Composing and Deploying Grid Middleware Web Services using Model Driven Architecture

Aniruddha Gokhale[1] and Balachandran Natarajan[1]

Institute for Software Integrated Systems, Vanderbilt University,
PO Box 36, Peabody,
Nashville, TN 37221
{a.gokhale, b.natarajan}@vanderbilt.edu

Abstract. *Rapid advances in networking, hardware, and middleware technologies are facilitating the development and deployment of complex grid applications, such as large-scale distributed collaborative scientific simulation, analysis of experiments in elementary particle physics, distributed mission training and virtual surgery for medical instruction. These predominantly collaborative applications are characterized by their very high demand for computing, storage and network bandwidth requirements. Grid applications require secure, controlled, reliable, and guaranteed access to different types of resources, such as network bandwidth, computing power, and storage capabilities, available from multiple service providers. Moreover, they demand multiple, simultaneous end-to-end quality of service (QoS) properties, such as delay guarantees, jitter guarantees, security, scalability, reliability and availability guarantees, and bandwidth and throughput guarantees, for their effective operation.*

Existing grid infrastructure middleware, such as Globus, ICENI, and Legion, offer simplified application programming interfaces (APIs) for deploying grid applications. However, grid applications using these APIs become tightly coupled to their respective middleware infrastructure creating an impediment to interoperability, portability, maintenance and extensibility. Moreover, existing grid infrastructure middleware offer only the means and not the solutions for reserving and securely accessing resources. Thus, the onus of actually reserving and provisioning these different resources while also ensuring end-to-end QoS still lies on the grid applications. These low-level concerns increase the accidental complexities incurred developing complex grid applications.

A promising solution to remedy these problems is to use the Model-Integrated Computing (MIC) paradigm to model the resource and QoS requirements of grid applications and integrate it with grid component middleware. MIC tools can perform feasibility analysis of the application's resource and QoS requirements and determine the right resource provisioning strategies. The MIC tools can subsequently synthesize, assemble and deploy QoS-enabled grid middleware components configured with the resource reservation and service provisioning strategies tailored to the needs of the grid application, while also delivering end-to-end QoS. Moreover, MIC tools can also be used to expose the deployed grid middleware as a Web service thereby decoupling grid applications from any particular middleware API.

The paper provides three contributions to the study of a model-driven approach to assembling and deploying QoS-enabled grid middleware capable of provisioning resources and delivering QoS end-to-end to grid applications. First, we describe our Grid component middleware called GriT, which is based on the Object Management Group's (OMG) CORBA Component Model (CCM). Second, we explain how we are using the OMG Model Driven Architecture (MDA), which is a standardization of the MIC technology, to develop a tool called CoSMIC. CoSMIC is used to simplify composition of semantically compatible components of GriT to provide end-to-end QoS and resource guarantees to grid applications. Third, we show how the CoSMIC tools expose the deployed GriT middleware as a Web service that enables grid applications to use ubiquitous web protocols, such as Session Initiation Protocol (SIP) to create, join, or leave collaborative grid applications.

Keywords: Model-Integrated Computing, Model Driven Architectures, CORBA Component Model, Grid Computing, QoS

1 Introduction

The term *grid applications* applies to a special class of distributed applications that have very high computing and resource requirements, and are often collaborative in nature. Grid computing [1] is an emerging paradigm that seeks to harness the power of the internet and the sophisticated resources spread across it, such as super computers, storage devices, and others to support grid applications.

The grid computing paradigm envisages a distributed hardware infrastructure and a wide range of software infrastructure for services, programming models, tools, programming languages and methodologies capable of providing the massive computational requirements (Petaflops) and massive storage capacities (Petabytes) required by grid applications. Moreover, it is also expected to support high-fidelity, real-time collaboration between geographically distributed *virtual organizations* (VOs) [2], that comprise researchers, scientists, other users, and organizations.

For grid applications to operate effectively, however, they simultaneously require:

- secure and controlled access to many different resources available from multiple resource and service providers, including networking resources (such as bandwidth and buffers), operating system resources (such as threads, CPU and kernel buffers), storage resources (such as high performance databases and RAIDs), computing resources (such as supercomputers), display resources capable of 3-D rendering, and many other specialized types of resources, such as sensors, telescopes, oscilloscopes, and other electronic equipment

– end-to-end multiple quality of service (QoS) properties, such as *delay guarantees*, *jitter guarantees*, *security*, *scalability*, *high reliability and availability guarantees*, and *bandwidth and throughput guarantees* to grid applications.

The *infrastructure middleware* that hosts grid applications is called a *Computational Grid* or simply a *Grid* [1]. Examples of grid infrastructure middleware include Globus [3], Legion [4] and ICENI [5] among others. The Grid provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities useful for *distributed super-computing*, *on-demand computing*, *high-throughput computing*, *data-intensive computing* and *collaborative computing*.

Although existing grid infrastructure middleware seems suited to supporting next-generation grid applications, however, developing distributed grid applications using these is fraught with the following challenges.

Challenge 1: Tight coupling with grid infrastructure middleware: Grid applications are developed using one of existing grid infrastructure middleware technologies, such as Globus, Legion, ICENI, and others making them tightly coupled to the underlying middleware. However, with advances and sophistication in grid middleware technologies, it is imperative for grid applications to avail of these advances. This in turn implies that grid applications be seamlessly portable across different middleware without significantly affecting existing applications thereby preserving investments. This proliferation of grid middleware choices has raised the level of accidental complexity by increasing the amount of effort required to interoperate and port applications between grid middleware technologies.

An effective approach to decouple grid applications from the underlying grid middleware is to expose the grid middleware as a Web service [6]. Grid applications can then use *standards*-based ubiquitous web protocols such as *http* and Session Initiation Protocol (SIP) to access the underlying grid middleware.

Challenge 2. Accidental complexities in integrating software systems. To reduce lifecycle costs and time-to-market, application developers are attempting to assemble and deploy distributed grid applications by selecting the right set of compatible grid middleware components, which in itself is a daunting task. The problem is further exacerbated by the existence of myriad strategies for configuring and deploying the underlying component middleware to leverage the environment advantages. Moreover, integrating applications using multiple middleware technologies demands multiple skill sets which makes the task even more complicated. Application developers therefore spend non-trivial amounts of time debugging problems associated with the selection of incompatible strategies and components. What is needed is an integrated set of processes and tools that can (1) select and validate a suitable configuration of middleware components and (2) generate optimized Web service configurations automatically.

Challenge 3: Satisfying multiple quality of service requirements simultaneously: As noted earlier, grid applications demand varying degrees and forms of QoS

support from their grid middleware. For example, collaborative scientific applications involving geographically dispersed scientists, engineers, and physicists working on real-time experiments and data require the infrastructure to be efficient, predictable, scalable, secure, and fault tolerant. Owing to the complex nature of these QoS requirements, it is not feasible for a single grid infrastructure middleware to provide an end-to-end solution that addresses all these challenges. Instead, highly configurable, flexible, and optimized higher-level, grid middleware components based on *standards*, such as CORBA Component Model (CCM) [7], must be used to assemble and deploy middleware tailored to the needs of the grid application.

Challenge 4: Lack of well-defined patterns for resource reservation: As mentioned earlier, grid applications require simultaneous access to several different types of resources available from multiple resource and service providers that own them. These service providers include internet service providers (ISPs), storage service providers (SSPs), content service providers (CSPs), and others. For example, a distributed virtual surgery application involving geographically dispersed doctors, radiologists, medical professionals, and medical students will require high bandwidth for collaboration, large storage databases to hold patient records and radiology images, expensive display devices for precise 3-D modeling and rendering of images, virtual reality equipment for simulating surgeries, and telephony equipment to maintain multi-leg call sessions.

Applications that require these resources must maintain Service Level Agreements (SLAs) with each individual service provider that provide the resources and services. Moreover, today's grid applications must authenticate themselves with each service provider everytime they access resources owned by the provider.

Conventional grid infrastructure middleware provide only the means to securely access the resources from different service provider. However, the responsibility of reserving and accessing the resources is still the responsibility of the grid applications. A possible solution to address this problem is for the grid middleware to provide a set of generic resource reservation strategies that grid applications can use. However, such a solution fails to serve the needs of all grid applications, each of whom might have differing end-to-end resource and QoS needs. What is therefore needed is an ability to compose patterns-based strategies for multiple resource reservations while assuring the end-to-end QoS requirements of the grid aplications. Moreover, these strategies should be deployed within the grid middleware and made available to the grid applications as a Web service.

Challenge 5: Provisioning and managing resources is hard: As mentioned in challenge 4, grid applications must make reservations for several different resources while ensuring that the end-to-end QoS requirements are met. Even if this problem is resolved by deploying custom resource reservation strategies within the grid middleware as outlined above, provisioning and managing multiple resources from multiple providers is a daunting task that existing grid infrastructure middleware currently do not handle and leave it to the grid application.

What is required is an ability to model the resource and QoS requirements of grid applications using Unified Modeling Language (UML) [8] modeling tools or Statecharts [9]. Model analysis tools can be used to determine if provisioning such a system is feasible or not. If it is, then a separate set of tools can synthesize the appropriate resource provisioning and management strategies composed from a library of higher-level QoS-enabled grid middleware components.

A promising way to address the challenges developing grid applications described above is to use *Model-Integrated Computing* technologies [10]. Understanding how to integrate Model-integrated Computing (MIC) and grid component middleware is essential to resolve the configuration, management, and deployment challenges of deploying QoS-enabled grid middleware as Web services. This paper provides the following three contributions toward the successful integration of Model-Integrated Computing and grid component middleware that is essential to develop QoS-enabled Web services to address the challenges presented above:

- We illustrate how the Model-Integrated Computing paradigm can be applied to simplify the development of large-scale grid applications that integrate components of our QoS-enabled reusable component middleware, called Grid TAO (GriT) [11].
- We discuss how emerging standards, such as the Object Management Group (OMG)'s Model Driven Architecture (MDA) [12] and the CORBA Component Model (CCM) [7] can be used to provide a standards-based approach to assemble and deploy grid middleware Web services.
- We describe how QoS-enabled component middleware enables modeling and synthesis tools to rapidly develop, assemble, and deploy flexible Web services that support heterogeneity, yet can be tailored readily to meet the needs of grid applications with multiple simultaneous QoS requirements.

The rest of the paper is organized as follows: Section 2 presents an overview of the MIC paradigm and MDA, and describes our MDA tool, called CoSMIC; Section 3 explains the GriT component middleware architecture; Section 4 explains how the CoSMIC tools is used to compose, assemble, and deploy GriT middleware components as a Web service that is tailored to the needs of grid applications; Section 5 describes related research; and finally Section 6 provides concluding remarks.

2 Model Integrated Computing and Component Middleware Synthesis: The Key to Developing Next Generation Grid Applications

Model Integrated Computing (MIC) [10] is a paradigm for expressing application functionality and QoS requirements at higher levels of abstraction than is possible using third-generation programming languages, such as Visual Basic, Java, C++, or C#. In the context of grid applications, MIC tools can be applied to

1. **Analyze** different—but interdependent—characteristics of system behavior *e.g.* resource requirements, such as network bandwidth, CPU processing speed, and storage capacity, and QoS requirements, such as scalability, predictability, safety, and security. Tool-specific model interpreters translate the information specified by models into the input format expected by analysis tools. These tools can check whether the requested behavior and properties are feasible given the constraints.
2. **Synthesize** platform-specific code that is customized for specific grid middleware and grid application properties, such as end-to-end timing deadlines, throughput requirements of simulations, and authentication and authorization strategies modeled at a higher level of abstraction.

The Object Management Group (OMG) has recently adopted the Model Driven Architecture (MDA) [12] to standardize the integration of MIC paradigm with component middleware technologies and web services. This section provides an overview of the MIC and MDA technologies. We then describe a tool called CoSMIC [13] we are developing to model grid application resource and QoS requirements.

2.1 Overview of Model-Integrated Computing

Model-Integrated Computing (MIC) [10] is a development paradigm that applies domain-specific modeling languages systematically to engineer computing systems ranging from small-scale real-time embedded systems to large-scale distributed enterprise and grid applications. MIC provides rich, domain-specific modeling environments, including model analysis and model-based program synthesis tools [14]. In the MIC paradigm, application developers model an integrated, end-to-end view of the entire application, including the interdependencies of its components. Rather than focusing on a single, custom application, therefore, MIC models capture the essence of a class of applications. MIC also allows the modeling languages and environments themselves to be modeled by so-called *meta-models* [15], which help to synthesize domain-specific modeling languages that can capture the nuances of domains they are designed to model.

When implemented properly, MIC technologies help to:

- Free application developers from dependencies on particular software APIs, which ensures that the models can be used for a long time, even as existing software APIs become obsolete and replaced by newer ones.
- Provide correctness proofs for various algorithms by analyzing the models automatically and offering refinements to satisfy various constraints.
- Synthesize code that is highly dependable and robust since the tools can be built using provably correct technologies.
- Rapidly prototype new concepts and applications that can be modeled quickly using this paradigm, compared to the effort required to prototype them manually.
- Save organizations significant amounts of time and effort, while also reducing application time-to-market.

Popular examples of MIC tools being used today include the Generic Modeling Environment (GME) [14] and Ptolemy [16] (which are used primarily in the real-time and embedded domain) and UML/XML tools based on the OMG Model Driven Architecture (MDA) [12] (used primarily in the enterprise application domain thus far).

As shown in Figure 1, MIC uses a set of tools to

- Analyze the interdependent features of the system captured in a model and
- Determine the feasibility of supporting different non-functional system aspects, such as QoS requirements, in the context of the specified constraints.

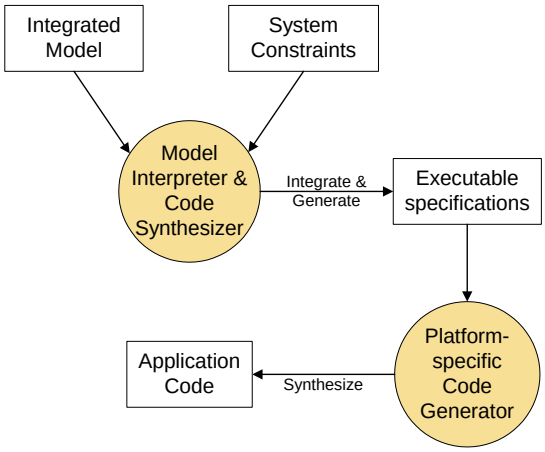

Fig. 1. The Model-Integrated Computing Process

Another set of tools then translates models into executable specifications that capture the platform behavior, constraints, and interactions with the environment. These executable specifications can in turn be used to synthesize application software.

2.2 Overview of the OMG Model Driven Architecture

The OMG MDA [12] defines standard ways to address many of the challenges facing complex applications, such as the grid applications, outlined in Section 1. The MDA builds upon years of research on model-integrated computing [10, 9, 17] to provide standard modeling notations based on the Unified Modeling Language (UML) [8]. Figure 2 illustrates the structure of the MDA.

The MDA defines platform-independent models (PIMs) and platform-specific models (PSMs) that streamline platform integration issues and protect investments against the uncertainty of changing platform technology. These two levels of models can be differentiated as follows:

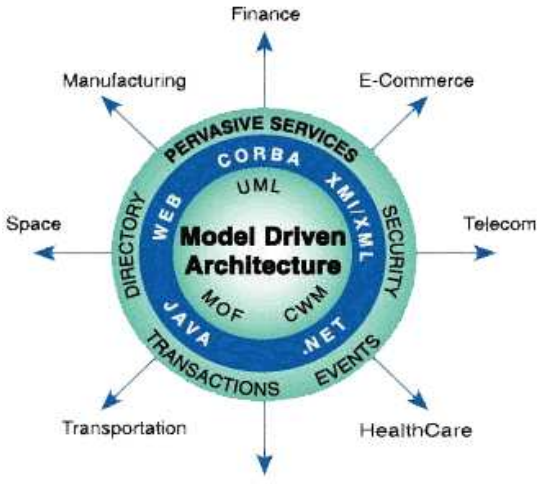

Fig. 2. Overview of the OMG Model Driven Architecture (Copyright OMG, reproduced by permission)

- The PIMs describe at a high-level how applications will be structured and integrated, without concern for the middleware/OS platforms or programming languages, on which they will be deployed. PIMs provide a formal definition of an application's functionality, as well as a representation of the application as a computation-independent business model, grid experiment model or a military strategy, also referred to as a *Domain Model*. For example, resource and QoS requirements of grid applications can be modeled generically using modeling tools based on UML.
- The PSMs are so-called *constrained* formal models since they express platform-specific details. The PIM models are mapped into PSMs via translators. For example, the generic operation that is specified in the PIM could be mapped and refined to the domain-specific operation, such as limits on response time accessing a resource, in the underlying Real-time CORBA platform.

Both PIM and PSM descriptions of applications are formal specifications built using modeling standards, such as UML, which can be used to model application functionality and system interactions. The MDA also defines a platform-independent meta-modeling language that allows platform-specific models to be modeled at an even higher level of abstraction.

2.3 Component Synthesis with Model Integrated Computing (CoSMIC)

Figure 3 illustrates how we are applying the MIC technology to build a MDA-based tool called CoSMIC (which stands for *Component Synthesis with Model*

Integrated Computing) suitable for modeling resource and QoS requirements of grid applications. The CoSMIC tool composes grid middleware tailored to grid application requirements from functional building blocks of the Grid TAO (GriT) middleware, which is explained in Section 3. Moreover, CoSMIC tools expose the deployed grid middelware as a web service.

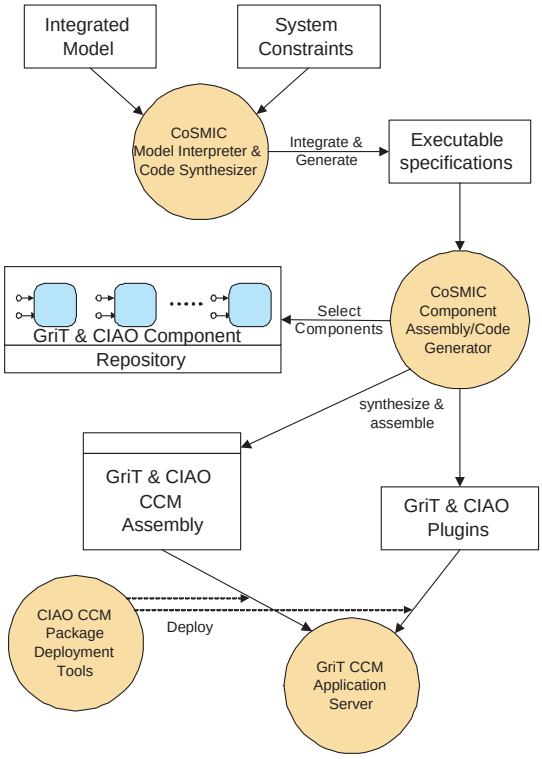

Fig. 3. The CoSMIC MDA Tool

In the CoSMIC approach, higher-level modeling languages, such as UML [8] are used to model grid application QoS and resource requirements. CoSMIC analysis tools then determine the feasibility of the requirements. Once a feasibility analysis is complete, CoSMIC translator tools are used to synthesize pattern-oriented, semantically compatible grid middleware code composed from a set of reusable, QoS-enabled GriT components. The decision on which patterns make most sense are made by the CoSMIC synthesis tools based on the input models and constraints.

3 The Grid TAO (GriT) Middleware Architecture

This section describes our next generation Grid component middleware called Grid TAO (GriT) [11]. GriT enhances the Component Integrated ACE ORB (CIAO) [18, 13] middleware, which is our CCM [7] implementation of the *The ACE ORB* (TAO) [19, 20]. TAO is an open-source, high-performance, highly configurable CORBA ORB that implements key patterns [21] to meet the demanding QoS requirements of distributed systems.

Figure 4 illustrates the components of the GriT middleware architecture. Below we explain each component of the architecture in detail.

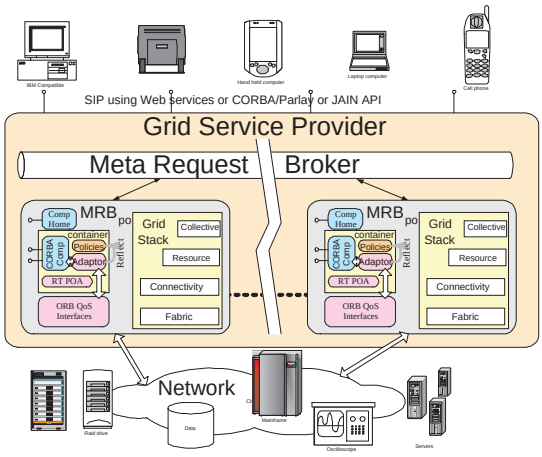

Fig. 4. Grid TAO (GriT) Middleware Architecture

Grid Service Provider (GSP) The GriT middleware comprises the notion of a Grid Service Provider (GSP) similar to other service providers such as ISPs, SSPs, CSPs and other providers providing specialized services such as access to advanced displays, virtual reality equipment, telescopes, oscilloscopes, etc. A fundamental difference between a GSP and other service providers is that a GSP is an abstract notion. The GSP does not actually own resources. The goal of the GSP is to provide a standard, unified view of resources to grid applications thereby eliminating the need for grid applications to require the knowledge and location of individual resource service providers.

Moreover, the GSP provides a *single sign-on* capability for grid applications, which eliminates the need for multiple SLAs and authentication mechanisms with multiple service providers. Applications use the GSP to delegate the responsibility of authenticating themselves with the individual specialized service providers. In addition, the GSP offers its user interface as a standard web ser-

vice, thereby enabling grid clients to use techniques such as SIP to create, join, or leave collaborative grid applications. Figure 5 illustrates the concept of a GSP.

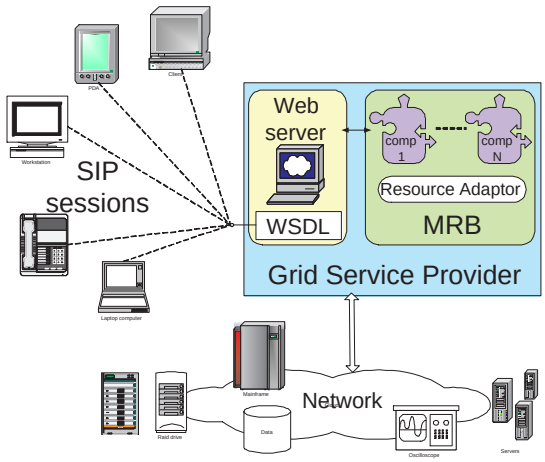

Fig. 5. Overview of Grid Service Provider (GSP)

Meta-Resource Broker Architecture At the heart of the GSP is a Meta Resource Broker (MRB), which is an enhanced the Common Object Request Broker Architecture (CORBA) Object Request Broker (ORB), that encapsulates resources from multiple providers as CORBA objects. The MRB exemplifies the actual GriT middleware. The MRB provides applications with standards-based, uniform interfaces and mechanisms to access and manage the underlying resources, and to create or join new or existing collaborative sessions, respectively.

The MRB is based on the CCM, where components represent the policies to manage the *virtual* resources . The resources are virtual since the GSP does not actually own any resources, but maintains only abstractions of them. These components therefore serve as a resource *proxy* of the actual resources thereby providing a uniform view to client applications. The internal structure of the MRB is illustrated in Figure 6.

The MRB mediates requests for different services and resources on behalf of grid applications and delivers them with the resources and guaranteed quality of service (QoS). This is accomplished by the MRB delegating the task of looking up individual resources required by the grid applications to MRB *part objects*. Figure 7 illustrates the use of part objects as defined in the Data Parallel CORBA (DP-CORBA) [22] specification.

When a grid application makes a reservation request to the GSP for all the different resources it needs and the QoS guarantees, this request is handed down by the GSP to its underlying MRB parallel object. The MRB parallel object

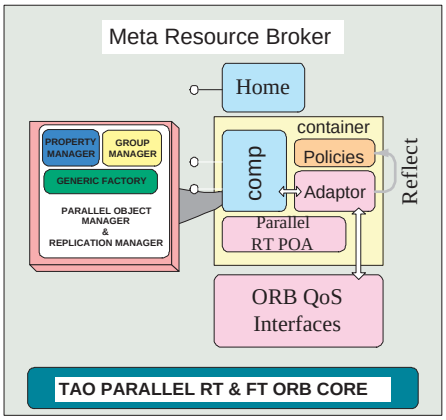

Fig. 6. Meta-Resource Broker Internals

will in turn partition the request using the techniques described in the DP-CORBA specification and the *Data Reorganization Effort* (www.data-re.org), such as *block distribution* or *cyclic distribution*. The partitioned request is then handed over to the MRB part objects. Each MRB part object is responsible to discover the appropriate resources that can meet the application's QoS requirements. This discovery process is performed in parallel thereby providing a highly scalable and predictable solution to determine the feasibility of resources and service provisioning.

The mechanism of resource discovery outlined above is akin to a *nested transaction*. If any one of the *child* transaction *i.e.,* resource discovery undertaken by a part object, is unsuccessful, then the *parent* transaction *i.e.,* the request initiated by the MRB parallel object, is rolled back.

If the request for resources is feasible, then the result of the MRB part object resource discovery operation is a collection of resources required by the application that provide it with the QoS guarantees. This collection of virtual resources is subsequently managed by the MRB as another *parallel object*. It is then upto the grid application to efficiently utilize these resources, although GriT will manage them on behalf of the grid application.

Figure 7 illustrates how the MRB reserves and manages different types of *virtual* resources, which are high-level abstractions of resources, such as network bandwidth, databases, or supercomputers, belonging to different service providers.

[11] provides detail information on the GriT middleware architecture.

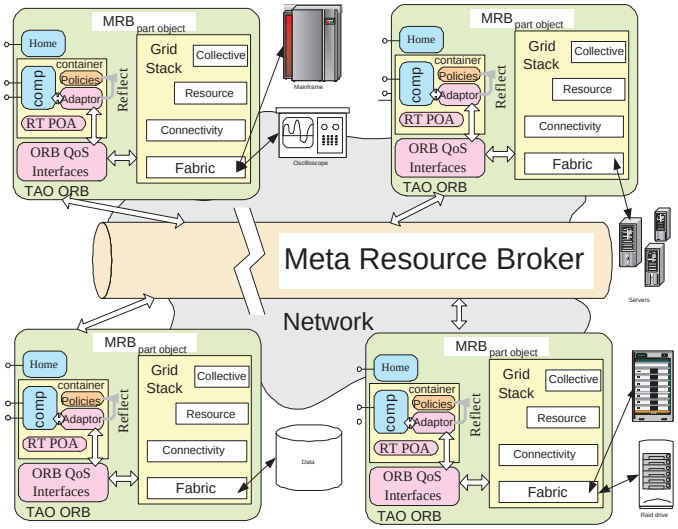

Fig. 7. Meta-Resource Broker Architecture

4 Resolving Grid Application Challenges Using Model Integrated Computing

This section describes how the MDA-based CoSMIC tool can be used to develop, assemble and deploy GriT middleware components as a web service, which is tailored to the needs of complex grid applications. In particular, we show how the application functionality specified as models can be used to synthesize new components that implement the web service, as well as to assemble them with semantically compatible reusable components provided by the GriT middleware.

First we describe how we provision the middleware components with appropriate strategies to reserve and manage the resources. Next we describe how these deployed components are made available as a web service so that grid applications can use it via standard web protocols such as *http* and SIP.

4.1 Model Driven Grid Middleware Deployment

Context. Today's grid applications are built using conventional grid infrastructure middleware, such as Globus or Legion. These middleware provide the APIs required to reserve the resources and securely access them.

Problem. Conventional grid infrastructure middleware make it hard to develop next-generation grid applications for the following reasons outlined in Section 1.

1. tight coupling with grid infrastruture middleware
2. accidental complexities in integrating software systems

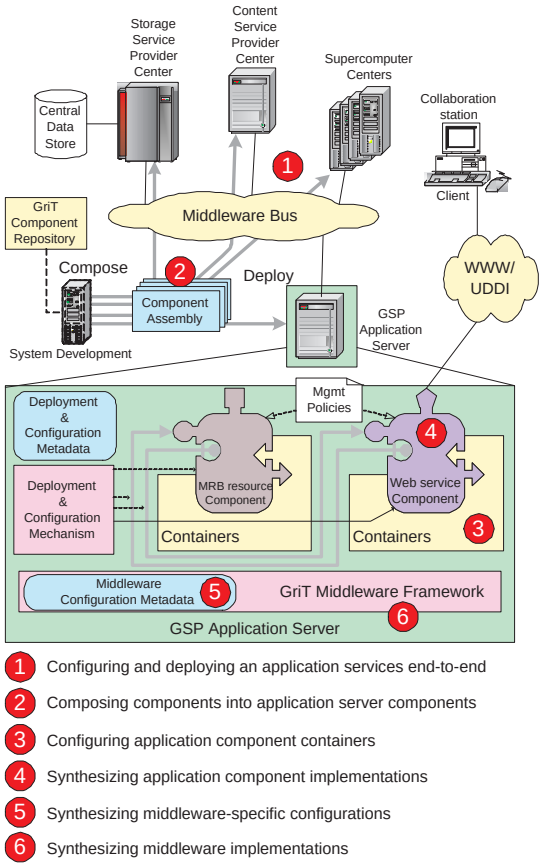

Fig. 8. Composing Grid Middleware from Models

3. satisfying multiple quality of service requirements simultaneously
4. lack of well-defined patterns for resource reservation
5. provisioning and managing resources is hard

In order to support next-generation grid applications effectively, there is a need to address these challenges. Our approach of using model integrated computing tools to assemble and deploy grid middleware as web services addresses these challenges.

Solution. Our solution involves using the MDA-based tool, called CoSMIC, to compose resource provisioning and QoS management patterns from building blocks of the GriT component middleware and to deploy them as web services. Our approach is illustrated in Figure 8.

Figure 8 illustrates six points at which Model-Integrated Computing, espoused by the CoSMIC tool, can be integrated into the grid middleware architecture, called GriT. We describe each of these six integration points below.

1. Configuring and deploying application services end-to-end: Assembling and deploying collaborative grid applications with stringent end-to-end QoS and resource guarantees is a daunting task. We are using CoSMIC to configure the right set of services to guarantee the QoS and resource requirement.

2. Composing components into application servers: Integration at this level will help compose the application server, which is responsible for hosting the application. We are using CoSMIC tools to compose grid application servers out of semantically compatible standard middleware components and possibly legacy components available as part of a component library.

3. Configuring application component containers: Application components use containers to interact with the application servers in which they are configured. Containers provide many policies that grid applications can use to fine-tune underlying component middleware behavior. Since grid applications consist of many interacting components, their containers must be configured with consistent and compatible policies.

Due to the number of policies and the intricate interactions among them, it is tedious and error-prone for an application to *manually* specify and maintain its component policies and semantic compatibility with policies of other components. We are using CoSMIC tools to automate the validation and configuration of these container policies by allowing system designers to specify the required system properties as a set of models. Another set of CoSMIC tools can then analyze the models and generate the necessary policies and ensure their consistency.

4. Synthesizing application component implementations: We are using modeling languages and tools to increase the automation in generating and integrating grid application components. The goal is to bridge the gap between specification and implementation via sophisticated aspect weavers and generator tools that can synthesize platform-specific code customized for specific application properties, such as resilience to denial of service attacks, robust behaviour under heavy load, and good performance for normal load.

5. Synthesizing middleware-specific configurations: In this step, the CoSMIC tools generate the deployment descriptors for the grid middleware. The deployment descriptors take into account the application's QoS and resource requirements along with the constraints.

6. Synthesizing middleware implementations: The CoSMIC tools can also be used to generate custom grid middleware implementations. This is a more aggressive use of modeling and synthesis than integration point 5 described above since it affects middleware *implementations*, rather than their configurations.

4.2 Model Driven Grid Web Service Deployment

Context. Both wireless and wireline client applications must be able to participate in collaborative grid applications. This requires *thin* client applications that can use the grid middleware interfaces to share resources.

Problem. As mentioned earlier, programming directly at the grid framework-specific protocols is too low-level and hence tedious and error-prone. Additionally, it ties the application to the underlying middleware API making portability infeasible. Moreover, for small footprint wireless clients and other embedded devices to use the grid framework, standards-based protocols and interfaces must be used.

Solution. The services offered by the GSP will be hosted as a web service as shown in Figure 5. This approach is similar to the ideas proposed in Open Grid Services Architecture (OGSA) [6]. The CoSMIC tools can be used to synthesize a Web Service Description Language (WSDL) description of the GSP's services. Moreover, CoSMIC tools can also help deploy these services and register it with naming services, such as Uniform Description, Discovery, and Integration (UDDI) or a CORBA Trader. Client applications can then access the GSP services via the web.

This web services approach provides grid application developers tremendous benefits when establishing SIP sessions [23]. SIP is designed to enable two or more participants to establish a session consisting of multiple media streams including audio, video, and other internet-based communication mechanisms such as distributed gaming, shared applications, whiteboards, etc. Participants in a collaborative application will use the GSP's interfaces and services to set up SIP-enabled collaborative sessions.

5 Related Work

Our previous work in collaboration with researchers at University of California, Irvine, on a high-performance, real-time CORBA ORB called the the ADAPTIVE Communication Environment (ACE) ORB (TAO) [24] has examined many dimensions of ORB middleware design, including static operation scheduling, event processing, I/O subsystem and pluggable protocol integration, both synchronous and asynchronous ORB Core architectures, IDL compiler features and optimizations, systematic benchmarking of multiple ORBs, patterns for ORB extensibility, high-performance fault-tolerant CORBA, and ORB performance.

The Component Integrated ACE ORB (CIAO) [18, 13] is our CCM-enabled version of the TAO ORB. The GriT middleware described in this paper enhanced CIAO by providing grid computing-specific components.

Grid computing is an emerging powerful paradigm to build large-scale, distributed, collaborative applications that require secure, controlled access to different resources from multiple providers. The GriT middleware is a distribution

middleware that complements and enhances the low-level grid infrastructure middleware such as Globus [3], Legion [4], and ICENI [5].

Our research is exploring the use of Model-Intergrated Computing (MIC) [10, 25, 17] to model and synthesize Grid middleware code for provisioning Grid applications.

Popular examples of MIC technology being used today include Generic Modeling Language (GME) [14] and Ptolemy [16] (which are used primarily in the real-time and embedded domain) and MDA [12] based on UML [8] and Extensible Markup Language (XML) [26] (which is used primarily in the business domain).

6 Conclusions

The key to the success of developing next generation grid applications lies in the integration of Model Integrated Computing and component middleware. This paper describes a MIC tool we are developing, called CoSMIC. We show how CoSMIC can be used to assemble and deploy grid middleware from fundamental building blocks provided by the GriT middleware. Moreover, we show how the same synthesis process can also be used to expose the grid middleware as a Web service, thereby decoupling grid applications from any particular middleware API.

References

1. Ian Foster and Carl Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Harper Collins, 1999.
2. Ian Foster, Carl Kesselman, and Steven Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, pp. 205–220, Apr. 2001.
3. I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.
4. Andrew S. Grimshaw and Wm. A. Wulf et al., "The legion vision of a worldwide virtual computer," *Communications of the ACM*, vol. 40, no. 1, pp. 39–45, Jan. 1997.
5. N. Furmento, A. Mayer, S. Gough, S. Newhouse, T. Field, and J. Darlington, "An integrated grid environment for component applications," in *Proceedings of the Second International Workshop on Grid Computing- Grid 2001, Denver 2001*. 2001, Springer-Verlag LNCS.
6. I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," www.globus.org/research/papers/ogsa.pdf, Jan. 2002, DRAFT.
7. Object Management Group, *CORBA 3.0 New Components Chapters*, OMG TC Document ptc/2001-11-03 edition, Nov. 2001.
8. Object Management Group, *Unified Modeling Language (UML) v1.4*, OMG Document formal/2001-09-67 edition, Sept. 2001.

9. David Harel and Eran Gery, "Executable Object Modeling with Statecharts," *IEEE Computer*, vol. 30, no. 7, pp. 31–42, July 1997.
10. Janos Sztipanovits and Gabor Karsai, "Model-Integrated Computing," *IEEE Computer*, vol. 30, no. 4, pp. 110–112, Apr. 1997.
11. Aniruddha Gokhale and Balachandran Natarajan, "GriT: A CORBA Based Grid Middleware Architecture," in *Proceedings of Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems Minitrack, HICSS 2003*, Honolulu, HW, Jan. 2003, HICSS.
12. Object Management Group, *Model Driven Architecture (MDA)*, OMG Document ormsc/2001-07-01 edition, July 2001.
13. Douglas C. Schmidt and Steve Vinoski, "Dynamic CORBA, Part 2: Dynamic Any," *C/C++ Users Journal*, Sept. 2002.
14. Akos Ledeczi, Arpad Bakay, Miklos Maroti, Peter Volgysei, Greg Nordstrom, Jonathan Sprinkle, and Gabor Karsai, "Composing Domain-Specific Design Environments," *IEEE Computer*, pp. 44–51, November 2001.
15. Jonathan M. Sprinkle, Gabor Karsai, Akos Ledeczi, and Greg G. Nordstrom, "The New Metamodeling Generation," in *IEEE Engineering of Computer Based Systems*, Washington, DC, Apr. 2001, IEEE.
16. J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *International Journal of Computer Simulation, Special Issue on Simulation Software Development Component Development Strategies*, vol. 4, Apr. 1994.
17. Man Lin, "Synthesis of Control Software in a Layered Architecture from Hybrid Automata," in *HSCC*, 1999, pp. 152–164.
18. Aniruddha Gokhale, Douglas C. Schmidt, Balachandra Natarajan, and Nanbor Wang, "Applying Model-Integrated Computing to Component Middleware and Enterprise Applications," *The Communications of the ACM Special Issue on Enterprise Components, Service and Business Rules*, vol. 45, no. 10, Oct. 2002.
19. Douglas C. Schmidt, David L. Levine, and Sumedh Mungee, "The Design and Performance of Real-time Object Request Brokers," *Computer Communications*, vol. 21, no. 4, pp. 294–324, Apr. 1998.
20. Douglas C. Schmidt, Bala Natarajan, Aniruddha Gokhale, Nanbor Wang, and Christopher Gill, "TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems," *IEEE Distributed Systems Online*, vol. 3, no. 2, Feb. 2002.
21. Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*, Wiley & Sons, New York, 2000.
22. Object Management Group, *Data Parallel CORBA Specification*, ptc/2001-11-09 edition, Nov. 2001.
23. Ubiquity Software Corporation, "White Paper: SIP and SOAP," www.sipforum.org/whitepapers/USC-SIPSOAP-WP2.pdf.
24. Institute for Software Integrated Systems, "The ACE ORB (TAO)," www.dre.vanderbilt.edu/TAO/, Vanderbilt University.
25. David Harel and Eran Gery, "Executable Object Modeling with Statecharts," in *Proceedings of the 18th International Conference on Software Engineering*. 1996, pp. 246–257, IEEE Computer Society Press.
26. "Extensible Markup Language (XML) 1.0 (Second Edition)," www.w3c.org/XML, Oct. 2000.