MOINC Agent, Dynamic Web Service Runtime Environment

N. T. Baranasuriya, M. B. R. C. Boralugoda and M. S. M. A. Nafran,
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka.

Abstract - **Project MOINC is an attempt to blend the web services paradigm with the grid computing paradigm to facilitate high available and high scalable web services deployments. MOINC Agent is one of the four main components of Project MOINC. This research paper presents the three main research areas that were covered during the attempt to build the MOINC Agent Component. The three main research areas are; Implementation of a Machine State Detection Algorithm, Re-implementation of Axis2 Kernel module and the Screensaver implementation.**

Index Terms - **Axis2, Grid computing, Idling resources, Volunteer computing**

I. Introduction

MORA Open Infrastructure for Network Computing (MOINC) is a project which focuses on designing and developing a Java web services deployment platform which provides high availability and scalability using the computing power of idling computers in a network. MOINC would ultimately enable service providers to offer high available web services with supreme throughput without having to deploy any mainframes, super computers or even traditional server clusters which are generally expensive, complex and difficult to maintain.

Similar applications to MOINC Agent can be named as SETI@Home and World Community Grid (WCG). These applications use the idling computing resources of computers connected to the internet to process complex tasks. The main difference between MOINC Agent and the above mentioned applications is MOINC Agent is a web service execution environment whereas applications such as WCG execute tasks. Hence if any problem requires processing of complex web service executions MOINC Agent can be a life saver by providing idling computing resources spread all over the world.

MOINC platform consists of four main components:

A. MOINC Server
B. MOINC Client agent
C. MOINC Server management module
D. Thisara Messaging Framework

This research paper presents the research areas that were covered during the process of implementing the MOINC Client Agent which was renamed *MOINC Agent*.

There were three main research areas that were covered in implementing MOINC Agent. They can be briefly explained as follows and each area will be thoroughly discussed in the following topics.

Machine State Detection Algorithm: MOINC Agent is an application which automatically launches itself once it detects that the user is no longer using his/her computer. To achieve this purpose it was necessary to build an algorithm that detects whether the user is using the machine or not. The research that was done to implement this algorithm will be discussed first.

Axis2 Kernel Module Re-Implementations: MOINC Agent is the web service execution environment of the entire platform. Axis2 is the web service runtime that was used in this project. However the default implementation of Axis2 did not meet our project requirements. Therefore we modified the Axis2 code in order to integrate it with the rest of our project artifacts. The changes were mainly done to the Kernel module of Axis2. The remainder was unchanged and used as it is. The research that was done to modify the Axis2 Kernel is discussed secondly.

Screensaver Implementation: Once MOINC Agent application launches itself after detecting user inactivity, a screensaver is displayed to the user. The screensaver gives the user information about what services were deployed and how many hits each service received. The research that was done to implement this screensaver is discussed thirdly.

II. Machine State Detection Algorithm

A. Introduction

This topic covers the research that was done to implement the Machine State Detection algorithm. The main purpose of this module is to monitor user activity and run the application when it detects that the user is no longer using the computer.

This research was done in the very first few weeks of the project because this was the entry point to the whole system. We came up with three different approaches to implement this module. The three approaches varied immensely on implementation and algorithms followed. We considered each of them in the light of easiness of usage, bug freeness and extensibility before selecting which of them was the most appropriate one. Let us now look at each of these in detail.

B. JNI Based State Detection

1) Implementation

In order to check for user inactivity our initial idea was to monitor the system idling process which exists on any operating system. Our algorithm was very simple and it was as follows.

```
int var = CPU_Usage_of_System_Idling_Process;
if (var > 95) {
        Machine_state = inactive;
} else {
        Machine_state = active;
}
```

Though the algorithm was simple, the implementation was hectic. Java is a platform independent programming language. Hence it is not tightly bound to any OS specific logic. In order to find the CPU usage of the system idling process we had to tap into the OS, but this was not a possibility through the JVM single handedly.

From the research material we went through we found that we had to write a programme in C which interacted with DLLs of the OS to retrieve the CPU usage of the system idling process. This C programme was then converted to a DLL using the MinGW C compiler [6]. Finally we coded a Java programme which got the information through this DLL via the Java Native Interface (JNI) [1] [2].

During our research we found a code base [2] [3] that dealt with JNI which helped our implementation. However we could not straightaway use the code that was available because the final CPU usage value it returned was of the current process which the programme was running on. In other words, it returned the CPU usage of the Java programme instead of the System Idling Process. To get around this problem we changed the existing implementation of the code that was available.

First we added a new method to the C programme to set a process id of which the CPU usage should be profiled. The method was added in such a way that it was visible to the Java programme via the JNI. Hence before calling the getProcessCPUTime() method, we called the newly added setPid (JNIEnv * env, jclass cls, jint pid) method with the pid variable set to zero. (System Idling Process is always given the process id zero by the OS). The method we added is given below.

```
JNIEXPORT jint JNICALL
Java_com_vladium_utils_SystemInformation_setPid
(JNIEnv * env, jclass cls, jint pid)
{
    DWORD errCode;
    LPVOID lpMsgBuf;
    s_PID = pid;
    s_currentProcess =
     OpenProcess(PROCESS_ALL_ACCESS,FALSE,pid);

    if (s_currentProcess==NULL) {
            errCode = GetLastError();
            FormatMessage(
               FORMAT_MESSAGE_ALLOCATE_BUFFER |
               FORMAT_MESSAGE_FROM_SYSTEM,
               NULL,
               errCode,
               MAKELANGID(LANG_NEUTRAL,
                          SUBLANG_DEFAULT),
                          (LPTSTR) &lpMsgBuf,
                          0, NULL );

            printf("[CPUmon] Could not attach to
native process.\n  Error code: %ld\n  Error
description: %s\n",errCode,lpMsgBuf);
            fflush(stdout);
            LocalFree(lpMsgBuf);
            return errCode;
    }
    printf("[CPUmon] Attached to native
process.\n");
    fflush(stdout);
    return 0;
}
```

2) Problems in this approach

The first problem in this approach was the OS compatibility problem. MOINC Agent application needed to be supported on both Windows and Linux platforms. However this approach only worked on the Windows platform.

The second problem was the difficultness of maintaining the code. Each change in the C programme required recompiling it to a DLL and attaching it to the Java programme via the JNI.

The third and the most troubling problem was the inappropriateness of this approach for the MOINC Agent. From the tests that we carried out we found that when the user is using a light weight application (e.g. typing text in

the notepad) the CPU usage of the SIP was very close to 100%. According to our algorithm the computer is now idling but actually it is not. Due to this problem we dropped this method of implementing the state detector module and looked for other means of implementation.

C. *Keyboard and Mouse Monitoring*

1) Implementation

In this approach we used keyboard hits and mouse movement monitoring programme to check for user inactivity. The algorithm we used here differed from the earlier one slightly.

```
int var =
mins_after_latest_keyhit_or_mouse_movement;
if (var > 10) {
        Machine_state = inactive;
} else {
        Machine_state = active;
}
```

To implement this we used classes from the java.awt package. The logic was simple. Each time the programme detected a keyboard hit or a mouse movement we reset a timer. Once the timer runs out due to no keyboard or mouse movements, we launched the application.

The advantage of this method of implementation was the easiness of implementation. The programme was 100% Java and no low level programming was necessary.

The code we used was a modified version of the code base which can be found at [11].

2) Problems in this approach

There was only one main problem in this approach and that was the inability of running the programme in the background. MOINC Agent had the requirement of running as a background process and constantly checking the status of the computer before launching the rest of the operations. In this approach in order for it to detect keyboard hits and mouse movements, the focus needed to be set to the program thus eliminating the chances of running as a background process.

Due to this reason we had to drop this approach as well and move on to another method of implementation.

D. *JNA Based Approach*

1) Implementation

By this time we knew the correct approach to detect user inactivity was to monitor keyboard hits and mouse movements. The remaining unsolved problem was to figure out how to achieve this goal while running in the background.

While researching thoroughly on this we came across a Java application [12] which was used in a chat application to set the status of the user automatically to 'away' when the computer was idling.

The programme was based on the open source class library called Java Native Access (JNA) [7]. Through JNA it was possible to communicate with the OS specific logic straight from Java without writing low level programmes in C or C++ [3].

Another advantage of this approach was, as it is purely Java it could be used both in Windows and Linux without changing much of the existing code. Hence the code was easily manageable and provided easy extensibility.

This programme worked on Windows by communicating with the *user32.dll* and *kernel32.dll* libraries. The user32.dll exposes a method called LASTINPUTINFO() which gives the information of the latest keyboard hit or mouse movement. Then we programmatically found the time since the last input and if it was above the threshold level we concluded that the system is idling and started the MOINC Agent operations.

By the use of JNA we were able to get rid of underlying operating system complexities. This made the state detection a breeze and the code we used is as follows.

```
public static void activateStateDetector() {
 MachineState state = MachineState.UNKNOWN;
 for (;;) {
  int idleSec = getIdleTimeMillisWin32() /
                                        1000;
  MachineState newState = MachineState.UNKNOWN;
  if (idleSec < 10) {
       newState = MachineState.ONLINE;
  } else if (idleSec > 10) {
       newState = MachineState.IDLE;
  }

  if (newState != state) {
       state = newState;
       Launcher launcher = Launcher.
                                  getInstance();
       launcher.launchScreenSaver(newState);
       launcher.launchAxis2(newState);
  }
  try { Thread.sleep(1000); }
  catch (Exception ex)
 }
}
```

This approach provided successful results and was used in the MOINC Agent's state detector module implementation. We encountered no problems in using this method because it could run in the background while listening for keyboard hits and mouse movements, and that is exactly what we wanted.

E. Evaluation

This section compares the three approaches that were used to implement the state detection algorithm. Four sample scenarios were created and it was checked how the three approaches classified each of the four scenarios. The four scenarios were:

1. User is idling i.e. no activity with the computer.
2. User typing text in the notepad.
3. User working with Adobe Photoshop, Windows Media Player at the same time.
4. User working with Eclipse.

The test results are as follows:

TABLE I
THE RESULTS OF THE CLASSIFICATION APPROACHES

Scena -rio	Correct Classifi -cation	JNI Approach Classifica -tion	Java awt Approach Classificat -ion	JNA Approach Classificat -ion
1	Idle	Idle	Idle	Idle
2	Active	Idle	Idle	Active
3	Active	Active	Idle	Active
4	Active	Active	Idle	Active

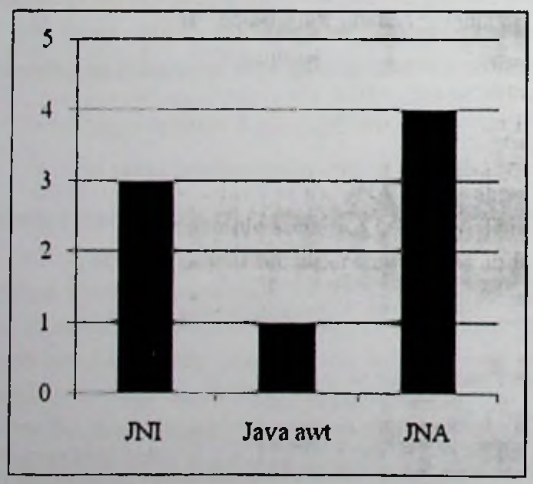

Fig. 1. Performance Comparison of the three Approaches

The first approach (JNI) failed to classify scenario 2 correctly because it could not detect light weight processes when they were running.

The second approach failed to classify the scenarios 2, 3 and 4 correctly due to the fact that it could not run in the background.

The third approach which is the JNA based approach, classified all the four sample scenarios correctly. This was due to the high accuracy of the algorithm and its ability to run as a background process.

Hence from these results and the above discussions on our research we decided to develop the state detection algorithm based on the JNA approach.

III. AXIS2 KERNEL RE-IMPLEMENTATION

A. Introduction

The topic will cover the research done on Apache Axis2 which is a Java-based implementation of both the client and server side of the web service equation. Apache Axis2 provides a complete Object model and a modular architecture that makes it easy to add functionality and support for new Web services related specifications and recommendations [14], [16]. This part can be considered as the core of MOINC Agent since Axis2 is used as the web service runtime.

After deciding to use Axis2 as our web services runtime environment of MOINC Agent, we wanted Axis2 to meet some of our requirements. We were able to achieve some of these via the axis2.xml and web.xml configuration files. To achieve the other requirements we had to dig into the code base of Axis2 and re-implement the parts we needed. MONIC Agent Requirements on Axis2 are listed bellow

1) Work with a remote repository
2) Deploy specified web services
3) Export statistical data for the screensaver & MOINC Server management module
4) Cluster enabled

B. Implementation

Work with a remote repository: MOINC Agent being an idle computer within a network and powering itself up when it is idle, will not have any available web services on the machine itself to deploy. Therefore it will require a common repository where the machine will be able to download the services and deploy them. To achieve this requirement we had to enable working with remote repository on web.xml on Axis2 [4] [5].

Deploy only the specified list of web service: This was one of the main requirements we had on Axis2. Since MOINC is a project which will be working with a grid of computers where all these computers will be pointing to a common repository. It will be impractical to download all the services in the repository and deploy it on each and every computer in the grid [5]. However if we used the default implementation of Axis2 to work with a remote repository, it will download all the services in the repository and deploy them on the local machine [15].

Therefore we had no other option other than changing the code base of Axis2. Through our research on the Axis2 source code, we saw that this functionality was done on the kernel module deployment package. We did the changes on this package so that only a specified list of services will be deployed instead of all the services in the repository. This change gave an enormous performance boost to the system and helped to avoid unnecessary network traffic. The code segment given below shows the changes we did to the loadServicesFromUrl method of the deployment engine of Axis2.

```
servicesDir = new URL(repoURL, path);
File filepath = new File("services.list");
URL filelisturl = new URL("file:///"+
                  filepath.getAbsolutePath());
ArrayList files = getFileList(filelisturl);
for (Iterator fileIterator = files.iterator();
    fileIterator.hasNext();) {
    String fileUrl = (String)
    fileIterator.next();
    log.info("File URL: " + fileUrl);

    if (fileUrl.endsWith(".aar")) {

       AxisServiceGroup serviceGroup = new
       AxisServiceGroup();

      URL servicesURL = new URL(servicesDir,
      fileUrl);

      ArrayList servicelist =
      populateService(serviceGroup,
      servicesURL,
      fileUrl.substring(0,
      fileUrl.indexOf(".aar")));

      addServiceGroup(serviceGroup, servicelist,
      servicesURL, null, axisConfig);

      log.info(Messages.getMessage(DeploymentErr
      orMsgs.DEPLOYING_WS,

      org.apache.axis2.util.Utils.getModuleName(
      serviceGroup.getServiceGroupName()),

      servicesURL.toString()
      ));
    }
```

Exporting Statistical data to screensaver & MOINC Server management module: The default Axis2 implementation does not provide any means of exporting statistical information about the services. (i.e. what services got executed and how many requests came for each service) However the MOINC Server Management Module required this information to be sent to it. Therefore we had to implement this requirement on Axis2. Since current request handling was done in the kernel module engine package we did our changes on the same and when Axis2 handled any request we captured that information and stored it in a file. The code segment given below shows one such instance where we capture the information on the engine package dispatcher phase class.

```
Axis2ServiceCount count =
Axis2ServiceCount.getInstance();
String Servicelist[] =
msgContext.getAxisService().
getFileName().toString().split("/");

for(int i=0;i <
msgContext.getAxisService().getFileName().toStri
ng().split("/").length;
        i++){
  if (Servicelist[i].endsWith(".aar")){
       count.addServiceCount(msgContext.getAxisS
       ervice().getName(), Servicelist[i]);
  }
}
```

We store this information in two formats, one where the web service file name and the request information are stored in another the service name and the number of hits each service got. The first file was used by the server management module and the other one was used to export information to be displayed on the screensaver. The structure of the file which exports information to the screensaver is given below.

```
<Screensaver>
       <Status>Deployed Web Services</Status>
       <Services>
              <Service      name      ="Version"
              count=""></Service>
              <Service       name="EchoService2"
              count=""></Service>
              <Service       name="EchoService"
              count=""></Service>
       </Services>
</Screensaver>
```

This xml file gets updated whenever a new request is handled by Axis2 so that the screensaver will display the up-to-date information to the user. The file that is used to send information to the server management module is written only once and that is when Axis2 stops.

Clustering: This was another major requirement of our project since MOINC Agent team needed the assurance that Aixs2 to will plug in to the cluster of computers without any issues. Even though Axis2 1.4.1 release did not fully supported our requirement, releases after 1.4.3 supported the clustering requirement and we only needed to enable the clustering settings on the axis2.xml file [17].

IV. SCREENSAVER IMPLEMENTATION

A. *Introduction*

Under this topic, the research that we carried out on implementing the screensaver will be discussed. The main purpose of the screensaver is to display the user, the information about what services were deployed and how many hits each service received. We started implementing the screensaver having three goals in mind. They were:

1) The screensaver should be simple.
2) It should be interactive as well as attractive.
3) We should be able to implement it without spending a lot of time.

Having these in minds we started the research work by looking at the BOINC screensaver.

B. *4.2 BOINC Screensaver*

BOINC is a non-commercial middleware system for volunteer and grid computing. It uses the idle time on computers to cure diseases, study global warming, discover pulsars, and do many other types of scientific research [18]. Due to the great similarities shown, we considered BOINC as one of the main resources that we could refer on implementing MOINC.

BOINC screensaver is implemented entirely in C++ and if you look at the source code of it you could see that there are a number of dependencies with other BOINC modules [19]. Furthermore the screensaver application needed some additional libraries to display graphics. For an example, to build the application you will need the JPEG library, a few other image libraries, and the OpenGL and GLUT graphics libraries. So it is clear that this sounds a very bulky program and moreover, though it could fulfill our first requirement of being simple, the other two goals appeared to be impossible to achieve.

So we decided to carry out more research in order to find another approach. There we took the path of blending ActionScripting with XML to come up with a simple, but attractive application that fulfils our requirements and would not take much time for implementation.

C. *4.3 MOINC Screensaver*

Keeping the attractiveness and the time factor in our minds, we tried the approach of using Actionscripting to load the data in an xml file which was generated by using the Axis2 code as mentioned in the section 3.2. XML alone may be easy. ActionScript alone may be easy. However once they were put together things became a little complicated. In other words this was a task of blending two technologies to come up with our requirement.

The following code was used to load the external service_counts.xml file in to the screensaver application.

```
function loadXML(){
        screenXML = new XML();
        da = new DataAccess();
        invoc = new Invocator();
        values = new Array();
        stat = new Array();
        screenXML.onLoad = checkStatus;
        screenXML.load ('../service_counts.xml');
        screenXML.ignoreWhite = true;
        updateAfterEvent();
}
```

In the checkStatus function, it will call two methods of the DataAccess class to get the required data fields to the two arrays.

```
function checkStatus(success){
  if(success){
        da.sendNodes(screenXML,'Status',da.handle
Status);
        da.sendNodes(screenXML,'Service',
da.handleServices);
        values = da.getServices();
        stat = da.getStatus(); }
}
```

Furthermore the sendNodes method of the DataAccess class will traverse through the xml file in order to get the values of the required fields as follows.

```
public function sendNodes(file, searchName,
callBack):Void{
for(var i = 0; i< file.childNodes.length;i++){
  var currentNode = file.childNodes[i];
    if(currentNode.nodeName == searchName){
        callBack(currentNode);}
    if(currentNode.childNodes.length > 0){
        sendNodes(currentNode,searchName,
callBack);}}
```

This approach provided successful results and was used in the MOINC Agent's screensaver implementation. Thus we were able to come with a simple but attractive screensaver in a very short period of time which certainly was our goal.

V. RESULTS

A. *Introduction*

This section presents some test results that were recorded during a large scale testing of the entire MOINC platform. In this test we increased the number of idling computers and tested the load the system can handle. By the term load we mean the number of concurrent web service requests the system successfully executed.

The test results are as follows.

TABLE II

THE NUMBER OF FAILED REQUESTS FOR EACH NUMBER OF IDLING COMPUTERS AND CONCURRENT REQUESTS. EACH CELL REPRESENTS HOW MANY REQUESTS FAILED TO EXECUTE FROM THE SENT NUMBER OF CONCURRENT REQUESTS.

Number of Concurrent Requests	Number of Idling Computers					
	1	2	4	8	16	18
1	0	0	0	0	0	0
4	0	0	0	0	0	0
8	0	0	0	0	0	0
10	0	0	0	0	0	0
12	0	6	0	0	0	0
16	16	8	0	0	0	0
20		13	10	0	0	0
24		18	16	0	0	0
28		28	21	0	1	0
30			22	1	2	0
32			24	2	2	0
36			28	19	3	0
40			40	30	5	0
44				34	5	0
48				40	6	1
50				41	6	2
52				43	7	2
60				52	8	3
70				61	16	10
80				71	46	20
90				89	80	55
100					90	88
110					101	98
120					112	106
130					121	118
140					133	130
150					141	139
160					160	147
180						164
200						191
220						220

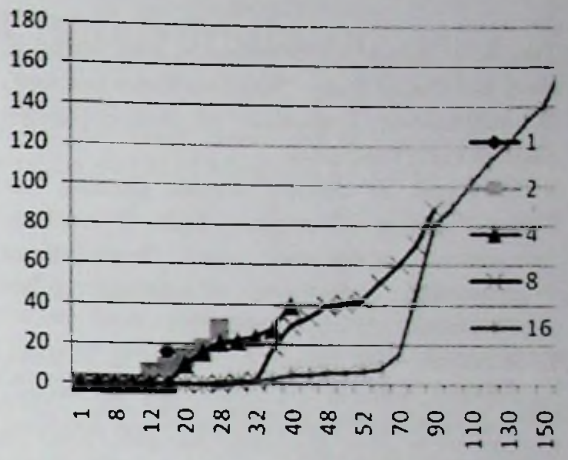

Fig. 2. The Failed Requests against the Sent Concurrent Requests

The test results indicate and represents how the load which can be handled by the system gets increased with the number of idling computers. Hence more the number of idling computers, the larger the load that can be handled by the system.

Thus it is evident that MOINC system can be used to enhance the performance and reliability of enterprise scale web service deployments.

VI. CONCLUSION

As discussed above through our research efforts we were able to conclude the fact that the JNA based approach is the most suitable and recommended method to accurately determine the current state of the computer i.e. whether it is idling or not. Through our attempts we managed to successfully modify the Kernel module of the Axis2 web service execution environment. Furthermore were able to implement a low resource consuming, attractive as well as an interactive screensaver using ActionScripting and XML, as a result of our research endeavors.

ACKNOWLEDGMENT

We would like to convey our heartfelt gratitude to quite a number of people who helped us in many ways to make this project a success. First of all we would like to thank Dr. Sanjiva Weerawarana, Chairman, WSO_2 for giving this wonderful project concept for us to work on. We would also like to thank our project supervisors, Mr. Shantha Fernando and Mr. Indika Perera for guiding us and providing us with necessary support. We would like to thank all the employees of WSO_2 as well for helping us out when we got stuck during development activities.

References

[1] Tan, A. W. Appel, S. Chakradhar, A. Raghunathan, S. Ravi, and Daniel Wang, "Safe Java Native Interface", In *International Symposium on Secure Software Engineering"*, March 2006.

[2] R M. Hirzel, R. Grimm, "Jeannie: Granting Java Native Interface Developers Their Wishes", In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications(OOPSLA)*, October 2007.

[3] L. Stepanian, A. D. Brown, A. Kielstra, G. Koblents, K. Stoodley. "Inlining Java Native Calls At Runtime", In *International Conference on Virtual Execution Environments,* June 2005

[4] S. Perera, C. Herath, J. Ekanayake, E. Chinthaka, A. Ranabahu, D. Jayasinghe, S. Werawarna, G. Daniels, "Axis2, Middleware of Next Generation Web Services", April 2006.

[5] D. Jayasinghe, *Quick start Apache Axis2*, Packt Publishing, first edition, May 2008.

[6] K.K.L Tong, *Deploying web services with Apache Axis2*, Tip Tech Development, second edition, March 2008.

[7] F. Peake, "The Hidden Treasure of Action Scripting", In *Alpha Conference,* 2006.

[8] MinGW-Minimalist GNU for Windows (2008, April) [Online] Available: http://www.mingw.org/

[9] Profiling CPU usage from within a Java Application (2008, May) [Online] Available: http://www.javaworld.com/javaworld/javaqa/2002-11/01-qa-1108-cpu.html

[10] Profiling CPU usage from within a Java Application (2008, April) [Online] Available: http://www.javaworld.com/javaworld/javaqa/2002-11/01-qa-1108-cpu.html?page=2

[11] Handling Key Press Event in Java (2008, April) [Online] Available: http://www.roseindia.net/java/example/java/awt/KeyPresK.shtml

[12] Detect the user's inactivity in Java using JNA (2008, April) [Online] Available: http://ochafik.free.fr/blog/?p=98

[13] Java Native Access (JNA) (2008, April) [Online] Available: https://jna.dev.java.net/

[14] Apache Axis2 (2008, April) [Online] Available: http://ws.apache.org/axis2/

[15] What is an Axis2 Repository? (2008, April) [Online] Available: http://wso2.org/library/tutorials/axis2-repository

[16] Axis2 Architecture Guide (2008, April) [Online] Available: http://ws.apache.org/axis2/0_94/Axis2ArchitectureGuide.html#servicearchive

[17] Enabling Apache Axis2 Clustering (2008, April) [Online] Available: http://www.swview.org/node/182

[18] Berkeley Open Infrastructure for Network Computing (2008, April) [Online] Available: http://en.wikipedia.org/wiki/Boinc

[19] BOINC Developer Trunk(2008, April) [Online] Available: http://boinc.berkeley.edu/trac/browser/trunk/boinc/clientscr/screensaver.cpp