

# Proposal for a Lightweight Generic Grid Platform Architecture

Jeyarajan Thiyagalingam\*, Nikos Parlavantzas\*<sup>‡</sup>, Stavros Isaiadis\*  
Ludovic Henrio<sup>‡</sup>, Denis Caromel<sup>‡</sup>, Vladimir Getov\*

\* Harrow School of Computer Sciences

and

CoreGRID Institute on Systems, Tools, and Environments

University of Westminster, Watford Road,

Northwick Park, Harrow HA1 3TP, U.K.

Email: V.S.Getov@westminster.ac.uk

<sup>‡</sup> INRIA and CoreGRID Institute on Systems, Tools, and Environments  
2004 Rt. des Lucioles, BP 93, F-06902 Sophia Antipolis, Cedex, France.

Email: Denis.Caromel@sophia.inria.fr

**Abstract**— One of the main shortcomings of existing Grid systems is their limited support for adaptability and scalability. We propose a generic, lightweight Grid platform with minimal context such that it could be adopted for different contexts of use and thus permitting scalability. In designing the platform we exclusively rely on component technology; specifically, our work builds on a reflective component model and composable component frameworks. This strategy enables us to keep the basic platform generic and lightweight yet extensible and adaptable through adding and reconfiguring components statically and dynamically. In this paper, we report on the analysis and design of the platform.

**Keywords:** Generic Grid, Lightweight platform, components technology

## I. INTRODUCTION

Grid technology has the potential to enable coordinated sharing and utilisation of resources in large-scale applications. In Grid systems the platform is the heart of the technology. It is the cohesive bond enabling virtualisation and control of resources, users, policies and applications regardless of geographic or administrative boundaries.

In many cases, existing, contemporary Grid platforms are feature-rich, such that the requirements of end users are a subset of the available features. This limits the scalability of the platform to be unidirectional – always scaling upwards. Additionally,

the feature-rich philosophy introduces considerable software and administrative overheads, even for relatively simple demands. As a consequence, the majority of the platforms cannot easily be adopted for a specific purpose; instead they involve substantial consideration for changes.

In our proposed architecture, we intend to address this problem by designing a generic, lightweight Grid platform, which could scale automatically based on the context of use. The platform relies on reflective component model and on composable component frameworks addressing different concerns of the platform functionalities. The component model supports flexible component adaptation statically and dynamically. The component frameworks facilitate designing the platform architecture, support extensibility of different aspects of the platform and allow easy to use and consistency-preserving reconfigurations. This paper builds on previous work [16], [2] which set out the basic principles for designing a lightweight Grid platform. Such a generic platform has also been considered in the context of hybrid systems integrating resource limited mobile devices into the Grid [9], [10].

The rest of this paper is organised as follows: In Section II we review some of the existing work, including component models, which are closely

related to the work described in this paper. In Section III we outline and discuss the minimalist features that the platform should support. The overall architecture of the platform is covered in Section IV. We illustrate the operational aspects of the platform using a real-world application in Section V. Finally, Section VI concludes the paper with directions for further research.

## II. RELATED WORK

Teo and Wang discuss a scalable run-time infrastructure for Grid, ALiCE [17], which is a lightweight Grid middleware facilitating aggregation and virtualization of resources. The modularised, object-oriented nature of its implementation supports possible extensions and varying levels of QoS, monitoring and security. The ALiCE architecture consists of multiple layers with the lowest, core-layer, providing resource discovery and system management while the second level layer supports application development and deployment. The ALiCE runtime system consists of consumers, resource brokers, producers and task-farm managers, which deploy and execute applications. However, their approach is not very generic and dynamic extensibility is partly a challenge as they entirely rely on an object-oriented approach as opposed to a component-oriented approach.

The Common Component Architecture (CCA) [6] specifies the means for interaction among components. In CCA, components interact using ports, which are interfaces pointing to method invocations. Components in this model define “provides-ports” to provide interfaces and “uses-ports” to make use of non-local interfaces. The enclosing framework provides support services such as connection pooling, reference allocation and other relevant services. Dynamic construction and destruction of component instances is also supported along with local and non-local binding. Though CCA enables seamless runtime interoperability between components, one of the main weaknesses of the CCA is the lack of support for hierarchical component composition and for control mechanisms thereof.

MOCCA [12], which is an implementation of the

CCA framework, is a lightweight distributed component platform implemented on top of the Java-based H2O [11] resource sharing platform. Though it is not a full-fledged Grid platform, it provides the basic mechanisms essential for implementing a lightweight Grid platform (see Section IV).

The Enterprise Grid Alliance reference model [8] is an attempt to derive a common model adopting Grid technologies for enhancing the enterprise and business applications. The model, which is aligned with industry-strength requirements, strongly relies on component technology along with necessary associations with component-specific attributes, dependencies, constraints, service-level agreements, service-level objectives and configuration information. One of the key features that the reference model suggests is the life-cycle management of components which could be governed by policies and other management aspects.

The CORBA Component Model (CCM) [15] is a language-independent, server-side component model which defines features and services to enable application developers to build, deploy and manage components to integrate with other CORBA [13] Services. The CCM is an extension of the CORBA object model defined to overcome the complexities of the same. The CCM specification introduces the concept of components and the definition of a comprehensive set of interfaces and techniques for specifying implementation, packaging, and deployment of components. The CCM provides the capabilities for composing components (through receptacles) and permits configuration through attributes. The CCM supports grouping interconnected components into assemblies. However, in contrast to the Fractal component model (see below), these CCM assemblies are not components and cannot be further composed — thus CCM does not permit hierarchical composition.

The Fractal specification [5] proposes a generic, typed component model in which components are runtime entities that communicate exclusively through interfaces. One of the crucial features of this model is its support for hierarchical composition. Another key feature is its support for extensible reflective facilities: each component is

associated with an extensible set of controllers that enable inspecting and modifying internal features of the component. Controllers provide a means for capturing extra-functional behaviours such as varying the sub-components of a composite component dynamically or intercepting incoming and outgoing operation invocations. The Fractal/ProActive component model [4] is an extension of Fractal that builds on the ProActive library for parallel and distributed computing. The proposal for a Grid Component Model (GCM) [1] is another extension of Fractal component model that specifically targets Grid environments.

### III. REQUIREMENTS FOR A GENERIC LIGHTWEIGHT PLATFORM

In this section, we outline the overall set of requirements for a lightweight, generic Grid platform. A complete list of features and requirements for a complex software, such as one we propose, evolve with user experiences and with time. To simplify this process, we discuss the requirements from the perspective of functional and non-functional requirements. In Section IV, we further refine and unify these and identify the features to be implemented to support these requirements.

#### A. Non-Functional Requirements

Specifying non-functional requirements, which often involve quality and contextual constraint properties of software, is very crucial for driving and validating the architectural decisions. Though run-time and development-time non-functional requirements are often considered equally important in the design process, we emphasise the run-time requirements.

- **Lightweight:** The platform should be available as a low overhead framework such that it could be deployed almost with little or no administrative and installation overheads. The purpose of this qualitative requirement is to enable easier deployment on different contexts.
- **Generic:** The platform should not be context bound nor infrastructure bound. This property guarantees that the platform is entirely context insensitive and independent upon deployment.

- **Configurability:** The platform should be re-configurable or should be adaptable for contexts, at the least by specification. This property is intended to guarantee the platform to satisfy two inter-related but different requirements: composability and evolvability. The platform should evolve with features over time and it should be possible to add or remove features, dynamically.
- **Security:** The platform should provide support for advanced, fine-grained, flexible and dynamically extensible authentication, authorisation and accounting mechanisms. Wherever possible, the platform should facilitate the utilisation of platform-level context based policies and application specific policies, with the former superseding the latter in case of conflicts.
- **Fault-tolerance:** The platform should have the ability to respond gracefully to unexpected failures. Although support for full fledged fault tolerance is not required, the platform should provide the core support for possible extensions and modifications.
- **Quality-of-Service:** The platform should deliver guaranteed or agreed level of QoS. This mandates the platform to exhibit acceptable performance and response behaviour.
- **Reliability:** The platform should remain as a reliable Grid platform. This property also covers the issue of correctness. The semantics of the applications running on this platform should be respected in every possible way and the platform should be counted as a reliable framework.

#### B. Functional Requirements

- **Application Management:** The platform should recognise and satisfy the requirements of applications. These requirements may constitute service level agreements, constraints to be satisfied or a specification how the application should be serviced. The application, in turn, may rely on a set of applications or parts of an application. For a given application, the platform should construct a workflow or ser-

vice plan meeting the requirements stipulated while meeting the governing policies. Further, the platform should act as an intermediary between the applications and between the application and resources.

- **Connectivity:** The platform should provide transparent support for distribution and connectivity of applications. With this support, applications and parts of applications will communicate with each other rather seamlessly regardless of the communication protocols or mechanisms.
- **Support for Resource Management:** The platform should act as an efficient resource manager, coordinating and orchestrating resources based on workflow or service plan for applications. The support also includes securing local or remote services and fair utilisation of resources.
- **Security Management:** In practice, considering security as a separate concern from other aspects is difficult. Security remains as a sub-system in all other core aspects of the platform. On its own as a separate concern, it is not possible to implement all the features as core of the platform. Instead, we rely on dynamically extensible scheme such that additional features will be added on demand — as we have pointed out earlier.

#### IV. DESIGN

In this section, we first outline the basic implementation aspects of the platform. We then discuss the adopted design approach and propose the overall architecture of the system.

##### A. Implementation Aspects

1) *Communication Management:* The platform will enable connectivity between applications, components and between peer platforms with support of the following:

- Dynamically changeable communication schemes/mechanisms: This ensures that any protocol or connection scheme is supported and can be introduced to the platform.

- Distribution and collection of applications, components and data: This guarantees the support for distributed execution of applications.
- Migration: The platform will be capable of migrating applications and/or data when necessary.
- Transparent Connection Introspection: This feature would guarantee that security policies could be enforced at every level of the system.

2) *Application Management:* For improved servicing of applications, the platform would require more than just the code. An application may provide an application specification, policies and any service level agreements. Hence, the platform will utilise them against servicing the application. When any of these are not provided, the platform will attempt to extract metadata (of components and applications) and will apply any platform level or generic policy and service level agreement.

Further, the platform will construct a workflow or service plan with reference to resources. In constructing the plan, the platform would use any information from policies, service level agreements and application specification.

However, the application management will not schedule applications for execution, which is left with resource management.

3) *Resource Management:* The resource management functionality involves managing the resources and scheduling applications according to workflow or service plan. Actual mapping of resources to applications takes place at this level.

4) *Security Management:* The security management provides the base trust and associated functionalities for applications and components to work on. Towards this, the platform would support the following :

- support for extensible security mechanisms for authentication, authorisation and accounting.
- support for single sign-on mechanisms
- support for adaptive, context based or ap-

- support for delegation of credentials among applications or components as appropriate
- support for profiling, logging, billing and notifications
- support for encryption

## B. Architecture of the System

1) *Approach*: In designing the platform, we follow the component-oriented development paradigm, which remains as an effective means of building flexible software systems. Towards this, our design approach draws on a previous work on configurable and reconfigurable middleware [7]. Specifically, the approach comprises two decisions: firstly we rely on a hierarchical, reflective component model and secondly we extensively use domain-specific component frameworks (CFs). The adopted component model conforms to the generic Fractal model (discussed previously) and is applied uniformly across both the platform architecture and applications. Using component frameworks helps structuring the platform architecture and provides a means for context specific reconfiguration.

In more detail, component frameworks define rules and interfaces that constrain the interactions between a set of *plug-in* components; for example, a framework for protocol stacking would constrain how protocol components are integrated to build protocol stacks. Component frameworks are represented by composite components that accept external plug-ins as subcomponents. Importantly, component frameworks include reconfiguration management functionality — implemented as a controller which handles the static and dynamic configuration of plug-ins. This allows framework designers to exploit domain-specific knowledge in order to support easy to use and consistency-preserving reconfiguration. For example, designers can expose high-level reconfiguration interfaces that rely on domain-specific abstractions, thus enhancing usability. Similarly, designers can provide reconfiguration functionality that validates changes using framework rules and invariants, thus preventing inconsistencies.

2) *Architecture*: In line with this design approach, the proposed architecture is structured as a first-level component framework (termed as platform framework) that is composed of multiple second-level component frameworks. The platform framework manages the configuration of second-level component frameworks, each of which addresses a different domain of grid middleware functionality. The architecture minimally includes second-level component frameworks that address application component management, connectivity, resource management, and security. Figure 1 illustrates the overall architecture for the proposed lightweight Grid platform, indicating the main supported functionalities.

The hierarchical structure of our architecture supports two dimensions of flexibility. First, the architecture can be customised by modifying, replacing, or adding new second-level component frameworks. In this way, different architecture customisations can be defined for different application domains and infrastructures. Secondly, a particular architecture supports customisation through external plug-ins which could be added, removed, replaced and/or reconfigured as necessary. As we have pointed above, this is feasible through the reconfiguration functionalities of component frameworks.

The *application container framework* is a key part of the architecture as it determines the programming model offered to application developers. This component framework follows the standard container-based architecture applied by Enterprise Java Beans (EJB) and CCM but, in contrast to these technologies, the framework supports extensible container-provider services. Specifically, the framework accepts as plug-ins both application components and middleware service components. A middleware service component provides a set of controllers that are applied to application components in order to transparently inject services, such as role-based authentication. In case that transparent injection is impossible or inadequate, service components may expose interfaces explicitly accessed by application components. At a minimum, the container framework includes as middleware services remote method invocation and dynamic

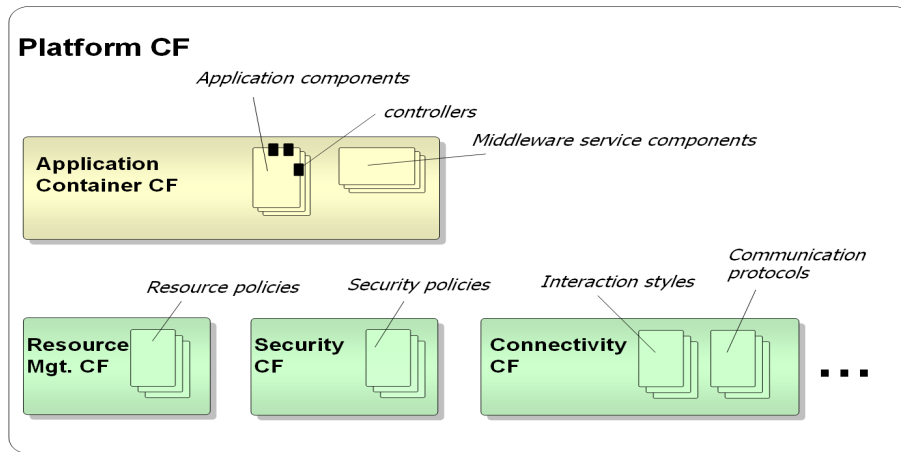


Fig. 1. Architecture for lightweight, generic Grid platform

component deployment. Service components build on functionality provided by second-level component frameworks (e.g., the role-based authentication service builds on the security component framework). The *connectivity framework* supports the establishment and management of interconnections between application components. It accepts two types of plug-ins: components that implement communication protocols, and components that build on such protocols to implement higher-level interaction styles, such as event communication, group communication, and continuous media streaming [14]. The *resource management framework* manages the resources provided by the grid fabric and supports the mapping of application components to those resources. It accepts resource allocation policies as plug-ins. The *security framework* supports authorisation, accounting, and authentication and accepts security policy components as plug-ins. Finally, the platform framework facilitates managing dependencies between second-level component frameworks and enables the dynamic addition, removal and replacement of component frameworks.

## V. USE-CASE SCENARIO: JEM3D

To demonstrate the effectiveness of the proposed platform architecture in meeting its requirements, we consider the scenario of developing and de-

ploying a high performance numerical application on the Grid. An example of such application is Jem3D [3]: a finite volume, time domain solver for the 3D Maxwell's equations modelling the propagation of electromagnetic waves. The latest version of this solver is grid-enabled and it is a component-based application, whose components include steering agents, visualisation agents, data collectors, and processing components that correspond to a geometrical decomposition of the computational domain.

This application imposes several requirements on the grid platform. Allowing users at separate workstations to participate at any time in steering the application requires that the platform supports dynamic component deployment and an extensible set of communication protocols. These requirements are respectively satisfied by the container and connectivity component frameworks. Accommodating the variations and unpredictability of the Grid fabric resources requires the platform to enable dynamic reconfiguration of both the application and the platform itself. For example, consider the data collector component that is periodically used to receive computed solutions from processing components. If the load imposed on the collector machine becomes excessive, the application may reconfigure itself to employ a hierarchical struc-

ture of collectors that exhibits better scalability. Reconfiguring applications is enabled natively by the component model that we are using and can be facilitated and managed through the introduction of a specialised, application-level, collector component framework. As another example, consider a drop in the bandwidth of the communication between two processing components. In this case, the application may reconfigure the platform to support compression of transferred data. Such a reconfiguration will involve the dynamic addition and use of a compression middleware service implemented as controllers that intercept transparently sending and receiving invocations.

Finally, simplifying the code of processing components requires that the platform supports group communication. This requirement is satisfied by the connectivity component framework, which supports the addition of multiple new interaction styles.

## VI. CONCLUSIONS

In this paper, we have outlined our initial findings in designing a generic, lightweight Grid platform. With a component-oriented methodology, we have proposed a set of requirements and features that a generic, lightweight Grid platform should support. In designing the platform, we have paid special attention to ensuring that the capabilities of the platform are extensible. To address this requirement, we based our design on a reflective component model and on composable component frameworks addressing separate concerns of the platform functionalities. The proposed design supports the development of flexible Grid platforms, capable of adapting to different contexts both statically and dynamically. This results in improved scalability of the platform. The effectiveness of the platform in meeting different requirements from the perspective of applications was demonstrated using a real-world application.

However, a number of issues remain for further investigation:

- Improving the grid platform architecture especially to adapt to more challenging contexts.
- Developing additional component frameworks and plug-ins, such as frameworks and plug-ins

related for data transfer, information management and fault-tolerance

- Self-adaptive, self-managing component frameworks and mechanisms for the platform
- Addressing context based and application specific policies and service level requirements
- Tools and supportive environments for using and porting non-Grid and legacy applications

## ACKNOWLEDGEMENTS

The authors would like to thank Maciek Malawski of Academic Computer Centre – CYFRONET, Kraków, Poland for his valuable feedback in shaping our understanding of some of the related work.

## REFERENCES

- [1] Proposal for a Grid Component Model, CoreGRID Deliverable, D.PM.002, Nov. 2005.
- [2] Rosa M. Badia, Olav Beckmann, Marian Bubak, Denis Caromel, Vladimir Getov, Ludovic Henrio, Stavros Isaiadis, Vladimir Lazarov, Maciek Malawski, Sofia Panagiotidi, Nikos Parlavantzas, and Jeyarajan Thiyagalingam. Designing a lightweight grid platform: Design methodology. Technical Report TR0020, CoreGrid Network of Excellence, 2005.
- [3] Laurent Baduel, Françoise Baude, Denis Caromel, Christian Delb, Sad El Kasmi, Nicolas Gama, and Stéphane Lanteri. A Parallel Object-Oriented Application for 3D Electromagnetism. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, New Mexico, USA, April 2004. IEEE Computer Society.
- [4] Françoise Baude, Denis Caromel, and Matthieu Morel. From distributed objects to hierarchical grid components. In *International Symposium on Distributed Objects and Applications (DOA), Catania, Italy*, volume 2888 of *LNCS*, pages 1226 – 1242. Springer, 2003.
- [5] E. Bruneton, T. Coupaye, and J. B. Stefani. Recursive and dynamic software composition with sharing. In *Proceedings of the Seventh International Workshop on Component-Oriented Programming (WCOP2002)*, 2002.
- [6] CCA Forum Home Page. The Common Component Architecture Forum, 2004. <http://www.cca-forum.org>.
- [7] G. Coulson, G.S. Blair, M. Clarke, and N Parlavantzas. The design of a highly configurable and reconfigurable middleware platform. In *ACM Distributed Computing Journal*, volume 15(2), pages 109–126, April 2002.
- [8] Enterprise Grid Alliance. Reference model. Technical Report Version 1.0, Enterprise Grid Alliance, 2005.
- [9] Stavros Isaiadis and Vladimir Getov. Integrating mobile devices into the grid: Design considerations and evaluation. In *Proceedings of EuroPar 2005 Conference*, volume 3648 of *LNCS*, pages 1080 – 1088. Springer, 2005.

- [10] Stavros Isaiadis and Vladimir Getov. A lightweight platform for integration of mobile devices into pervasive grids. In *Proceedings of HPC 2005 Conference*, volume 3726 of *LNCS*, pages 1058 – 1063. Springer, 2005.
- [11] Dawid Kurzyniec, Tomasz Wrzosek, Dominik Drzewiecki, and Vaidy Sunderam. Towards self-organizing distributed computing frameworks: The H2O approach. *Parallel Processing Letters*, 13(2):273–290, 2003.
- [12] Maciej Malawski, Dawid Kurzyniec, and Vaidy Sunderam. MOCCA – towards a distributed CCA framework for metacomputing. In *Proceedings of the 10th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS2005)*, 2005.
- [13] Object Management Group, Inc. CORBA, 2005. <http://www.corba.org/>.
- [14] Nikos Parlavantzas, Geoff Coulson, and Gordon S. Blair. An extensible binding framework for component-based middleware. In *Proceedings of 7th International Enterprise Distributed Object Computing Conference, (EDOC) 2003*, pages 252–263, Brisbane, Australia, September 2003.
- [15] Diego Sevilla Ruiz. The CORBA & CORBA Component model (CCM) page. web page. <http://ditec.um.es/~dsevilla/ccm/>.
- [16] Jeyarajan Thiyagalingam, Stavros Isaiadis, and Vladimir Getov. Towards building a generic grid services platform: A component oriented approach. In Vladimir Getov and Thilo Kielmann, editors, *Component Models and Systems for Grid Applications*, pages 39–46. Springer, 2005.
- [17] Y.M.Teo and X.B.Wang. Alice: A scalable runtime infrastructure for high performance grid computing. In *Proceedings of the IFIP International Conference on Network and Parallel Computing*, volume 3222 of *LNCS*, pages 101–109, Wuhan, China, October 2004. Springer-Verlag.