Project no. IST-033576

XtreemOS

Integrated Project

BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

Basic services for application submission, control and checkpointing
D3.3.3
Basic service for resource selection, allocation and monitoring
D3.3.4

Due date of deliverable: November 30th, 2007

Actual submission date: December 19th, 2007

Start date of project: June 1st 2006

Type: Deliverable
WP number:3.3
Task number (optional):
Name of responsible:Toni Cortes
Editor & editorš address: Julita Corbalan. Julita.corbalan@bsc.es
Gregor Pipan . gregor.pipan@xlab.si

Version 5.0/ Toni Cortes / 19/12/2007

Project co-funded by the European Commission within the Sixth Framework Programme

Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission	
RE	Restricted to a group specified by the consortium (including the	
CO	Confidential, only for members of the consortium (including the	

Keyword List:

Revision history:

Version	Date	Authors	Institution	Sections Affected / Comments
1.0	26/09/07	Julita Corbalan	BSC	First proposal
1.1	8/11/2007	Gregor Pipan, Matej Artac Ales Černivec Eva Milošev Uroš Jovanovič	XLAB	Installation guide Resource Management Config description Ganglia Resource Allocator
3.0	12/11/2007	Julita Corbalan	BSC	Re-organization and first validation
4.0	14/11/2007	Toni Cortes	BSC	Executive summary and review
5.0	14/12/2007	Julita Corbalán	BSC	Reviews included
5.0	19/12/2007	Toni Cortes	BSC	Include conclusions and future work

Reviewers: Yvon Jegou and Luis Pablo Prieto

Task related to this deliverable

Task number	Task description	Partners involved°
T3.3.3	Basic services for job management	BSC*, XLAB
T3.3.4	Basic services for resource management	XLAB*,BSC

° This task list may not be equivalent to the list of partners contributing as authors to the deliverable

*Task leader

Merging deliverables D3.3.3 and D3.3.4

When writing the DoW for XtreemOS, before the project started, we had the intention of making a clear difference between job and resource management services. For this reason, we planned parallel deliverables to describe the advances in both resource and job management. After a year and a half of working on the application execution services, we have concluded that we can make a much better work if we design and implement both kinds of services in a more coupled way.

On one hand, we can share the same architectural infrastructure (already presented in D3.3.2 and updated in this deliverable). Sharing this part of the code has many advantages: reduce the developing time, make more stable code (it naturally gets much more tested), simplify very frequent interactions, etc.

On the other hand, and in order to take good decisions, both sets of services need to cooperate very frequently and designing and implementing them as a whole (as opposed to two different set of services) allows a much better cooperation.

For all these reasons, we have worked both set of services as a single set and thus we believe that we should deliver them as a whole, thus we have merged D3.3.3 (job management services) and D3.3.4 (resource management services) into this single document.

Executive summary

In this deliverable we present the first prototype for the Application Execution Management (AEM) services (job and resource). In this very initial prototype, we support the execution of both sequential and parallel (master/slave) jobs but always restricted to a single resource (although it can be a cluster). We can also send events to a job (currently all Linux signals). And finally, it is also possible to request information on the status of a job and of a resource.

It is important to keep in mind that the current prototype has not been integrated with the rest of prototypes and thus runs in a standalone way. Integration with other services such as VOM (Virtual Organization Manager) and ADS (Application Directory Service) will be performed in the next 6 months.

After some initial tests, we have decided to move the architecture to a staged architecture (SEDA) and the implications it has on the architecture presented on D3.3.2 are also presented in this document. It actually does not mean great changes in the components, but just on how they are implemented. In addition, we also present XATI (XtreemOS Application Toolkit Interface), the interface AEM will export. The good thing of this interface is that it is automatically generated and thus exporting functionality to the user is as simple as adding some notation to the internal call and when compiled the user library to call it will be automatically generated.

After the architecture is presented, we also give a detailed description of the installation and user guides including commands, API and some examples.

Finally, a set of appendixes describing internal tools such as the resource console (which is also delivered in this prototype but it is mostly designed for internal use), the description of a new service that has not been presented in any previous deliverable (cron), and some additional information such as the java libraries used.

Table of contents

1 Description of prototype

1.1 Job Management services

All the services provided in this prototype are designed having in mind two types of scenarios. We will refer to the first one as *shell scenario* and the second one as *application scenario*. Most of the users will work in the shell scenario, using what we call *xcommands* to submit and monitor jobs, see Figure 1. Making a parallelism with Linux, it is the traditional exec command+ps way of work. To work in this scenario, we provide a set of xcommands with the most used functionality[1]. In the current prototype we include a shell script that offers users a working environment similar to a UNIX shell, the `xconsole`. In section 5.2 some examples can be found about how to use it. The `xconsole` offers a prompt where users can execute the different xcommands existing in the prototype both for job management and for resource management.

Figure 1 shell scenario

```
%vi example.jsdl
%xsub -f example.jsdl
801929a6-9ac1-4f1b-8c18-0267958a0a4a has been submitted
%xps -j 801929a6-9ac1-4f1b-8c18-0267958a0a4a
```

The application scenario is a context where an application, for instance a SAGA[5] application, acts as an orchestrator, submitting and managing jobs, see Figure 2. For this kind of applications, we will provide a more complex and powerful API. However, in this basic prototype, a subset of the xcommands and AEM API is provided. The prototype implements the AEM architecture and API mechanisms to be used with Java applications. In next releases support for C applications will be provided.

Figure 2 Application scenario

```
#include <XATI.h>
void main(int argc,char ** agrv)
{
     int ret;
     jobID j1,j2;
     statusJob exit1,exit2;
     ret=createJob(»test1.jsdl«,&j1);
     if (ret<0)   ...
     ret=createJob(»test2.jsdl«,&j1);
     if (ret<0)   ...
     waitjob(j1,&exit1);
     waitJob(j2,&exit2);
     printStatus(exit1);printStatus(exit2);
}
```

In this document, we present the API in its current status, only with features supported. Most of them will change in the next release, supporting extended/advanced functionality.

[1] From now on, the symbol `%` will represent the prompt of the Shell.

The prototype includes submission of sequential and parallel jobs. Job submission can be done either by the AEM API (`createJob` and `runJob`) or through the specific command line (`xsub`). Jobs are described using a JSDL file [2] that includes the description of the executable and resource requirements. We only support execution of parallel jobs where the AEM starts the first process of the job and the rest is created by the first one. In the case of MPI jobs, only parallel jobs running in a single resource is supported. We will also refer to applications using the AEM API as XATI (XtreemOS Application Toolkit Interface) applications, since this is the name of the automatic framework used by AEM developers to automatically generate the AEM API.

Concerning scheduling, a first scheduling algorithm is provided, where the first resource available is selected. In the current prototype, no advanced reservations are provided, so the only criteria to select the resources will be resource requirements.

Job Submission will return a jobID with format UUID [3]. In a similar way to PID's in Linux kernels, the jobID is unique within the VO and is the handler that allows users to manage the job after the submission (job control operations).

Concerning job monitoring, basic services are provided for accessing job information. In the case of a shell scenario, we provide a `xps` command that returns information for all the jobs of a user (in a VO) or for a single job. In the current prototype, we return the jobID, the list of nodes where the job is running, the list of processes, the status of the job, submission/user/system time, and the exit status.

In the application scenario, we provide the same functionality but divided in two AEM calls: `getJobID` (returns the list of all the jobIDs of the user), and `getjobInfo` (returns the information available for the job).

Concerning job control operation, we support cancelling jobs and implicit exit of jobs. Send/receive UNIX signals is also supported.

1.2 Resource Management services

In the current prototype, most of the resource management functionality is internal to the AEM. In the advanced version we plan to export some of the functionality supported internally in the AEM API.

Current resource selection is basically a filter of attributes. Since we are using the JSDL as submission language, we support JSDL resource requirements in the current prototype and we will include some extensions to support specific XtreemOS requirements such as geographical limitations in the next release. The resource filtering process is done by WP3.2. They will provide a list of resources that fulfill job resource requirements to start the negotiation process. The list of attributes currently supported in the JSDL schema is:

- ExclusiveExecution
- OperatingSystem (Maybe this should be ignored and restricted to XtreemOS)
- CPUArchitecture
- IndividualCPUSpeed
- IndividualCPUCount
- IndividualNetworkBandwidth

- IndividualPhysicalMemory
- IndividualVirtualMemory
- TotalCPUCount
- TotalPhysicalMemory
- TotalVirtualMemory
- TotalResourceCount

In the current prototype we support three features: resource selection based on JSDL requirements, weigthed resource resource selection based on JSDL requirements and basic resource monitoring. The resource monitoring is based on Ganglia [4]. In the current prototype, we only support resources belonging to a single VO.

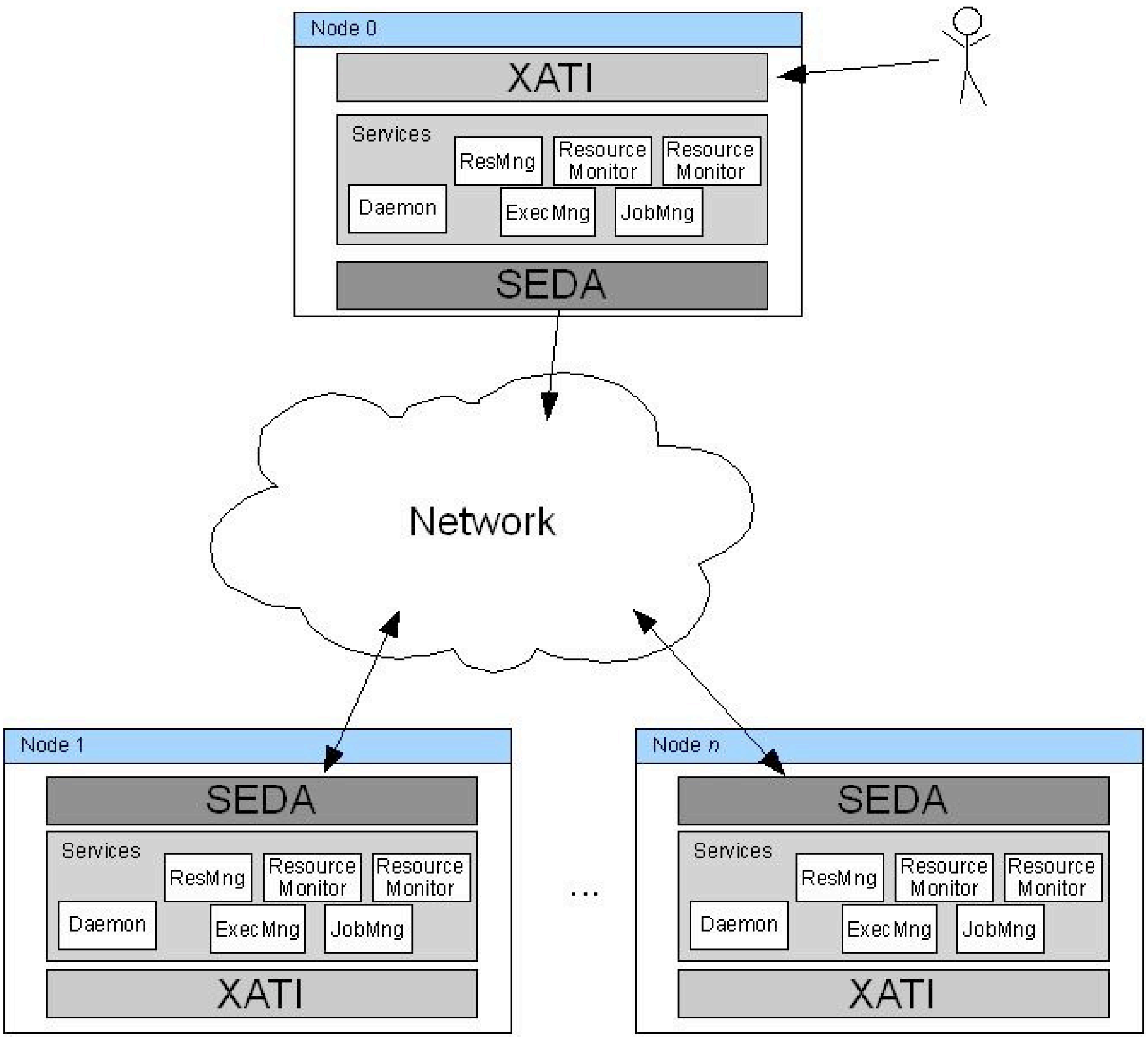
Resource allocation is supported without reservations, and no external resources managers (MAUI, SLURM) are supported. Resource allocation and enforcements is based on the `fork` and `ptrace` system calls.

2 AEM architecture

This section describes the main changes from what was described in D3.3.2 in regards message communication and service implementation. We have decided to follow the SEDA concept [1] to implement components described in the architectural document (see **¡Error! No se encuentra el origen de la referencia.**).

AEM architecture is based on the staged event driven architecture – SEDA concept [1] first published by Matt Welsh. The architecture enables us to decouple complex services into stages, and use concept of event – message passing for communication between them. In addition such architecture proves to be highly scalable, and provides good performance under high load. In a nutshell, SEDA is a set of classes, which implement event handler, which is automatically triggered as soon there are some incoming messages, and message queue, which is called by a stage, which wants to send a message to a particular stage.

Figure 3 AEM Architecture



AEM SEDA architecture is composed of following objects:

- ***Event Machine*** – is a root object, which is used to load stages, and connects queues.

- ***Message Bus Stage***– is an object, which provides communication capabilities between various stages. It parses through message header, and based on the information found, sends message to the appropriate service.
- ***Communication Stage*** – is used to send and receive messages from the network interface(s).
- ***A Service***– any service, which is handled by the SEDA server, is connected to message bus through a simple interface.

In order for AEM to run, there have to be running at least following stages: Communication Stage, and MessageBus Stage. These two stages are started by default, and can not be turned off. For each user service developed, a new stage is created as, for example, information about daemons network: Daemon (Stage). See below for more information about creating a service.

Network communication: By default Communication Stage creates an appropriate object which implements IServer interfaces (see CommunicationFactory and ServerNIO), and provides network connectivity. IServer is started in separate thread (automatically done so by CommunicationStage), and does listen to selected port. In parallel to IServer object a ISender object is created, which is used to dispatch out bound messages to network device - similar approach with factory is used here.

Event Messages: There are 4 important message types defined in the core architecture:

- *NetIncomingMessage - incoming network traffic*
- *NetOutgoingMessage - outgoing traffic*
- *ServiceMessage - to send messages between services in the daemon*
- *CallbackMessage - used to specify return path of the message*

2.1 Data flow

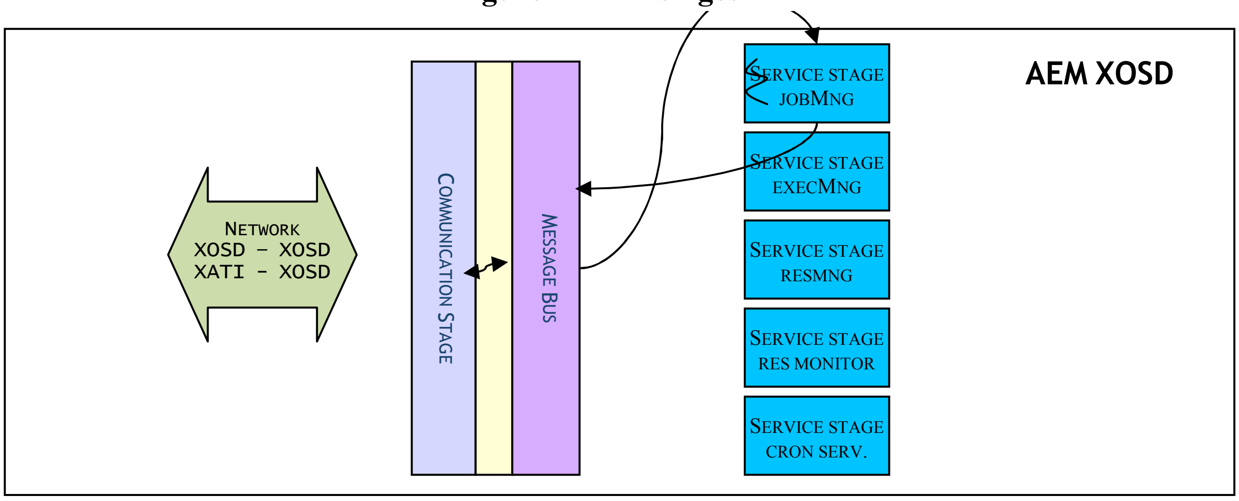
First a message is received on a network device and sent to ServerNIO. ServerNIO package call callback supplied by CommunicationStage with following information: `ByteBuffer message, SocketAddress localAddress, SocketAddress remoteAddress`. After this message is forwarded to Communication Stage, through a callback is called, the information is packaged into *NetIncomingMessage* and send to MessageBusStage. At this point the message is in the MessageBusStage queue and will be handled as one of the following kind of messages, which are accepted by MessageBusStage (*NetIncomingMessage, NetOutgoingMessage, and ServiceMessage*).

- As described earlier, where in the case of *NetIncommingMessages*, strips out the message buffer, and serialize it to ServiceMessage, which is than dispatched to the appropriate service.
- One a message is received by a Service, where the message is of *ServiceMessage* type, a switch is used to call appropriate method.
- When some information is to be send outside the daemon a *NetOutgoingMessage* is created, which embeds ServiceMessage, and possibly CallbackMessage, and then in similar (but reversed) way dispatched to CommunicationStage, which uses ISender object to send the message to network device.

2.2 AEM architecture – Service Stage

The design of the AEM is meant to be simple, but provide good and stable support for running various services (i.e. Job Manager). As SEDA objects have already been described, we shall focus on the design of the Service stage.

Figure 4 AEM stages



There are several utilities, components, which helps in the process of writing a service.

- **Abstact stages** - provides basic stage functionality, with or without input output message queues and handlers.
- **Service Processor** - generates the APIs required to allow calling the service from other services or from user code.
- **xosd** (EventMachine, CommunicationStage, MessageBusStage) are facilities, which provide access to network devices, and handle all the tedious work with creating threads, handling queuing, sending and receiving messages.

At the end of the day, the goal of the framework is, to provide developer all necessary tools, which would speed up the service development, and reduce unnecessary time spending debugging distributed architecture. Figure 4 shows some of the current stages in AEM architecture. Cron service is a new service used to schedule the execution of any code, see Appendix for more details.

2.3 Creating a new service

Service by definition has to implement IStage interface, which allows the system to be handled by event machine. For the developer convenience we have provided several abstracted version of the stages, which already implement some common functionality.

- **AbstractStage** - provides basic thread handling
- **AbstractReceivingStage** - extends AbstractStage, and provides facilities to receive requests (incoming queue, abstract event handler for the incoming messages). Incoming queue is, when used in xosd automatically connected to MessageBus Stage, which will automatically forward all received messages addressed to this service.
- **Abstract2wayStage** - extends AbstractReceivingStage with sending capabilities, where the outgoing queue (named sink), is linked to MessageBus Stage (MessageBus Stage will receive all the messages, send by this stage).

Implementation of a service:

The Service skeleton, see Figure 5, does implement basic service, which can be included in the xosd daemon

Figure 5 Example of simple xos service

```
@XOSDGLOBALSERVICE
public class Daemon extends Abstract2wayStage {

     public Daemon(){
            // this constructor provides name to the
thread
            super(getShortName(Daemon.class.getName()));
     }

     public void init(){
            //TODO: provide initialisation for the
service
     }


     @XOSDXATICALL
     public ArrayList<CommunicationAddress> getDaemons(){
            return list;
     }

     public void handleEvent(Object event) throws
Exception{}
     public String getHandledEventType(){return "";}

}
```

First of all, Daemon class extends one of the provided abstract stages, which gives it various capabilities. Additionally we have to tag the whole class with "`@XOSDGLOBALSERVICE`", which will generate following code:

- `'pwd'/service/${Name}Handler.java` Linkage code to the MessageBusStage
- `'pwd'/service/${Name}CB.java` Methods to create specialised CallbackMessages
- `eu/xtreemos/xati/API/${Name}.java` XATI interface for client program developer convenience

Second important information which the developer has to provide is information of the method visibility.

XATI access: In order for XATI to access a function in the service a method has to be annotated with "`@XOSDXATICALL`" (see Figure 5). After completing the service you have first to run service processor, which will generate the aforementioned files, which will allow writing all the code, which calls these services.

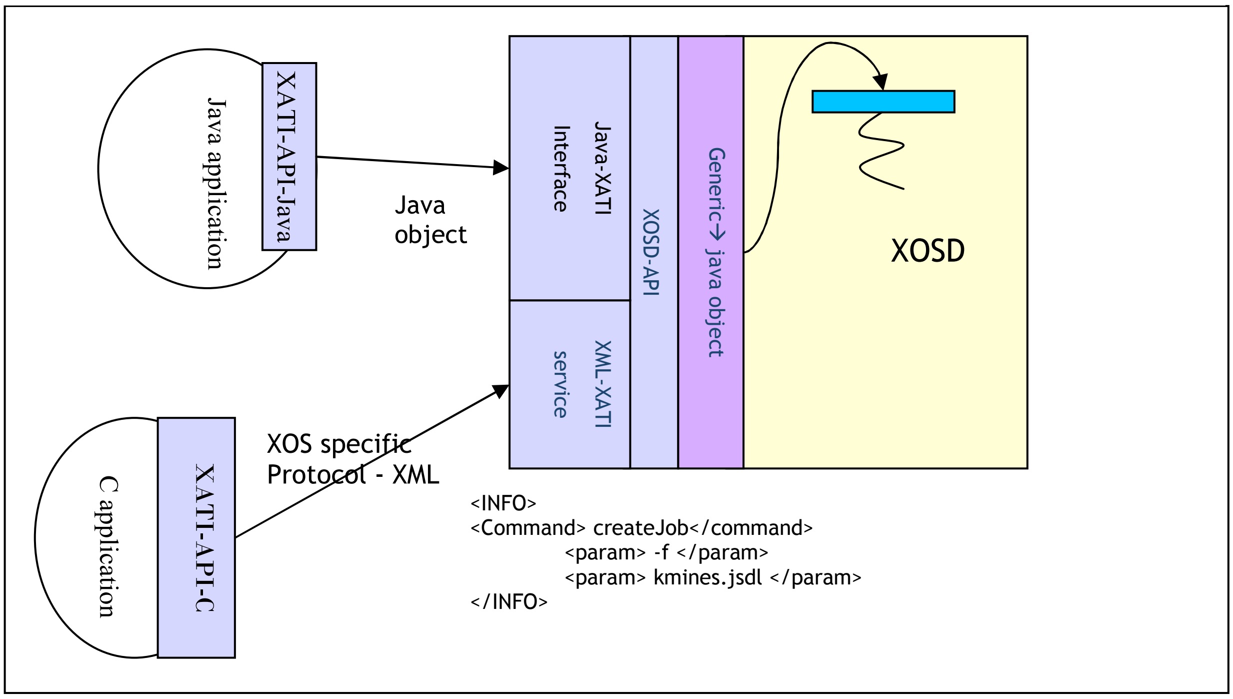
2.4 Linking External Services to the AEM

The AEM services depend also on several external services – i.e. directory service or resource selection service provided by WP3.2. The access to these services is provided through proxy services, which decouple the AEM code from an API provided by a specific service. Integration with external services will start after month 18.

3 XATI: XtreemOS Application Toolkit Interface

AEM API is automatically generated by the XATI framework developed in the context of WP3.3. XATI framework will provide an easy access to the public part of the AEM functionality, where currently we foresee supporting XATI for Java and C applications. At the current time, it is only available for the java API, however, we plan to have it accessible also for C code. General XATI architecture is depicted on Figure 6:

Figure 6 XATI application → AEM interface



```
<INFO>
<Command> createJob</command>
          <param> -f </param>
          <param> kmines.jsdl </param>
</INFO>
```

First important feature of XATI is that it is capable of operating in two distinct modes. In the first mode, XATI is composed of a Java client side, which is capable of wrapping XATI calls to native XOSD format, where on the other hand we have C applications, which do not have capability to create Java objects, and therefore communicate with XOSD through a XML formatted messages. This distinction is presented in the figure above. In this prototype we will only provide Java interface, however, the same mechanism is suitable for any other language.

3.1 Writing XATI applications

As we have mentioned before service deployed in XOSD can easily annotate methods as XATI methods, which will generate XATI client code, which is than called by the Java client. An application that wants to communicate with XOSD, only has to call appropriate methods, and all the processing is done in the background, see Figure 7.

Figure 7 Example: Using XATI to call AEM XOSD

```
public class XATItest {
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		ArrayList resourceList = 
(ArrayList)ResMng.getAllResources();
		System.out.println(resourceList);
	}
}
```

Developer is provided the whole object model, which enables compile time check on the calls to the XOSD. In section 5 we present main XATI methods offered with this prototype.

4 Installation and configuration guide

This is a description of how to install the AEM code and to have it run. Installation instructions for various Linux distribution might vary (especially for system paths or system variables), but based on these instructions it should be easy to adapt to other distributions. Steps to have a running AEM environment are:

- Configure the system environment
- Install AEM XOSD java code, either from the INRIA svn or as eclipse project
- Compile C libraries used in Execution Manager Service.
- Install Ganglia monitoring framework used by the Resource Manager service.
- Run the AEM XOSD daemon

4.1 System Environment

- Linux
- Java Virtual Machine (v6), see Appendix – Java6
- Eclipse (v3.2)
- gcc (v4)
- subversion client
- additional java libraries see Appendix – External Libraries

4.2 Installing AEM from svn

- Create a folder under your home directory `(~/wp3.3`) for AEM to be checked out.
- Get AEM source code from the inria svn server:
```
%svn checkout
%svn+ssh://<user>@svn.gforge.inria.fr/svn/xtreemos/WP3.3/xOS/trunk
```
- This results into the folder `xOS-Code` being created with subfolders:

The checkout should create at least following folders in your directory:

- **AllocationMng** - Allocation Manager, a service
- **CDAMng** - an interface service to access CDA functionality developed by 3.5
- **Daemons** - a temp service, which stores all the currently connected XOSDs
- **JobMng** - Job manager
- **ResMng** - Resource Manager
- **ResourceMonitor** - a Monitoring service, which collects information from ganglia and transforms it to GLUE schema[2][6]. Resource manager uses this information to provide dynamic information about the resources

[2] The GLUE Schema is an abstract modeling for Grid resources and mapping to concrete schemas that can be used in Grid Information Services

- **Support** - jars needed to compile the projects
- **VOPS** - VO Policy Service - mostly part of the 3.5 functionality.
- **XML_access** - this will be merged with XCLIService, and provides parsing functionality to decode the XML.
- **XMLExtractor** - a low level XML parsing service, which is capable of transforming JSDL, GLUE, and other XML schema based XML files (probably policy language, ...)
- **XOS_SEDA** - Staged Event Driven Architecture - core daemon framework, which is capable of dynamically load any service. It also includes network communication layer.
- **XOSd** - starting of the daemon - configuration
- **XOSdCommon** - some XML parsing functionality, which will be integrated to some other project
- **XServiceProcessor** - generator of the Automatically generated code, includes configuration class generator

In the SVN are also some additional projects, which are result of the code generation process,

- **XATI** - Interface for client side applications.
- **XOS_Services** - Interface for communication service-2-service
- **Config files** see appendix

4.3 Installing AEM as Eclipse Project

- Ensure java compiler of version 6 is used! (Window→Preferences→Java→Compiler). (See section 4.6 for Java v6 installation)
- Each of the points mentioned before creates one Java project. To import AEM source code into new Eclipse projects:
  - Go to “new project” wizard,
  - select SVN project type (subclipse required for performing check out directly in Eclipse)
  - Enter aforementioned SVN URL into the repository path,
  - Select projects you want to CO,
  - Finish.
  - In the Workspace you have to define a user library XOSLIBS, which includes all the jars provided in the “Support” subfolder.

4.4 Installation of C-Libraries

The Execution Manager uses some C code for low level process management: process creation, monitoring, etc. To use it, you have to compile it.

```
%cd ~/wp3.3/xOSCode/ExecMng/JNI
%gcc XExecMng.c -o XExecMng.so -shared -I $JAVA_HOME/include/
     -I $JAVA_HOME/include/linux/ -o libXExecMng.so
```

4.5 Installation of Ganglia Monitoring Framework

Ganglia[4] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. We use Ganglia as the basis for resource monitoring.

Installing Ganglia is elementary. In some Linux distribution, you can simply find the Ganglia package in your package manager of choice (YAST, Synaptic,...). If not, you can download the source (currently `ganglia-3.0.5.tar.gz`) from http://sourceforge.net/project/showfiles.php?group_id=43021&package_id=35280 or follow the download link on http://www.ganglia.info. In the case of package install, there is nothing to do but click OK.

Installation from source amounts to:

```
 % ./configure
 % make
 % make install
```

It is not necessary to install `gmetad`, only `gmond` (= Ganglia Monitoring daemon) for ResourceManager to work. Once there is a `gmond` binary in the `/usr/sbin` directory, the `gmond` is started by typing

```
 % ./gmond
```

Installation can be tested by typing

```
 % telnet localhost 8649
```

If the installation was successful, and gmond is running, you should see an XML with the description of the monitored machine. For more detailed instructions (see Appendix Ganglia-Installation Guide).

4.6 Installation of Java v6

Java 6 is used instead of Java 5. We recommend using SUN Java to be used to run AEM modules.

Manual installation: Get the JDK from http://java.sun.com/javase/downloads/index.jsp (`jdk-6u2-linux-i586.bin`) and execute it. Copy resulting folder to `/usr/lib/jvm`. Set links to new java, javah, javac:

```
%cd /us r/bin/
%ln -s /usr/lib/jvm/jdk1.6.0_02/bin/java java
%ln -s /usr/lib/jvm/jdk1.6.0_02/bin/javac javac
%ln -s /usr/lib/jvm/jdk1.6.0_02/bin/javah javah
Enter bash.bashrc and export path to new jdk:
%cd /etc
%export JAVA_HOME=/usr/lib/jvm/jdk1.6.0_02
```

4.7 Running XOS daemon - configuration

When running the AEM XOSD server, a configuration file to initialize daemons variables can be provided:

- `rootaddress` which is comprised of IP address/port.
- `services` is list of strings, which enables user to choose which stages daemons architecture will load at start-up.

User can not change IP (*rootaddress.ip*) because it is bounded to the local machine where daemon with services is started, meanwhile *rootaddress.port* property can be an arbitrary value. *rootaddress.port* property is passed into *CommunicationStage* class where server is started on provided port value.

If user does not provide configuration file, a default file is created in `daemons` home directory. Values of default variables which will be written in newly created file are set automatically. Listing of automatically created `XOSDConfig.conf` is shown in Figure 8:

Figure 8 AEM XOSD configuration file

```
#Properties File for the client application
#Thu Nov 15 11:14:32 CET 2007
rootaddress.host=/84.88.50.104
services.8=eu.xtreemos.service.test.service.TestHandler
services.7=eu.xtreemos.xosd.execMng.service.ExecMngHandler
services.6=eu.xtreemos.xosd.jobDirectory.service.JobDirectoryHandler
services.5=eu.xtreemos.xosd.jobmng.service.JobMngHandler
rootaddress.port=60000
services.3=eu.xtreemos.xosd.resourcemonitor.service.ResourceMonitorHandl
er
services.2=eu.xtreemos.xosd.xmlextractor.service.XMLExtractorHandler
services.1=eu.xtreemos.xosd.resmng.service.ResMngHandler
services.size=10
services.0=eu.xtreemos.xosd.daemon.DaemonGlobal
xosdRootDir=/home/martag/workspace-xos
```

If user wants to run just some of these stages, he/she should remove lines which contain services he does not want to run (numbering order of services is not restricted, but should be less than *services.size* property).

4.8 How to run XOS daemon?

Script for running XOS Daemon is located in `Support/runscript` folder. Execute the following line in a bash shell:

```
$ xosd_run.sh
```

Daemon is now running. In `Support/runscript` new files are created:

- `support_bin_dirs` (list of root binaries of all projects)
- `support_jars` (list of jars from Support project)
- `XOSdConfig.conf` (configuration file, see D3.3.3 deliverable for description; documentation of configuration classes are in `Support/Documentation/configGenerator.txt`)

These files are used by `xosd_run` script to run java class XOSd with `XOSdConfig.conf` configuration file (if something goes wrong, user can cat these files to see if all paths are correct). It is important that developer has directory structure of projects like on AEMs repository.

5 User guide

5.1 Xcommands and XATI methods

Table 1 and Table 2 make a brief description of xcommands and XATI methods supported in the current prototype.

Table 1 Xcommands supported

Xcommand	Description
`Xsub -f jobdefinition.jsdl`	In the current prototype, we provide a command, `xsub`, for job submission. It receives a JSDL file name as parameter. The `xsub` command creates and runs the job. It waits until the job id is returned and prints it.

`Xps -j jobID`
 	
`Xps -a`
	-Shows the info for the job with id jobID
	-Shows the info for all the submitted jobs

`Xwait jobID`	It waits for the finalizations of a specific job specified by jobID
`Xkill event jobID`	Sends the specified event (currently UNIX signals) to the jobID

`xrs -a`
 	
`xrs -f jsdl_file`
 	
`xrs -cpu architectureName -numcpu cpuCountInterval -ghz cpuSpeedIntervalGHz -os osName -ram RAMSizeIntervalGB`
	-Shows all the resources available
	
	-Shows the list of resources that fits into the JSDL requirements
	
	- Shows the list of resources that fits the query expressed in the command line. At least one of the cpu, numcpu, ghz, os or ram parameters has to be provided. The values of cpuCountInterval, cpuSpeedInterval and RAMSizeInterval can be either a single value (e.g. 1.5) or an interval (e.g. 1.5-3). For example:
	xrs -cpu x86 -os Linux -ram 1-3
	selects all resources with x86 architecture, Linux operating system and with physical memory between 1 and 3 GB

`xmonctr -node address -m`
 	
`xmonctr -node address -i`
	- Shows the list of attributes that are being monitored at the node. The address has the form of IP:port. The port can be obtained using the `xrs -a` command.
	
	- Shows the details on the node by listing the values of the monitored resource attributes. The address has the form of IP:port

Table 2 XATI methods supported

Scope	XATI call	Description
Job Submission	`createJob(String jsdlFilePath, boolean run)`	Creates a job based on a job definition (jsdlFilePath parameter). If run is true, the job is created and submitted for execution. Otherwise, it is only created.
	`runJob(String jobId)`	Starts the execution of the jobId
Job Control	`jobControl(String jobID, int operation)`	Only CANCEL operation is implemented
Job Events	`sendEvent(String jobId, int event)`	Send the specified event (only UNIX signals) to the jobID
Job Monitoring	`getJobIDs()`	Returns the list of jobIDs of jobs existing in the system
	`getJobInfo(String jobID)`	Returns the attributes associated with the jobID
Resource Management	`getResources(String jsdlQuery)`	Returns the resources that match the provided resource query.
Resource Monitoring	`getResInfo(CommunicationAddress nodeAddress)`	Returns the information on the nodeAddress. The returned value consists of the resource attributes currently selected and their values.
	`getResMetrics(CommunicationAddress nodeAddress)`	Retrieves a list of the node's attributes that are currently being monitored.

5.2 Job Submission

- Start the AEM**:**
```
%cd ~/Support/runscript/
%./xosd_run.sh
%./xconsole.sh
```
- This results into the following output, the Xconsole coming up:
```
XTreemOS Console
Version 1.0
$
```
- Once started the Xconsole, you will get a prompt where xcommands can be executed. Now you can submit a job for execution using xsub command (see Figure 9 for a JSDL example) :
```
%xsub -f tests/kmines.jsdl
Job submitted succesfully: 53c92b29-9a58-4620-b7ab-2dc6d363de35
```

Figure 9 Example of JSDL file. File `kmines.jsdl` has to be saved under `~/workspace/xOS/tests/`

```
<?xml version="1.0" encoding="UTF-8"?>
    <JobDefinition xmlns="http://schemas.ggf.org/jsdl/2005/10/jsdl">
        <JobDescription>
            <JobIdentification>
                <Description>Execution of Kmines KDE game</Description>
                <JobProject>BSC_Test</JobProject>
            </JobIdentification>
            <Application>
                <POSIXApplication
xmlns:ns1="http://schemas.ggf.org/jsdl/2005/06/jsdlposix">
                    <Executable>/usr/games/kmines</Executable>
                    <Output>out_kmines.txt</Output>
                    <Error>err_kmines.txt</Error>
                </POSIXApplication>
            </Application>
        </JobDescription>
    </JobDefinition>
</JobDefinition>
```

- The output confirms a job has been submitted successfully.
- In order to display the process being in the system issue xps command on the Xconsole (`xps -a` to show all the processes).
- Output is:

```
$xps -a
JOB ID      -  COMMAND -      JOB STATE   * RESOURCE ADDR:PORT
           + PID    - USER TIME - SYS TIME - PROC STATE

53c92b29-9a58-4620-b7ab-2dc6d363de35 -/usr/games/kmines -           Done
95f6ad50-b918-4b71-8795-18ab78efe67a -office -LocalSubmited * 84.88.50.104:60000
          +   9164 - 00:01.29 - 00:00.13 -          S
          +   9149 - 00:00.00 - 00:00.00 -          S
```

5.3 Resource management

In the same xconsole, you can type

```
$ xrs -a
  Address = [:///192.168.0.178:60000]
  Address = [:///192.168.0.219:60000]
```

The output shows two resources. To get the details on the resources we first check which attributes are being at one of the resources.

```
$ xmonctr -node 192.168.0.178:6000 -m
    operatingSystemName
    processorArchitecture
    CPUCount
    RAMSize
```

We then query the details on each of the nodes.

```
$ xmonctr -node 192.168.0.178:60000 -i
[hostIP={Address = [:///192.168.0.178:60000]}, hostUniqueID={localhost},
operatingSystemName={Linux}, processorArchitecture={x86}, CPUCount={1.0},
RAMSize={2.146435072E9}]

$ xmonctr -node 192.168.0.219:60000 -i
```

```
[hostIP={Address = [:///192.168.0.219:60000]}, hostUniqueID={localhost},
operatingSystemName={Linux}, processorArchitecture={x86}, CPUCount={1.0},
RAMSize={7.43440384E8}]
```

If we would like to select the resource with physical RAM between 1 GHz and 4 GHz, we invoke:

```
$ xrs -ram 1-4
  Address = [:///192.168.0.178:60000]
```

6 Conclusions and future work

In this deliverable we have presented the first prototype for the Application Execution Management (AEM) services (job and resource). In this very initial prototype, we support the execution of both sequential and parallel (master/slave) jobs but always restricted to a single resource (although it can be a cluster). We can also send events to a job (currently all Linux signals). And finally, it is also possible to request information on the status of a job and of a resource.

It is important to keep in mind that the current prototype has not been integrated with the rest of prototypes and thus runs in a standalone way. Integration with other services such as VOM (Virtual Organization Manager) and ADS (Application Directory Service) will be performed in the next 6 months.

In addition, in the next 6 months we plan to work on the following issues: porting of the XATI interface to C (including the supported commands in a standalone – not Java – version), start developing very basic reservation and SLA mechanisms, improving the semantics to send events, and start defining the monitoring and negotiation architectures,

7 Bibliography

[1] Matt Welsh, SEDA: An Architecture for Highly Concurrent Server Applications

URL: http://www.eecs.harvard.edu/~mdw/proj/seda/

[2] JSDL specifications. http://www.ggf.org/documents/GFD.56.pdf

[3] UUID . http://en.wikipedia.org/wiki/UUID

[4] Ganglia. http://ganglia.sourceforge.net/

[5] T.Goodale, S.Jha, H.Kaiser, T.Kielmann, P.Kleijer, G.von Laszewskik, C. Lee, A.Merzky, H.Rajic, J.Shalf. SAGA: A Simple API for Grid Applications. High-Level Application Programming on the Grid. http://wiki.cct.lsu.edu/saga/

[6] GLUE schema. **http://glueschema.forge.cnaf.infn.it/**

Appendix – Resource Management Console

XResourceConsole is an utility that enables the access to the functionality of the Resource Manager service and other relevant services such as the Resource Monitor service. The utility offers a command line interface in either an interactive mode or through command line parameters. Using this utility, we can manually test any new functionality we introduce for resource management and monitoring. We include its description as appendix because it is included in the code prototype.

Name

`xrc` – XResourceConsole, a utility for managing the resources from the console.

Synopsis

```
 xrc [OPTION]...
```

Options

`-I`	Run in an interactive mode.
`-l`	List all available node addresses.
`-s JSDL_FILENAME`	List nodes selected by a resource query. Use the file with name and/or path JSDL_FILENAME as an input for the resource query. The file has to contain an XML following a JSDL schema.

Examples

Interactive mode

Interactive mode of XResourceConsole consists of a set of menus that let the user form a resource query and test it live with the currently available computation nodes. The following menus are available:

- Main menu
    - Query editing
        - String constant selection
        - Numerical range selection
        - Upper bound selection
        - Lower bound selection
        - Range selection
        - Exact value selection
    - Attribute selection for ordering the results
    - Attribute weight selection for scoring the query hits

Main menu

The main menu has the following options:

```
k  edit current query
n  create new query
a  list all nodes
s  list selected nodes (using current query)
o  list ordered selected nodes (using current query)
w  list selected nodes (using current query) ordered by hit scores
m  print this menu
q  quit
```

Entering the letter next to an option enters a submenu or performs the selected option. The options in this menu are as follows:

- Edit current query – enters a menu for managing the current resource query.
- Create new query – clears the current resource query and enters a menu for managing the query.
- List all nodes – calls the Resource Manager to obtain all the available nodes, and lists their addresses. Stores the results as the current list of nodes.
- List selected nodes – uses the current query to call the Resource Manager service to obtain the list of nodes, selected by the query. Stores the results as the current list of nodes.
- List ordered selected nodes – uses the current query to call the Resource Manager service to obtain the list of nodes, selected by the query and ordered by attributes as defined by the current ordering. Displays the list and stores the results as the current list of nodes.
- List selected nodes ordered by hit scores - uses the current query to call the Resource Manager service to obtain the list of nodes, selected by the query. The results are ordered by the hit scores in the descending order. Displays the list and stores the results as the current list of nodes.
- Print this menu – prints again the options of the menu.
- Quit – exits the XResourceConsole.

Query editing

The menu for editing the resource query enables an interactive way to add the query entries. The query is structured as an XML that follows a JSDL schema, and the menu reflects the schema's structures. The substructure relevant to the node selection is the Resource structures with entries that represent either string types or JSDLš RangeValue_Type structure. The menu enables entering the queries for the resource attributes currently supported by the Resource Manager service.

When we enter the menu, we get the following options:

```
p  print the current query
o  select OS
i  select CPU instruction set
c  select CPU clock speed
n  select the number of CPUs
r  select the RAM size
m  print this menu
x  exit this menu
```

Entering the letter next to an option enters a submenu or performs the selected option. The options in this menu are as follows:

- Print the current query – displays the XML structure currently entered by the user.

- Select OS – enters the string constant selection menu for selecting the operating system name required by the resource query.
- Select CPU instruction set – enters the string constant selection menu for selecting the instruction set required by the resource query.
- Select CPU clock speed – enters the numerical range selection menu where the user can select acceptable values for the CPUš clock speed. The values entered are in GHz.
- Select the number of CPUs – enters the numerical range selection menu where the user can select acceptable values for the number of physical CPUs or the available cores.
- Select the RAM size – enters the numerical range selection menu where the user can select acceptable values for the physical memory size. The values entered are in MHz.
- Print this menu – prints again the options of the menu.
- Exit this menu – returns to the previous menu.
- Attribute selection for ordering the results

This menu lets the user enter the sequence of resource attributes which the nodes will be sorted by. The first attribute in the sequence will define the primary ordering of the results. The nodes that have equal values of the first attribute are then sorted by the second attribute in the sequence, etc. In this menu the user can also select the direction of the ordering (ascending or descending).

The menu has the following options:

```
p  print the current attribute sequence
z  restart the ordering attribute sequence
o  append attribute: OS name, ascending
O  append attribute: OS name, descending
i  append attribute: CPU instruction set, ascending
I  append attribute: CPU instruction set, descending
c  append attribute: CPU clock speed, ascending
C  append attribute: CPU clock speed, descending
n  append attribute: number of CPUs, ascending
N  append attribute: number of CPUs, descending
r  append attribute: RAM size, ascending
R  append attribute: RAM size, ascending
m  print this menu
x  proceed
```

Entering the letter next to an option enters a submenu or performs the selected option. The selection letters are case-sensitive. The options in this menu are as follows:

- Print the current attribute sequence – displays in the console the sequence of attributes entered by the user so far.
- Restart the ordering attribute sequence – clears the sequence of attributes entered by the user so far, letting the user to enter a new sequence.
- Append attribute: <attribute name>, ascending – the next attribute in the sequence will be the one denoted by the attribute name, and the ordering will be ascending (e.g., 0, 1, 2, ...).
- Append attribute: <attribute name>, descending – the next attribute in the sequence will be the one denoted by the attribute name, and the ordering will be descending (e.g., 42, 41, 40, ...).
- Print this menu – prints again the options of the menu.
- Proceed – confirm the current sequence and proceed with the workflow.

Attribute weight selection for scoring the query hits

The menu for selecting the attribute weights lets the user to select the importance of the resource attributes as they take part in the resource query. Each attributeš weight is a value from the interval [0..1], where a higher value means a higher importance. The sum of all attributes' weights has to equal 1.0.

The menu presents us with the following options:

```
  p  print the current attribute weights
  z  reset to uniform weights
  o  set weight for OS name
  i  set weight for CPU instruction set
  c  set weight for CPU clock speed
  n  set weight for number of CPUs
  r  set weight for RAM size
  x  proceed with selected weights
  u  proceed without selected weights (uniform weights)
```

Entering the letter next to an option enters a submenu or performs the selected option. The options in this menu are as follows:

- Print the current attribute weights – displays in the console the currently entered attribute weights.
- Reset to uniform weights – sets all attributes' weights to identical values.
- Set weight for <attribute name> - assign a weight for the attribute with attribute name.
- Proceed with selected weights – confirm the weights and use them in the proceeding with the workflow.
- Proceed without selected weights – proceeds with the workflow, but without the weights currently assigned to the attributes. This effectively works as if the attributes that take part in the resource query are set to an equal non-zero weight, while the attributes not in the query have zero weights.

Usage example

The following example shows a possible usage of XResourceConsole. We first start the utility in the interactive mode. We start by listing all available nodes.

```
  k  edit current query
  n  create new query
  a  list all nodes
  s  list selected nodes (using current query)
  o  list ordered selected nodes (using current query)
  w  list selected nodes (using current query) ordered by hit scores
  m  print this menu
  q  quit

 # a
 Getting resources
 Found the following resources:
  - Address = [:///192.168.0.178:60000]
  - Address = [:///192.168.0.190:60000]
```

The call resulted in a list containing two nodes, one with 2 GB RAM and a single 3 GHz CPU, and another one, an old computer with 256 MB RAM and 900 MHz CPU clock. We proceed with creating a query that will filter out the slower computer.

```
  k  edit current query
  n  create new query
  a  list all nodes
```

```
 s  list selected nodes (using current query)
 o  list ordered selected nodes (using current query)
 w  list selected nodes (using current query) ordered by hit scores
 m  print this menu
 q  quit

# n
```

This puts us in a menu where we can select the attributes to be used in the query.

```
Edit query menu:
 p  print the current query
 o  select OS
 i  select CPU instruction set
 c  select CPU clock speed
 n  select the number of CPUs
 r  select the RAM size
 m  print this menu
 x  exit this menu
```

We would like the selected node to use an x86 CPU architecture.

```
# i
CPU Instruction Set selection
 0  sparc
 1  powerpc
 2  x86
 3  x86_32
 4  x86_64
 5  parisc
 6  mips
 7  ia64
 8  arm
 9  other
Please type the number next to the selected architecture: 2

Edit query menu:
 p  print the current query
 o  select OS
 i  select CPU instruction set
 c  select CPU clock speed
 n  select the number of CPUs
 r  select the RAM size
 m  print this menu
 x  exit this menu
```

We want the selected node to have the CPU clock higher than 1 GHz and lower or equal to 5 GHz.

```
# c
Menu for Range_Type of the CPU speed [GHz]
 u  upper bound
 l  lower bound
 r  range
 e  exact value
 m  print the menu
 x  exit menu

# r
Range entry
Lower bound: 1
Exclusive lower bound: [y/n] y
Upper bound: 5
Exclusive upper bound: [y/n] n
```

The exclusive bound modifier sets whether the boundary value should be within acceptable values (exclusive bound **y**) or not (exclusive bound **n**).

```
Menu for Range_Type of the CPU speed [GHz]
```

```
 u  upper bound
 l  lower bound
 r  range
 e  exact value
 m  print the menu
 x  exit menu
```

Next, we would like to set the condition for the memory size resource attribute.

```
# x
Left current menu.
Edit query menu:
 p  print the current query
 o  select OS
 i  select CPU instruction set
 c  select CPU clock speed
 n  select the number of CPUs
 r  select the RAM size
 m  print this menu
 x  exit this menu

# r
Menu for Range_Type of the RAM size [GB]
 u  upper bound
 l  lower bound
 r  range
 e  exact value
 m  print the menu
 x  exit menu
```

We need the computer to have at least 512 MB of physical memory.

```
# l
Range entry
Lower bound: 0.5
Exclusive lower bound: [y/n] n
Menu for Range_Type of the RAM size [GB]
 u  upper bound
 l  lower bound
 r  range
 e  exact value
 m  print the menu
 x  exit menu

# x
Left current menu.
Edit query menu:
 p  print the current query
 o  select OS
 i  select CPU instruction set
 c  select CPU clock speed
 n  select the number of CPUs
 r  select the RAM size
 m  print this menu
 x  exit this menu
```

We display the XML that forms the query we have built.

```
# p
Printing the current query:


Done printing the current query.

Edit query menu:
 p  print the current query
 o  select OS
 i  select CPU instruction set
 c  select CPU clock speed
 n  select the number of
CPUs
```

```
<?xml version="1.0" encoding="UTF-8"?>
<JobDefinition xmlns="http://www.example.org/"
            xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
            xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<JobDescription>
     <JobIdentification>
            <JobName>Interactive Test Job</JobName>
     </JobIdentification>
     <Application>
            <POSIXApplication>
                   <Executable>/usr/temp/test</Executable>
            </POSIXApplication>
     </Application>
     <Resources>
            <IndividualPhysicalMemory>
                   <Range>
                          <LowerBound>5.36870912E8</LowerBound>
                   </Range>
            </IndividualPhysicalMemory>
            <IndividualCPUSpeed>
                   <Range>
                          <LowerBound
exclusiveBound="true">1.073741824E9</LowerBound>
                          <UpperBound>5.36870912E9</UpperBound>
                   </Range>
            </IndividualCPUSpeed>
            <CPUArchitecture>
                   <CPUArchitectureName>
                          x86
                   </CPUArchitectureName>
            </CPUArchitecture>
     </Resources>
</JobDescription>
</JobDefinition>
```

```
 r  select the RAM size
 m  print this menu
 x  exit this menu

# x
Left current menu.
 k  edit current query
 n  create new query
 a  list all nodes
 s  list selected nodes (using current query)
 o  list ordered selected nodes (using current query)
 w  list selected nodes (using current query) ordered by hit scores
 m  print this menu
 q  quit
```

Finally, we list the nodes selected by our resource query.

```
# s
Getting selected resources
Found the following resources:
 - Address = [:///192.168.0.178:60000]
```

To quit the program, we enter **q**.

```
 k  edit current query
 n  create new query
 a  list all nodes
```

```
  s  list selected nodes (using current query)
  o  list ordered selected nodes (using current query)
  w  list selected nodes (using current query) ordered by hit scores
  m  print this menu
  q  quit

# q
```

Appendix – External Libraries

List of the libraries used in the AEM:

axis2-kernel-1.3.jar	Used to retrieve parameter names of methods in XServiceProcessor
djep-1.0.0.jar	(deprecated) The DJep collection of packages offers a number of extensions to the standard JEP package (library for expression evaluation)
commons-logging-api-1.1.jar	Logging facility API
junit-4.4.jar	JUnit testing framework
vivaldiLib.jar	Library for Vivaldi support
mina-core-1.1.2.jar	MINA core communication library
slf4j-api-1.4.3.jar	Simple Logging facility
velocity-1.5.jar	Velocity library for code generation (XServiceProcessor)
commons-logging-1.1.jar	Logging core facility
log4j-1.2.14.jar	Apache logging facility
commons-logging-adapters-1.1.jar	Apaches common logging, provides thin-wrapper Log implementations for log4j
vsuite-findbugs-0.4b.jar	(deprecated) VSuite is a project that aims at finding and reporting problems in your Java code
dom4j-1.6.1.jar	XML parsing framework
bcprov-jdk16-137.jar	The Bouncy Castle Crypto APIs for Java
tools.jar	Common tools classes used in AEM (XServiceProcessor)
java_cup-0.10k.jar	Parsing tool
sunxacml.jar	XACML support (VOPS)
velocity-dep-1.5.jar	Velocity library dependencies
bamboo.jar	Bamboo - DHT implementation
oncrpc.jar	Used by Bamboo API
slf4j-log4j12-1.4.3.jar	Logging facility
jep-2.3.0.jar	Java Expression Parser
mina-filter-ssl-1.1.3.jar	SSL filter for SSL support in MINA communication library

Appendix – Cron Service

CronDaemon is a service that provides functionality to schedule the execution of any user code. Depending on the type of the entry, the code is executed at a defined time or repeated in some predefined intervals. The entries can also be tagged, making it easy to distinguish and group different entries. We give an overview of the architecture and the components, finishing with some code examples how the CronDaemon service can be used.

Entries

The user defines the execution code by extending one of the basic classes shown in the Figure 1. The basic and most primitive class is the CronDaemonEntry class. The CronDaemonEntry defines the time of execution, where the class itself is executed only once.

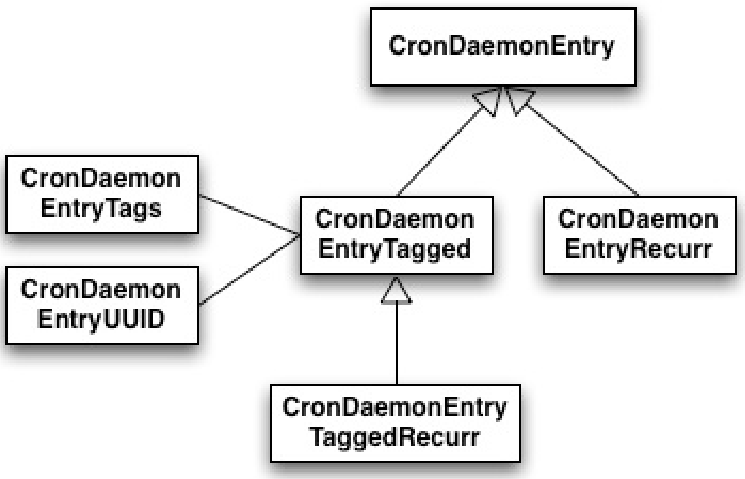


Figure 1 Basic classes for cron daemon

The user creates an extended class of CronDaemonEntry and overrides execute() method. An example of an extended class and its usage:

```
class MyCronEntry extends CronDaemonEntry {
     public MyCronEntry(GregorianCalendar at) {
            super(at);
            //todo: add your initialization code here
     }
```

If the code needs to be executed repeatedly, the CronDaemonEntryRecurr class is used, which defines the interval (in milliseconds) of execution and the starting and ending time of execution. If the ending time is not defined, it is expected, that the entry does not have a time limit. An example of usage:

```
class MyRecurrCronEntry extends CronDaemonEntryRecurr {
     public MyCronEntry(GregorianCalendar startAt, GregorianCalendar finishAt,
int interval) {
             super(startAt, finishAt, interval);
             //todo: add your initialization code here
     }
     ...
}
```

An example of an entry that starts within next 5 minutes and ends after an hour and is executed every 2 seconds:

```
GregorianCalendar startAt = new GregorianCalendar(); //set to now
startAt.add(GregorianCalendar.MINUTE, 5);
GregorianCalendar endAt = (GregorianCalendar)startAt.clone();
endAt.add(GregorianCalendar.HOUR, 1);
cron.addEntry(new MyRecurrCronEntry(startAt, endAt, 2000));
```

When the total timeframe is not known, set the end time to null. The CronDaemon is informed to never remove the entry by itself and waits for the user to manually remove the entry:

```
cron.addEntry(new MyRecurrCronEntry(startAt, null, 2000));
```

Both types of entries described so far can be seen as anonymous entries. They can not be distinguished and therefore can not be identified and acted upon. In order to provide additional information, CronDaemonEntryTagged class must be used. The class provides tagging functionality, from basic UUID to user defined complex tags. The CronDaemonEntryTaggedRecurr provides the basis for recurring tagged entries.

The tagging is provided by the CronDaemonTags class. The tags can also have one or more objects associated with it. One can use tags as a description of an activity, while the values of the tags describe different attributes of the activity. For example, many entries can be tagged as "test" with values describing the stages of the testing, giving the user complete control over the execution process.

Entries can be identified by:

- UUID
- simple tags (no values)
- a set of tags and their dependent values.

In the last case, the entry is a match if the given entries and their values are proper subset of entries tags (there is no such tag or value, that is present in the query but not in the entry).

An example of basic UUID tagging:

```
class MyTaggedEntry extends CronDaemonEntryTagged {
 ...
}

MyTaggedEntry myTagged = new MyTaggedEntry(...);
myTagged.getTags().generateUUID();
CronDaemonEntryUUID uuid = myTagged.getTags().getUUID();
//TODO: store the uuid for later use
```

An example of more complex tagging:

```
myTagged.getTags().addSimpleTag("testing process");
GregorianCalendar startTime = new GregorianCalendar();
myTagged.getTags().addTag("Process A", startTime)

mySecondTagged.getTags().addSimpleTag("testing process");
int numThreads = 100;
mySecondTagged.getTags().addTag("Process B", numThreads);

myThirdTagged.getTags().addSimpleTag("testing process");
numThreads = 50;
myThirdTagged.getTags().addTag("Process B", numThreads);
```

An example of tag search:

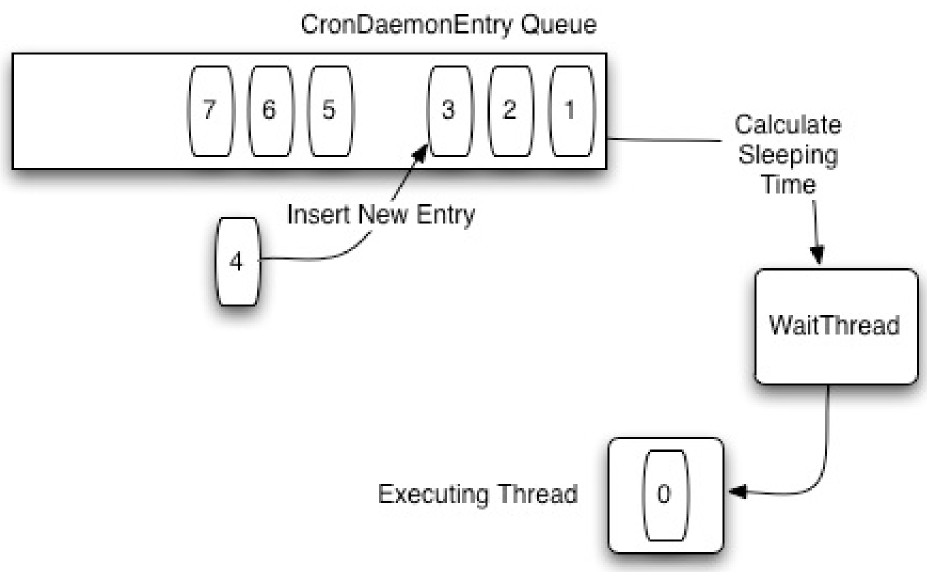
```
ArrayList<CronDaemonEntry> group =
cron.gorupByTags("testing process");
//return all three entries

ArrayList<CronDaemonEntry> group = cron.gorupByTags("Process B");
//return only second and third

ArrayList<CronDaemonEntry> group = cron.groupByTags("Process B", 50);
//returns only the third one
```

Cron Architecture

The CronDaemon service is designed to handle as many simultaneous requests for execution as possible. The entries are added to the sorted queue, from the entries that will be executed sooner in the front and the ones executed later in the back of the queue. The first entry of the queue determines the time of execution. The CronDaemon holds a special thread which waits in suspension until the first entry is scheduled. After this given interval, the WaitThread notifies the CronDaemon that it needs to process its first entry. The CronDaemon creates a special execution thread and executes the entries code inside this threads context. As soon as the execution thread is created and the first entry is removed from the queue, the waiting thread is notified of new suspension time, defined by the new top entry in the queue. The process is shown in the Figure 2.



1. Figure 2 Cron architecture

User's entries can be inserted into the queue at any given time, therefore an entry that is added right in the beginning of the queue must interrupt the waiting thread and force new suspension time recalculation. The described process is shown on Figure 3.

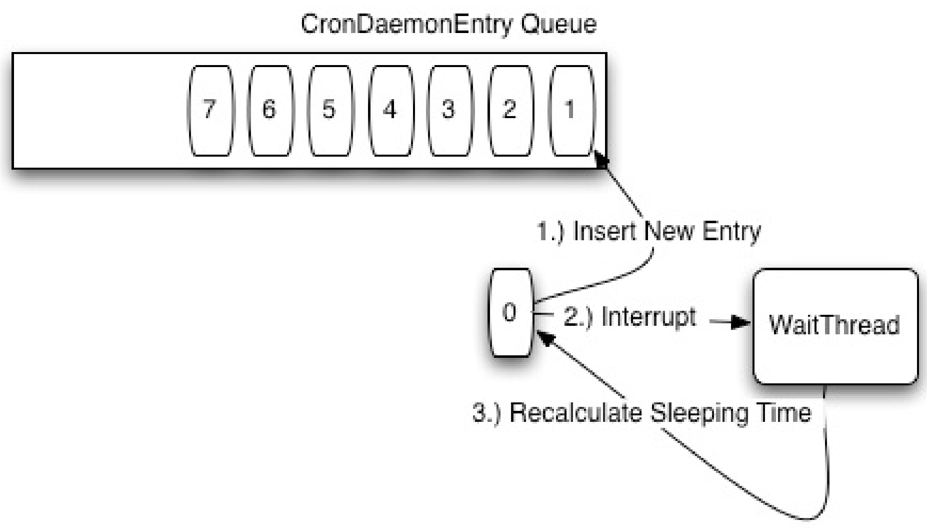


Figure 3 Insertions in the cron queue

Besides adding entries to the queue, the CronDaemon also provides the following functionality:

- pause/resume
- remove all entries
- remove tagged entries

- group tagged entries

The last two features enable the user to control the entries that depend to a certain process. For example, the user can remove an entry defined by its UUID. Grouping of tagged entries enable the user to act on a group of entries, as shown in Figure 4. For example, one can group all entries with a specific tag or tag value and then corrects the recurrence interval, or add new tag or simply remove the group of entries from the CronDaemon.

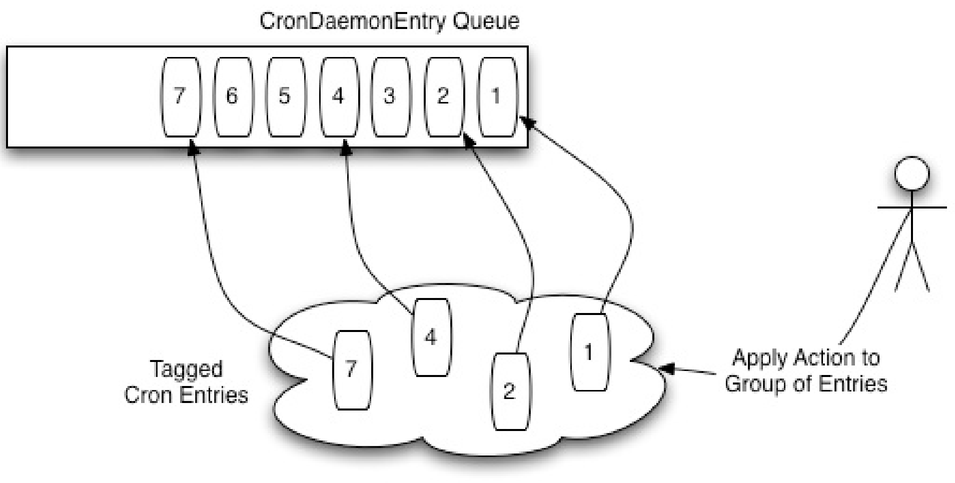


Figure 4 Grouping entries

The grouping of tagged entries is useful especially in cases of entries that belong to a certain process in the system. Suppose userš job is composed of three subtasks, task A, task B and C. Each of the tasks requires recurrent execution of some code during the taskš processing time (such as monitoring its progress, gathering statistical data, etc). When the task terminates, the CronDaemon must remove dependent entries and possibly change the entries from the other task, where the described functionality comes to use.