ICE: A service oriented approach to uniform the access and management of clusters environments

Clarissa Marquezan , Rodrigo Righi, Lucas Schnorr,
Alexandre Carissimi, Nicolas Maillard, Philippe Navaux
PPGC/Instituto de Informática - Universidade Federal do Rio Grande do Sul - Brazil
{clarissa, rrrighi, lmschnorr, asc, nmaillard, navaux}@inf.ufrgs.br

Abstract—This paper presents ICE (Integrated Cluster Environment) which is target to deal with several clusters. Generally, when users and clusters administrators have access to several clusters they have to deal with different kind of tools in each cluster. For example: different job schedulers, resource monitoring and administrative configuration issues. Sometimes these tools are used to perform similar tasks but using different interfaces with different parameters. ICE environment provides an alternative to uniform the access to different cluster tools, allowing users and administrators to interact and manage them in a transparent manner. Besides transparency, it also provides extensibility, since tools already installed in clusters can be managed by ICE environment just by adopting the integration framework designed. Uniformity, transparency and extensibility are reached due to ICE architecture and Web service usage as the environment middleware. This paper describes ICE architecture, the developed prototype, ICE usage and also a comparison with related work.

I. INTRODUCTION

This work focuses on the intersection of cluster [1] and grid areas, adapting grid concepts (such as uniformity access, usage transparency, capability of using different tools and the idea of grid portals [2], [3], [4]) to cluster context. The considered context is formed by users that have accesses to more than one cluster. However, these clusters are not integrated in a grid environment, they have a standalone usage and management. We think that using grid infrastructure environments, like Globus Toolkit [5], in order to provide the abstractions to deal with different clusters are not appropriated in this context. Adopting them means to insert an overhead of protocols and controls to provide grid infrastructure and that are not required when dealing with clusters. For example, to take profit of WSGRAM in GT4 (a module responsible for providing resource management for the grid) it is necessary to use the GT4 core. According the considered context, users are aware about what resources they are using, but they do not know what tools are used to perform their requests.

In order to provide a user-friendly environment target to clusters, Integrated Cluster Environment (ICE) was developed and is presented in this paper. It is based on a Web Portal and on Web services [6]. ICE’s main goals are: to uniform the way that operations of tools and clusters are accessed; to afford access and usage transparency for its users; and provide extensibility, enabling the incorporation of new services on ICE environment and also new tools under just implementing a defined Web service interface and the designed framework. The target users of ICE environment are: clusters administrators and mainly non-native computer science users. Considering the first kind of users, ICE offers administrative facilities for managing several clusters. The second kind of users generally want to run their applications but they are not used to deal with clusters tools. Examples of these kind of users are researchers from: hydrology, bioinformatics and physics areas. It is target to applications already tested and debugged that are ready to be executed.

The remainder of this paper is organized as follow. Section 2 presents related work. Section 3 describes ICE architecture and section 4 the implemented prototype. Section 5 shows ICE evaluation, composed by a usage demonstration and the comparison between ICE and related work. Finally, section 6 presents the conclusion, further considerations and future works.

II. RELATED WORK

In this section two high performance computing environments are described: HPC2N [7] and M3C [8].

High Performance Computing Center North (HPC2N) is an environment that provides a single point to interact with its functionalities. HPC2N is an user-centric environment that allows users to deal with their jobs (submitting, manipulating, monitoring status) and with monitoring features. It does not provide administrative configuration and debugging issues. HPC2N was developed to be used with high performance computing systems, this includes one or more clusters, and according to their authors it could be extended to a grid environment. Its batch system is based on OpenPBS with Maui as scheduler. It was developed to be accessed via Internet through CGI.

Managing and Monitoring Multiple Clusters (M3C) is designed to be used with several clusters. It provides a Web-based interface for cluster administration and also provides a framework for the development of the underlying management system. M3C was designed to be extensible, since cluster functionalities can be added following a defined framework. Although it presents a framework to extend its functionalities it does not provide support to extend the environment to operate with different tools for the same functionality. It was implemented using Applets and Servlets Java, and also uses configuration files to store information concerning with cluster resources. M3C is composed by the following modules:
node monitoring, node reservation and cluster partition. It is composed by tools that are developed by the same group that support this environment.

Comparing HPC2N with ICE approach it is possible to notice that although HPC2N would appear as a good alternative to treat several clusters environments, it does not support most services related to cluster environments, such as administrative configurations and specific monitoring issues as ICE provides. Another problem is its backend system, although being able to access different clusters, HPC2N is not extensible. Its backend system is restrict to its supported tools. There is not an uniform interface allowing other job schedulers to be used, for example. Through ICE architecture it is possible to extend the environment and adapt it to cluster established tools. As well as HPC2N, M3C could appear as an alternative instead of ICE. However, even providing a framework for extending its functionalities it does not provide the capability to integrate different tools with the same functionality within its infrastructure. Moreover, to use M3C it is necessary to change the established cluster environment. With ICE it is possible to integrate different functionalities and tools in the same environment without changing the established cluster underlying.

Considering HPC management environment context, we have identified the necessity of developing an environment actually focused on dealing with several clusters, transparent, extensible and user-friendly. For this reason we propose ICE, a Web services based environment for managing several clusters through a unique entry - a Web Portal. The Web service is decisive in ICE environment. Using this middleware it was possible to maintain the compatibility with clusters legacy systems and to define a uniform interface for accessing the clusters functionalities. This last feature provides transparency when interacting with tools installed on the clusters and extensibility to incorporate new services and tools in ICE environment.

III. ICE ARCHITECTURE

ICE environment architecture is presented in Figure 1. It is a modular environment based on SOA (Service Oriented Architecture) model. It is divided in three main components: Portal, Unified Service Interface (USI) and Service Implementation (SI). Each of them are described below.

**Portal** component is a Web application whose goal is to encapsulate the access to clusters functionalities in a unique Web interface. It is divided in two modules: System Management Module (SMM) and Service Module (SM). SMM is responsible by a set of tasks and Web interface presentation. The tasks include: profiles and authentication user management, maintenance of information related to clusters managed by ICE; management of functionalities offered by ICE; management of relationship among users, clusters and functionalities. The Web interface is dynamically rendered according to: user profile, clusters that each user is allowed to access and the functionalities that these clusters provide for that user.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Capabilities inside ICE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ICE_ROOT</strong></td>
<td>Handle users, clusters and functionalities (insert, remove, change)</td>
</tr>
<tr>
<td><strong>CLUSTER_ROOT</strong></td>
<td>Specify cluster functionalities; Assign functionalities to a cluster; Assign users to a cluster functionality; Maintain users, cluster and functionalities relationships</td>
</tr>
<tr>
<td><strong>USER</strong></td>
<td>Use assigned functionalities</td>
</tr>
</tbody>
</table>

Profile is a concept used to defined ICE user access policies. These policies are related to clusters and SM usage permissions. A user can be assigned to more than one profile. Table I describes the actions performed by each ICE profile.

SM is a Web service consumer that follows a Web service interface. Each cluster functionality can be mapped to a SM. It is responsible for contacting the Web service provider of a cluster functionality, to treat the response of this request and to transform this information to be presented in ICE Portal. Clusters functionalities can be, for example, job management, application and cluster monitoring.

**Unified Service Interface (USI)** component is the definition of a Web service interface for a cluster functionality. Actually, defining a USI can be compared to define a WSDL for a cluster service. However, defining a WSDL means also to inform the address of this service. When a USI is defined the address of its service provider is not informed, just operations and parameters are defined. Through a USI it is possible to uniform the way that operations of different tools with the same purpose are requested. This USI should represent typical operations executed by users when using a functionality, i.e. a service. The process of creating a USI consists on analyzing tools with the same functionality and trying to find out an intersection among their operations and parameters. Using this approach ICE environment can enclose as many services as USIs were defined.

**Service Implementation (SI)** component is the Web service provider of a USI. Considering the same USI it is possible...
to implement several SIs. However, it is possible that an exactly matching between USI and target tool does not exist (concerning its operations and parameters). So, it is necessary to adapt them to support USI features. This could lead to insert extra operations in target tool.

ICE architecture was designed to be extensible and transparent. Within ICE, extensibility was carried out in two levels. The first one is the SM level, since ICE architecture allows the incorporation of as many SMs as services (and USIs) were defined. The second level of extensibility is achieved with SI modules, since it is possible to integrate inside ICE as many tools (with the same functionality) as SI modules were implemented. Access transparency was reached due to Web Portal usage that uniforms the access to clusters functionalities. Usage transparency were accomplished using SM, USI and SI modules that provide Web services underlying.

IV. PROTOTYPE IMPLEMENTATION

A prototype was developed based on ICE architecture. This section describes the implementation of this prototype which is composed by ICE Portal, Job Management Service Module (an instance of SM), Job Management Service Interface (an instance of USI) and three implementations of Job Management Service Implementation (instances of SI). Other service modules compose ICE environment, but they are not discussed in this paper.

A. ICE Portal Implementation

ICE Portal was developed using Servlets, Java Server Pages (JSP) and Tomcat. Servlets are used in SMM and SM modules, enclosing business logic and Web services consumers. Web interface presentation is provided using JSP technology.

Considering that information of users and clusters is transmitted through different administrative domains, and that the SOA model allows users to access services from anywhere, some security issues must be considered in ICE environment. The first issue considered was HTTPS protocol. All ICE communications are performed using this protocol. These communications concern with users and ICE Portal and ICE Portal with service providers of cluster front-ends. Furthermore we are considering authentication and authorization security features. Before accessing ICE environment and its functionalities, users must log in the system informing a username and password. This information is checked and when it is valid the user will be allowed to enter in ICE environment. The set of functionalities granted to a user depends on his profile and the clusters he is allowed to use. With this approach it is possible to guarantee that just authenticated and authorized users are using ICE environment. Although ICE Portal presents an authentication process, each cluster managed by ICE can present its own authentication mechanism with different information. Thus, inside ICE environment users can have different usernames and passwords (or other authentication mechanism) for the clusters they have access. SMM controls the whole process of authentication and authorization of ICE users. Information regarding security issues is stored in ICE database. PostgreSQL technology has been used to maintain ICE database.

SMM is also responsible for dynamically rendering of ICE Web Portal. Due to profile adoption the Web interface is not static. It is built according to the: user profiles, clusters that the user has access and the functionalities this user has permission to use. This information is also stored inside ICE database. Furthermore, all operations and parameters of a SM components are also stored in this database. This feature provides a high level of flexibility to ICE, once it is possible to change or to insert operations and parameters in each SM without changing the SMM code.

Figure 2 shows an snapshot of ICE environment regarding a user that has all supported ICE profiles (ICE_ROOT, CLUSTER_ROOT and USER).

IV. PROTOTYPE IMPLEMENTATION

A. ICE Portal Implementation

ICE Portal was developed using Servlets, Java Server Pages (JSP) and Tomcat. Servlets are used in SMM and SM modules, enclosing business logic and Web services consumers. Web interface presentation is provided using JSP technology.

Considering that information of users and clusters is transmitted through different administrative domains, and that the SOA model allows users to access services from anywhere, some security issues must be considered in ICE environment. The first issue considered was HTTPS protocol. All ICE communications are performed using this protocol. These communications concern with users and ICE Portal and ICE Portal with service providers of cluster front-ends. Furthermore we are considering authentication and authorization security features. Before accessing ICE environment and its functionalities, users must log in the system informing a username and password. This information is checked and when it is valid the user will be allowed to enter in ICE environment. The set of functionalities granted to a user depends on his profile and the clusters he is allowed to use. With this approach it is possible to guarantee that just authenticated and authorized users are using ICE environment. Although ICE Portal presents an authentication process, each cluster managed by ICE can present its own authentication mechanism with different information. Thus, inside ICE environment users can have different usernames and passwords (or other authentication mechanism) for the clusters they have access. SMM controls the whole process of authentication and authorization of ICE users. Information regarding security issues is stored in ICE database. PostgreSQL technology has been used to maintain ICE database.

SMM is also responsible for dynamically rendering of ICE Web Portal. Due to profile adoption the Web interface is not static. It is built according to the: user profiles, clusters that the user has access and the functionalities this user has permission to use. This information is also stored inside ICE database. Furthermore, all operations and parameters of a SM components are also stored in this database. This feature provides a high level of flexibility to ICE, once it is possible to change or to insert operations and parameters in each SM without changing the SMM code.

Figure 2 shows an snapshot of ICE environment regarding a user that has all supported ICE profiles (ICE_ROOT, CLUSTER_ROOT and USER).

**ICE - Integrated Cluster Environment**

<table>
<thead>
<tr>
<th>ICE Configuration</th>
<th>Cluster Administrator Interface</th>
<th>Cluster Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Profiles</td>
<td></td>
<td>ICE Cluster Configuration</td>
</tr>
<tr>
<td>Services Modules</td>
<td></td>
<td>Cluster Users</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Functionalities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Job Management Services</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cluster Monitoring</td>
</tr>
</tbody>
</table>

Fig. 2. ICE Portal main interface for a user with all supported profiles.
will receive the parameters of the operation from the JSP pages. These parameters are treated and then the Web service consumer of SM module is invoked. This SM consumer is implemented according the USI defined for the functionality.

Job Management Service Module (JM-SM) was implemented according to Job Management USI (JM-USI) defined for this functionality. Its Web page content reflects JM-USI, but some graphical simplifications were done to let it user-friendly. Through this module it is possible to submit and finalize jobs, verify job output and visualize the script generated by the service provider of this functionality in target cluster.

Considering job submission operation it is possible to set the parameters for launching sequential or parallel jobs; to launch batches of these two kind of jobs; to upload applications and auxiliary archives of this application; and the reservation of a future date to execute the application. The uploaded application must be ready to be executed. The process of uploading files in ICE is done using MTOM (SOAP Message Transmission Optimization Mechanism) mechanism of Axis2 [9]. Using finalization operation the user must choose, from a table of his running and queued jobs, some job to be finalized or removed from the queue of the job management scheduler running inside the target cluster. Job status operation is activated every time the user access JM-SM, once he will be presented with a table containing his submitted job on the target cluster. Regarding job output verification it is possible to retrieve the standard and error output of a given job. Its content is presented in ICE Portal in a user-friendly manner. Finally, visualizing scripts operation returns the application launch script generated in the service provider; i.e. if this operation is requested in a cluster using OpenPBS scheduler this operation will return the launching script generated for this tool. This operation was provided because there are some users (generally the ones that are more familiarized with clusters tools) that want to be sure if what they are asking in ICE Portal will be actually translated in the service provider of the target cluster. Job management operations and its parameters will be better explored in Job Management USI description, and the demonstration of ICE JM-SM will be presented in Section V.

B. Job Management Unified Service Interface

The USI for job management (JM-USI), called JobManagementServiceInterface, was defined considering the parameters passed by users for job schedulers. They were chosen through the analysis of the most used schedulers, such as OpenPBS [10] and SGE [11].

Security is also considered in JM-USI definition. It is necessary to guarantee that just authenticated users will be able to perform the requested operation in a cluster. For this reason all operations of this interface have username and password parameters. Before performing any operation it will be necessary to check user authenticity. Authorization parameters are not provided in JM-USI yet, but we are working with the possibility of employing WS-Policy and WS-Security specifications [12], [13] to provide authentication and authorization in ICE Web services middleware.

JM-USI is composed by the following operations:
• jobSubmissionOperation() is used to launch applications in cluster environment.
• jobFinalizationOperation() finishes a job or remove a job from the job scheduler queue.
• jobStatusVerificationOperation() is used to visualize job status.
• jobOutputRetrieveOperation returns output file content of a given job. This operation can be used to get standard and error outputs.
• submissionScriptVerificationOperation() returns the script generated by the service provider of this functionality in target cluster.

With these operations users can perform basic actions of cluster job schedulers. Some of the specified parameters in JM-USI can not be directly mapped into job management schedulers parameters or functions. Job management SI implementations will have to treat these kind of problems, but ever following JM-USI semantic and syntax.

C. Job Management Service Implementation

Following JM-USI we have implemented three Job Management Service Implementations (JM-SI): OpenPBS JM-SI, SGE JM-SI and OAR JM-SI [14]. In order to keep the extensibility of SI we have defined a job management framework (JMF) of abstract classes that must be extended and implemented in each JM-SI. Figure 3 shows ICE JMF defined for JM-SI. The classes inside dark boxes on Figure 3 composes the framework.

JobManagementProviderInterface is the Web service provider interface that must be implemented. JobManagementProvider abstract class implements this interface and there must be an association between this class and Job abstract class, and also between it and UserAuthentication abstract class. These associations are necessary since JobManagementProvider operations utilizes methods of Job class when operating over job features, and the methods of UserAuthentication class whenever an operation has to be performed.

Job is an abstract class that handles information and methods related to jobs. It is used to set information through setJobInfo() method; to generate job submission script, through generateSubmissionScript() abstract method; and to submit it to job scheduler through launchSubmissionScript() abstract method.

UserAuthentication is an abstract class that contains two abstract methods: setUserInfo() and confirmPassword(). They must be implemented according cluster authentication mechanisms.

In order to enable authentication mechanism in the clusters where ICE is installed, LoginUserAuthentication class was implemented. It extends UserAuthentication abstract class and is based on Unix login authentication (username and password). Inside it,
confirmAuthentication() method was implemented based on information of /etc/passwd file system. All developed JM-SIs use this authentication implementation.

Following JMF definition three JM-SIs were developed:
- OpenPBSJobManagementProvider,
- SGEJobManagementProvider and
- OARJobManagementProvider.

Most of the problems ICE developers face on mapping JM-USI to job schedulers functionalities are enclosed on jobSubmissionOperation(). Thus, the mapping process of this operation will be described.

OpenPBSJob, SGEJob and OARJob are abstract classes that extends Job abstract class. These classes contain methods needed to implement generateSubmissionScript() and launchSubmissionScript() abstract methods. However, these two methods are not implemented inside these classes because there are differences on the script generation regarding sequential and parallel jobs. For example, when launching a parallel job, it is necessary to create a file containing the name of the nodes that will be used by the application, define the number of parallel processes that will be launched, etc. For this reason sequential and parallel jobs were distinguished and implemented in different classes. Parallel jobs were also distinguished according to the parallel environment used. At this moment, there is support for MPICH and DECK [15] parallel environments, but it is possible to extend ICE parallel library support for other ones. Then, generateSubmissionScript() and launchSubmissionScript() methods are implemented in OpenPBSSeqJob, OpenPBSMPICHParallelJob and OpenPBSDECKParallelJob classes (this infrastructure will be the same for SGE and OAR JM-SI implementations as presented in Figure 3).

Concerning the JM-USI, OpenPBS presents the most complex mapping process to JM-SIs. This job scheduler has support for most JM-USI submission operation parameters. However, there is no idea of reserving a predetermined date for launching a job on its approach. For this reason a daemon called sched_pbs_daemon was developed. It implements this functionality and operates with OpenPBS. In fact, every requisition of jobSubmissionOperation is first analyzed by the daemon. After this, the requisition is submitted to OpenPBS. The daemon controls every running and queued job on OpenPBS. When a job submission request arrives containing reservation features it compares if this request fits among already OpenPBS running and queued jobs. If the job request fits it will be submitted to OpenPBS scheduler, else an error will be returned to the Web Portal. The information of running and queue OpenPBS jobs and the features of them have to be stored a in a persistent repository. Thus, beyond the daemon a database is also required in OpenPBS JM-SI implementation. PostgreSQL database was used to store this information.

SGE JM-SI and OAR JM-SI development was facilitated since submission operation parameters could be mapped almost directly to their parameters. Just OAR JM-SI development had to adapt to perform the standard and error output join in a single file because this job scheduler does not have native support. Different from OpenPBS, SGE and OAR provide job reservation functionality. Thus, there was not the necessity of using a database in the clusters that utilize these job schedulers.

Despite adapting some parameters, there was not the necessity of changing any job scheduler or cluster basic infrastructure. The developed JM-SIs were built to lead with all mapping necessities without changing any aspect of underlying job schedulers. The extensibility of ICE is maintained since it is possible to create JM-SIs for other tools than OpenPBS, SGE and OAR.

V. ICE EVALUATION

This section is divided in two sub-sections. The first one presents ICE usage, where we will describe a submission job request being performed in our cluster testing scenarios. The second part concerns the comparison of ICE environment with related work presented in Section II.
A. ICE Usage Demonstration

This demonstration considers a user that has USER profile and is authorized to use JM-SM provided by three clusters of our research group, whose front-ends are: corisco (using OpenPBS), testcluster (using SGE) and frontal-minuano (using OAR). This user will launch the same application in the clusters through ICE Portal. Figure 4 shows job submission Web page inside ICE Portal.

The application that will be launched is a parallel job using MPICH parallel library, uploading main application and auxiliary files, four cluster nodes will be reserved for this job, and two MPICH processes will be launched in each reserved node. It is not necessary to fill appointment fields, since no reservation will be requested. This job must be executed three times inside the walltime reserved for it. This means that there won’t be generated launching scripts for each time the application will be executed. In order to explain this situation Figure 5 and 6 shows the difference between launching just a single script with several executions of the same application (Figure 5) and multiple scripts for the same application (Figure 6). These figures were generated through Preview Script button of Submission Web page that calls the submissionScriptVerificationOperation() of the Web service provider inside the target cluster; in this case inside corisco cluster.

This feature was provided inside ICE because there are some times that users need to execute their application several times in order to get statistical measurements. However, it is possible that the execution time of all iterations overpass the maximum time of the job scheduler allocation policy from the target cluster. Using this ICE feature the user can adapt his execution requirements according to the allocation policy of the cluster.

Figures 7 and 8 show the submission script generated by the others JM-SI for the same job submission of Figure 4. To get these scripts from the target clusters the Script Visualization Web page of ICE was used. The same information of Submission Web page is required to call the visualization operation.

Analyzing these figures it is possible to see each specific configuration issue of the considered job schedulers. Although there are differences when launching jobs in each scheduler the user is not aware about them. Through ICE environment the user is able to interact with clusters without regarding with tool commands.

B. ICE x Related Work

Comparison between ICE environment and related work will regard the challenges of constructing distributed systems
highlighted by Coulouris in [16], applying his concepts to the context ICE and related work environments. The list of the comparison criteria is exposed as follows.

- **Heterogeneity** - Considering this aspect of a distributed system, the comparison was done based on the following features of each environment: (i) platform independency, the system is capable to be running in different hardware and operational system platforms; (ii) language independency, the modules that compose the environment can be written in any programming language; (iii) interoperability, the middleware that compose the environment is capable to integrate legacy systems and deal with the differences among modules that compose the system.

- **Openness** - This issue regards to system extensibility. We are considering two kind of environment extensibility: (i) functionality extensibility, that is related to the support of providing an infrastructure to increase the system with new functionalities; and (ii) tool integration extensibility, that considers the support to provide an infrastructure to enclose new tools, with the same functionality, inside the environment.

- **Transparency** - Considers the following types of transparency: access, location, concurrency, replication, failure, mobility, performance and scaling.

- **Failure handling** - Considers the capability that environment components presents to deal with failures.

- **Scalability** - Represents the capability of adding users and resources without affecting the system.

- **Concurrency** - Takes into account the design of safe resources in a concurrent environment.

- **Security** - The analysis of this parameter was based on the existence of an encrypted communication channel and user authentication.

Table II presents the existence or not of the assumed comparison parameters among HPC2N, M3C and ICE environments.

<table>
<thead>
<tr>
<th>Comparison Parameters</th>
<th>HPC2N</th>
<th>M3C</th>
<th>ICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Platform independency</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2 Language independency</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3 Interoperability</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Functionality extensibility</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5 Tool integration extensibility</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Access transparency</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7 Location transparency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 Concurrency transparency</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9 Replication transparency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 Failure transparency</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Mobility transparency</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 Performance transparency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 Scaling transparency</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 Failure handling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 Scalability</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16 Concurrency</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>17 Security</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

The first five parameter of Table II are related with heterogeneity and openness features of distributed systems. ICE environment provides them due to Web services usage as ICE middleware. M3C can reach platform independency since it is built using Applets and Servlets Java. M3C can also provide functionality extensibility since its framework afford users with the possibility to insert new functions to M3C environment. However it does not provide either a framework to deal with different tools with the same functionality or the capability to deal with legacy cluster systems. HPC2N can not lead neither heterogeneity nor openness issues because it is restrict to a closed number of tools. It was developed with technologies that do not provide the capability to be interoperable and do not provide any framework to achieve extensibility.

All compared environments present access, concurrency and mobility transparency issues. Their users can access the environment without regarding the used operation to reach the resource. Many users can access them via browser concurrently and these users can change their physical location without affecting the system. However, none of them presents location transparency because the user knows which cluster is being used. Replication and failure transparency, as well as failure handling, are not provided for all of these environments because their components were not designed to be aware and deal with the failures of the components that they depend on.

Considering scalability issue it is not possible to ensure
anything, since there are not studies or experiments that present whether these environments are scalable. Taking into account that they are Web-based environment, and that are executed under Web servers like Apache or Tomcat, their scalability should be limited to the configuration of these components. Concurrency is another issue also related to these underlying system of ICE, M3C and HPC2N. Each access to these environment are handled first by the Web server. Finally, all of them utilize HTTPS to encrypt their exchanged data. But each one employ distinct techniques to assure authentication and authorization issues.

VI. CONCLUSION AND FUTURE WORK

This paper presents ICE environment. It was designed to manage clusters underlying tools and to provide high level abstractions to the users at the moment they are interacting with several clusters. ICE main goals are: (i) provide a uniform way to use cluster tool operations and to uniform the access of cluster users; (ii) offer access and usage transparency for its users; and (iii) provide extensibility either regarding the increase of new functionalities on the system or the addition of different tools with the same functionality within the environment.

ICE architecture was defined to be modular and to follow SOA (Service Oriented Architecture) model. ICE is composed by a Web Portal and Web services modules. Web services were used since they are build according SOA model. In order to endorse ICE architecture this paper presented a prototype containing a job management service module (JM-SM), the Web service interface for this cluster functionality (JM-USI) and the framework designed for implementing this service. Based on this framework three versions of this service were implemented (JM-S1), one version for each job scheduler used in our research clusters: OpenSC, SGE and OAR. Furthermore, ICE usage was presented and it was also compared with other similar environments. Security issues were considered in ICE in order to avoid that non-authenticated and non-authorized users have access ICE features. However, some works are being done to change current security mechanism implementation. The intention is take profit of WS-Policy, WS-Security specifications. Moreover, the adaptation of Grid Security Implementation (GSI) [17] to clusters context is being evaluated.

Not all ICE features are yet implemented. One of them regards on time information of clusters managed by ICE. Since it is an environment that manages several clusters, and perhaps they can be spread along different physical locations, there are some problems related to front-end time synchronization and how to present time information for its users. As ICE presents a Web Portal for its users, they can be located in any place on the Internet. For this reason it is necessary to treat time differences from where the user is located and where the cluster he is using is placed. Another issue is UDDI usage. Currently, information concerning with Web service is stored in a database, but we are considering to use of UDDI register. In order to complete ICE environment we are implementing other service modules, such as resource monitoring, job monitoring and administrative configurations.

VII. ACKNOWLEDGMENT

Thanks HP Brazil for partial support on Clummsy project.

REFERENCES