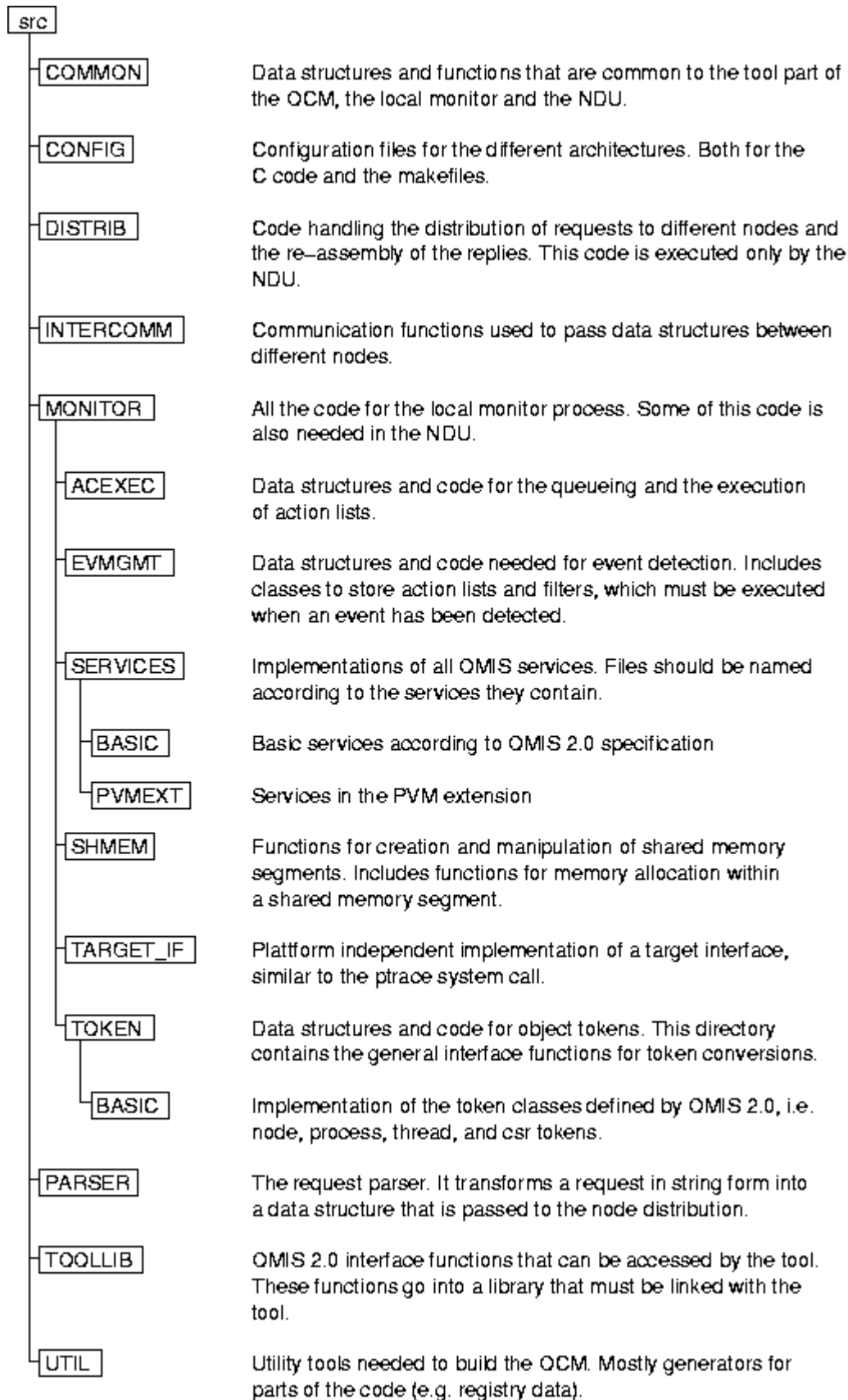
CGD32 CrossGrid Deliverable 3.2, [...] CrossGrid Document CG-3.3.1-SDD-0100.
CGD33T35 CrossGrid Deliverable 3.3, [...] CrossGrid Document
CG3.5-D3.3-v1.0-CSIC021-TestIntegrationPrototype.
OMIS *OMIS – On-line Monitoring Interface Specification*. Version 2.0. Lehrstuhl für Rechnertechnik und Rechnerorganisation Institut für Informatik (LRR-TUM), Technische Universität München.
<http://wwbode.informatik.tu-muenchen.de/~omis/>
PVM02 Bartosz Baliś, Marian Bubak, Włodzimierz Funika, Tomasz Szepieniec, and Roland Wismüller, *An Infrastructure for Grid Application Monitoring*. In Proc. EuroPVM/MPI 2002 Conference, Linz, April 2002.

3. IMPLEMENTATION STRUCTURE

The implementation of FOMP is entirely based on an existing implementation of an OMIS-compliant monitor – the OCM [OCM]. The OCM is a monitoring system for clusters of workstations. Since FOMP is intended to work on one site only in a shared filesystem, i.e., practically a cluster, the functionality and architecture we needed for FOMP was exactly the one already provided by the OCM. In a few cases, however, some implementation effort was needed to make the monitoring system suitable for a Grid environment. First, the top-down start-up mechanism in the OCM was replaced by a new bottom-up one, suitable for the Grid (see [CGD32] for a detailed description); second, some new services were added (see chapter 4).

The design of the OCM differs at some points from what was described for the OCM-G in [CGD32]. First of all, the OCM is written in C, while the OCM-G was planned to be object-oriented and written in C++. Originally we planned to rewrite FOMP from scratch and use the existing OCM code as implementation for classes. However, in practice it turned to be impossible, or at least not realistic due to a limited timescale for the implementation.

On the following page the structure of the OCM modules is shown. These correspond to the original OCM source code structure, which slightly changed in the OCM-G, however the modules and their functionality remained exactly the same.



4. PROTOTYPE FUNCTIONALITY

Below is a summary of the functionality and limitations of FOMP.

- FOMP is able to start-up as described in [CGD32];
- FOMP functions only on one site with a shared filesystem;
- FOMP provides all monitoring services specified by OMIS 2.0 except of those which are undesirable in the Grid (e.g., for starting a new process).
- additionally, the following new Grid-specific services has been implemented in FOMP:

1. **app_attach2** attach to an application

```
Token
app_attach2(String appname)
```

This service attaches to an application identified by its name `appname` and returns the application's token.

2. **app_get_proclist** return process list of an application

```
TokenList
app_get_proclist(Token app)
```

This service attaches to an application identified by its name `appname` and returns the application's token

3. **thread_executes_probe** a probe has been executed

```
void
thread_executes_probe(TokenList threadlist, String probe)
```

This service represents an event of executing a probe (i.e. an arbitrary place in code) named by name `probe`.

- probes are available in FOMP;
- only one user is supported (i.e., each user has to have a separate instance of FOMP);
- since only one site is supported, only one Service Manager will be running; furthermore, as only one user will be handled, the SM will run as a user process;
- with the functionality described above, there are no security issues in FOMP at all; consequently, no security layer is implemented.

5. USER MANUAL

This section is intended to explain how to install and use the FOMP.

5.1. INTRODUCTION

The OCM-G application monitoring system is designed for monitoring distributed Grid applications which in our case are MPI applications. The monitoring system consists of three distinct parts. They are each created at compile time, but the set-up and functionality of each one is considerably different. These parts are as follows:

1. Service Managers – persistent system daemons which should be started at the boot-time (in the final version – see below). On each site of the Grid, there should be one SM on a selected and well-known host of that site.
2. Local Monitors – services started on each host of the Grid when the processes of the monitored application are started.
3. Application linked library – part of the OCM-G which is linked to an application being monitored.

5.1.1. Use-case of application monitoring with FOMP

Let us start with a simple application monitoring use-case which may be helpful in understanding the system's functionality.

There are two preconditions which must be met for an application to be monitored: first, it should be linked against libraries enabling communication with the monitoring system; second, it should be linked against the instrumented version of the MPI library (provided with FOMP). A small utility named "ocm" is available to help in recompiling the user's application. When it is done, the application is ready to be run and monitored with FOMP.

Next, the application is normally submitted to run on the Grid. In the first prototype this should be simply the `mpirun` command. Each application process checks if there is a Local Monitor already running on the local host (existence of `/tmp/ocmg_${UID}` file is checked); if so, the process registers in it, otherwise the LM is forked off first.

As soon as a new LM starts, it registers itself to its Service Manager, which must have been started before. The SM is found via an external information system. In the first prototype, since there is one SM started as a user process, information about its location is written in a file located in the user's home directory.

Next, the user can use a tool to attach to FOMP and perform some monitoring activities.

5.1.2. Service and tool details

First, we briefly describe how system modules are started:

1. The Service Managers should be started at the system's start-up and run as unprivileged user's account processes, but in the first prototype they are not permanent yet. One SM should be started by hand by the user, before running the application. The executable for the SM is called 'omis_monitor'.
2. The Local Monitors are started automatically, being forked off from the application processes, (unless they have been started before somehow).
3. Application linked library – this part of FOMP is embedded in the application. Before being monitored application must register to FOMP. To do this, the following function call is required at the very beginning of the application: ``ocmg_register(&argc, &argv)``. Additionally, two command line parameters must be passed to the application:

- `'--ocmg-appname name'`, where *name* is an arbitrary application name given by the user. (Giving the same name to a tool the user later specifies which application he wants to monitor.)
- `'--ocmg-regcont'`; this parameter currently does nothing but is required for the future compatibility.

5.2. DEPENDENCIES

FOMP does not need any extra hardware; a typical testbed environment should be sufficient.

The software prerequisites for the FOMP follows.

1. Shared filesystem between all nodes on which application will be run (Network File System).
2. ~~Kernel patch which enables named pipes (FIFO queues) to issue the SIGIO signal when the FIFO has been read or written. The patch can be downloaded from:~~
<http://www.cirelemud.org/~jelson/linux/fifo-sigio-2.4.patch>
3. MPICH (ver. 1.2.0) installed on all nodes where the OCM-G runs.
4. "C" and "C++" language compilers (gcc, g++). Tested version: egcs-2.91.66.
5. GNU-make (ver. 3.78.1), ~~automake (ver. 1.4.8) and autoconf (ver. 2.3-10) tools.~~

5.3. INSTALLATION

The FOMP source codes are available in the FZK CVS repository. See <http://gridportal.fzk.de/cvs/?group=cg-wp3-3> for a detailed description how to get them.

5.3.1. Building FOMP

Executables, monitoring libraries, and the instrumented version of the MPI library are created by the compilation of the source codes. The locations of these components are as follows: binary files are put in `$(OCMGROOT)/bin`, libraries dynamically linked to `omis_monitor`, in `$(OCMGROOT)/lib`, libraries linked to the application (including the instrumented MPI library) – in `$(OCMGROOT)/applib`. (In the first prototype OCMGROOT is fixed and, as stated, points to `/opt/cg/ocmg`).

Building of FOMP is done by issuing `'make'` command in the `$(OCMGROOT)/src` directory.

5.3.2. Installing FOMP

To install FOMP, type `make install` in `$(OCMGROOT)/src` directory. This will copy executables and libraries to the appropriate `/opt/cg/ocmg/*` directories.

It is possible to generate rpm package of FOMP by typing `'make rpm'`.

5.4. RUNNING

Using FOMP is possible for any user, who has an account on machine where FOMP is installed. No additional permission nor configuration is needed.

5.4.1. Starting up the FOMP components

The Service Managers must be started by hand by issuing the following command: `/opt/cg/ocmg/bin/omis_monitor`; Local Monitors are started automatically, they are forked by an application processes.

5.4.2. Using the `ocm` utility

To compile the application with monitoring support, the user issues the following command: ``/opt/cg/ocmg/bin/ocm mpicc [...]'`, where ``mpicc [...]'` is the command the user normally issues to compile the application. The ``ocm'` tool changes the command line to link additional libraries.

5.4.3. Step-by-step instruction

In this section, we provide a simple step-by-step description of how to prepare application for monitoring and how the basic functionality of FOMP can be tested. The full functionality of FOMP will be tested in the cooperation with the G-PM tool (Task 2.4).

1. Preparing the application.

Let assume the user have the source code of an MPI application.

The tool ``ocm'` helps to compile the application in way the executable is linked against every library necessary for both monitoring and MPI. The user types the command he usually issues to compile the application (e.g. `mpicc ...`) preceded by ``ocm'`, e.g.:

```
[ymradeck@zeus MPI]$ ocm mpicc pgauss.c -o pga
gcc -I/home/ymradeck/OCMG/include -
I/home/ymradeck/OCMG/include/OCM -c /tmp/ocm_app_info.9039.c -o
/tmp/ocm_app_info.9039.o -DOCM_IN_APP_CONTEXT
mpicc -L/tmp/ocm_app_info.9039 -L/home/ymradeck/OCMG/applib.LINUX
/tmp/ocm_app_info.9039.o -lAppMonitor -lAppOcmbasic -lAppOcmcore
-lAppOcmdetop -lAppOcmpa -lAppOcmpatop -lAppOcmtool -lAppOcmtrace
-lAppOcmvistop pgauss.c -o pga -lAppMonitor -lAppOcmcore -ldl
```

2. Starting Service Managers

Service managers are currently started by hand. Since SM prints some debugging information it is convenient to start it in a separate terminal. Just type:

```
[ymradeck@zeus ymradeck]$ omis_monitor
*** Master
```

3. Running the application.

When the application is ready, it can be run in a usual way; only additional parameters are required, e.g.:

```
[ymradeck@zeus ymradeck]$ mpirun -np 2 ./pga 2000 --ocmg-regcont
--ocmg-appname gaussapp
```

The command line parameter ``2000'` is the application-specific one. The other parameters are meant for monitoring system. See section 5.1.2 for details about the ``--ocmg-*'` parameters.

4. Attaching the tool.

The OCM-G should be run with the G-PM tool developed by Task 2.4, but for testing purposes one can use a very simple console-based one (`/opt/cg/ocmg/bin/tool`). If the application is running, and Local Monitors have been started the tool can be started too. It should then automatically connect to FOMP (SM):

```
[ymradeck@zeus MPI]$ tool
```

After a successful start-up, the command prompt is displayed.

```
*** Tool
ocm.1>
```

5. Sending OMIS requests

Now the tool is ready to send OMIS monitoring requests to FOMP. The replies will be displayed accordingly. The first request should attach the tool to the specified application.

```
ocm.1> :app_attach2("gaussapp")
Got reply 1:
  Element 0:
                |  0 |
  Element 1:
      app_1 |  0 |
```

Note that the application name passed as request parameter is the same as passed via the `--ocmg-appname` option. This request returns the application *token*, which should be from now on used as the application's identifier. In the example above the token is: `app_1`.

Next, we can get the information about all the application processes:

```
ocm.2> :app_get_proclist([app_1])
This should return a list of application process tokens, e.g.:
Request 2: :app_get_proclist([app_1])
Got reply 2:
  Element 0:
                |  0 |
  Element 1:  app_1 |  0 | 2,[n_2,p_29026_n_2,n_1,p_21598_n_1]
```

This list contains two process tokens (`p_29026_n_2`, `p_21598_n_1`). The tokens are composed of two parts: the first one identifies the process itself (`p_29026`), while the second one – the node it is located on (`n_2`).

We can get information about each node, e.g. `n_1`, by two following requests:

```
ocm.3> :node_attach([n_1])
Got reply 3:
  Element 0:
                |  0 |
  Element 1:
                |  0 |
```

```
ocm.4> :node_get_info([n_1],3)
Got reply 4:
  Element 0:
                |  0 |
  Element 1:
                n_1 |  0 | "zeus26.cyf-kr.edu.pl", "Linux", "#1
SMP Fri Jan 10 11:08:19 CET 2003", "2.4.20"
```

5.5. INTERFACE DESCRIPTION

The monitoring services provided by the OCM-G are available via a standard protocol – OMIS. The OMIS 2.0 specification [OMIS] defines also a set of six API functions, but they are merely for connecting to the monitoring system, sending the OMIS requests, etc. There is no real API which encapsulates the protocol – the user must know it and specify the monitoring requests practically at the level of protocol messages. For a detailed description of the API mentioned above, refer to [OMIS].

Additionally, another API function was added in the OCM-G. While the above mentioned six functions are to be used by tools using the monitoring system, in the OCM-G each application process has to add an explicit function call at the very beginning to register in the OCM-G. The prototype of the mentioned function is as follows:

```
ocmg_register(int *argc, char ***argv)
```

The user should pass the argc and argv parameters which contain command-line parameters. These parameters will be modified, i.e., all parameters specific for the OCM-G will be removed. For MPI applications, there is a possibility to call the above function from within MPI_Init, so that even this call will be transparent to the user. This can be achieved by instrumenting the MPI_Init call.

6. INTERNAL TESTS

Since the majority of code used in FOMP was taken from the OCM, it was already tested during many years of using the OCM. The parts which were added to FOMP include the new start-up scheme and a couple of new services. The start-up procedure was tested thoroughly, since it was a precondition for everything else to work at all. The new services were also tested for small applications using a console-based tool. The test procedure is described in section 5.4.3.

There is also a task 3.5 document [CGD33T35] containing description of tests and integration for the whole WP3.

7. ISSUES

N/A