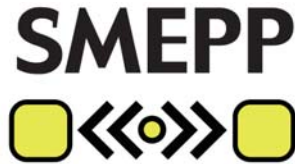


INFORMATION SOCIETY TECHNOLOGIES (IST) PROGRAMME

Project n°: FP6-IST-033563



SMEPP

Secure Middleware for Embedded Peer-to-Peer systems

WP5.2

SMCOM-Monitorization Tool

User Manual

Author(s): José Angel Dianes

Status -Version: Review version -1.0

Date:

Distribution - Confidentiality: Confidential

Code:

Disclaimer

This document contains material, which is the copyright of certain SMEPP contractors, and may not be reproduced or copied without permission. All SMEPP consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

The SMEPP Consortium consists of the following companies:

Participant no.	Participant name	Participant short name	Country
1 (Co-ordinator)	Universidad de Málaga	UMA	Spain
2	Tecnatom, S. A.	TEC	Spain
3	Technische Universität Graz	TUG	Austria
4	Siemens AG	SIEM	Germany
5	Valtion Teknillinen Tutkimuskeskus	VTT	Finland
6	Università di Pisa	UPI	Italy
7	Telefónica I+D	TID	Spain
8	Institute for Infocomm Research	I2R	Singapore

The information in this document is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Document Revision History

Date	Issue	Author/Editor/Contributor	Summary of main changes
October 2008	0.1	Jose Ángel Dianas (UMA)	Document structure and basic contents

Table of contents

1	Monitorization Tool.....	7
1.1	Functional requirements overview.....	8
1.1.1	Tool User Use Cases.....	8
1.1.2	SMCOM Interactions.....	8
1.2	Architecture summary.....	9
1.3	Deployment.....	9
1.4	Graphical User Interface.....	10

1 Monitorization Tool

As described, the code of SMCOM components follows the pattern $(\{wait|call|raise\}.seqblk.\{wait|call|raise\})^*$ where the using of a synchronization primitive is followed by a sequential block and the optional using of another synchronization pattern. This pattern allows synchronization points to be distinguished by the use of the synchronization primitives. These points can be taken into account for the design of monitorization tools in terms of memory usage, execution times or similar properties. These tools will be used in combination with the Runtime Component Framework (RCF) which is the basis for the execution of the different SMEPP components and which can be seen as a kind of scheduler where the execution of the components is scheduled.

Figure 1 shows an example of monitorization points in an active component. When these points are reached, the scheduler can provide information to the monitorization tool.

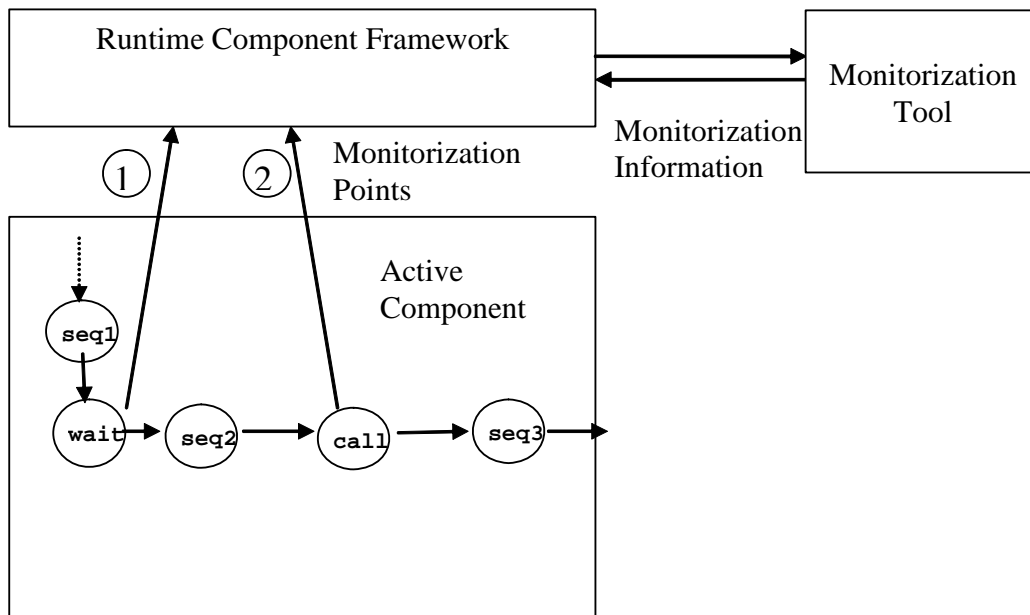


Figure 1 Monitorization Points

This way, the runtime component framework works as a type of virtual machine catching the using of the synchronization primitives. When this happens, the RCF must decide new actions to be performed, but in addition, it can provide information related to the execution of the components such as:

- Components connected to the RCF
- Interconnection between components
- Methods invocation
- Events publishing and subscribing

- Event creation

1.1 Functional requirements overview

1.1.1 Tool User Use Cases

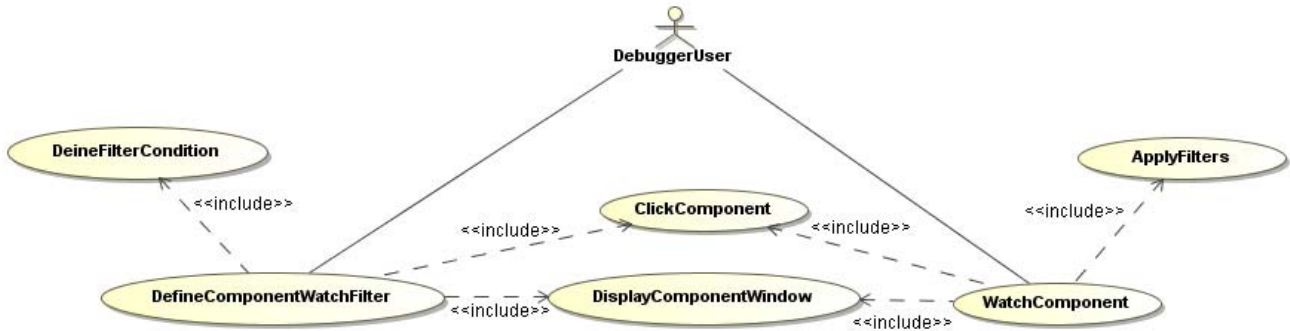


Figure 2 Tool User use cases

DebuggerUser actor represents a Monitorization Tool user. Basically, it can perform two basic tasks:

- WatchComponent: display a window to visualize the component received primitives (wait/call). This may include applying filters to the displayed messages.
- DefineComponentWatchFilter: define the filter to the previous display window. This includes the definition of some condition (f. e. only wait primitives).

1.1.2 SMCOM Interactions

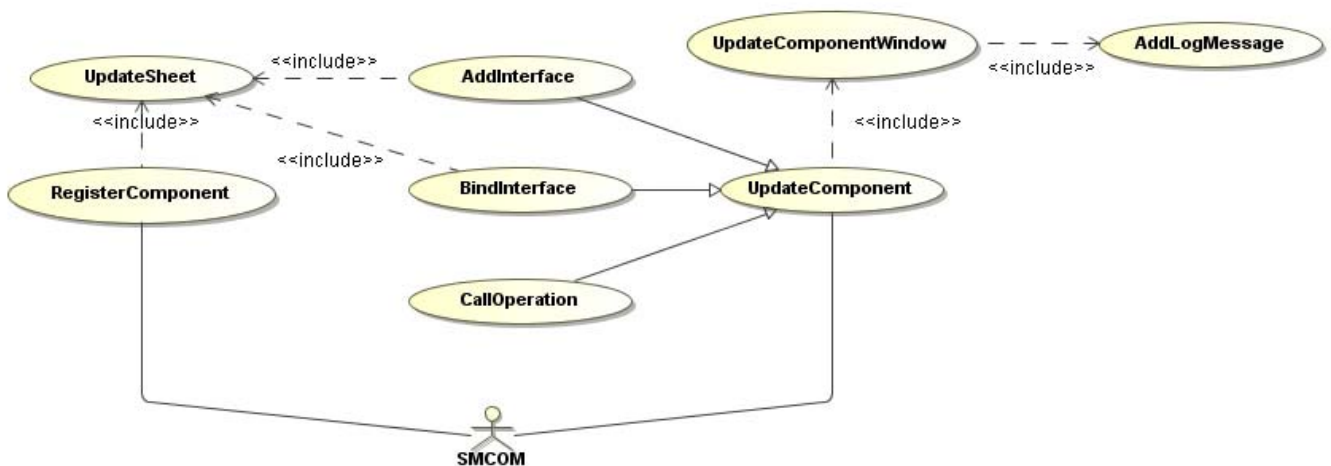


Figure 3 SMCOM Interactions

The SMCOM actor represents the SMCOM runtime environment. It can interact with the monitorization tool in two different ways:

- Registering a new component, that will be added to the displayed sheet in the GUI of the tool.
- Update component related information, including interfaces, operations and bindings with other component interfaces. It also includes to update the component window with the interaction activity between components (invocation of primitives).

1.2 Architecture summary

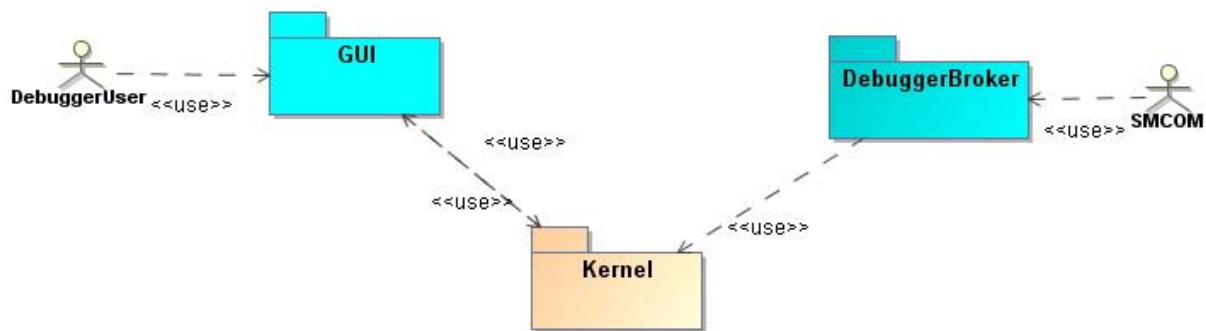


Figure 4 Monitorization Tool Architecture

Figure 4 Monitorization Tool Architecture summarize the Monitorization Tool architecture. A Model-View-Controller like architecture is applied, with a Kernel representing the model and the GUI representing the view and one side of the controller that interacts with the Tool User. But the controller is split in two parts. The GUI one and the most important part of the SMCOM side, that is represented by the *DebuggerBroker*, that interacts with SMCOM runtime environment.

1.3 Deployment

The Monitorization Tool is contained in a Java jar that interacts with the SMCOM Runtime environment jar. Figure 5 shows these interactions. The Monitorization is silent to SMCOM applications, because are the own runtime environment the responsible to inform to the Monitorization Tool (or SMCOM-Debugger) with the application interactions, that is, the interactions between the SMCOM Components that are executing while the SMCOMApplication is active.

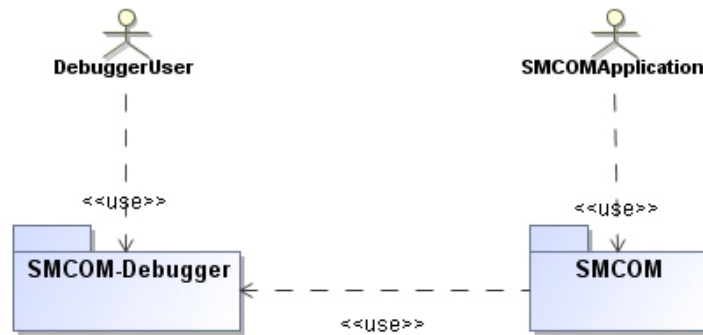


Figure 5 Monitorization Tool Deployment

The Monitorization Tool Graphical User Interface is performed in a separate window, and is managed in a totally independent manner from the SMCOMApplication point of view.

1.4 Graphical User Interface

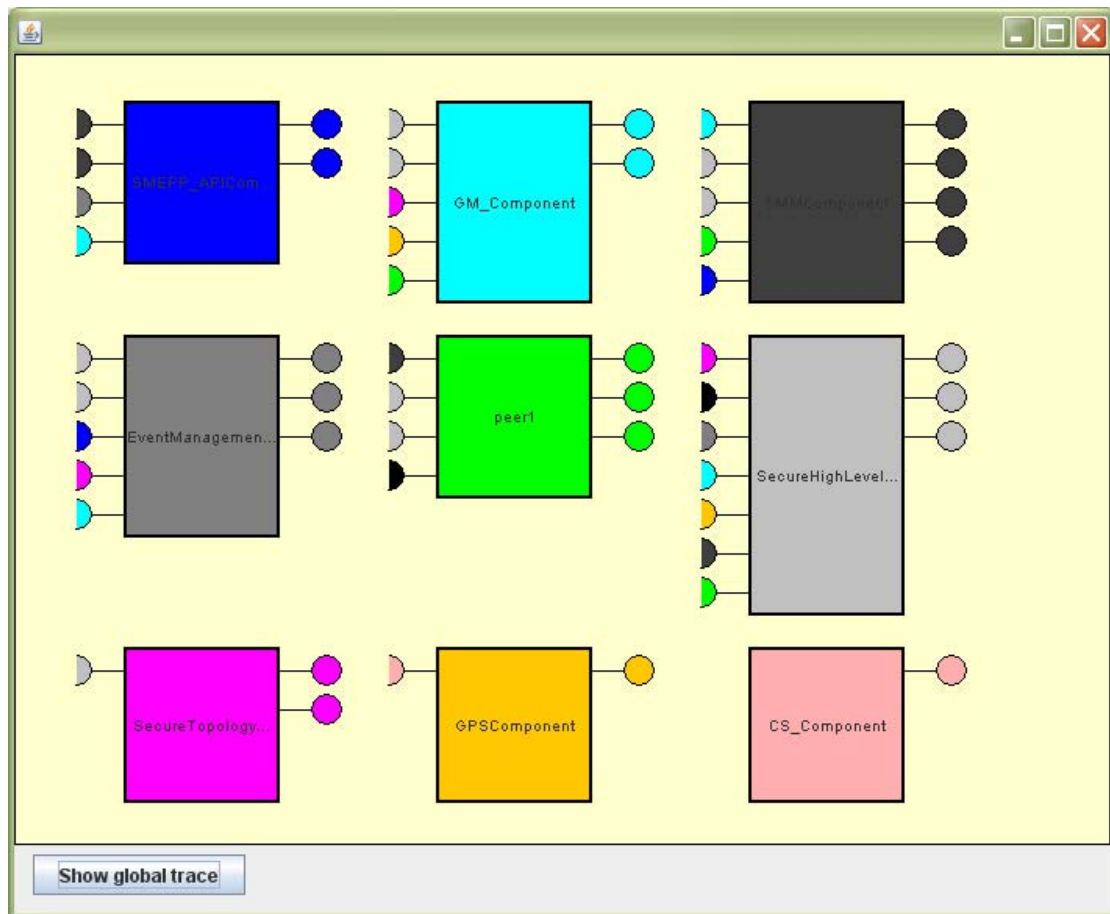


Figure 6 GUI Interface

The main GUI window of the Monitorization Tool (Figure 6) shows a component sheet. This component sheet contains all components in different colours. Each bound required interface is painted with the colour of its provider (only correct interface binding can

be performed correctly in SMCOM). Interfaces not bound are painted in black. Clicking on an required interface will highlight its provider (Figure 7).

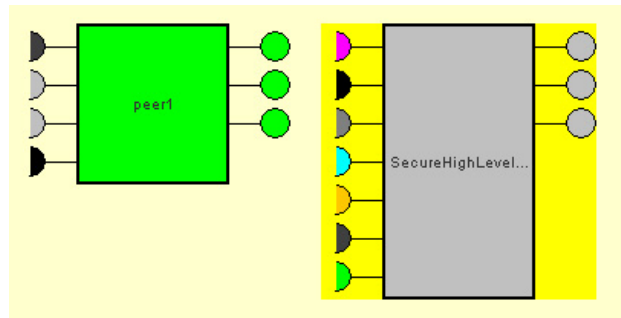


Figure 7 Interface provider highlight

Provided interfaces also have tooltips with its names and providers (Figure 8).

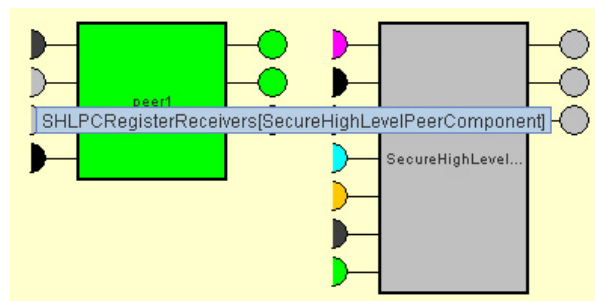


Figure 8 Provided interface tooltip

Clicking in a component makes the Component Window (Figure 9) be displayed. The Component Window contains the trace of all primitives performed in the scope of this component. The GUI also provides a similar window with the global trace.

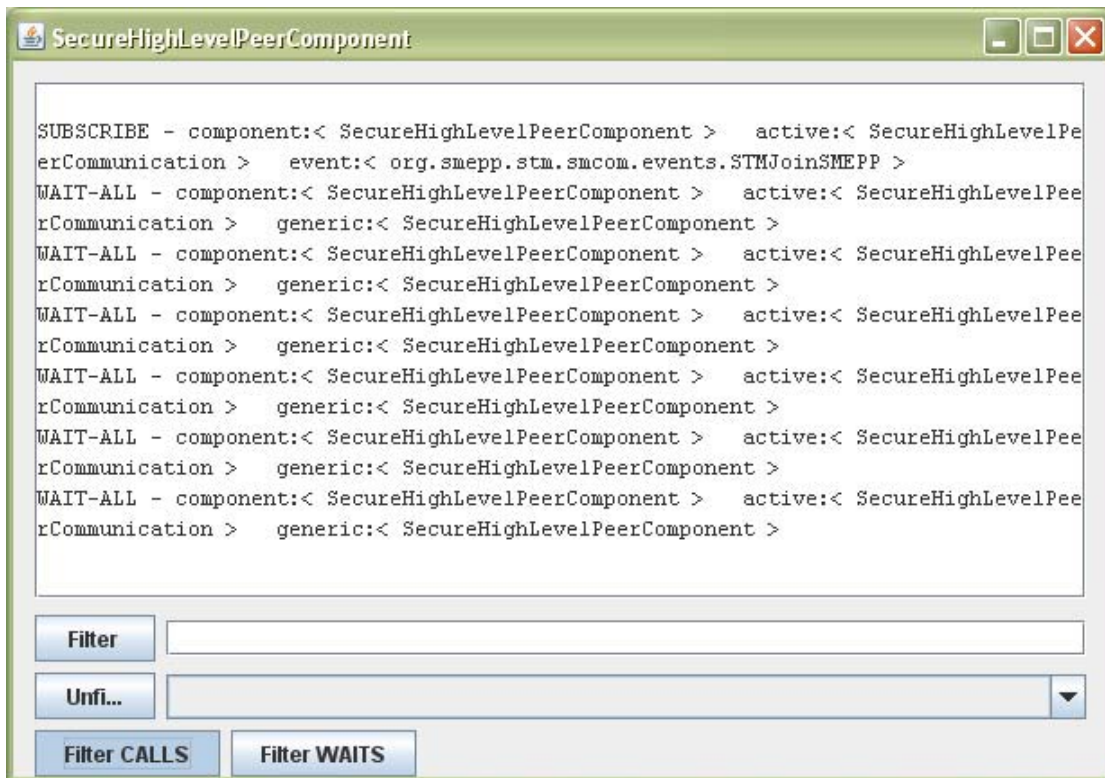


Figure 9 Component Window