JMonitor: A monitoring tool for distributed systems

Maurício G. Penteado
Computer Science Department
Federal University of São Carlos
São Paulo, Brazil
mauriciopenteado@dc.ufscar.br

Luís Carlos Trevelin
Computer Science Department
Federal University of São Carlos
São Paulo, Brazil
trevelin@dc.ufscar.br

Abstract — **Immersive, interactive, and collaborative distributed applications, due to their real time and multi-platform characteristics, require a monitoring structure of their functionalities during the development period, which is not fully available in the current monitoring systems, which are usually specific of the languages and platforms in which they are made available. Monitoring systems that require synchronism of the distributed objects during the processing is a complex task. It is often difficult to check the correct execution of the system. It might be useful for developers and users of such systems to have the ability of identifying, for example, if the reason for the system delay is due to the abusive use of a resource or due to programming mistakes; the identification of which object has crashed, compromising the system as a whole. Distributed object systems, also known as SOA (Service Oriented Architecture), generally use middleware of the broker type in their infrastructure and can have a dynamic number of connected clients or servers when running. There are, in the literature, several tools for the monitoring of performance, resources, debugging, and support, among others, of such systems. Nevertheless, the tools available in the literature are limited to programming languages or platforms that are able to process the monitoring agents related to them. In the paper, a tool for the monitoring of distributed object systems is proposed, based on communication described in XML documents, which allows language and platform independence in the such systems development.**

Key-words: Distributed Systems; Broker; JAMP; SOA; Monitoring Systems

I. Introduction and Motivation

The union of object orientation with distributed systems originated the area of the distributed object systems. The base for this model is the fundamental concept of objects, which can be clients, servers, or both. Objects defined as entities with specific behavior and configurable attributes can be combined in a simple way in order to provide the ability to execute customized services.

The middleware's of broker type can be seen as essential components of distributed object systems. These broker's have the function of managing the communication among the different objects of the system.

Distributed objects can storage their communication interfaces through the brokers, providing a way of tracking them. Objects can also search for interfaces that have been already registered in the broker. Through such interfaces, the consumption of the provided services by remote objects can be performed.

When the architecture of the distributed system uses provider and consumer objects, it is possible confirm that such systems are structured under the SOA model (Service Oriented Architecture).

There are several available technologies for the development of SOA systems, such as: CORBA, RMI, JINI, Web Services, among others [1][2].

The JAMP (Java Architecture for Media Processing) platform [5], developed to provide services for the distribution of different types of media for distributed object systems. The JAMP platform have a broker for the management of the distributed objects, as well as a set of frameworks and servers that help the conception and maintenance of such systems.

For the development of systems with interactive, immersive, and collaborative characteristics, the LAVIIC (Immersion, interactivity, and collaboration lab) is available at the Federal University of São Carlos, which provide a CAVE and resources for the processing of virtual distributed environments [3][4].

Virtual Distributed Environments (VDEs) are characterized by the fact that they have resources that are accessible, synchronous, as well as distributed, in different computers connected by communication networks. Since these environments allow the development using different programming languages, the task of monitoring them is a complex activity.

Monitoring such systems can be critic for the development and maintenance of such types of systems, allowing the identification of errors that lead objects to the abusive use of resources, faults regarding their functions, or even the interruption of their operation. Such characteristics can compromise the execution of the system as a whole, in case of synchronous systems.

In face of the resources provided by the JAMP and similar platforms, it was identified the need for a tool that is able to monitor: (*i*) simple and customizable data to the distributed object systems; (*ii*) complex data, such as the usage of the available resources. This tool must also be ready to monitor different programming languages.

The development of the JMonitor tool, showed in this paper, is aimed to fulfill this need. JMonitor allows to follow up the information of the distributed object systems. From simple data, such as variable values, to complex data, such as the availability and usage of resources, as well as other information, can be reported using JMonitor.

II. RELATED WORK

There are several projects that investigates different types of monitoring functions for distributed object systems. The techniques and presentations of the project of Karunamoorthy [6], PerfMoon[7], CENNI [8], among other existing ones, are examples in this category.

In Karunamoorthy's project [6], approaches for the acquisition of descriptive data regarding the resources provided to distributed object systems are reported. Such approaches use the JINI platform as infrastructure. The approaches consist of recognize and provide the computational resources used where the distributed objects are being executed, in a centralized way. Nevertheless, monitoring customization is not allowed.

PerfMoon [7], on the other hand, uses monitoring modules at a kernel level in its architecture. These modules must be enabled in the systems to be monitored, providing RMI communication interfaces prepared to receive invokes originated from the central monitoring daemon. However, developing new modules for the monitoring of customized characteristics are highly complex tasks.

In CEMMI[8], the JMX platform and MBeans components are used to provide a distributed monitoring system based on rules, which can be customized according to the needs of the object to be monitored. For CEMMI, the use of the JMX platform does not allow the monitoring of simple variables for "non Java" applications.

In the architecture of the JMonitor tool addressed in this paper, any data provided by distributed object systems can be monitored by means of XML files. This allows an easy customization, as well as language and platform independence.

III. CHALLANGES

The architecture of distributed object systems are quite similar to the client-server architecture, which are distinguished by the presence of a broker. If a broker is available in a computer network, any computer in the network can use it in order to provide or use services. This is can be performed of its server objects or by means of its client objects.

The computers in the network using the broker can be geographically distant from each other. Also, the amount of client objects using services can be dynamic. A tool can be added to the broker to improve it, if it provide data about: (*i*) which services are registered in a certain moment; (*ii*) the amount of the usage from available resources and used resources, and; (*iii*) debugging data.

The challenges in acquiring and making available such information are closely related to the development of the distributed object system to be monitored.

In a development level, client objects of a given distributed system must periodically send the information required by the Server object periodically. The server object must create a document for the monitoring task. This document will make its information available, as well as the information from all its clients. Such document must be made available periodically.

IV. JAMP PLATAFORM

The JAMP platform is contextualized in Figure 1, among other platforms that provide middleware of broker type. It is also possible to identify in Figure 1 that the JAMP broker is named JBroker.

JBroker extends the Java RMI and Java Serialization technologies. Such extension provides mechanisms to find and distribute the objects. These mechanisms allow the storage of the service providers and the invoking of such services by the consuming objects.

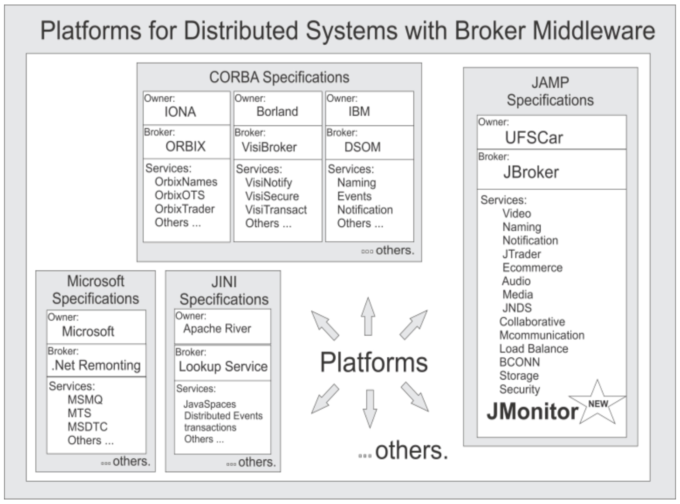

Platforms for Distributed Systems with Broker Middleware
CORBA Specifications
Owner: IONA
Broker: ORBIX
Services: OrbixNames OrbixOTS OrbixTrader Others ...
Owner: Borland
Broker: VisiBroker
Services: VisiNotify VisiSecure VisiTransact Others ...
Owner: IBM
Broker: DSOM
Services: Naming Events Notification Others ...
others.
JAMP Specifications
Owner: UFSCar
Broker: JBroker
Services: Video Naming Notification JTrader Ecommerce Audio Media JNDS Collaborative Mcommunication Load Balance BCONN Storage Security
JMonitor
NEW
Microsoft Specifications
Owner: Microsoft
Broker: .Net Remonting
Services: MSMQ MTS MSDTC Others ...
others.
JINI Specifications
Owner: Apache River
Broker: Lookup Service
Services: JavaSpaces Distributed Events transactions Others ...
others.
Platforms
...others.

Figure 1. Comparisons among platforms for distributed systems.

The JMonitor tool was codified over the infrastructure of the JAMP platform and it will be made available as an additional JAMP service, as shown in Figure 1.

Although JMonitor is structured based on the JAMP platform, it is important to emphasize that programmers applications intending to use it do not necessary need to be structure your applications based on the same platform. The only requirement is that such applications must make available the information to be processed by means of XML documents, generated according to the validating specifications.

V. Implementation

The implementation of the JMonitor tool can be divided into three modules and two auxiliar files that control the communication, as shown in Figure 2.

The modules are denominated: JMonitorManager, JMonitorAgent, and JMonitorAppletAgent. The auxiliar files are JMonitor.dtd and JMonitor.xml.

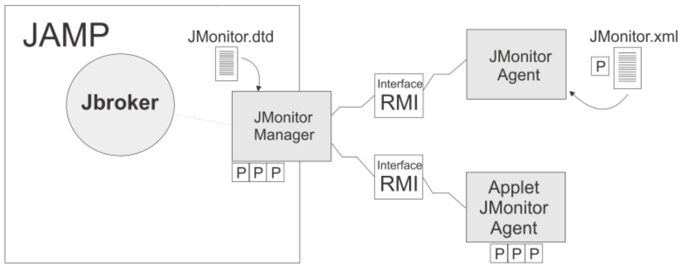


Figure 2. JMonitor, communication among modules.

A. The JMonitorAgent Module

This module was prepared to be executed with the distributed object thats providing services. As explained in item III, programmers intending to use the JMonitor tool must prepare their systems for this.

In such systems, server objects must collect the information of all client objects by periodic and customized means. Once the information is collected, the servers must generate or update an XML file reporting the information to be monitored.

A thread was developed in the JMonitorAgent module to constantly check whether the markup file was generated or received any update by the server object. This file is named JMonitor.xml.

When any modification is detected in this file, the JMonitorAgent module transforms the file into a set of bits prepared to be sent to the managing module. This operation is carried out by means of serialization techniques.

After the file serialization, the JMonitorAgent invokes an RMI call to the receiveProperties method, from the JMonitorManager. This way, the JMonitorAgent sends a set of bits representing the XML file to the managing module.

B. JMonitorManager Module

The JMonitorManager managing module was prepared to receive RMI calls originated in remote objects. By means of the receiveProperties call, the module receives the set of bits sent by the JMonitorAgent. Once this set of bits is received, JMonitorManager performs the de-serialization of the JMonitor.xml file.

After finishing the de-serialization of the file, the JMonitorManager module checks whether the same follows the established rules in the JMonitor.dtd file.

In JMonitor.dtd, the rules that validate the JMonitor.xml file are defined. This file allows checking the compatibility of the received file with the type of file expected by the JMonitor tool. Once the validation is accepted, the received file is then transformed into an object of the Properties class.

An object of the Properties class represents all the properties of the distributed system that were marked in the JMonitor.xml file. This object has the identification and information related to the server object. It also has the identification and information of each client connected to it.

Together all the information regarding the markup file, an object of the Properties class also receives a lifespan, identifying how long such information is taken as valid. If the information is not updated within the lifespan of the object, the object is destroyed.

The JMonitorManager module has the ability to store Properties objects representing all the systems that are being monitored at a given moment. By means of a thread linked to the module, the lifespan checking of such stored properties is performed.

In case the lifespan is expired, the Properties object is destroyed. Otherwise, the Properties object remains available to be sent to the visualization module. This is performed by means of invokes periodically originated in the visualization module.

C. The JMonitorAppletAgent Module

The JMonitorAppletAgent visualization module was developed with an applet application accessible to any computer in the network that can connect to the JBroker. This module has the function of making the monitored information available to the developers and users of the monitored systems, in an intuitive way.

In the monitoring applet there are graphic generators that intuitively show: (*i*) RAM memory loads; (*ii*) swap memory loads; (*iii*) average processor usage load, and; (*iv*) network interface usage. The system that runs the distributed object of interest contains such information. This object can be selected from the list of objects that are being monitored.

The selectable list of distributed objects is disposed in an alphabetic and hierarchical way. In the first level, it demonstrates a first service provider distributed, as well as all its clients. In the second level, a second service provider object, also with its clients, and so on, until all the monitored systems are selectable.

The list of monitored properties is displayed under the list of selectable distributed objects. When a distributed object is selected, its properties are made available in this second list, keeping constant update when new values are registered.

The selection of the first list also updates the graphic generators in order to correctly display the data of the selected system.

Besides the applet, the JMonitorAppletAgent also has a thread that periodically invokes the informProperties call of the managing module, through the RMI. This way, all the valid Properties objects stored in the manager are retransmitted to the applet.

Through the applet, it is possible to visualize the information of the server distributed objects, as well as information of the remote clients connected to it. Such information is related to: (*i*) usage and availability of the processors used in the system; (*ii*) variable values, and; (*iii*) informative details, such as the email of the person in charge of a system. Such examples of information can be visualized by any computer in the network, through the applet being executed in union with the JBroker and the remaining modules of the JMonitor tool.

D. JMonitor.dtd and JMonitor.xml Files

These two files are necessary for the JMonitor tool runs. By using them, it is possible to guarantee that documents sent by users are in accordance with the documents the tool is prepared to process.

A part of the JMonitor.dtd file can be seen in Figure 3. This file is used by the managing module and has rules that validate the JMonitor.xml file. Figure 4 shows a JMonitor.xml example file reporting information of a fictitious system.

It is possible to observe in Figure 3 that the systems must, obligatorily, inform the name of the service server object to be monitored, as well as its lifespan. Such information is used by the JMonitor tool in order to determine how long the information in this document should be considered valid.

In case the system to be monitored might want to use the graphic generators in the applet, memory-ram-total, memory-ram-free, memory-swap-total, memory-swap-free, cpu-used, recv_packages, and trans_packages elements must be informed. Nevertheless, the use of such graphic generators is optional to the use of the tool.

```
<!ELEMENT monitoring ( serverName, vality, memory-
ram-total?, memory-ram-free?, memory-swap-total?,
memory-swap-free?, cpu-used?, recv_packages?,
trans_packages?, static-properties, dynamic-
properties, client+ ) >
        . . . .
<!ELEMENT client ( clientName, memory-ram-total?,
memory-ram-free?, memory-swap-total?, memory-
swap-free?, cpu-used?, recv_packages?,
trans_packages?, static-properties?, dynamic-
properties ) >
        . . . .
```

Figure 3. JMonitor.dtd.

According to the rules of the JMonitor.dtd file, the client element is also optional. In case it exists, it must obrigatorily have a unique name among the client objects connected to a same server object. The use of graphic generators related to the system that processes the client distributed object is also optional.

The data of the static-properties element are registered in pairs of name and value. These data are prepared to receive information such as: (*i*) email of a person in charge of the system; (*ii*) where the access to the source code can be obtained; (*iii*) relation of dependencies to be installed, etc. Such information varies according to the requirements of the developers and users of the systems.

The change of the defined values with the dynamic-properties element is checked and loaded at each update performed by the periodic internal events of the JMonitor tool. These dynamic-properties elements are suitable to follow the change of values of the variables for debugging purposes.

The graphic generator elements also use this way of checking and loading information at each update.

It is possible to observe in Figure 4 that the JMonitor.xml file was generated with a single connected client object. This information is marked by the <client> and </client> tags. In case no client object exists, the tool is ready to keep the monitoring of the server object only.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE monitoring SYSTEM "JMonitor.dtd">
<Jmonitor>
  <serverName value="Demo Server 1" />
  <vality value="20" />
  <memory-ram-total value="3062808" />
  <memory-ram-free value="923938" />
  <memory-swap-total value="6256636" />
  <memory-swap-free value="4654613" />
  <static-properties>
    <property name="Email" value="mauricio@dc.ufscar.br" />
    <property name="Source" value="http://www.laviic.dc.br" />
  </static-properties>
  <dynamic-properties>
    <property name="Dynamic-test" value="80" />
  </dynamic-properties>
  <client>
    <clientName value="Client 1" />
    <cpu-used value="43" />
    <static-properties>
      <property name="Email" value="mauricio@dc.ufscar.br" />
      <property name="Source" value="http://www.laviic.dc.br" />
    <dynamic-properties>
      <property name="Dynamic-test" value="4" />
    </dynamic-properties>
  </client>
</Jmonitor>
```

Figure 4. JMonitor.xml.

VI. CASE STUDY

In order to illustrate the usage of the JMonitor tool, a system that allows the visualization and iteration in real time of 3D distributed modeling was developed. This system can be used at different computers of the network. This developed system follows the standards of distributed virtual environments [9].

The case study also allows the navigation under environments prepared to provide a higher immersion sensation, such as CAVES. Such environments usually have graphic clusters that control their operation.

Graphic clusters are sets of computers that have projectors and graphic cards, which are set in a customized way in order to aggregate its resources and provide the illusion of a single computer with large processing capability.

These environments parallelize the image processing in such a way that each computer of the graphic cluster becomes responsible for the processing and projection of a part of the image. This technique allows the processing division among the nodes of the cluster improving the quality of the final image [10]. Such technique is known as multi-projection.

In the case study each node of the graphic cluster processes a client distributed object of the VDE under test.

The server distributed object is kept ready to receive interaction actions, originated in any network client. It also has the function of distributing the performed interaction in the environment to passive clients of this action. This server object keeps the environment synchronized.

In order to support the server object, JBroker performs the function of coordination and locate the other client and server objects spread in the network.

In the case study, there are client distributed objects ready to be executed in graphic clusters. Such objects have camera tuning limited to its portion of the image.

Also, for the case study, there are client objects prepared to execute in personal computers, set to process the whole image.

For the case study execution, two portable personal computers were used, as well as a graphic cluster with three nodes, totalizing five client objects connected to the server. The graphic cluster can be observed in Figure 5.

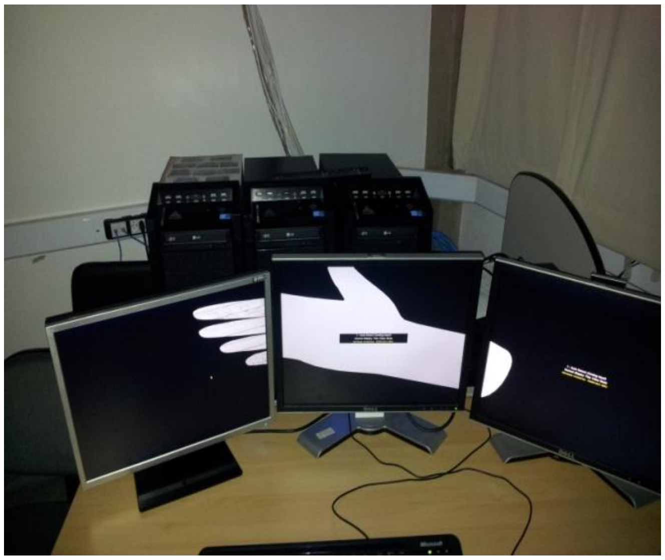


Figure 5. Graphic cluster used for tests.

Figure 6 shows an image of the whole environment in execution, including the two portable computers.

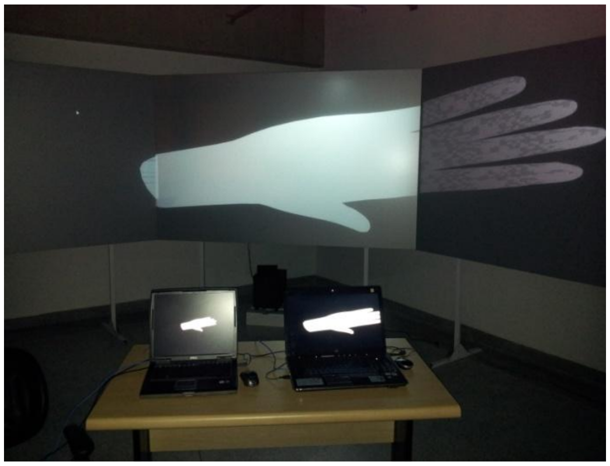


Figure 6. All clients interacting.

It is possible to observe in Figure 6 that the images area placed in the same direction, independently of the multi-projection of the CAVE or of the generated projections in the portable computers. The movement interaction propagated by all clients is a characteristic of the developed test system.

It is also possible to observe in Figure 6 that the multi-projections of the CAVE and the projections generated in the personal computers have different approximation values (zoom). This characteristic was developed in order to improve the exploration of the image limits offered by the CAVE.

The communication of the JMonitor tool can be visualized in Figure 7. In this figure JMonitor is monitoring the distributed objects of the case study.

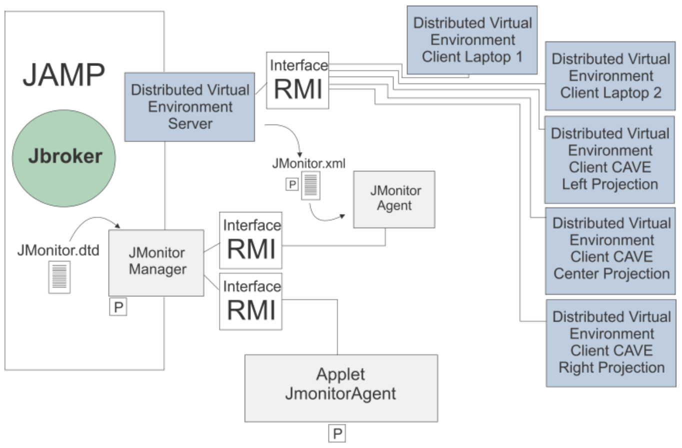

JAMP
Jbroker
Distributed Virtual Environment Server
Interface RMI
Distributed Virtual Environment Client Laptop 1
Distributed Virtual Environment Client Laptop 2
Distributed Virtual Environment Client CAVE Left Projection
Distributed Virtual Environment Client CAVE Center Projection
Distributed Virtual Environment Client CAVE Right Projection
JMonitor.xml
P
JMonitor Agent
JMonitor.dtd
JMonitor Manager
P
Interface RMI
Interface RMI
Applet JmonitorAgent
P

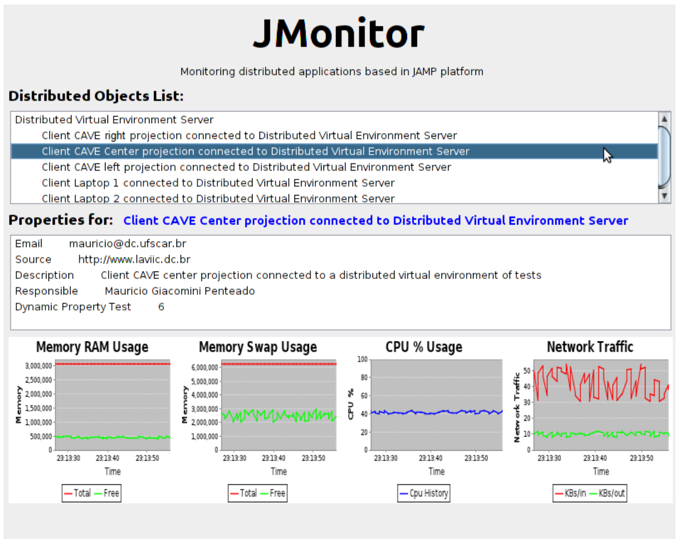
Figure 7. JMonitor making information of the case study available.

Together with the code of the client objects, statistical verifications of the systems and customized information collection it was also coded. According to the codification, such data are periodically updated and sent to the server object.

The verification and collection of data referring to the system that executes the server distributed object were also developed.

Periodically, the server object gathers the information collected from all clients. Together with the information of the server object itself, the server generates or updates the JMonitor.xml file prepared to be captured by the JMonitor tool. This tool is responsible to provide the available information in the network. It's performed in an intuitive way, to developers and users of the system.

In Figure 8, the visualization applet of the JMonitor tool is demonstrated under execution, monitoring the distributed objects of the case study. In this figure, it is possible to observe that a client object is selected. This action activates the listing of the static and dynamic properties of the selected object, as well as the generation of the graphics related to the resources of the system that processes it.


JMonitor

Monitoring distributed applications based in JAMP platform

Distributed Objects List:

Distributed Virtual Environment Server
Client CAVE right projection connected to Distributed Virtual Environment Server
Client CAVE Center projection connected to Distributed Virtual Environment Server
Client CAVE left projection connected to Distributed Virtual Environment Server
Client Laptop 1 connected to Distributed Virtual Environment Server
Client Laptop 2 connected to Distributed Virtual Environment Server

Properties for: Client CAVE Center projection connected to Distributed Virtual Environment Server

Email	mauricio@dc.ufscar.br
Source	http://www.laviic.dc.br
Description	Client CAVE center projection connected to a distributed virtual environment of tests
Responsible	Mauricio Giacomini Penteado
Dynamic Property Test	6

Figure 8. The JMonitor tool monitoring information of the case study.

VII. RESULTS

The results regarding the use of the JMonitor tool in the case study were satisfactory. The JMonitor tool helped in the identification of several programming errors during the development of the VDE system under test. It also allowed the monitoring of the variables values codified in different languages.

The communication interruption of a client is easily observed by using JMonitor. In this case, the lifespan of the information that is not updated does not take long to expire. This way, the object with interrupted execution becomes unavailable in the listing of the monitoring applet, which is quite intuitive.

The graphic generators of the tool showed similar graphs to the real use of computational resources used in the remote computers.

These graphics are similar, although they present delays caused by the network and by the processing of XML documents. Such delays can be minimized with the correct tuning of the periodicity defined for the updates.

VIII. CONCLUSIONS

In this paper, techniques for the acquisition and demonstration of different properties aggregated to distributed object systems were addressed.

Such properties can be useful in the identification of abnormal behavior in the use of resources, in the processing of the systems, as well as in the dissemination of the information in the network.

The JMonitor tool was implemented based on infrastructure offered by the JAMP platform. It can be used by any distributed system that generates information marked through XML documents. This allows the systems to be monitored to be independent of platform and programming language.

REFERENCES

[1] Al Belushi, W.; Baghdadi, Y.; , "An Approach to Wrap Legacy Applications into Web Services," *Service Systems and Service Management, 2007 International Conference on* , vol., no., pp.1-6, 9-11 June 2007

[2] Rafe, V.; Rafeh, R.; Fakhri, P.; Zangaraki, S.; , "Using MDA for Developing SOA-Based Applications," Computer Technology and Development, 2009. ICCTD '09. International Conference on , vol.1, no., pp.196-200, 13-15 Nov. 2009

[3] Prado, G.M.; Zorzo, S.D.; Trevelin, L.C.; de Paiva Guimaraes, M.; Gnecco, B.B.; , "Interactive architecture for interactive social inclusion applications," Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on , vol., no., pp.1280-1285, 9-12 Oct. 2011

[4] M. Francischetti-Corrêa, L.C. Trevelin, and M P Guimarães, "Molecular Visualization with Supports of Interaction, Immersion and Collaboration among Geographically-Separated Research Groups" in Enterprise Information Systems. Springer Berlin Heidelberg, v. 221, p. 128-135, 2011.

[5] Corrêa, M.F.; Schpector, J.Z.; Trevelin, L.C.; de Paiva Guimarães, M.; , "Immersive environment for molecular visualization to interaction between research groups geographically dispersed," *Applied Sciences in Biomedical and Communication Technologies (ISABEL), 2010 3rd International Symposium on* , vol., no., pp.1-5, 7-10 Nov. 2010.

[6] Karunamoorthy, D.; Devinuwara, N.; , "Monitoring & manging dynamic distributed systems," Electronics, Circuits and Systems, 2005. ICECS 2005. 12th IEEE International Conference on , vol., no., pp.1-4, 11-14 Dec. 2005.

[7] Jian Xu; Manwu Xu; , "A Performance Monitoring Tool for Predicting Degradation in Distributed Systems," Web Information Systems and Mining, 2009. WISM 2009. International Conference on , vol., no., pp.669-673, 7-8 Nov. 2009.

[8] Jingxin Peng; Jian Cao; , "ECA rule-based configurable frame of distributed system monitoring," Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on , vol.1, no., pp.674-677, 10-12 Dec. 2010.

[9] Ali, A.E.E.; El-desoky, A.I.; Salah, M.; , "An allocation management algorithm for DVE system," Computer Engineering & Systems, 2009. ICCES 2009. International Conference on , vol., no., pp.489-494, 14-16 Dec. 2009.

[10] Colombo Dias, D.R.; La Marca, A.F.; Moia Vieira, A.; Neto, M.P.; Brega, J.R.F.; de Paiva Guimaraes, M.; Lauris, J.R.P.; , "Dental arches multi-projection system with semantic descriptions," Virtual Systems and Multimedia (VSMM), 2010 16th