GRID OBJECT DESCRIPTION: CHARACTERIZING GRIDS

Gerd Lanfermann
Max Planck Institute for Gravitational Physics – Albert Einstein Institute
Am Mühlenberg 1, 14476 Golm, Germany
lanfer@aei.mpg.de

Bettina Schnor
University of Potsdam, Dept. of Computer Science
August-Bebel-Straße 89, 14482 Potsdam, Germany
schnor@cs.uni-potsdam.de

Edward Seidel
Max Planck Institute for Gravitational Physics – Albert Einstein Institute
Am Mühlenberg 1, 14476 Golm, Germany
eseidel@aei.mpg.de

Abstract: We present a new data model approach to describe the various objects that either represent Grid infrastructure or make use of it. The data model is based on our experiences and experiments conducted in heterogeneous Grid environments. While very sophisticated data models exist to describe and characterize e. g. compute capacities or web services, we will show that a general description, which combines *all* of these aspects, is needed to give an adequate representation of objects on a Grid. The ***G**rid **O**bject **De**scription **L**anguage (GODsL)* is a generic and extensible approach to unify the various aspects that an object on a Grid can have. GODsL provides the content for the XML based communication in Grid migration scenarios, carried out in the GridLab project. We describe the data model architecture on a general level and focus on the Grid application scenarios.

Keywords: Grid Computing Information Model, Interoperability, Grid Migration

Introduction

Computational Grids are becoming increasingly common, promising ultimately to be ubiquitous and thereby change the way global resources are accessed and used. We have investigated several large scale Grid scenarios for autonomic applications, based on a previous prototype, dubbed the "Cactus Worm" [1]. Two of the major ones are *nomadic migration* [18], which describes the autonomic migration of applications from one compute resource to another, and *application spawning* [20] which allows an application to accelerate the main execution thread by taking advantage of workflow parallelism. In this case internal routines are "outsourced" to external compute resources. These scenarios are realized through service environments, which offer file transfer, resource selection and job submission capabilities to autonomic applications by interfacing with a wide spectrum of existing infrastructure of Grid middleware, batch systems and basic transport and shell methods.

The "Grid substrate" that these services operate on is heterogeneous in availability and type. We address service availability issues in [19, 20] through a fault-tolerant service topology and focus here on the problem that all objects which constitute the Grid come with different access types, batch submission systems, resource selection processes etc. Clients of such an environment are e. g. characterized through their non-web service compliant functionality or resource requirements.

The problem of heterogeneous environments is well known and projects like Globus or Legion address it through a Grid middleware that provides a uniform access to machines. However, many sites do not install this software, they fail to maintain it properly, or test installations are never integrated into the normal production environment. Nevertheless, most of these sites support launching of jobs through the standard login procedure and batch submission systems. The direct interaction with Grid middleware is often the only way to access resources if sophisticated packages like Globus fail.

Before we can integrate existing Grid infrastructure as well as proprietary user applications, we need a way to describe such "objects" on a Grid. This description goes beyond the current scope of web service or resource characterization. While sophisticated information models exist for the individual aspects, we are lacking a lean data model, that combines these aspects into a single entity which is simple enough to be used not only for server communication but also within client applications. In the following case study, we will motivate such a common description system with a concrete Grid migration scenario.

1. The Grid Migration Scenario

Nomadic Migration denotes the self-controlled and automatic relocation of large applications to provide faster or more efficient code execution [20]. *Nomadic* illustrates the low frequency of this event and the self-contained operation: It is an application-initiated and self-determined process in a service environment. Nomadic migration is investigated by the Max Planck Institute for Gravitational Physics to relocate numerical relativity simulations to increase the job throughput of their large-scale computations [2].

Migration mechanisms and strategies were well studied in the context of cycle-stealing clusters in the mid 90s (see for example [22, 24, 3]). The motivation for migration has more or less stayed the same: *Administrative reasons*, when the compute requirements of an application exceeds the queue time limits offered at a compute center and *Performance reasons* when "better" compute capacity becomes available. Our scenario goes beyond the scope of migrating within a single cluster or a homogeneous intra-Grid machine pool: we relocate across global Grids.

Traditional Job Re-submission. If the resource demands of an application exceed the granted limits (e. g. memory or processors), the application needs to relocate to a different machine or batch queue. The user engages in a tedious process of instructing the application to checkpoint, securing the checkpoint files and archiving them if necessary. If the application can only be continued on another host, the checkpoint files need to be transferred. After deriving the new resource requirements the program is resubmitted to the queuing system. At all steps, the user's involvement makes the process prone to failure:

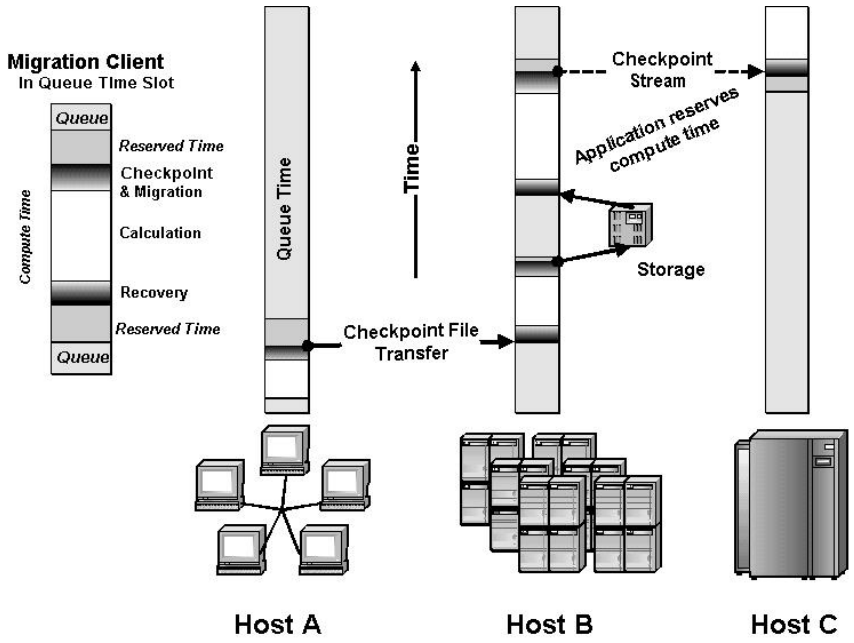

Figure 1. "Nomadic Migration".

1 Checkpoint files can be erased by disk purging policies and lack of bandwidth can make the checkpoint transfer unacceptably slow.

2 Resource requirements have to be correctly analyzed, otherwise the job will be rejected by the resource management system, either at submission time or worse – during runtime. Conservative estimates of resource requirements usually lead to longer waiting times.

3 The researcher is required to remember usernames and passwords as well as the interfaces to a wide range of different machines, architectures, queuing systems and shell programs.

From experience, the overhead of this procedure encourages the researcher to resubmit a job on the same machine, discarding potentially faster machines.

Automated Job Re-submission. The process of resubmitting to an arbitrary host, which fulfills the application's minimum resource requirements, is a prime candidate for automation: In Figure 1 we illustrate a migration process for an application. The migration progresses from left to right across three different host types *A*, *B* and *C*, indicating a network of workstations, a cluster of PCs and a traditional supercomputer, respectively. The left inset shows the different phases of an application in a queue: a migration client receives reserved compute time, called a "slot". Within this time slot, all code execution has to take place: the recovery from a previous checkpoint, the cal-

culation phase, and saving the new state to a new checkpoint file. Applications which exceed their time slots are terminated.

In the illustration an application is started on host *A*. A migration server (not shown), which is responsible for the relocation of applications, receives information on the client's resource consumption and its location. It monitors the availability of new machines which meet the requirement. As "better" resources become available, the application on *A* is informed and it checkpoints. The checkpoint files are transferred to host *B*. The migration client is restarted or submitted to the queuing system. As the application runs out of compute time on *B*, the last checkpoint is archived in a storage facility and the program is resubmitted to the queue on the same machine. The application will execute a second time on *B*.

Some advanced applications are able to receive the checkpoint as a socket stream instead of reading from file. In combination with advanced reservation scheduling, this transfer mode allows for fast checkpoint transfer, shown in the migration to machine *C*: The program on *B* is aware of the expiration of its time slot and requests in advance a slot on machine *C*, which overlaps with the compute slot on *B*. By the time the application is about to finish on *B*, the migration server starts an uninitialized application on *C*, which receives the application state through the streamed checkpoint and continues the calculation. GridLab [25] is currently investigating these prototypes to study the functionality which they will make available in a generic way to a wide class of applications.
This case study identifies some of the problems that have to be solved:

- *File Transfer*: A migration service stages executables and other files to a new host. Therefore, it must be able to determine and support different file transfer methods for each of the source and target machines.

- *File Description*: The service must provide a compact description of multiple files, including information on where the files are located and how they can be accessed.

- *Job Submission:* The migration server is responsible for submitting the new job to individual queuing systems.

- *Resource Description*: It is essential to characterize resource requirements of an application as well as resource capacities of machines or queues. The migration service must be able to evaluate and compare them.

- *Hardware Description*: A migration service must describe the location of machines, including the ways to access it.

2. Related Work

Quite a number of different software packages exist to assist in interfacing with heterogeneous environments. Resource management systems are primarily aimed at users, while web services and related technology are designed for applications. In this section we list the most important packages and discuss how we can incorporate their functionality in our information model.

2.1 Resource Management Systems and Tools

These systems focus on describing the compute capacities of machines and the requirements of applications. They provide advanced scheduling policies to achieve high job throughput and provide a uniform access to machines. Typically each of these submission systems comes with its own way of characterizing resources.

Globus. Globus [13] makes a significant contribution to the Grid infrastructure. We use Globus functionality for our migration service wherever and whenever possible. Globus requires a working and maintained installation. Within the Globus installation base, a well specified job submission and resource selection system is supplied to the user.

The Globus Resource Specification Language (RSL) [23] provides a common interchange language to describe resources. It is used to define resource requirements for applications which are submitted through the Globus Resource Allocation Manager (GRAM) [16]. The Meta Directory Service (MDS) [10] is the information service component of the Globus Toolkit. MDS services can deposit and extract information on the current resource situation of a machine or queue. MDS is based on LDAP and represents information as a set of objects organized in a hierarchical namespace, in which each object contains key-value pairs. MDS introduces yet another set of vocabularies to express resources.

Condor. The Condor [4] system is designed as a "cycle-stealing" middleware to harvest unused compute capacity for single processor applications. It supports transparent checkpointing and operates primarily on homogeneous machines within a single administrative domain, called a 'flock". Condor-G [14] is a modified Condor package, that joins multiple condor "flocks" from diverse sites through Globus.

Condor Classified Advertisement (Class-Ad) is a sophisticated data model to describe resources in general and includes a matchmaking ability which makes it a valuable tool to compare resource requirements with constraints. Condor does not come with a fixed set of attributes but requires the information in the Class-Ad syntax.

Batch Submission System. Packages like PBS, LSF, Load Leveler, Sun Grid Engine, etc. are responsible for submitting the application to a computer. They come with a proprietary script syntax and offer different capabilities. Submission scripts are traditionally prepared by users – either manually or through graphical user interfaces.

2.2 Web Services

Most of the services listed above are designed to be accessed by a user – either from the command line or through graphical interfaces. They are not primarily designed to be accessed by an application. The *application-centric* utilization of services has been a very recent trend and builds on the concept of web services. Technologies like "eXtensibleMarkup Language" (XML) [6], "Simple Object Access Protocol" (SOAP) [5], XML-Remote Procedure Call (XML-RPC) [27], "Web Service Description Language" (WSDL) [7] and "Universal Description Discovery and Integration" (UDDI) [26] enable applications to autonomously connect to other programs to exchange information and services. WSDL, which is an XML language used to describe

network service endpoints, has become the standard to describe web services. The recently released Open Grid Service Architecture (OGSA) [12], which is a specification that integrates the concepts of web services and Grid technology, builds heavily on these web service standards.

2.3 Related Information Models

There are some unification efforts for the description of Grid entities under way: Globus GRAM provides interfaces to address the most common batch submission systems. The Global Grid Forum features the Job Submission Description Language (JSDL) working group which investigates a standard description for job submissions to incorporate the different resource dialects. The Common Information Model (CIM) [8] is developed by the Distributed Management Task Force (DMTF). CIM's data model is an implementation-neutral scheme for describing management information in a network/enterprise environment. CIM has similarities regarding our goal to provide a common description system.

The JSDL is similar in spirit to what GODsL aims for, but is restricted to job description. The Globus RSL, which unifies access to different submission systems through Globus GRAM has only rudimentary support for file information. CIM information models on the other hand offer a functionality which reaches far beyond what we feel is necessary to describe entities on the Grid and which makes it difficult to deploy such models on clients. We follow a more simplistic approach to describe Grid Objects.

3. Grid Objects

The design of the Grid Object Description Language is based on our experiences with the migration service environment and in particular on these observations:

- To use an information model, we cannot require or rely on the global installation of a single Grid middleware. Because of the *administrative autonomy* of sites, we will be faced with anything from modern Grid web services to traditional interactive commands.
- Web service technology has excellent solutions for providing information on services. But our system must accommodate "legacy-services", like ssh-based machine access as well. Furthermore, we need to describe non-web service compliant user programs and have to associate service descriptions with file or machine information.
- We require a solution which abstracts a compute capacity from the hardware that provides it and which is able to describe resource requirements as well.
- A description of files must be included for migrating an application: a checkpoint file contains the "hibernating" state of a migrating application and is of equal importance as the application's executing state, which can be characterized as a service.

With this motivation, we do not attempt to reproduce any of the software packages or Grid middleware listed above. We target a uniform description of the various systems and their abilities. We motivate an information model that allows for a precise definition of an "object" on the Grid independently of the circumstances under which it is

realized: this object can be a distributed file, a compute capacity offered by a machine or the resource requirements of an application. We attempt to capture these different aspects in a single data model.

3.1 Grid Object Examples

The following examples illustrate how the different aspects of items on the Grid are interchangeable:

1 The information on the name and directory of a data file is only sufficient if we know on which physical machine the file resides. However, we have no information on how we can access that particular computer, we are not able to look at the file, copy it or treat it in any other way.

2 The file on that machine may have been generated by an application. Unsurprisingly, the machine which hosts the service is identical with the machine location of the file.

3 The application has a minimal resource requirement, and a computer may provide resource capacities. The host can execute the application if and only if the hardware's resource capacity is greater or equal than the application's resource requirement.

4 If requirements and constraints are matching, the application can be hosted on that particular hardware. The description of the application remains the same, but the host information changes.

In these simple examples we can identify four core components of an object, which are service, hardware, file and resource properties.

3.2 Grid Object Description Language

We propose *Grid Objects* as a general description of objects on a Grid. We call the structure to describe such an entity the *Grid Object Description Language*. A Grid Object is a collection of sub-objects, termed a *Container*, each of which holds one or more *Profiles* that reflect the different properties of this object, as illustrated in Figure 2. The container structure in a Grid Object is optional and depends on the background situation that is described.

For our purposes we define a Grid Object as follows:

$$\text{Grid Object} \rightarrow \text{UId}\ \ \text{Label}\ \ \{\text{Machine Profile}\}\ \{\text{Resource Profile}\}\ \{\text{Service Profile}\}\ \{\text{File Profile}\}$$

The profiles and container can be optional. Multiple profiles are collected in a *container* structure. Profiles can be copied (example 2, in which the file's machine profile can be inherited from the description of the application) or compared (example 3, where two profiles are put in relationship). Profiles can be added or replaced (example 4, adding or replacing the description of the hardware as the application starts or migrates, respectively).

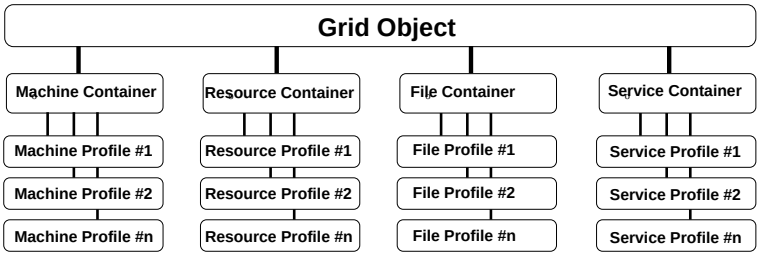

Figure 2. Grid Object structure

3.3 Machine Profiles

Machine profiles contain the information which are obligatory to address the hardware on the internet. Typical information items are the type of the hardware, the hostname, the domain name and the IP address. Multiple machine profiles can be grouped in a *Machine Container*. A machine container can e.g. describe a network of workstations or the participating machines of a meta-computer.

3.4 Resource Profiles

The *resource profile* structure characterizes the compute and storage capabilities or requirements of a Grid Object. Key attributes of a resource profile are e. g. the *number of processing elements*, *memory*, information on the *operating system* and *CPU* type.

The reason to treat the resource characteristics separately from a machine profile is justified by the fact that resource profiles are not necessarily tied to a hardware device. Instead they can be used to characterize abstract resource situations: An application may provide a resource profile to interested parties to relay information on its *resource consumption*. In this context the resource profile is not attributed to a hardware (machine profile) but to an application, which is described through its location (machine profile) and functionality (service profile). As hinted above, multiple resource profiles may be used to characterize the different batch queues which partition the total compute resource of a machine, specified through a machine profile.

3.5 Service Profiles

A *Service profile* is a description of a functional ability which is available on a machine or provided by an application, usually through a port or range of ports. We aim at describing modern web services as well as traditional interactive commands like secure-shell based access to a hardware or queue submission system. Key attributes, which are used to describe such a profile are *operation* for traditional command line programs, *binding* specifying contact information on web service based operations, *method* the name of the method under which a rpc-typed service can be accessed, followed by port information. Multiple service profiles can be grouped into a *Service Container*. Such a container may e. g. list all access methods (e. g. ssh, rsh, http, Globus gatekeeper) that permit access to a hardware.

The Grid Object information model that we suggest does not try to supersede or copy WSDL technology. Instead we look for a pragmatic way to include non-webservice compliant, legacy applications as well as proprietary user codes in the description of Grid entities. While WSDL has some capabilities to store service related information, we do not regard WSDL as a suitable place e. g. to store the description of a migratable entity (application, checkpoint, parameter files). We would suggest using WSDL for describing the interface to retrieve such an object description from a database.

3.6 File Profiles

File profiles describe properties of files and directories. Their main attributes are *Filename* and *Directory* to locate files on a filesystem. *Size* and *Compactification* information allow a transfer operation to favor certain services over others or to rule out potential target hosts, e. g. due to the lack of bandwidth.

3.7 Grid Objects

The different profiles and containers that we introduced above are combined to provide a unified description of the different aspects of objects on the Grid. The Grid Object structure collects the various containers into a single entity. Container information is optional, containers are appended to the Grid Object depending on the situation which is described. The Grid Object may hold a *Machine Container*, *Service Container*, *File Container* and *Resource Container*. Each component (profile, container, object) provides space for a human readable label and a unique identifier (UId) to distinguish the different components in a machine readable style. Multiple Grid Objects can be collected into an array of Grid Objects, called a *Grid Object Container*.

The classification into the four fundamental types is inspired by our underlying migration scenario, whose communication content we intend to express with this data model. For other scenarios, different profiles may become important while others can be omitted. In section 5 we suggest several important additions to the Grid Object information model.

3.8 GODsL Toolkit

The *GODsL Toolkit (GODsL-Tk)* provides a number of routines to manage Grid Objects, container and profile structures for the C programming language. It can be used to copy profile and container constructions and attach them to already existing objects. The toolkit provides routines to serialize the Grid Objects from C to XML and deserialize objects in XML into the appropriate C structures. Our migration service is such an XML-RPC based service, which uses XML representation of Grid Objects as data for the migration requests.

4. Grid Objects Applied: Grid Migration Service

In this section we give actual examples of how Grid Objects are used in the context of the migration service, which we introduced as our main motivation for developing this data model.

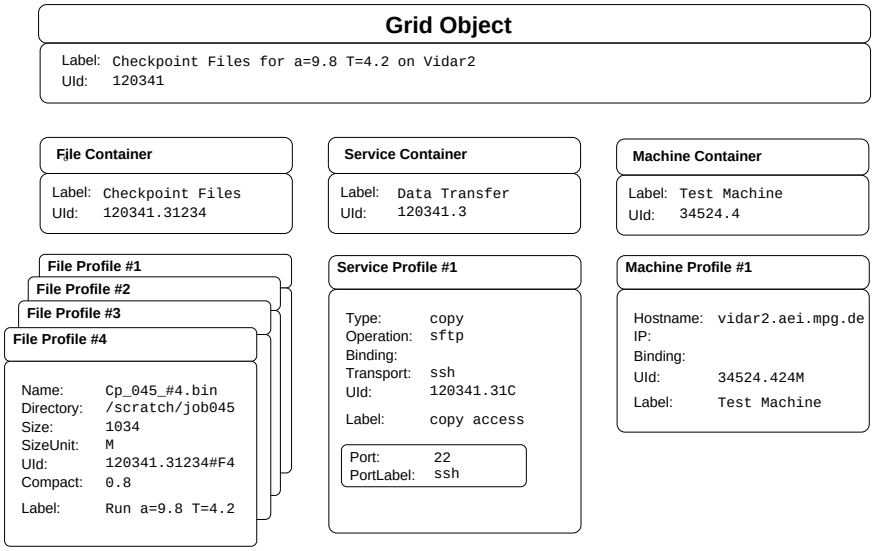

Figure 3. Grid Objects describe distributed files generated through parallel I/O as well as the hosting machine and the access methods.

4.1 Multiple Files on a Single Machine

Large data files in parallel applications are usually brought to disk through parallel I/O methods to speed up the process of data writing. It has the drawback in that it generates multiple files, as sketched in Figure 3, which shows the four checkpoint file chunks which compose a single logical checkpoint. In the same way that a service container can hold several access profiles for a machine, we can group multiple file profiles in a file container.

In Figure 3 we show the resulting Grid Object, which holds the file container, that stores the four file profiles. The Grid Object can be completed with information *where* the file is located through a machine profile and *how* it can be accessed: this service profile identifies secure-shell ftp as the only transfer method. We can now pass such a Grid Object as the source part in a copy request to a migration server, e. g. to store the files in a storage facility. With the method call, the copy service receives all the necessary information to move this file from the specified machine to a new site. The service can also decline the copy operation, if it does not provide the transfer methods which are required to access the files on the machine.

4.2 Resource Identification, Evaluation and Request

Before any job is executed on a supercomputer, a three stage process needs to be completed, which is traditionally done by the user, interactively and intuitively: The user determines the resource requirements of his application, typically through educated guesses or trial and error. Secondly, the user has to familiarize himself with the compute capacities on the machines in question. This knowledge is usually gathered

by reading up on the site's batch submission configuration. The user then chooses a particular supercomputer, where he selects a queuing system, whose constraints will not be violated during the runtime of the program. The user requests the resources, usually by filling out a batch submission script, in which he states the requirements, like number of processors and memory. The user has to obey the particular queue syntax. The job is finally submitted.

If we express the diverse resource requests and constraints, job submission interfaces, etc. as profiles in a Grid Object, we have accomplished a first step to automate this user-centric process. In Figure 4 we show the use of Grid Objects in each of the three stages and explain the mapping between the different vocabularies of MDS [10], Condor's Class-Ads [9], PBS [17] and LSF [29]. The attributes of a resource profile (abbreviated `RP`), a service profile (`SP`) and a machine profile (`MP`) are being exchanged with the corresponding third party vocabulary. The automated process consists of these procedures:

Resource Identification. In the identification phase, the available resources are reported by monitoring systems like MDS, and the mapping of the MDS vocabulary to the Resource profile's attributes is performed. This step is illustrated in the top section. If more than one resource is obtained, the lookup service provides a Grid Object for each of the results. The resource requirements of an application can be measured at runtime with tools like Performance-API [21] and are also expressed in a resource profile of a Grid Object (bottom part).

Resource Evaluation. To determine which resource provides the best compute capacity for a given resource constraint, we prefer using existing technology, like Condor's Class-Ads: A migration service rewrites the resource profile of the application and the resource profiles of the available resources as Class-Ads. The Class-Ads parsing algorithm compares the job requirement to the constraints and returns a match if possible.

Resource Request. When a match is found, the migration server can proceed to submit the job and needs to fill out the proper batch submission script. Depending on the scheduling system found on the particular machine, the server provides the resource requirements as arguments to the batch scheduler: The content of the resource profile is now mapped onto the vocabulary of the batch submission systems. As shown for PBS and LSF in the bottom part of Figure 4, this is a straightforward translation process.
Reusing existing software is an indispensable ability and the Grid Object Description Language permits to converse with already existing software packages and grid middleware. For example, a statement on the amount of memory is expressed as `physicalmemorysize` (MDS), `Memory` (Class-Ads, user defined), `#BSUB -M` (LSF directive), `#PBS -l mem` (PBS directive). The mapping between different vocabularies is performed through modules in the individual programs.

5. Conclusions and Future Research

In this paper we have described the hierarchical and modular architecture of the Grid Object Description Language. Grid Objects represent the different aspects of an

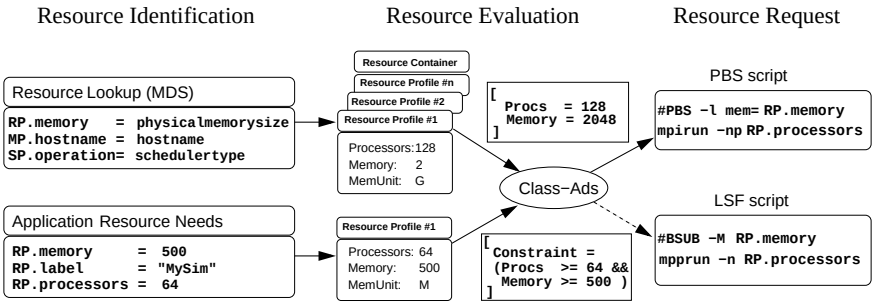

Figure 4. Grid Objects provide the translation capabilities to make use of already existing resource management technology. Resource profiles are used as a mediator between the different systems.

abstract entity on the Grid. Its hierarchical structure allows Grid Objects to scale with the increasing complexity of the described situation. Profiles and container components can be exchanged between Grid Objects which permits uncomplicated generation, updating and management of Grid Objects. The attributes of its profiles can be mapped onto existing Grid middleware in a straightforward manner, which is an important requirement to take advantage of sophisticated legacy middleware, which are omnipresent in today's Grid infrastructure.

We have illustrated in several examples that Grid Objects can be used to describe complex situations on Grids in a flexible way. These examples are taken from experiments with the Grid Migration Service, which uses Grid Objects to communicate the state of files, services and resources. Migration experiments have been conducted on the European Testbed (EGrid) [11] to develop and test communication through Grid Objects.

Open Grid Service Architecture. The Grid Object description approach falls short in respect to the capabilities of today's web service technology. For the future development of GODsL, the interfacing with web service technologies, like OGSA, will become an important field of research. The OGSA code base has been changing quite drastically over time and has now reached a stage where it has started to stabilize. We intend to take advantage of OGSA services in migration servers, for example to utilize Grid copy operations.

Network Profiles. The current version of GODsL does not support the characterization of network performance. The Network Weather Service [28] is a distributed system that periodically monitors and dynamically forecasts the network conditions like bandwidth. Currently, we investigate, how Grid Objects can be extended through *Network Profiles* [15], which will allow a migration service to select a potential migration host based on its network connectivity. Such a profile should also be capable to typify *internal* networks to give users a way of requesting a preferred network interconnect within a cluster or parallel computer.

Time. Future work will include the notion of time and time intervals. The present system of profiles is static in the sense that profiles do not timeout or become invalid.

Especially in the field of meta-computing and advanced reservation scheduling it is necessary to describe the beginning and expiration time of a resource. We are working on methods to complete the Grid Object structures to express these dynamic properties.

With GODsL we do not intend to provide the community with yet another way of solving the problems of resource detection or web service description. However, from our application background, we see the need to express a unifying view on general objects on a Grid: We see compelling demand to glue together both the advanced and historical software technologies, which are the fabric of today's Grid infrastructure, as well as user applications. The Grid Object Description Language suggests a solution to this problem.

Acknowledgments

We are pleased to acknowledge support of the European Commission 5th Framework program, which is the primary source of funding for the GridLab project, but also the German DFN-Verein, Microsoft, the NSF ASC project (NSF-PHY9979985), and our local institutes for generous support for this work. We also thank Gabrielle Allen, Thomas Dramlitsch and Ian Kelley for helpful discussions. We are grateful for using the compute resources of the EGrid and at NCSA.

References

[1] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *Int. J. of High Performance Computing Applications*, 15(4):345–358, 2001. `http://www.cactuscode.org/Papers/IJSA_2001.pdf`.

[2] G. Allen, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, W. Benger, C. Hege, A. Merzky, J. Massó, and J. Shalf. Solving Einstein's Equations on Supercomputers. *IEEE Computer*, 32(12):52–59, 1999.

[3] A. Barak, A. Braverman, I. Gilderman, and O. Laaden. Performance of PVM with the MOSIX Preemptive Process Migration. In *Proceedings of the 7th Israeli Conference on Computer Systems and Software Engineering*, pages 38–45, Herzliya, June 1996.

[4] J. Basney, M. Livny, and T. Tannenbaum. High throughput computing with Condor. *HPCU News*, 1(2), June 1997.

[5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. Simple Object Access Data Protocol (SOAP) 1.1. W3C Note, May 2000. `http://www.w3.org/TR/SOAP/`.

[6] T. Bray, J. Paoli, C. Sperenberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0. W3C Recommendation, October 2000. `http://www.w3.org/TR/REC-xml`.

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerarawana. Web Service Description Language (WSDL). W3C Note 15, March 2001. `http://www.w3.org/TR/wsdl`.

[8] Common Information Model (CIM) Standards. The DMTF webpage: CIM Specification v2.7 and Standards, September 2002. `http://www.dmtf.org/standards/standard_cim.php`.

[9] The Condor Classified Advertisement. The Condor Webpage. `http://www.cs.wisc.edu/condor/classad/`.

[10] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing, (HPDC-10)*. IEEE Press, August 2001.

[11] T. Dramlitsch, G. Lanfermann, E. Seidel, A. Reinefeld, et al. Early Experiences with the EGrid Testbed. In *IEEE International Symposium on Cluster Computing and the Grid*, 2001. Available at `http://www.cactuscode.org/Papers/CCGrid_2001.pdf.gz`.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Service Architecture for distributed systems integration, June 2002. `http://www.globus.org/ogsa/`.

[13] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, 11(2):115–128, 1997.

[14] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Journal of Cluster Computing*, 5:237–246, 2002.

[15] C. Fricke. Characterizing networks through the Grid Object Description Language. Student's Thesis, University of Potsdam, 2003. To appear.

[16] Globus Resource Allocation Manager. The Globus GRAM Webpage. `http://www.globus.org/gram`.

[17] R. Henderson and D. Tweten. Portable Batch System: External reference specification. Technical report, NASA Ames Research Center, 1996.

[18] G. Lanfermann, G. Allen, T. Radke, and E. Seidel. Nomadic Migration: A New Tool for Dynamic Grid Computing. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing, HPDC-10*, pages 435–436. IEEE Press, August 2001. `http://www.cactuscode.org/Papers/HPDC10_2001_Worm.ps.gz`.

[19] G. Lanfermann, G. Allen, T. Radke, and E. Seidel. Nomadic migration: Fault tolerance in a disruptive grid environment. In *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 280–281, 2002.

[20] Gerd Lanfermann. *Nomadic Migration – A Service Environment for Autonomic Computing on the Grid*. PhD thesis, University of Potsdam, Potsdam, 2003. To appear.

[21] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, and T. Spencer. End-user tools for application performance analysis, using hardware counters. In *International Conference on Parallel and Distributed Computing Systems*, August 2001. `http://icl.cs.utk.edu/projects/papi`.

[22] S. Petri and H. Langendörfer. Load Balancing and Fault Tolerance in Workstation Clusters – Migrating Groups of Communicating Processes. *Operating Systems Review*, 29(4):25–36, October 1995.

[23] The Globus Resource Specification Language RSL v1.0. The Globus Webpage. `http://www-fp.globus.org/gram/rsl_spec1.html`.

[24] B. Schnor, S. Petri, R. Oleyniczak, and H. Langendörfer. Scheduling of Parallel Applications on Heterogeneous Workstation Clusters. In Koukou Yetongnon and Salim Hariri, editors, *Proceedings of the ISCA 9th International Conference on Parallel and Distributed Computing Systems*, volume 1, pages 330–337, Dijon, September 1996. ISCA.

[25] E. Seidel, G. Allen, A. Merzky, and J. Nabrzyski. GridLab – A Grid Application Toolkit and Testbed. *Future Generation Computer Systems*, 18(8):1143–1153, 2002.

[26] Universal Description, Discovery and Integration (UDDI). `http://www.uddi.org`.

[27] D. Winer. XML-RPC Specification, June 1999. `http://www.xmlrpc.com/spec`.

[28] Richard Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[29] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software - Practice and Experience*, 23(12):1305–1336, 1993.