

TOWARDS BUILDING A GENERIC GRID SERVICES PLATFORM:

A COMPONENT-ORIENTED APPROACH

Jeyarajan Thiyaalingam, Stavros Isaiadis, Vladimir Getov
Harrow School of Computer Science, University of Westminster, London, U.K.
{t.jeyan,s.isaiadis,getov}@wmin.ac.uk

Abstract

In recent years, significant efforts have been made towards designing and building advanced Grid infrastructures. One of the main priorities in building new Grid systems is to assure that longevity and flexibility exist. In order to support these two seamlessly, the underlying Grid platform is built with a rich set of features, such that the requirements of any Grid application need a subset of the complete list provided by the platform. As a result, the notion of a lightweight platform has not been addressed properly yet, and current systems cannot be transplanted, adopted or adapted easily. With the promise of the Grid to be pervasive, it is time to re-think the design methodology for next generation Grid infrastructures.

Instead of building the underlying platform with an exhaustive rich set of features, we describe an alternative strategy following a component-oriented approach. Having a lightweight core platform, built only with the minimal essential features is the key to our design. We identify and describe the very minimal and essential features that a modern Grid platform should offer and then provide the other functionalities as pluggable components. These pluggable components can be brought on-line, whenever necessary or as demanded by the applications. With the support of adaptiveness, we see our approach as a solution towards a flexible dynamically reconfigurable Grid platform.

Keywords: Generic Grid Platform, Light Weight Grid Platform, Adaptive Grid, Adaptive Grid Service

1. Introduction

Grid applications using advanced Grid infrastructures benefit from very rich set of features - as these Grid platforms are designed with built-in exhaustive set of functions. Standards (such as [8]) and software for implementing Grid [15, 12] are also motivated to provide a generic computational Grid with all possible features built-in. The Open Grid Services Architecture [8], on which most of the Grid platforms are based, is built as a feature rich platform. This approach ensures that service request from applications is included of complete set of features offered by the platform.

Complexity (in terms of interactions, manageability and maintainability) of the implementation of any Grid platform based on this philosophy will be very significant. For example, upgrading a service component in this model is a difficult task. When one service component is modified, other components also need to be modified. Further, deployment of these Grid systems demand considerable computing resources. Managed and/or un-managed migration of these Grid platforms is also a challenging task. Nor can they be extended very easily in terms of functionalities and capabilities. For example, layering an existing Grid platform on a lab of machines involve considerable effort in configuring. Difficulties in configuring the platform involves removing or disabling the unnecessary features and in extending the system capabilities. In summary, current Grid systems are failing to address the issue of generality and reconfigurability. This is not a design flaw; instead they are designed with exhaustive set of services targeting longevity and flexibility – resulting in highly complex platform, impeding the expandability.

This paper summarises the current status of our work in progress on the design and architecture of a Generic Grid platform or Core Grid Platform. We use these two terms interchangeably in this paper. We identify a generic set of features that should be common to any Grid system, while addressing the issue of longevity and flexibility. Further, we also address the issue of “light weight platform”. The motivation in identifying this common set of features of generality is to standardise the road map for development of future Grid systems, which should be adaptive and intelligent while retaining the features of flexibility, longevity and expandability.

The overall contributions of this paper are:

- Proposing a Core Grid Platform with minimal complexity but with essential features.
- Providing seamless way of extending the platform capabilities by means of component introduction.
- Means to provide more flexibility to the end users.

- Defining a standard for future Grid Systems.

This paper is organised as follows: Section 1.2 provides the background for the paper. Section 1.3 identifies the common set of features found across different Grid platforms. Section 1.4 discusses the architectural aspects of this core Grid platform while Section 1.5 concludes the paper with future works.

2. Background and related work

The original motivation behind the OGSA [8] development was to offer ubiquitous support for Grid infrastructures by converging Web Services and Grid Services. The OGSIS specification [9], on which the OGSA relies, included necessary extensions to support stateful web services. However, the fact that these extensions were heavily object oriented and the interoperability issues with the Web Services and XML, impeded the adoption by the Grid Community.

Refactoring the OGSIS led to consider the WSRF [5] which constitutes specifications for different web services and management services. WSRF retains all OGSIS functionalities but all these functionalities are enhanced to meet the web services specification, for example WS-Addressing [6].

The idea of adaptivity in Grid Systems have been discussed in [17] and main emphasis was given either at the very low level, middleware or at the application level. The idea of service level adaptivity for heavily componentised Grid systems has not been addressed by these works.

Reconfigurability at software components level, especially in the context of Grid systems, has not been addressed in the literature. The notion of Web Services is included in our proposed Generic Grid Services Platform both at the higher level and at the lower level. In other words, the platform offers the Grid Service as a web service. Further, componentised functionalities can also be represented as web services. However, the lower level of service interaction is transparent to the end-user or applications.

These web-service components are adaptive and an extra layer of flexibility is provided by permitting these components to be re-wired as necessary to provide the reconfigurability.

3. Generic Services

OGSA was derived from use-cases of e-business and e-science applications [10]. These applications require more functionality in addition to the fact of being computationally demanding. This has influenced the architectural aspects of OGSA and resulted in functionally-rich and thick platform specification. To identify the minimal set of core functionalities, an equal emphasis must be given to small scale applications and devices as well, contrary to the approach that OGSA has taken.

The idea of the component-oriented design approach is to componentise the functionalities from the core set of features that the platform has to offer. Later, the functionality of the platform can be extended by plugging in these components on an on-demand basis. Enabling the Generic Platform to secure the foreknowledge on these pluggable components, permits the platform to extend the capability as necessary. Further, the platform should also be pro-active when components are introduced in order to inter-relate the operations of different components. For example, the platform should be able to recognise additional operations when a self-healing functionality is plugged in, so that any further negotiations with the fault-tolerant component can be done effectively.

Definition 1: An implemented feature as part of the Generic Grid platform is defined as **Feature**.

Definition 2: The information relating to a new component (or new feature) which might be plugged in at a future time is defined as **Feature Knowledge**.

Feature Knowledge related to a specific component provides only the information about the component, expected interface/interaction and interaction map across components, which enables cross-component operation. *Feature Knowledge Set* is a collection of *Feature Knowledge*. Any member in the *Feature Knowledge Set* does not implement any of the functionalities. Instead, functionalities are separately and exclusively implemented inside the respective components. In other words, *Feature Knowledge Set* is the foreknowledge of the engine about pluggable components.

With these definitions, the idea is to design the generic Grid platform with minimal and essential *Feature Set* and with necessary *Feature Knowledge Set*. The platform has to be engineered in such way that new *Feature Knowledge* can also be added later on. However, to realise a functional Grid platform, it may be necessary to include some of the components. For example, it is essential to include a resource management component to the generic Grid platform to realise a fully working Grid platform. The reason why it is being added through the knowledge set is to enable the development of tailored components. It is possible to include the resource management component inside as part of the Grid core, but such a resource manager should be rich in features, some of them may not be utilised at all. Consider a use-case of a computer laboratory with PCs turning into a Grid system during the middle of night. Resource management function for such a system is completely different than for a supercomputer based lab.

3.1 Feature Set

The following set of features must be available as part of the core of the proposed Grid platform.

- **Core Operating Support:** This results from feature extraction from the Native Platform Services and Transport Mechanisms and OGSA Hosting Environment from OGSA specification. This feature forms the concrete resource-hosting environment. However, a main difference is that the approach taken in building this layer is similar to building the Java Virtual Machine [14], building the core support for underlying operating system/hardware pair. Once they are in place this feature enables the system to handle the hosting of resources specific to the supported operating systems or hardware components, and the native resource managers manage them. Effectively this feature provides the basic operating skeleton and a hosting environment — an essential feature for a Grid Platform.
- **Core Connectivity Services:** (The connectivity services can also be the part of core operating support) Core connectivity services are to offer networking and transport functions for data transfer across multiple Generic Grid platforms and within the Grid domain. By default, it uses the platform specific connectivity/network/transport parameters (such as protocols) but can be varied by *Feature Knowledge*.
- **Knowledge Engine:** This part interprets and understands the knowledge sets discussed in the next section. This also permits addition of new knowledge sets.
- **Component Management Engine (CME):** This part manages the different components and triggers actions wherever applicable to handle the cross-component interaction.
- **Service Management Engine (SME)/Service Manager (SM):** All service operations are orchestrated and coordinated by this kernel. It is also responsible to direct the Component Management Engine.

These components and their interactions within a Grid system is shown in Figure 1.

3.2 Feature Knowledge Set

As outlined above, effective operation of a Grid System inherently depends on multiple capabilities of the Grid Platform, which we decided componentise. An application, such as one from [10] may require introduction and interaction of multiple components for the operation. A careful inspection of [10, 8] reveals that the following set of functionalities must be available as separate components so that, whenever necessary, any component providing a required functionality can be brought on line.

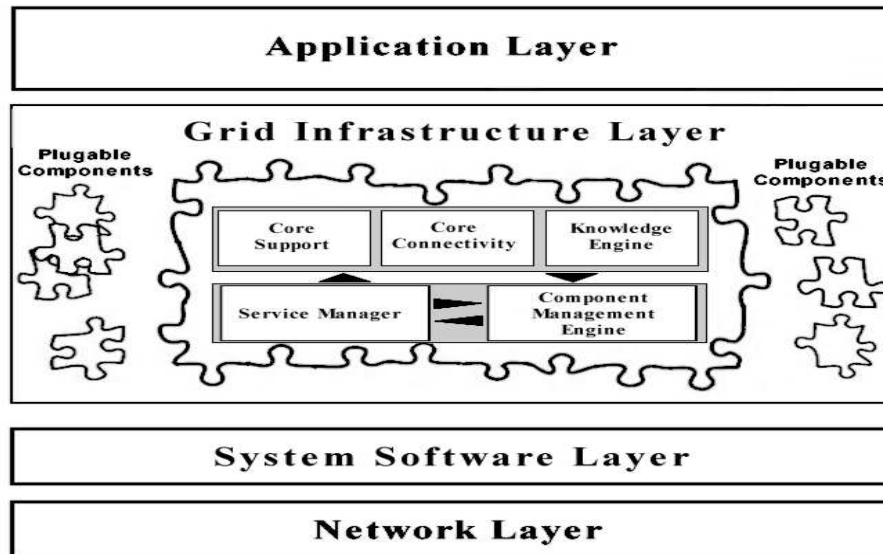


Figure 1. Simplified layered diagram focusing on fundamental or skeleton features to be implemented inside the proposed Generic Grid Services Platform

■ Basic Functionality Extension Components

– Resource Discovery

When a new resource enters a Grid environment must let the rest of the Grid know what type of services it provides and also to find other available services in the Grid. Mechanisms have to be provided to support such a dynamic resource discovery scheme. This is usually achieved with the use of a registry along with relevant registration and query mechanisms.

– Accounting/Metering and Pricing of Services/Resources

These services meter the usage of the resources, while for commercial Grids a pricing/billing component should also be in place to control resource utilisation — (perhaps based on price limits) producing pricing reports and necessary bills. Logging mechanisms are also required for the provision of more advanced services like for example forecasting which makes use of resource usage logs.

– Monitoring

In a complicated and dynamic Grid environment, Monitoring services for applications, resources and usages can assist in maintaining an “environment”, providing valuable information for trou-

troubleshooting in case of failures, supplying data regarding user applications and resource usage among other information.

- Data Management

Data management techniques such as data deployment/migration, data replication and data sharing are common in a Grid environment and should be supported by specialised components. Data migration (or deployment), sharing and replication are important techniques and are sometimes used to support failure/disaster recovery, higher performance through parallel data processing, service continuation through data mirroring (replication), job scheduling and work load balancing and many other procedures.

- Notification/Reporting/Messaging

Notifications and messages are very important in emergency situations like component or resource failures, but can also help in troubleshooting and prevention of unwanted conditions like heavily loaded resources/services or data inconsistencies.

- VO Component

Virtual Organisations (VO) can contribute in more scalable and richer in terms of available resources Grids. Mechanisms have to be provided that achieve automatic, dynamic VO creation (by merging collaborative networking environments) and VO management.

- Component based Policy Management and Application

Policies play a very important role in any Grid environment and can be present in almost every aspect of a Grid: resource management, security, accounting, pricing and data management just to name a few. Components/mechanisms that enforce the application of all these policies in an automated manner can be provided here.

- **Security**

- Authentication/authorisation and Accounting

The most fundamental notions of security in a distributed environment are those of authentication and authorisation. Authentication requires both the consumer of a service and the service to authenticate themselves to each other. This can be achieved with the use of a Public Key Infrastructure for example. Authorisation controls who has access to which resources and can be provided by simple access lists or more sophisticated techniques.

- Certification

Every resource needs to present a certificate in order to register to

the Grid. This certificate can be acquired from an independent Certificate authority and provides such information about the resource as the type of service provided, owner of the resource and other.

- Encryption/Decryption

A cryptographic infrastructure is important in order to maintain confidentiality of sensitive messages. This is usually achieved using a Public Key Infrastructure to support digital signatures and encryption/decryption of messages.

- Various security infrastructure-supporting components

The nature of the Grid assumes that many companies, organizations or individuals will participate in a Grid environment. Each of these parties will probably make use of different security infrastructures and techniques. Support services have to be in place to ensure secure interoperability across these different platforms. Such services should minimally include single sign-on, delegation of credentials and intrusion prevention and detection.

- Secure Inter-Grid communications

Different Grid platforms should be able to communicate with each other in order to utilise available resources and scale. We have to provide mechanisms to support inter-Grid collaboration and interoperability without compromising security (mainly) or functionality.

- **Resource Management**

- Provisioning

Components providing services such as scheduling of resources, reservation and termination are included in this category. Advance reservation, scheduling and provisioning as well as termination mechanisms provide the necessary support for the smooth and efficient utilisation of the available resources. Complimentary services like deadlock resolve mechanisms or freeing resources bound to processes that terminated abnormally, can enhance functionality and increase availability.

- Load Management/Balancing

Such services can increase performance and resource availability by eliminating possible communication bottlenecks and redistributing workloads of heavily loaded resources to ensure that all resources are used uniformly. Also, load balancing components can ensure that certain requirements are met (or at least at the highest possible degree) by reallocating resources depending on

the Grid user demands. For example more resources could be provisioned for a critical or highly prioritised application to ensure increased performance.

- Scavenging

Most workstation nodes present in a networking environment will remain idle most of the time according to many recent studies. In a Grid environment, utilising these idle resources is of great importance. These resources can be combined to create a huge secondary storage, memory or CPU pool that could substantially improve performance in demanding Grid applications. Scavenging mechanisms, however, should manage idle resources very delicately since the end user response times should not become unacceptable when the user decides to use his machine again.

- **Optimisation**

In addition to providing services, the platform should also optimise various operational aspects of the system, applications running on them and the interaction of different components in order to provide smooth and efficient operation.

- **Added Services**

- Fault Tolerance

Fault tolerance requires mechanisms for fail-over, workload redistribution, service continuation and notification of other relevant services like self-healing and disaster recovery. Fault tolerance is of extreme importance to real time environments or critical applications where even the lowest possible percentage of down-time might be unacceptable and/or disastrous.

- Disaster Recovery

Disaster recovery mechanisms are also important in sensitive Grid environments and should ensure continuation of at least the most vital services. They should also take actions to restore system operation and service, resource and application states as soon as possible (perhaps using backup data, previous checkpoints and last known state information).

- Self-Healing

Self-healing is the ability of a system to monitor its resources, detect failures and plan and apply necessary changes to ensure resource availability and service continuation. Human intervention should be kept to a minimum level and all operations should be

performed automatically, with human administrators only being notified in emergency or unresolved situations.

- Forecasting/prediction

Forecasting components cooperate with and may require the presence of scavenging, scheduling, workload balancing, metering and logging mechanisms. They can then extract valuable usage pattern information that can be used to predict the amount of time an idle workstation will remain idle and assist that way in job scheduling and workload balancing and reduce execution time costs.

4. Engineering the Core Grid Platform

In this section, we use two operational examples to illustrate the operation of the Generic Grid Platform and then we discuss the design aspects of the platform.

4.1 Functionality

Although the functionality built inside the Core Grid platform is very minimal, when engaged in supporting a Grid application, the platform must offer necessary features that the application requires. If such services are not available from the platform itself (either as part of the core-feature set, or as pluggable components), the internal mechanism may decide to secure a specific service from a remote site because of the limited local resources or because it is more efficient to act as a client rather than download and plugging-in this particular service component. If the remote site decides to not allow the component to migrate, the Generic Grid Platform may react according to a pre-configured policy or can act adaptively. However, submission of a clear job description along with the job, is an essential part of the whole process, as in [15].

The overall operation of the Core Grid platform is solely dependent on the capability of the SME to accommodate, anticipate and to reconfigure the components plugged in. Successful engineering of such a platform requires clear understanding of the operation of the proposed platform. Here, we consider two different applications with differing requirements to illustrate the operation of the platform. Figure 2 shows very generalized operation of the platform.

Operational Example 1

Consider the Use-Case example 3 from [10]. In this example, a severe storm prediction is considered. In the proposed platform, functionally, the following sequence of operations will take place:

- 1 The Core Grid platform announces the service availability through UDDI [7].

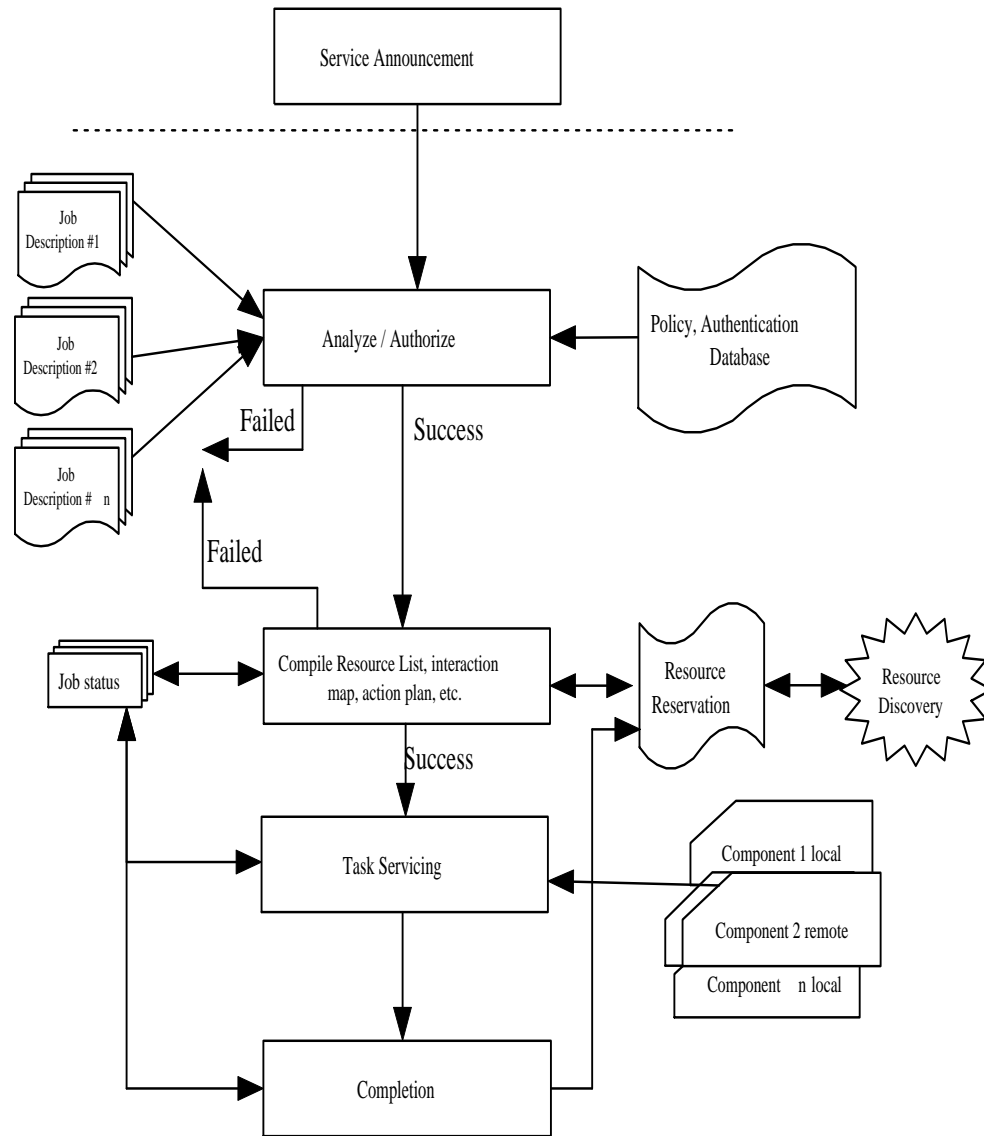


Figure 2. Generalized operation of the Core-Grid Platform. The platform dynamically demands and manipulates components as required by each job description submitted.

- 2 An interested client forwards the job description request to the platform.
- 3 The SM/CME part of the platform analyses the job description. The job description need to state all the requirements of features and should

supply a handle to any proprietary features. These proprietary features can replace an already existing feature in the Grid platform, or can be a completely new feature. It also verifies whether the set of features requested by the application is not empty. This means that, a request for a feature should exist on either the platform side or the client side. If the platform fails to secure any service (either locally or remotely) the request is terminated with a negative acknowledgement and positively otherwise.

- 4 Having accepted the job, the SM/CME should authenticate the user and authorise the job for further manipulation. This requires a feature which is not present in the Core Grid platform, but available through on-demand loading (in case of components) or through intra-Grid web-service (in case of web-service). SM/CME identifies the end-point where the “AAA” service is available and forwards the request and obtains the response before proceeding further.
- 5 The policy component needs to be loaded and consulted to determine the operational policy, if there are any, to be applied.
- 6 Once the job is authenticated and authorised, the SM/CME builds a list of resources and features and builds the interaction and dependency map between components or services.
- 7 This in turn requires a consultation with the resource reservation. This necessitates the launching of resource management service which along with the discovery, brokers the requests.
- 8 In case of reservation is favoured, the SM/CME updates the job status and continues further with the other operations that does not break the dependency or it waits until it is indicated that the resources are available.
- 9 If the advance reservation was done, then the SM/CME also delegates the provisioning and management tasks to the resource management component.
- 10 The SM/CME is notified about the resource availability.
- 11 If the customer wants to monitor the progress, the SM/CME loads and launches the Monitor (Application Part) component, which reports the status of the job at various time steps. This also triggers the launching of Reporting (or Notification or Messaging component) and logging components.
- 12 SM/CME also launches the pricing/metering component.

- 13 As this application requests complete reliable service, the self-healing, disaster recovery and fault tolerant components also loaded for this application.
- 14 Application also involves access to the large databases and this mandates the loading of data management component.
- 15 High Performance and fair pricing strategy requires application and workload to be distributed evenly and to be balanced as much as possible across the machines. This requires launching “load management and balancing” component.
- 16 With an application that can spawn multiple other applications, it is necessary to have proper synchronisation. This requires efficient orchestration of the tasks executed on the Grid - and thus launching “Work-Flow” component is inevitable.
- 17 The actual task servicing begins.
- 18 Along the time line, the SM/CME also should launch scavenging and resource optimiser components to guarantee the all free cycles are harnessed.
- 19 Execution Terminates, modules are unloaded one by one, and the result is forwarded to the customer.

Operational Example 2

Contrary to the large scale Grid application discussed above, here we consider a very small, but computationally demanding application. The application is CFD simulation of a moving car and the end-user is interested only on the final results. The job request is just to run the simulation on a single machine with a supplied set of data. Further, assume that the service is provided free of charge. Functionally, following sequence of operations will take place:

- 1 The Core Grid platform announces the service availability (through UDDI).
- 2 An interested client forwards the job description request to the platform.
- 3 The SM/CME part of the platform will analyse the job description. The job description will see that there are no proprietary features and all features to be available inside the platform. The request is positively acknowledged.
- 4 Having accepted the job, the SM/CME should authenticate the user and authorise the job for further manipulation. SM/CME loads the “AAA”

service is available and forwards the request and obtains the response before proceeding further.

- 5 The policy component need to be loaded and consulted to determine the operational policy, if there are any, to be applied.
- 6 Once the job is authenticated and authorised, the SM/CME builds a list of resources and features and builds the interaction and dependency map between components or services. In this case, the resource requirements are rather minimal.
- 7 This in turn requires a consultation with the resource reservation. This necessitates the launching of resource management service which along with the discovery, brokers the requests.
- 8 In case of reservation is favoured, the SM/CME updates the job status and continues further with the other operations that does not break the dependency or it waits until it is indicated that the resources are available.
- 9 If the advance reservation was done, then the SM/CME also delegates the provisioning and management tasks to the resource management component.
- 10 The SM/CME is notified about the resource availability.
- 11 The SM/CME deploys the data and code on the target machine, by invoking the data management engine.
- 12 The actual task servicing begins.
- 13 The SM/CME is notified when the execution finishes.
- 14 Execution Terminates, modules are unloaded one by one, and the result is forwarded to the client.

4.2 Advanced Features

From the above examples, it is clear that constantly loading and retaining all the services and features in the platform is not optimal. This is especially true with the case of offering the Grid services in a “plug-and-go” fashion, where there will be large number of small tasks or less demanding applications.

The design and engineering of the Core Grid platform should encompass adaptivity and intelligence. We have not stated this very clearly until this point. The reason for such a delayed introduction of engineering philosophy is to make the cases and requirements very clear.

The engineering of this platform can no longer track the traditional techniques, where the system behaves in a predicted pattern. Instead, the platform should be intelligent and should adapt itself to the changing conditions. These are:

- 1 As with the case of where applications or customers using their own services (or nominates their service providers), the SM/CME should be able to delegate the operation with those foreign components. This introduces, tremendous amount of freedom and flexibility in the Grid environment. Keeping aside the issue of trust and security, the main challenging issue is the delegation phase. The components inside the Grid system, should adapt themselves (at least to certain extent) to match the interaction with those foreign components.
- 2 The current state of the Grid system should be taken into consideration by these components. For instance, a pricing component should change the pricing strategy it is using when the system cannot meet the estimated target timing. Similarly, other components need to adapt and re-wire themselves according to the current state.

Another example is, one component is forced to adapt itself due to a change or adaptation occurring in another component.
- 3 Next generation application software will be adaptive in nature. This entails that the platform it is relying on also to be adaptive.

4.3 Relevant and Necessary Software Technologies

Section 1.4.1 has demonstrated the operational principles of the Core Grid platform and the fact that fixed functionality infrastructure would render the Grid platform to be unsuitable for future generation. Most of the software technologies required for engineering the Core Grid platform are readily available for us. This includes Jini [16] for resource discovery, WSDL [4] for service descriptions and so on. The most demanding aspect is the way that the components have to be built. As illustrated above, the platform is adaptive and requires the components to be adaptive for an effective operation. With this requirement, the interface of service components can vary during the runtime and the “Service Management Engine” should be able to understand the interface to initiate the interaction. This is not possible, unless the service engine adapts the interface presented in the “knowledge feature set” to the new set of interfaces announced or the component changes the interface dictated by the service management engine.

In a flexible Grid environment, services should be adaptive and the engine should be able to adapt to the current situation, without compromising the security. Such a requirement, where the component interfaces change while they

evolve presents us a challenge of interface morphing and/or with the need for dynamically reconfiguring software components [3]. This is well beyond what is offered by the Jini [16] or multi-paradigm techniques discussed in [11]. However the techniques for adaptive and customisable software components discussed in [1, 13] can be applied here - whose approaches tend to provide the necessary level of adaptability required by emerging applications and components – such as one we have outlined above.

5. Conclusions and directions for further research

Using a components-specific technology, we have outlined a possible direction towards a generalised Grid platform. We have addressed the flexibility, longevity and expandability issues by:

- identifying core set of features which should be built inside as part of the platform.
- identifying essential feature knowledge set, about which the platform should be aware of.
- separately plugging these components on demand in order to provide an effective operation.
- and by using the adaptive/reconfigurable software technology to permit these components to adapt themselves to the changing environment needs.

However, the design philosophy mentioned here still need to be checked against the available software technologies and to be considered at deeper level. A number of interesting issues remain:

- The list of features made available in the knowledge feature set need to be optimised.
- The proposed design may or may not render the existing protocols and software technologies obsolete. For example, adaptive nature of the platform requires much smarter discovery protocols. We need to investigate the possible side effects that this feature can have on other components and on the overall design and operation of the platform.
- The adaptive nature of components demands more on the software technology side. For instance, inter-component dependency analysis is a critical part of the adaptive concept. Cross component interaction opens up lots of questions for research, such as cross-component optimisation, security and trust issues to be enforced by the Grid platform and so on.

- The choice of customers or applications providing their own components involve additional challenges. In essence, the platform is provided with a component, which it has never seen. As mentioned in Section 1.4, this requires dynamic reconfiguration and interface morphing of the components. Related works mentioned in Section 1.4.3 are not yet mature enough, or at least their capability in addressing this problem is yet unknown.
- The Core Grid platform provides an extra flexibility to the applications or to the end users by permitting them to redefine the feature knowledge set and to update it on the fly. This requires the feature knowledge part to be retained in a separate database - say an XML file.
- The service manager is a critical component of the proposed platform. We have not looked into the details of the design aspects of this component, and in particular how it directs the CME.
- Techniques for announcing and invalidating interfaces and feature knowledge sets need to be determined.

In addition to the issues mentioned above, clearly there are lot of issues need to be addressed when designing and implementing such a platform.

Acknowledgments

We would like express our special thanks to Paul Kelly at Imperial College, London for his valuable comments on this paper.

References

- [1] Agha, G. A. (2002). Introduction: Adaptive middleware. *Communications of the ACM*, 45(6):30–32.
- [2] Astley, M., Sturman, D. C., and Agha, G. A. (2001). Customizable middleware for modular distributed software: Simplifying the development and maintenance of complex distributed software. *Communications of the ACM, CACM*, 44(5):99–107.
- [3] Bagchi, S., Whisnani, K., Kalbarczyk, Z., and Iyer, R. K. (1998). The chameleon infrastructure for adaptive, software implemented fault tolerance. In *Seventeenth IEEE Symposium on Reliable Distributed Systems (SRDS '98)*, pages 261–270, Washington - Brussels - Tokyo. IEEE.
- [4] Chinnici, R., Gudgin, M., Moreau, J.-J., and Weerawarana, S. (2003). *Web Services Description Language (WSDL) 1.2*. World Wide Web Consortium. [tp://www.w3.org/TR/wsdl12/](http://www.w3.org/TR/wsdl12/).
- [5] Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Maguire, T., Snelling, D., and Tuecke, S. (2004). *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution*. Version 1.1.

- [6] Bosworth, A., et al. (2003). *WS-Addressing Specification*. World Wide Web Consortium. <ftp://www6.software.ibm.com/software/developer/library/ws-add200403.pdf>.
- [7] Curbera, F., et al. (2002). Unraveling the Web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Distributed Systems Online*, 3(4).
- [8] Foster, I., Gannon, D., and Kishimoto, H. (2004a). The open grid services architecture. *GGF-WG Draft on OGSA Spec, 014*.
- [9] Tuecke, I., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maquire, T., Sandholm, T., Snelling and Vanderbilt, P. (2003). The Open Grid Services Infrastructure. *GWD-R OGSi Spec, Version 1.0*.
- [10] Foster, I., Gannon, D., Kishimoto, H., and von Reich, J. J. (2004b). Open grid services architecture use cases. *GGF-WG Draft on OGSA Use Cases 2.0*.
- [11] Getov, V., von Laszewski, G., Philippsen, M., and Foster, I. (2001). Multiparadigm communications in Java for grid computing. *Communications of the ACM*, 44(10):118–125.
- [12] Grimshaw, A., Ferrari, A., Lindahl, G., and Holcomb, K. (1998). Metasystems. *Communications of the ACM*, 41(11):46–55.
- [13] Kon, F., Costa, F., Blair, G., and Campbell, R. H. (2002). The case for reflective middleware. *Communications of the ACM*, 45(6):33–38.
- [14] Lindholm, T. and Yellin, F. (1999). *The Java Virtual Machine Specification*. Addison-Wesley, Reading, USA.
- [15] The Globus Toolkit, <http://www.globus.org/>.
- [16] Waldo, J. and Arnold, K. (2001). *The Jini specifications*. Jini technology series. Addison-Wesley, Reading, MA, USA, Second edition.
- [17] Alan Su, Dmitrii Zagorodnov, Gary Shao, Graziano Obertelli, Holly Dail, Jennifer Schopf, Jim Hayes, Marcio Faerman, Neil Spring, Richard Wolski, Shava Smallen, Silvia Figueira and Walfredo Cirne. (2003). Adaptive Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382.