A Formal Model-Driven Approach for Grid Application Architectures

David Manset [1,2], Hervé Verjus [1]

[1] LISTIC University of Savoie, Annecy, France
[2] Maat Gknowledge, Toledo, Spain

Abstract

This paper advocates a formal approach to Grid systems development in an effort to contribute to the rigorous development of Grids software architectures. The presented approach addresses cross-platform interoperability and quality of service and applies the model-driven paradigm to a formal architecture-centric engineering method in order to benefit from formal semantic description power (using an Architecture Description Language based on π-calculus) in addition to model-based transformations. The result of such a novel combined concept promotes re-use of design models and eases developments in Grid computing.

Keywords

MDE, Software Architectures, ADLs, Grid computing, SOA.

1. Introduction

The new Grid paradigm has been described as "*a distributed computing infrastructure for advance science and engineering*" that can address the concept of "*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*" in [1, 2, 3]. This coordinated sharing is not only file exchange but can also provide direct access to computers, software, data and other system resources. Grid applications bundle different services using a heterogeneous pool of resources in a so-called *virtual organization*. This makes it very difficult to model and implement Grid applications. In addition, one of the major issues in today's Grid engineering is that it follows a code-driven approach. As a direct consequence the resulting source code is neither re-usable nor does it promote dynamic adaptation facilities as it should, since it is a representation of the Service Oriented Architecture (SOA) paradigm [4, 5].

As it is directly extracted from the definition of the Grid concept, Grid design should ensure cross-platform interoperability by providing ways to re-use concretely systems in a heterogeneous context. Having no guidelines or rules in the design of a Grid-based application is a paradox since there are many existing architectural approaches for distributed computing applications which could ease the engineering process, could enable rigorous engineering methods and could promote the re-use [6, 7] of components in future Grid developments. Although it has been proven from past experience that using structured engineering methods would ease the development process of any computing system and would reduce complexity when building large Grid systems, the hype of Grid computing has been and still is forcing brute force coding and rather unstructured engineering processes. This always leads to a loss of performance, interoperability problems and generally results in very complex systems that only dedicated developers can manage.

It is our belief that semi-formal engineering methods in current use are insufficient to tackle tomorrow's requirements in Grid computing. In this paper we aim at extending the OMG's model-driven vision by providing a set of models enacted within a novel design process. This paper focuses mainly on the implementation of the model-driven philosophy inside a formal engineering approach. The key idea of our study is to investigate established concepts in formal architecture-centric software engineering methods and model-driven concepts in order to tackle Grid engineering problems. Inside a well-defined and adapted formal approach, we investigate the enactment of our model-driven engineering process to complete our design framework and provide tools to build the next generation of Grid applications. The remainder of this paper emphasizes different aspects which are, in our view, essential to Grid engineering:

- that of offering a user-friendly vision to Grid architects by providing re-usable conceptual building blocks,
- that of hiding the complexity of the final execution platform through abstraction models, and finally
- that of promoting design re-use to ease further developments.

In order to achieve these objectives, we combine two approaches together and seek advantages from both of them. On the one hand, we use a formal semantic descriptive power to model-check Grid applications in terms of correctness and properties; on the other hand, we use a model-driven approach to promote model re-use, to hide the platform complexity and to translate abstract software descriptions to concrete usable ones.

The remainder of this paper is structured as follows. Part 1 presents the different approaches we are using. In part 1.1, we introduce briefly the model-driven paradigm through the OMG's MDA™ [8, 9, 10] architecture. In part 1.2, we introduce the architecture-centric approach and then conclude on a design approach which is a combination of these two. Part 2 explains how model-driven engineering is enacted to design Grid applications. Part 3 presents our formal architecture-centric model-driven approach and the means used to achieve it. Finally, we conclude with identifying future work to be done with the framework and state the benefits of using it.

1.1 Model-Driven Engineering

Model-driven approaches provide tools to transform models by means of transformation rules in order to describe and design complex systems at a very high level of abstraction. The main objective of such methods is to hide the complexity and constraints induced by the targeted execution platform, during the design phase. Thus, an architect can focus mainly on functional requirements of his application rather than on non-functional ones. In this paper, we discuss the OMG's Model-Driven Architecture, and then introduce our own approach dedicated to Grid applications.

1.2 The OMG MDA™ – Model Driven Architecture

The Object Management Group recently introduced the Model-Driven Architecture. This engineering method is based on generic models called *PIM* (Platform Independent Model) which are subsequently translated into *PSM* (Platform Specific Model) by mapping the *PIM* to a specific platform using transformation rules that preserve system correctness. Providing *PIMs* lowers the coupling between the design and the underlying technologies and enables interoperability at a high level, the *PIM* being completely independent of the final platform. This way of handling interoperability is totally new, interoperability being addressed at the design level rather than at the technology level. In order to enact this method, the OMG also provides a set of core standards and technologies: UML™ (Unified Modelling Language [11]), MOF™ (Meta Object Facility [12]), XMI™ (XML Metadata Interchange [13]) and the CWM™ (Common Warehouse Metamodel [14]). The main objective of the OMG is the achievement of a complete design process by introducing an AOM [15] (Adaptive Object Model) which is the sum of the above listed tools. Beyond the OMG's vision [10], the idea of *autonomic systems* [16, 17] is also appearing. An autonomic system is a system capable of discovering automatically its environment, and self-adapting as a consequence. Such systems are very high level applications that only dynamic architectures and well-defined standards for storing, recovering and managing metadata could handle. In that context the MDA™ constitutes a first attempt to tackle metadata problems, however this paper focuses not on metadata but on enacting the model-driven concept in a concrete domain such as the automated generation of Grid applications.

By convention and in order to separate clearly the concept described here from that of the OMG, we will call our approach a *model-driven engineering* approach *(MDE)* and will use a Grid-specific vocabulary. As a matter of fact, our approach differs from that of the OMG in many aspects. The OMG describes a design method based on model transformations according to meta-models, which is generic enough to fulfil many requirements in terms of modelling and re-use. In our study we define a concrete set of key models to design a Grid application from the high level descriptions of each architectural element to its final deployment on a physical grid of resources. In addition to this we specify the necessary tools to generate, transform and check models among the whole design process.

Enacting the model-driven paradigm is not an easy route to follow since as yet there are no available tools or frameworks designed with MDE in mind. For this reason one needs to be aware of available software engineering methods and paradigms. It is the position of this paper that an MDE approach can be enacted using architecture-centric methods with relatively little effort.

1.3 Architecture-Centric Engineering Approach

The architecture-centric approach [18] focuses on the software architecture description used to organise development activities. Thus every stage of the software life cycle – including specification, implementation, quality attributes and also architectural style [19, 20] – are considered as part of the process. The work on architecture-centric approaches for software development has been very fruitful during the past years, leading, amongst other results, to the proposition of a variety of Architecture Description Languages (ADLs) [21, 22, 23, 24], usually accompanied by analysis tools. These languages are used to formalize the architecture description as well as its refinement.

The benefits of using such an approach are various. They rank from the increase in architecture comprehension among the persons involved in a project (due to the use of an unambiguous language),

to reuse at the design phase and to property description and analysis (properties of the future system can be specified and the architecture analyzed to check their verification). The following *Figure [1]* introduces the architecture-centric development process.

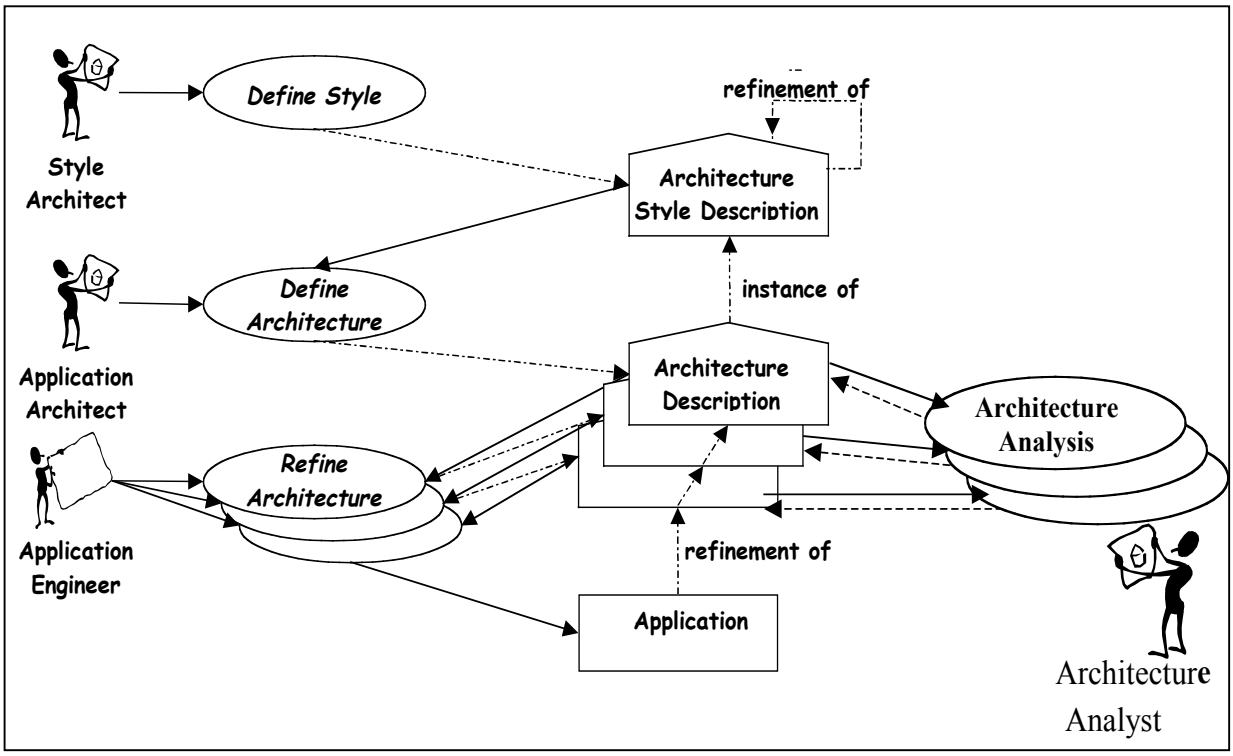

Figure 1 : The Architecture-Centric Development Process

The enthusiasm around the development of formal languages for software architecture descriptions comes from the fact that such formalisms are suitable for automated handling. As discussed later in this paper, most of these aspects are essential to the enactment of the model-driven paradigm. In our approach we use this architecture-centric vision as a strong basis to further investigations in enabling MDE. The formal dimension, in addition to adapted tools, gives our MDE approach its robustness.

1.4 A Combination of Approaches

Focusing on the model transformation aspects, one can notice similarities with the refinement concepts found in formal architecture-centric software engineering paradigms. Since the main purposes of both concepts are slightly different (the refinement of an abstract software architecture aimed at making it more and more concrete) the MDE could benefit from refinement to handle model transformations and to ensure the model's correctness. As a matter of fact, given the hypothesis that models of the system are expressed in a well-formed and standard architectural language capable of refinement, it is potentially possible to apply refinement actions on models to adapt them with respect to platform constraints. Thus investigating different platforms can lead to the creation of transformation and constraint models applicable to an abstract system model. This is the reason why formal architecture-centric software engineering concepts are very well suited to the enactment of the MDE process.

Thus our position is to combine the architecture-centric and model-driven paradigms to provide the model transformation and architecture description power for MDE. Combining these two paradigms makes them more complete with respect to Grid application domain. This combination of approaches is explained in detail below.

2 A Formal Architecture-Centric MDE Approach

Following our MDE paradigm, we address the challenge of designing, optimizing and refining a Grid abstract architecture, with respect to different criteria, in order to automatically generate a partial or complete set of Grid services. From the study we conducted in Grid engineering we consider the Grid as a SOA and define a set of properties related to Quality of Services (QoS) [25]. Using a formal approach to describe these views we build a set of models and investigate the feasibility of enacting this model-driven process. The remainder of this paper is based on a Grid domain-specific vocabulary. From the Grid requirements and issues faced in previous Grid-based and Grid applications developments, we define a set of major views of the system and their orchestration along the MDE process.

2.1 Defining the key models

In Grid engineering, design is largely effected by many constraints; these constraints are of different types and are introduced either by the architect himself when specifying properties like quality of service or by the target execution platform. Thus the MDE process dedicated to Grid engineering must take into account all of these aspects in providing the necessary views or models. As an extension to the OMG's concept, we introduce the following models and explain their interactions and transformations in the drawings 2 and 3. By proposing several models, our approach separates concerns and addresses different aspects of the application. Thus expertise management and capture is better than in classical approaches [26, 27]. Each model represents an accurate view of the system useful for conceptual understanding, analysis and refinement. Unlike the software engineering refinement process where the system architecture is iteratively refined by the architect, many of the transformations of models in our MDE are automated. The different models composing our process are defined as follows:

- *GEIM* – the Grid Environment Independent Model: an abstract architecture of the Grid application based on a formal ADL (Architecture Description Language) – a high level description using domain specific constructs.
- *GESM* – the Grid Environment Specific Model: a concrete architecture close to the final code and optimized according to a selected execution platform (a refined ADL description),
- *GECM* – the Grid Execution Constraint Model: one or more transformation models specifying which of the quality of service properties to emphasize on and how to do it, (refinement operations) and
- *GETM* –the Grid Environment Transformation Model: one or more transformation models specifying modifications to operate on the GEIM to meet the requirements of the targeted platform of execution, (refinement operations).

In order to simplify our approach, we will not discuss in detail the other models; however as a clarification of the concept these models can be defined as follows:

- *GEMM* – the Grid Environment Mapping Model: a model of translation between an abstract description and an implementation language (i.e. in charge of defining the mapping between the semantics of the GESM and a given programming language, for instance Java TM).
- *GERM* – the Grid Environment Resource Model: a model representing the physical constitution of the Grid.
-

- *GEDM* – the Grid Environment Deployment Model : a model specifying how to deploy the resulting application onto the grid set of resources and
-
- *GESA* – the Grid Environment Specific Application: the auto-generated source code of the application (i.e. obtained after GEMM translation).

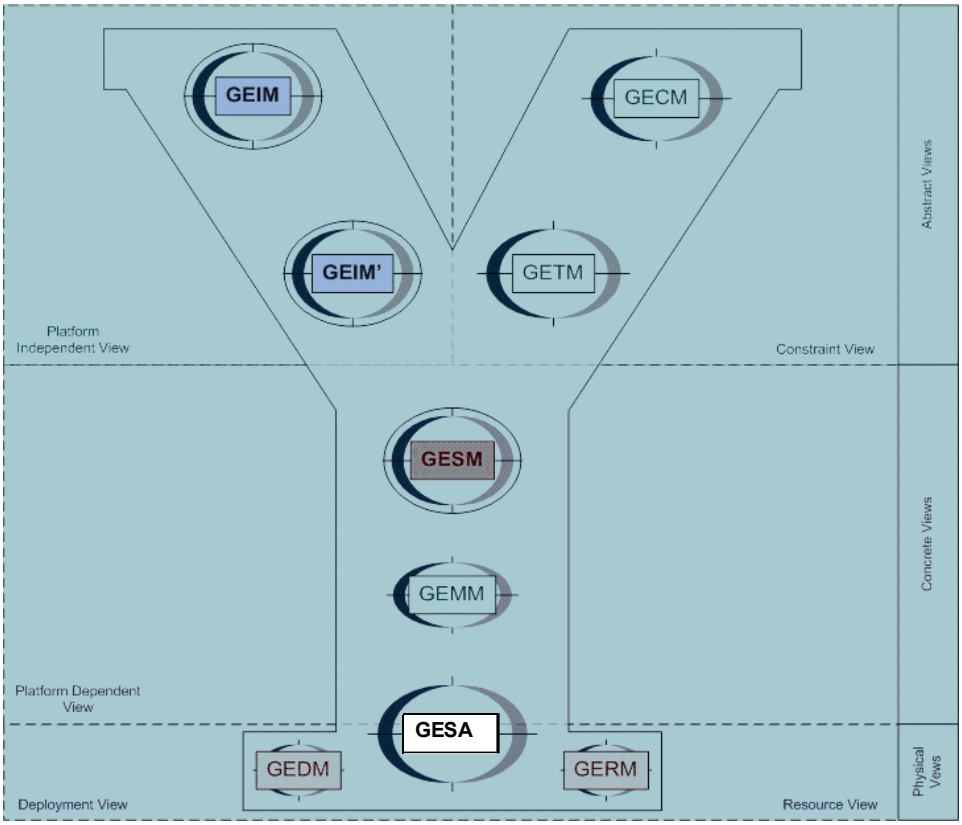

Figure 2 : MDE Key Models

Some of these models, such as the GEIM and the GECM, are defined by the software architect during design, unlike others which are automatically built from the transformation of previous models. As mentioned above, our model-driven approach uses the architecture-centric refinement concept to handle model transformations. Automating these transformations helps in decoupling architect's functional specifications and platform non-functional requirements. In other words, once the system architecture has been defined properly and meets the architect's requirements, the MDE handles the rest of the process to adapt it to the chosen platform of execution by applying models of transformations. The next section details these transformations in terms of nature and objectives, whilst introducing the whole process.

2.2 The Architecture-Centric MDE Process

As explained in the previous section, our approach enacts a set of models. These models can be of two distinct types; either the model is manually created or it is automatically obtained by transformation. The transformation itself can then be of two different types too; either the transformation is a composition of one or more refinement actions or it is a translation mapping. The following figure [3] details these different models and transformations types.

Here, we make a distinction between two major levels, one is the architecture level of transformation and the other is the implementation level of transformation. Respectively, the upper part of the drawing concerns models independent of the platform which are re-usable for further design, whereas the lower part concerns models dedicated to platforms, which are, in consequence not re-usable.

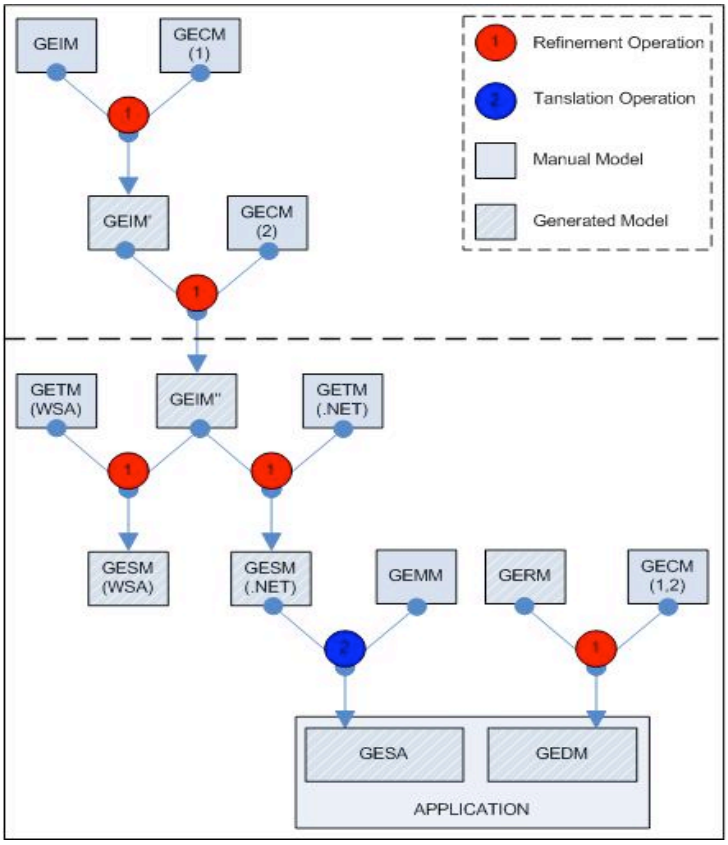

Figure 3 : MDE Process

Having introduced the approach and models, we come now to a short example to demonstrate a full workflow of model creation, transformation and enactment. Based on our knowledge of Grid systems, we take as an example a well-known service: the File Replication Service (FRS).

2.3 The File Replication Service Example

In many Grid systems, there is a service (the File Replication Service, FRS) charged with remote file replication for local processing. Whenever a service or user asks for a remote file, the FRS is called to download and create a replica of the file. Using our approach, an architect would create a GEIM model describing the topology of the service in terms of ports, connections, attributes as well as its internal behaviour. Note that the underlying conceptual view used to describe architectures in our method is the Component and Connector Style. Basing our descriptions on this paradigm allows us to tackle most distributed systems' descriptions. As shown in the following Figure [4], our architecture contains two main services, one is the "User Interface" and the other is the "File Replication Service".

The User Interface is used to request the Grid for a remote file to be copied locally. Thus both services are linked by means of Connectors named 'toFRS' and 'fromFRS'. The connectors handle data format processing and communication exchanges.

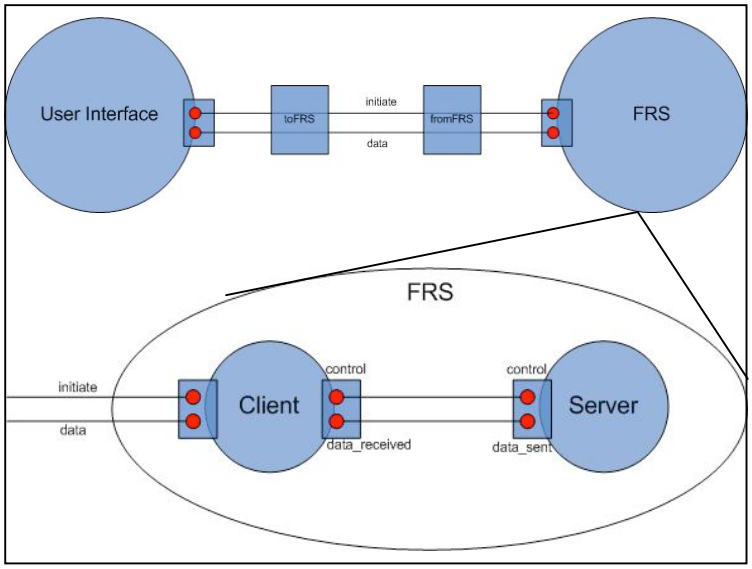

Figure 4 : The FRS, Simple Authentication

The specification of the FRS (the lower part of Figure 4) shows that the architect has described a client/server architecture with a basic authentication mechanism. An architect can assign constraints to each architectural element via the GECM model. In the current example, the architect could specify a QoS attribute for the FRS component requiring a higher level of security between the Client and the Server components (e.g. the Public Key Interchange – PKI - model [29]). The system would then apply transformations to the GEIM model corresponding to specific constraints in the GECM. After interpretation and transformation of the models, the architect would obtain a new enhanced GEIM model that satisfies his requirements as detailed in Figure 5. This initial level of transformation corresponds to the upper part of Figure 3.

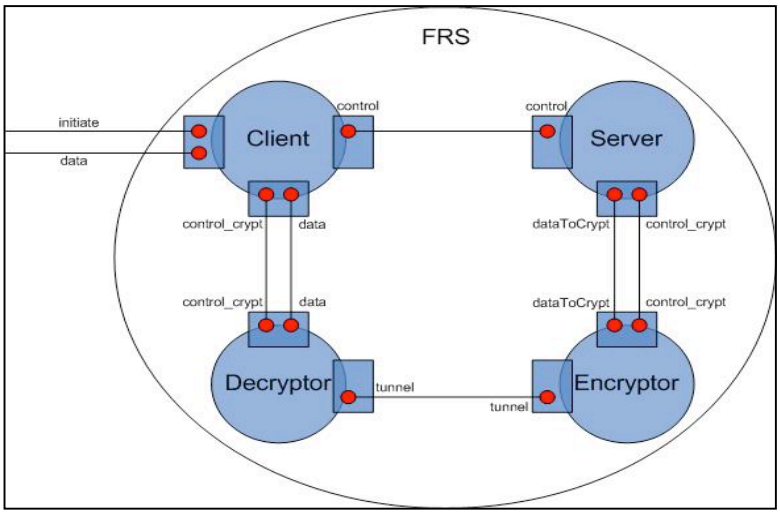

Figure 5 : The FRS, PKI-based Authentication

In the transformed specification of Figure 5, one can see the PKI security model. Here some refinement operations have been applied to the GEIM in order to make it compliant to the expressed constraints:

- "***add***": two new components were added (Decryptor and Encryptor in charge of encrypting data transmissions between the Client and the Server),

- "***remove***": the connection data has been removed between Client and Server components,

- "***unifies***": new connections were added and linked together.

Once the GEIM has been completed, a platform of execution is selected. From this choice, a known constraint model (GETM) is applied to the GEIM in order to adapt it. This latter operation is represented in the lower part of Figure 3.

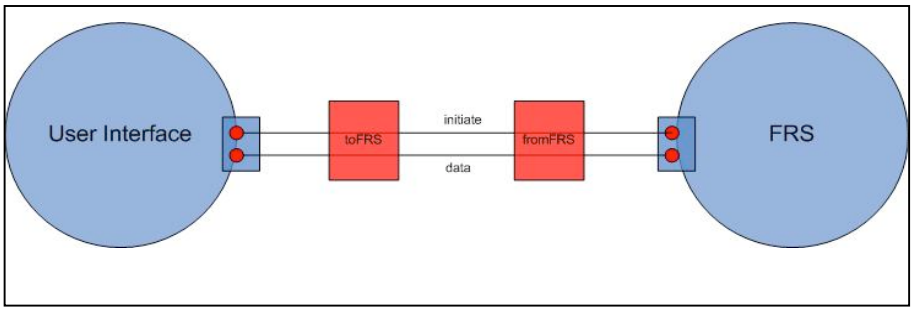

Figure 6 : The FRS, Concrete Model

Again, some refinement operations are applied to the GEIM, in our case:

-"***replace***": connectors linking elements "Graphical User Interface" and "FRS" are replaced by ones specifying concrete data format and message exchanges.

Thus the concrete system model (GESM) is obtained, ready to be translated into a suitable programming language (according to the platform), compiled and deployed.

This example has introduced two model transformations to our approach. The transformation from Figures 4 to 5 has allowed the introduction of the architect's constraints related to QoS (i.e. security). The second transformation shows a simple case of system model adaptation to a given platform. One can notice that the example GEIM is re-usable; this model is independent of any execution environment. In Grid computing, the second transformation partially resolves the well-known problem of moving from Web Services to Grid Services as a result of following the OGSA [30, 31] standard.

3 Enacting MDE, A Concrete Framework

3.1 ArchWare: the Formal Architecture-Centric Approach and Toolkit

ArchWare [32] is an architecture-centric engineering environment allowing design from formal descriptions of software. Such a formal method enables the support of critical correctness requirements and provides tools to guarantee system properties and reliable execution. By adding the model-driven dimension to this formal approach, Grid architects can benefit from a complete and reliable software engineering environment promoting system correctness, cross-platform interoperability and optimization facilities. ArchWare delivers a set of formal languages and corresponding tools to enable reliable design; among them we find a refinement specific language:

- the ***ArchWare Architecture Description Language***, defined as a layered language used to describe software architectures (supporting both structure and behaviour as well as properties like QoS, constraints, etc,

- the ***ArchWare Architecture Refinement Language*** (ARL), used to describe software architectures and refine them accordingly to transformation rules.

These languages used together with the corresponding tools constitute the ArchWare environment framework. In our approach we demonstrate how a model-driven process can be enacted using ArchWare facilities. As mentioned in section 1.3 there are noticeable similarities between MDE model transformations and architecture refinement operations. As a matter of fact, refinement consists of adapting and modifying an architecture through a set of transformation rules based on some rewriting logic. From this point of view, refinement can be seen as an architecture-level transformation within the MDE process. In the rest of this paper, we investigate the ArchWare refinement process which is, we believe, essential to the enactment of a formal architecture-centric MDE.

3.2 The ArchWare Refinement Concept

Complex systems cannot be designed in one single step. In a stepwise architecture refinement, a sequence of steps starting from an abstract model of the architecture leads to a concrete, implementation-centred model of the architecture. These refinement steps can be carried out along two directions: "vertical" and "horizontal". The concrete architecture of a large software system is often developed through a "vertical" hierarchy of related architectures. An architecture hierarchy is a linear sequence of two or more architectures that may differ with respect to a variety of aspects. For instance, an abstract architecture containing functional components related by data flow connections may be implemented in a concrete architecture in terms of procedures, control connections, and shared variables. In general, an abstract architecture is simpler and easier to understand; a concrete architecture reflects more implementation concerns. "Vertical" refinement steps add more and more detail to abstract models until the concrete architectural model has been described. A refinement step typically leads to a more detailed architectural model that increases the determinism while implying properties of the abstract model. "Horizontal" refinement concerns the application of different refinement actions at different parts of the same abstract architecture, for instance, by partitioning an abstract component into different parts at the same abstraction level.

Refinement is the formal relation that relates two different models, one abstract another more refined:

$$abstractModel \Rightarrow refinedModel$$

Refinement relations are transitive and reflexive:

$$abstractModel \Rightarrow refinedModel1 \Rightarrow refinedModel2 \Rightarrow \ldots \Rightarrow concreteModel$$

Thus, a concrete model is a refinement of an abstract model via intermediate steps:

$$abstractModel \Rightarrow concreteModel$$

An architecture concrete model can be thought of as just another architectural model in a style suitable for implementation.

Thus, the refiner [33] handles an exhaustive set of refinement actions and types. Architecture refinement can be carried out in a series of steps, a basic step being defined in terms of basic refinement actions that can transform an architecture. An architectural model can be refined into another more concrete (i.e. more refined) architectural model. A refinement step can be carried out by the application of one or many refinement action(s). The ArchWare ARL language is the formal expression of these transformations which aims at preserving upper abstract architecture properties while modifying it. This language provides a refinement calculus on architecture descriptions where

application of actions on architecture descriptions yields architecture descriptions that are related by the refinement relation.

The ArchWare refiner is a wrapper to the programming language Maude [34, 35]. Maude is a "formal" declarative programming language based on rewriting logic in the field of algebraic specification and concurrency modelling.

A refinement – as shown in Figure [7] - is a more detailed description of a parent abstraction. What makes the ArchWare project original is the facility to ensure that decompositions preserve any rigorously defined properties of the parent. At each level ArchWare supports the re-use of existing architectural models and, at the concrete level, architecture-based code generation. Furthermore, ArchWare supports the analysis of architecture descriptions with respect to syntactic and semantic checks to ensure the relative correctness of two architectures (in possibly different architectural levels) in the refinement process.

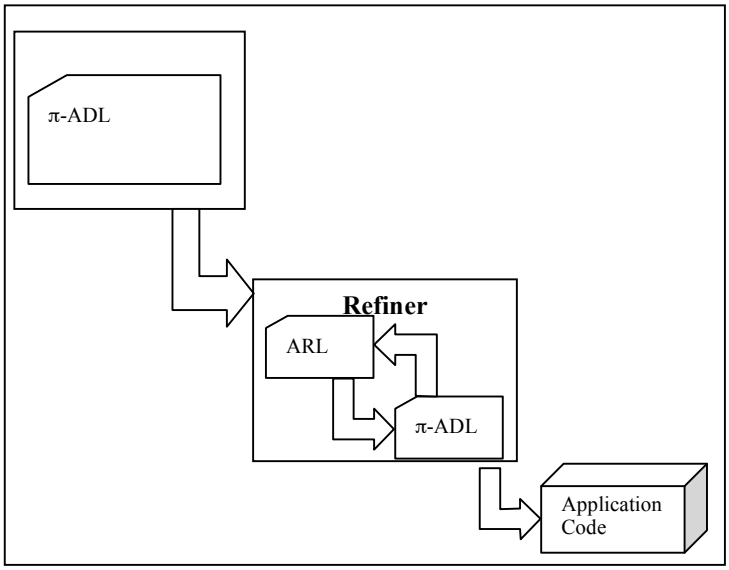

Figure [7], The ArchWare Architecture-Centric Development

In terms the description of refinement operations, a transformation model is a combination of typical refinement actions like add, remove, replace, etc. For instance, the following is the formal expression in ARL of a "remove" action:

```
on a : architecture action removeTypeArchRef is refinement (
    t : type ) {
    pre is { a::types includes? t }
    post is { a::types excludes? t }
} as { a::types excludes t }
```

3.3 A Refinement Process in an MDE perspective

In our approach, we focus on both aspects of the refinement – i.e. the "vertical" and the "horizontal". Our implementation of the model-driven concept is different and goes beyond its main purpose. Our objective is not only to refine an architecture to a concrete and "close to final" code form specific to a given hosting environment, but also to optimize it. We propose two ways of using the model-driven process in Grid engineering. The first consists of optimizing a given abstract architecture according to expressed users' requirements in terms of quality of services (QoS). The second consists of optimizing and modifying an architecture according to the targeted execution environment. As mentioned in

Figure [8], we build transformation models – directly interpreted by the Refiner – which ensures the correctness of the resulting model and satisfies platform specific requirements.

As explained in previous sections, ArchWare provides reliable refinement facilities through the Refiner tool, based on the formal language ARL. The implementation of the ARL operational semantics (the refinement actions) is achieved by Rewriting Logic in Maude where refinementActions become rewrite rules. The whole Refiner system has been formally specified in Core Maude and successfully tested in the ArchWare project. The Refiner is in fact a Maude specification of about 1500 lines, introducing system and functional theories to the understanding of the ARL language.

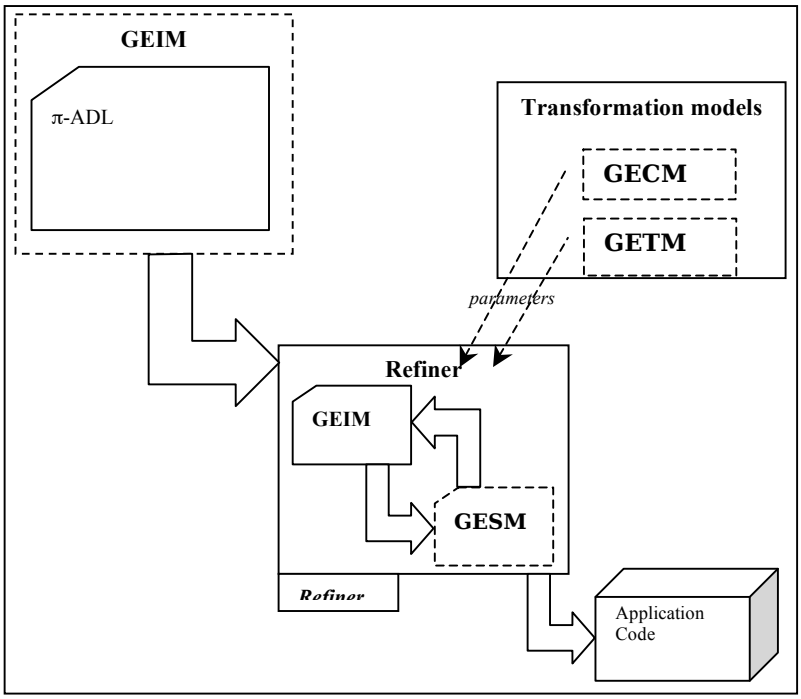

Figure [8], The Refinement Process in an MDE Perspective

Here, our objective is not to re-develop existing ways of refining models but rather to investigate the usability of existing methods in the context of MDE. From our investigations, the Refiner tool developed in ArchWare appears to be a very good candidate. The Refiner addresses the main type of transformation in our process. In that context, enabling MDE requires the expression and consideration of external models of transformations described with the ARL formalism. The Refiner accepts transformations to apply on an architecture as parameters; its internal architecture remains unmodified and still relevant to our approach. This is what is discussed in Figure [8]. Thus enacting the MDE in the ArchWare Refiner architecture is fairly straightforward; the biggest task in that process being the generation of the GECMs and GETMs models themselves.

Given the flexibility of our generic Refiner and relying on the correctness of our transformation models, the resulting tool is able to tackle every aspect of model transformations needed in Grid development.

4 Future Work

In this paper we presented a technique for specifying Grid applications by modeling and by transforming these models to automate the adaptation to specific platforms. Since the development of this approach is nearing completion, we are now focusing on QoS attributes and their corresponding views. We have started defining an extension to the ARL language in order to increase its descriptive power while not modifying its core semantics. This work is done in collaboration with Web Services practitioners, to make it re-usable in the context of Service composition. As a matter of fact, QoS

properties defined at the design level can be useful when achieving Service compositions.

In the previous sections, we described a study that illustrates how our approach can tackle QoS specifications in addition to platform requirements. Our study leads to an investigation of the most frequently used platforms in Grid computing which will result in the required GETM models. The power of our approach depends mainly on the correctness of these models; consequently great care is being taken to ensure this. As a proof of concept, the engineering framework being developed is enacting the combination of the formal architecture-centric and model-driven approaches introduced previously. In its current state, it is already capable of handling most of the presented models and transformations. In future we shall investigate case studies to validate its usability and promote its user-friendliness. Since this approach is based on the concepts of re-use and execution platform independence our engineering framework is not limited to the Grid domain. The same approach could tackle other developments based on the Service Oriented Architecture vision such as web services based applications (i.e. online traders, booking systems, video on demand system etc).

Thus, the benefits of using our approach are numerous. Application models designed using our framework are persistent and re-usable, as long as they are independent of the platform. For instance, one can use libraries and previously stored models to design new applications. The approach is scalable; one could extend the scope limitation of the framework by simply providing corresponding platform constraint models. And finally, from establishment of well-known architectural concepts, the framework brings the user to a high level of description while promoting user-friendliness through a simple semi-automated graphical user interface.

Finally, with respect to model transformations, another interesting area of future research is the development of a decision support system to help users through model-driven transformations. Indeed, some of the adaptations needed to satisfy platforms can lead to critical decisions. We are using the example described previously and others, to elicit the framework requirements.

5 Conclusion

In this paper we investigated model-driven process enactment using a formal architecture-centric approach to designing Grid systems. We analyzed the needs for this paradigm and shown clearly the feasibility of its implementation using the ArchWare tools. Our method was also applied to more elaborate models specific to the Grid domain in order to demonstrate that MDE can be used from design to deployment of an application. In this vision, our model-driven approach covers all the aspects required in the development of complex distributed systems such as Grids. The approach described here extends the OMG's vision by concentrating on the detail of models and transformations; and on categorizing them into different types. This paper is a first investigation of the model-driven paradigm enactment using reliable, established formal architecture-centric concepts.

Besides supporting the usefulness of ArchWare ARL and related tools, we are able to draw a number of general conclusions. We learned that the model-driven approach is a very useful paradigm when addressing cross-platform developments and problems of re-use but it must be dependent on a rigorous basis to be efficient. The formal dimension brought by ArchWare is one of the key points of our successful implementation. Similarly we learned that QoS attributes are not easy to quantify in models. There is a true lack of standards that could help significantly when considering resource comparisons. In the context of other engineering frameworks and given the basic concepts we have now in hand, our approach can provide directly relevant benefits to the practice of Grid system engineering. From our experience, we believe that MDE approach is an important contribution to the development of new Grid systems.

ACKNOWLEDGEMENTS

The authors wish to thank their Home Institutions and the European Commission for financial support in the current research and to gratefully acknowledge Karim Megzari for his contribution on the refinement aspects of software architectures.

REFERENCES

[1] I. Foster, C. Kesselman & S. Tueke., "The Anatomy of the Grid – Enabling Scalable Virtual Organisations", Int. Journal of Supercomputer Applications, 15(3), 2001.

[2] I. Foster, C. Kesselman J. Nick & S. Tueke., "The Physiology of the Grid – An Open Services Grid Architecture for Distributed Systems Integration". Draft document at.

[3] Dennis Gannon, Kenneth Chiu, Madhusudhan Govindaraju, Aleksander Slominski., "An Analysis of the Open Grid Services Architecture", Department of Computer Science Indiana University, Bloomington, IN andSteven Tuecke, Karl Czajkowski, Jeffrey Frey, Ian Foster, Carl Kesselman, Steven Graham., "GRID Service Specification" (2002).

[4] SOA – Service-Oriented Architectures An Introduction. See http://www.developer.com/design/article.php/1010451 and

http://www.developer.com/services/article.php/1014371.

[5] Kishore Channabasavaiah, Kerrie Holley., "Migrating to a Service-Oriented Architecture", IBM.

[6] Aoife Cox., "An Exploration of the Application of Software Reuse Techniques to the Location of Services in a Distributed Computing Environment", thesis report, University of Dublin, September 2004.

[7] Thomas Hemmann., "On the Reuse of Software Engineering, Reuse Approaches and Techniques in Knowledge Engineering".

[8] Object Management Group, MDA website, http://www.omg.org/mda/ and http://www.omg.org/mda/specs.htm

[9] "MDA Guide Version 1.0.1", OMG

[10] Anneke Kleppe, Jos Warmer, Wim Bast, "MDA Explained: The ModelDriven Architecture™: Practice and Promise", Addison Wesley Professional (2003), and Rick Kazman, Steven G. Woods, S. Jeromy Carrière, "Requirements for Integrating Software Architecture and Reengineering Models: CORUM II". Software Engineering Institute, Carnegie Mellon University

[11] Object Management Group, "Unified Modeling Language Specification, Version 1.4", September 2001.

[12] Meta Object Facility, Document -- ptc/04-06-11 (MOF 2.0 Core Specification), See http://www.omg.org/cgi-bin/doc?ptc/2003-10-04

[13] XML Metadata Interchange (XMI), version 1.2, See : http://www.omg.org/cgi-bin/doc?formal/2002-01-01

[14] Common Warehouse Metamodel, See : http://www.omg.org/technology/documents/formal/cwm_mip.htm

[15] Adaptative Object Model, See : http://adaptiveobjectmodel.com/

[16] "Enabling Autonomic, Self-Managing Grid Applications", Z. Li, H. Liu, and M. Parashar, The Applied Software Systems Laboratory, Dept of Electrical and Computer Engineering, Rutgers University.

[17] "Enabling Autonomic Grid Applications: Requirements, Models and Infrastructure", M. Parashar, H. Liu, Z. Li, C. Schmidt, V. Matossian, N. Jiang. The Applied Software Systems Laboratory, Dept of

Electrical and Computer Engineering, Rutgers University.

[18] “An Introduction to Software Architecture”. D.Garlan and M.Shaw. Advances in Software Engineering and Knowledge Engineering, Volume 1, World Scientific Publishing Co. 1993.

[19] Garlan, D. “What is Style?”, Proceedings of Dagshtul Workshop on Software Architecture, February 1995, and G.Abowd, R.Allen and D.Garlan., “Formalizing Style to Understand Descriptions of Software Architecture”. ACM Transactions on Software Engineering and Methodology, pp. 319-364. October 1995.

[20] Garlan, D., Monroe, R. and Wile, D., “Exploiting style in architectural design environments”, Proceedings of SIGSOFT’94 Symposium on the Foundations of Software Engineering, pp 179-185, ACM Press, December 1994, and G.Abowd, R.Allen and D.Garlan., “Using Style to Give Meaning to Software Architectures”. Proceedings of SIGSOFT'93 : Foundations Software Eng., ACM. New York, 1993.

[21] F.Oquendo, S.Cimpan, and H.Verjus., “The ArchWare ADL: Definition of the Abstract Syntax and Formal Semantics.” ARCHWARE European RTD Project IST-2001-32360. Deliverable D1.1b. December 2002.

[22] Medvidovic, N., and Taylor, R.N., “A Classification and Comparison Framework for Software Architecture Description Languages”, Technical Report UCI-ICS-97-02, Department of Information and Computer Science, University of California, Irvine, February 1997.

[23] Neno Medvidovic., “A Classification, Comparison Framework for Software Architecture Description Languages”. Technical report, Dept of Information, Computer Science, Uni of Irvine California 96.

[24] Sorana Cîmpan (InterUnec), Fabien Leymonerie (Thésame), Flavio Oquendo, “State of the art on Architectural Styles: Classification and Comparison for Software Architecture Description Languages”, (InterUnec), 2003.

[25] Rikard Land, Mälardalen,. “Improving Quality Attributes of a Complex System Through Architectural Analysis – A Case Study”, University, Department of Computer Engineering.

[26] Luigi Guadagno and Xiaoping Jia., “PSM Advisor: A Method for the Selection and Evaluation of Platform-Specific Models”, DePaul University, School of Computer Science, Telecommunications and Information Systems, Chicago.

[27] “Using object-oriented typing to support architectural design in the C2 style”. N. Medvidovic, P. Oreizy, J. E. Robbins, R. N. Taylor 4th ACM Symposium on the Foundations of Software Engineering (SIGSOFT), Oct 1996.

[29] Public Key Interchage, See : http://www.ietf.org/html.charters/pkix-charter.html

[30] I. Foster, D. Gannon., “The Open Grid Services Architecture Platform”, GWD-R (draft-ggf-ogsa-platform-2), February 2003.

[31] I. Foster, C. Kesselman, S. Tueke, K. Czajkowski, J. Frey, S. Graham, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt, “Open Grid Services Infrastructure (OGSI)” Version 1.0, June 2003.

[32] The EU funded ArchWare – Architecting Evolvable Software - project : http://www.arch-ware.org.

[33] Chaudet C., Megzari K., Oquendo F., A Formal Architecture-Driven Approach for Designing and Generating Component-Based Software Process Models, 4th World Multiconference on Systemics, Cybernetics And Informatics (SCI 2000), Track on Process Support for Distributed Team-based Software Development (PDTSD), Orlando, Floride, USA, July 2000, pp. 700-706 and

[34] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, and J. Meseguer. «Metalevel Computation in Maude ». In 2nd International Workshop on Rewriting Logic and its Applications (WRLA'98). Electronic Notes in Theoretical Computer Science, Vol. 15. 1998, and

[35] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J.F. Quesada. «Maude:

Specification and Programming in Rewriting Logic ». User's Guide Maude 99, and M. Clavel, F. Durán, S. Eker, J. Meseguer, and M.-O. Stehr. «Maude as a Formal Meta-Tool. In World Congress on Formal Methods » (FM'99). Lecture Notes in Computer Science, Vol. 1709, pp. 1684-1703. 1999 and M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J.F. Quesada. «A Maude Tutorial ». Manuscript, Mars 2000.