# Policy-based Resource Topology Design for Enterprise Grids

Sven Graupner, Akhil Sahai
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2005-59
March 14, 2005*

E-mail: {sven.graupner,akhil.sahai}@hp.com

enterprise, Grid, design tool, policy, system management, model-based automation

One of the challenges Grid technology faces today is the adoption in other domains than scientific computing, where it originated. However, in order to be usable and useful into other domains, it needs to be adapted and extended to fit the needs of the domain and yet maintains the standards defined by OGSA [1] and WSRF [27]. We focus on the domain of *Enterprise Grids* [2], which are comprised of the interconnected data centers as they exist in commercial enterprises forming large pools of geographically distributed IT resources that are shared among numerous IT applications. In this paper, we compare the commonalities and differences between compute Grids [15] and Enterprise Grids. We then introduce the concept of *Resource Topologies* which is important for Enterprise Grids. A design tool is presented for designing Resource Topologies. In the second part of the paper we introduce the problem of *Resource Composition* and present a solution for exploring a space of possible resource compositions for meeting application requirements using a policy-based constraint satisfaction method.

* Internal Accession Date Only

# Policy-based Resource Topology Design for Enterprise Grids

Sven Graupner, Akhil Sahai

Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94304

{sven.graupner,akhil.sahai}@hp.com

Abstract

One of the challenges Grid technology faces today is the adoption in other domains than scientific computing, where it originated. However, in order to be usable and useful into other domains, it needs to be adapted and extended to fit the needs of the domain and yet maintains the standards defined by OGSA [1] and WSRF [27]. We focus on the domain of *Enterprise Grids* [2], which are comprised of the interconnected data centers as they exist in commercial enterprises forming large pools of geographically distributed IT resources that are shared among numerous IT applications. In this paper, we compare the commonalities and differences between compute Grids [15] and Enterprise Grids. We then introduce the concept of *Resource Topologies* which is important for Enterprise Grids. A design tool is presented for designing Resource Topologies. In the second part of the paper we introduce the problem of *Resource Composition* and present a solution for exploring a space of possible resource compositions for meeting application requirements using a policy-based constraint satisfaction method.

## Introduction

Collaboration across locations and organizations, sharing of resources, security and service-orientation are general goals associated with Grids today [1]. These goals do not only apply to the typical Grid domain of scientific computing, they also apply to other domains such as the commercial enterprise IT environment [2]. This environment is characterized by data centers providing large pools of resources in which enterprise applications and services are hosted. Hewlett-Packard, for example, operates 120 data centers world wide that are hosting in the order of 7,000 applications for supporting its own business operations. This infrastructure forms a large Enterprise Grid of distributed, interconnected pools of resources located in data centers. Besides the similarities between the typical scientific compute Grids and Enterprise Grids, there are substantial differences leading to gaps in existing Grid technology that need to be filled.

This paper addresses one of those gaps, the design and selection of resource sets for commercial enterprise applications based on notions of *resource topology* and *policy-based resource composition*.

The paper is organized as follows. First, the enterprise IT environment is briefly characterized before the problem is defined by describing the specific conditions in regard to resource management in enterprise data centers and the requirements enterprise applications have for resource management functions. Second, the concept of a resource topology is introduced. Third, a tool for designing resource topologies is presented. And finally, the process of policy-based resource composition is described.

## Applying Grid Technology in the Enterprise Environment

HP and the IT industry have agreed to leverage concepts, standards and technologies developed for Grids for commercial enterprise environments.

As an *enterprise IT environment* is understood the entirety of IT systems performing information processing functions in an enterprise including services comprised of applications, hard- and software resources as well as associated management functions.

Examples of enterprise applications are infrastructure applications such as email or business applications such as supply-chain applications. Those application systems can be large and can be comprised of a multitude of components which are themselves applications (e.g. databases). The concept of services has been introduced unifying the diversity of applications and resources in enterprise IT environments.

Grid standards, initially defined by GGF and now continued in OASIS, such as Web-Services Resource Framework (WS-RF) [27] and Web-Services Distributed Management (WS-DM) [28] are provide the standardized web-services-based middleware layer for integrating enterprise applications as well as their associated management systems. Implementations of those standards exist in form of the Globus Toolkit [4] and the WS-RF/WS-DM Reference Implementation developed by HP. Initial management interfaces have been built based on Web-Services standards [5,6].

The concepts of the Open Grid Services Architecture (OGSA) [1] and the Service-Oriented Architecture (SOA) are also reflected in the strategies IT vendors have articulated for their product roadmaps.

The main motivators for adopting these technologies in enterprise environments are goals of lowering operational costs by increasing automation and of improving agility of IT systems in their ability to adapt faster to changes occurring in the businesses they drive.

One of the main enablers of these goals is the better integration between enterprise systems and their management systems replacing proprietary integration points with standards-based interfaces that would allow "plug-and-play" connectivity among application systems and management systems, both rendered as services. Substantial problems have to be solved along the journey towards service-oriented systems in enterprise IT environments. Grid technology is seen as an enabler for those systems.

## Enterprise Grid

The following discussion explains differences between typical compute Grids and Enterprise Grids, which lead to the mentioned gaps when it is tried to simply use Grid technology as it exists today in other domains without adaptation to those domains.

**Resource pools in enterprise data centers are large heterogeneous resource sets.** For instance, different kinds of servers exist in an enterprise data center with different processors, architectures, resource properties, operating systems, etc. Different kinds of storage resource exist. Multiple interconnect fabrics exist that are important resources and sometimes bottlenecks for connecting machines (via LAN) to machines and for connecting storage to machines (via SAN). Specialized devices such as firewalls or load balancers are important resources as well that need to be assigned and configured for enterprise applications.

**Enterprise resources provide substantially more capabilities** that must be configured. Networks can be "programmed" as well as storage associations to machines. Different configurations can be applied to machines and devices depending on application requirements.

**Enterprise applications are composed of heterogeneous components** that themselves are applications requiring resources. Enterprise application components requires different resource sets and different configurations (such as for the web tier, the application server tier, and the database backend).

**Applications have specific needs for resources** in terms of resources they require at different times as well as in terms of specific configurations they assume on resources.

**Resources specifically need to be configured** for enterprise application components. Examples are attaching specific disk images to servers, or creating and linking servers into specific networks.

**Resources need to be isolated** when shared among different enterprise applications. Although this is a desirable goal for compute Grids as well, clusters typically do not provide application isolation due to lacking support in the infrastructure. A variety of techniques exist in enterprise data centers for isolating applications from each other. Virtualization is one important technique, ranging from using virtual LAN isolating IP address spaces to encapsulating applications in virtual machines. Application server containers are another example providing light-weight isolation among application components.

**Resource may need to be constructed** and may be constructed **in different ways** for enterprise applications. For instance, an application component may require an "IA32 Linux PC". This resource may simply be taken from a pool as a physical resource, or it may be constructed using a Virtual Machine. Choices for creating virtual machines may exist (VMWare VM or Virtual PC (Microsoft's VM). Another option may be a user-level Linux partition. Choices for constructing resources must be explored.

Resource constructions such as virtual machines are application processes themselves posing their own requirements onto resources. When a construction is chosen, the additional resource requirements of the constructions must be considered. Different costs can be associated with different construction choices in terms of resources constructions consume or licenses they require.

**Requested resource may not physically exist** also for other reasons than constructions are being chosen. Resources may simply have not become part of the resource inventory in a data center yet, but are known to exist in future (e.g. when purchases are planned).

**Requests for Resources** for enterprise environments **are longer-term** than Grid compute jobs (months/years vs. hours/days) leading to longer-term planning and allocation cycles for applications in enterprise data centers. Resource schedulers must accommodate long allocation cycles during which resource inventory may change.

This also links to the consequence that **resource requests may refer to resource inventory that does not currently exist**, but is know to exist in future and hence can already be allocated. This case is important for enterprise environment facing continuous inventory replacement and turnover cycles.

## Resource Topology

These differences and implied problems necessitate extending Grid technology from dealing with simple resource sets that currently exist to complex graph structures which include all the resource requirements in terms of resource elements, specific configurations and the different relationships among elements. Such a graph structure describing resource requirements from an application point of view is referred to as a *Resource Topology*. Nodes in the graph represent resources. Arcs (relationships) typically represent connections among resources (such as servers to networks, or storage to servers). Configuration information is derived from arcs which will create specified connections when applied to switches and routers.

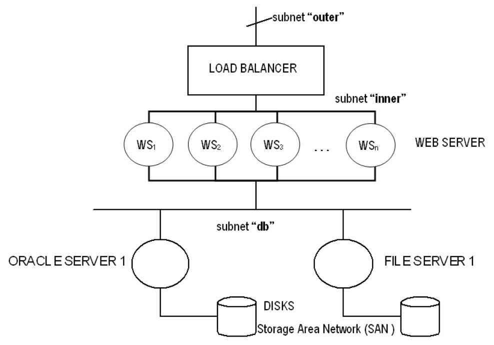


**Figure 1: Example of a Resource Topology.**

Figure 1 shows an example of a resource topology consisting of a load balancer in front of a web server tier behind which a file server resides for serving static content as well as a database containing dynamic content.

A Resource Topology encompasses the entirety of resources (the resource set) with types and quantities of types as well as their relationships of how they are connected.

## Problem Statement

From the general problem space of extending and adapting Grid technology to other domains than scientific computing, one problem is selected and addressed in this paper. This problem is how resource requirements for enterprise applications can be formalized, described, and requested from data center resource pools taking different choices for resource constructions into account.

In scientific Grids, requests for resources are specified in terms of quantities of compute nodes meeting certain constraints. Examples of constraints are: [CPU=IA32, MEM>512MB, DISK> 20GB, OS=Linux]. A grid request could include 128 nodes of this type. Grid's early Resource Specification Language (RSL) [11] allowed to describe resource requirements in those terms.

Resource requests in RSL could then be sent to a Grid Resource Allocation Manager (GRAM) for allocation from a pool. The GRAM evaluated the request and decided about acceptance. A variety of GRAM implementations exist in form of Grid schedulers such as PBS [7] and Maui [29].

Co-allocation allowed exchanging resource request matching across GRAM's (with or without broker), each GRAM representing multiple resource pools for identifying a pool in a Grid that would support the set of requested resources [8].

More recent work extends match-making between resource requests and available resources using the semantic web framework [25].

Described techniques in Grids are not sufficient for enterprise environments for following reasons:

- Heterogeneous sets of resources must be made available for request as entirety accommodating the entire enterprise application.
- Resource requirements requested for applications must be separated from realizations allowing for considering choices of resource constructions.
- Choices that exist for realizing required resources must be evaluated in terms of conflicts that may exist, in terms of cost associated with constructions, and in terms of overall impact constructions have on the application system.
- Further requirements may need to be obeyed in terms of isolation and the overall resource assignment in the data center.

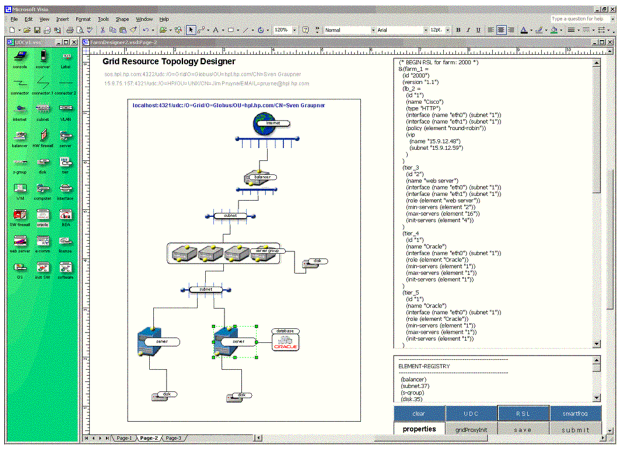
The first problem addressed in this paper refers to tool support for designing resource topologies, which may be complex. Section "Resource Topology Designer" presents a graphical design tool for designing resource topologies.

The second problem refers to making choices for selecting and constructing the resources needed in a resource topology. This is discussed in section "Policy-based Resource Composition". The presented policy-based resource composition technique is accepts a resource topology design which includes application-oriented components as requirements, evaluates them against possible combinations (choices) of constructions or elementary resources (resource atoms) that exist in a data center.

This method of policy-based resource construction is also a technique for realizing the concept of Resource Grounding or RSL Specialization in [16]. This concept allows formulation of resource requests in higher terms than properties of physical resource inventory, such as [IA32 Linux] server, and translating those requirements into physical machine properties that can be matched against in the server pool. We extend this concept to the notion of resource topologies and extend the translation from higher into physical properties to also evaluate combinations and choices of resource constructions.

## Resource Topology Designer

Graphical design tools are needed for creating and specifying requirement for resource topologies for enterprise IT environments. Similar design tools have long been in use in other domains such as circuit design or general engineering. It is surprising that design tools have not emerged for the design of IT solutions and resource environments yet, mainly due to the lack of formalism used in today's practice of creating IT solutions. We developed a Resource Topology Designer prototype that can be used for graphically designing resource topologies required for enterprise IT applications.


Grid Resource Topology Designer
sos.hpl.hp.com:4322/udc:/O=Grid/O=Globus/OU=hpl.hp.com/CN=Sven Graupner
15.9.75.157:4321/udc:/O=HP/OU=UNIX/CN=Jim Pruyne/EMAIL=pruyne@hpl.hp.com
localhost:4321/udc:/O=Grid/O=Globus/OU=hpl.hp.com/CN=Sven Graupner
internet
balancer
subnet
clear
UDC
RSL
smartfrog
properties
gridProxyInit
save
submit

```
(* BEGIN RSL for farm: 2000 *)
&(farm_1 =
  (id "2000")
  (version "1.1")
  (lb_2 =
    (id "1")
    (name "Cisco")
    (type "HTTP")
    (interface (name "eth0") (subnet "1"))
    (interface (name "eth1") (subnet "1"))
    (policy (element "round-robin"))
    (vip
      (name "15.9.12.48")
      (subnet "15.9.12.59")
    )
  )
  (tier_3
    (id "2")
    (name "web server")
    (interface (name "eth0") (subnet "1"))
    (interface (name "eth1") (subnet "1"))
    (role (element "web server"))
    (min-servers (element "2"))
    (max-servers (element "16"))
    (init-servers (element "4"))
  )
  (tier_4
    (id "1")
    (name "Oracle")
    (interface (name "eth0") (subnet "1"))
    (role (element "Oracle"))
    (min-servers (element "1"))
    (max-servers (element "1"))
    (init-servers (element "1"))
  )
  (tier_5
    (id "1")
    (name "Oracle")
    (interface (name "eth0") (subnet "1"))
    (role (element "Oracle"))
    (min-servers (element "1"))
    (max-servers (element "1"))
    (init-servers (element "1"))
  )
```

```
-------------------------------------------
ELEMENT-REGISTRY
-------------------------------------------
 (balancer)
 (subnet.37)
 (s-group)
 (disk.35)
```

**Figure 2: Designer with resource topology.**

Figure 2 shows the screen of the Resource Topology Designer. An IT solutions architect can design the resource topology required for supporting a desired IT application, a simple three-tier application in the shown case. The design tool provides a palette of possible resource types on the left-hand side that can be dragged into the drawing panel, where further requirements can be specified by clicking on the icon opening the configuration window for the resource. Resources can then be connected creating the topology. Connection information is used later for programming network and SAN switches establish-ing the desired connectivity among resources. This process leads to the graphical design of a resource topology as shown in the figure. From the graphical design, the tool generates a formal description of the request specification, which is shown in the text panel on the right-hand side. Resource topology specifications currently can be generated in three formats: FML [10], RSL [11], and SmartFrog [13].

The prototype design tool is based on Visio, a standard graphical drawing and design environment from Microsoft. Code generation is implemented within Visio in VB. Generated textual representations of resource topology designs can be stored as files and also submitted into allocation tools using an OGSI/GT3 [3] gateway (upgrade to WS-RF/GT4 [27] is indented when GT4 is available and stable). This way, the design tool allows designing resource topologies for enterprise applications off-line and independently from a specific resource pool or resource management system.

### Resource Request Submission and Flow.

After the textual description for a resource topology has been created, the <submit> button will initiate sending the description from the design tool to a specified Grid address (GSH – Grid Service Handle, similar like a URL) via GT3 middleware.

The destination of a resource topology design may be a resource allocation service (implemented as a Grid service) of a specific data center resource pool (such as shown in [14]); or the destination may be a broker service proposing choices of target data centers offering required resources.

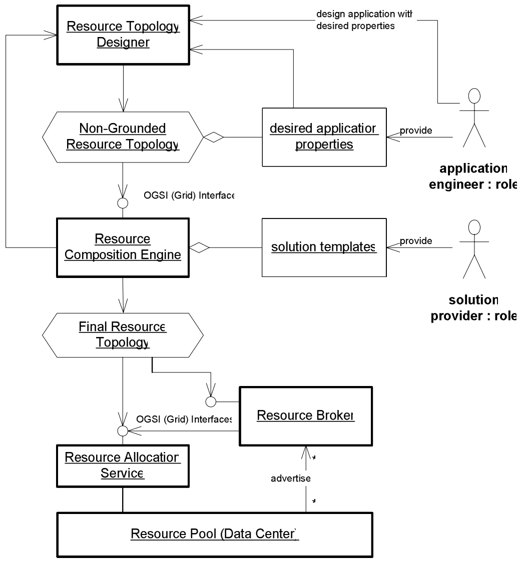

Resource Topology Designer
design application with desired properties
Non-Grounded Resource Topology
desired application properties
provide
application engineer : role
OGSI (Grid) Interface
Resource Composition Engine
solution templates
provide
solution provider : role
Final Resource Topology
OGSI (Grid) Interfaces
Resource Broker
Resource Allocation Service
advertise
Resource Pool (Data Center)

**Figure 3: Resource topology design/request flow.**

Figure 3 shows the workflow for resource topology information. The figure shows a scenario where a "non-grounded" resource topology will be routed to a *Resource Composition Engine* (presented in the following section) that performs the process of resource composition. After this, the final resource topology is either forwarded directly to a resource allocation service or to a broker for allocation.

The novelty of this approach is that an application engineer can design the resource environment for the application in terms of application-oriented properties as an iterative process potentially going through multiple design and redesign stages.

Design templates can be created for commonly used applications for typical resource topologies. For instance, an SAP procurement application with all its components can be sized on different sets of machines for supporting various capacities, e.g. defined by the numbers of supported users (such as 100, 500, or 10.000 users). Resulting resource topologies are stored as templates and selected based on the requirement provided by the application engineer when constructing a specific solution. The template search in this case would require input for the kind of application (SAP procurement) and a desired capacity (for 300 users) matching the 500 users template, which is returned to the Resource Topology Designer for final configuration (indicated as design cycle by the back arrow from the Resource Composition Engine to the Topology Designer). The composition engine performs the matching, resolves conflicts and binds configuration parameters.

## Policy-based Resource Composition

Resource composition is the process of identifying resources from a pool in a data center based on their primary specifications and the ability to built constructions of resources upon them incorporating all possible combinations. Possible combinations are ones that are not prevented by constraints.

Grid resource grounding as a simple one-way translation from a "higher" specification into a set of properties that can be matched is not sufficient here because of the problem of choices that exist for providing resources and the need to exploring choices. A decision space must be traversed for possible solutions meeting all constraints. The decision space can be loosely constraint such that multiple valid solutions exist and simply one of them is chosen. The decision space can also be strictly constraint such that only one or no solution exists. If no solution exists, the solver will report it.

### Policy-based model for resource construction.

When resources are combined to form other higher-level resources, a variety of constraints need to be obeyed. For example, when operating systems are loaded on a host, it is necessary to validate that the processor architecture assumed in the operating system is indeed the architecture on the host. Similarly, when an application tier is composed from a group of servers, it may be necessary to ensure that all network interfaces are configured to be on the same subnet or that the same version of the application is loaded on all machines in the tier.

We map the problem into a goal satisfaction problem that can be addressed using a constraint satisfaction formulation. A constraint solver is used for evaluating solutions [17].

Constraints form the core of the policy specification and are defined using expressions that use policy attributes as variables. Constraints contain first order predicates, arithmetic and logical operators, data types and quantifiers (for-all, exist, etc.) and other structural constructs such as let in, if then else etc.).

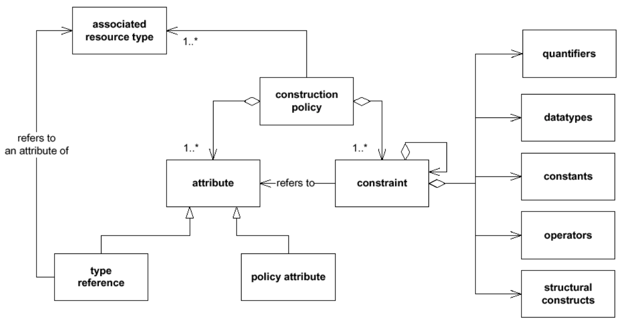


**Figure 4: Elements of construction policy.**

Expressions that can be used in policy specifications support the following primitives:

*Data types:* Data types may be imposed on attributes as constraints that have to be satisfied by the corresponding attribute, e.g. constraints can specify if a particular attribute should be a String, integer, float etc. This allows validation of data types when different underlying components are used to provide similar functionality.

*Constants:* Numeric or string constants define the values or thresholds for attribute values.

*Quantifiers:* Quantifiers are often used in constraints, e.g.  (for all),  (there exists), etc.

*Operators:* A number of operators can be used to combine attributes in defining constraints. These operators fall in the following categories:

- Arithmetic operators (+, -, *, /): These operators can be used for constructing arithmetic expressions on literals of the allowed data types.
- Comparison operators (<, >, <= , >=, =, !=): Comparison operators can be used to compare other expressions, and result in a boolean value.
- Boolean Operators (&&, ||, ! (unary not)): Provide logical expressions in constraints.
- Implication Operators (=> (logical implication), <= (reverse implication), <=> (equivalence, or if-and-only-if)): These operators allow

expression of dependencies between attributes, e.,g. (name = Solaris) => version \in {5.7,5.8};

- The instanceOf Operator: The <: operator is used to denote "an instance of" relationship. This allows constraints to be created that enforce data types on components or their attributes, e.g., // ensure that component "server" is an instance of type Appserver server <: AppServer;
- Set Operator: \in operator is used to constrain values of an attribute to be always in a set.

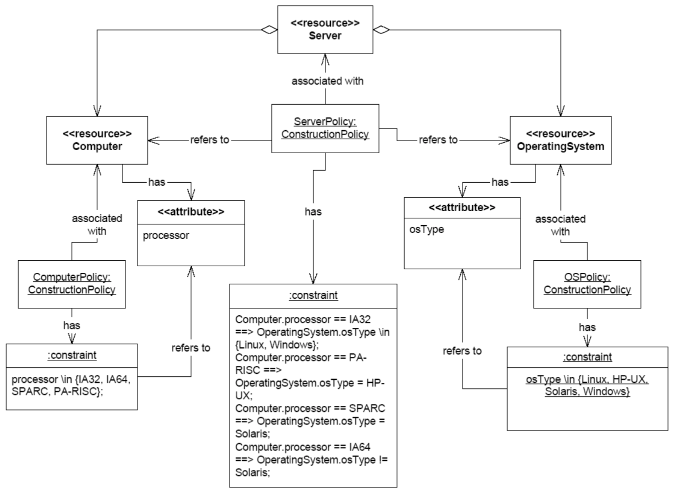
*Structural Constructs:* Other structural constructs (e.g., let in, if then else etc.) are used mostly for syntactic convenience. These familiar programming constructs simplify the task of the constraint writer when complex constraints have to be expressed in policy. This model allows constraints to be specified in a distributed and hierarchical manner

**Example for Resource Composition.**

When composing higher-level resources from other resources (such as the SAP procurement application from servers), a variety of resources may need to be put together. However not all possible combinations are valid. Policies can be attached to component types to ensure that the resulting construction is valid. To illustrate this principle, let us assume that we are trying to create servers. A server is simply defined as a computer system with an operating system on it. In order to create servers we need to select a computer and install an operating system image on it. A Server resource entity is thus constructed out of an underlying Computer resource and an OperatingSystem resource. However, not all computer types may be available in the resource pool, and not all operating system images would work on a given computer. Thus we need to define constraints that identify which computer and operating system image combinations are valid for Server construction.

In Figure 5, a resource composition is shown (as a model in UML notation) with all the components and all the constraints that are imposed on components. The example shows the composition of a specific resource Server. A Server in this example is a resource type that is composed from two other resource types: a Computer, and an OperatingSystem. A Computer has an attribute processor while an OperatingSystem has an attribute called osType. A policy associated with the Computer type states that the attribute processor can only take values in the set {IA32, IA64, SPARC, and PA-RISC}. Note that this constraint is specified by the system operator (perhaps because only these instances are available in the resource pool). Similarly, the construction policy associated with the OperatingSystem resource type states that the attribute osType can only take values in the set {Linux, HP-UX, Solaris, and Windows}. Again, the set is defined by the operator based on available operating systems within the utility. When the Server resource type is created, the type definition includes policy constraints that specify which osType can exist with which processor architecture.

Suppose a request specifies that it needs a resource of type Server with the additional constraint that Server.Computer.processor = PA-RISC. From the constraints specified as part of the Server, the policy system determines that the only valid value for OperatingSystem.osType = HP-UX.



<<resource>> Server
associated with
<<resource>> Computer
refers to
ServerPolicy: ConstructionPolicy
refers to
<<resource>> OperatingSystem
has
<<attribute>> processor
has
has
<<attribute>> osType
associated with
associated with
ComputerPolicy: ConstructionPolicy
:constraint
Computer.processor == IA32 ==> OperatingSystem.osType \in {Linux, Windows}; Computer.processor == PA-RISC ==> OperatingSystem.osType = HP-UX; Computer.processor == SPARC ==> OperatingSystem.osType = Solaris; Computer.processor == IA64 ==> OperatingSystem.osType != Solaris;
OSPolicy: ConstructionPolicy
has
:constraint
processor \in {IA32, IA64, SPARC, PA-RISC};
refers to
refers to
has
:constraint
osType \in {Linux, HP-UX, Solaris, Windows}

**Figure 5: Example of a Server composition.**

This example shows a number of aspects of constraint-based construction policies:

- By associating policy constraints with the individual component types, construction using those types can be controlled. By changing the allowed policy constraints, valid (or available) configurations can be maintained by the operators, and easily changed as needs change without extensive modifications about how the new types are handled.
- Policy constraints for a resource type depend only on the attributes of that type, or the attributes of the underlying resource types. This simplifies hierarchical specification of types, because such dependencies can usually be localized in the type hierarchy. The designer of a new type only has to deal with the preceding types it is using and the corresponding constraints on them. The policy system automatically accounts for dependencies that are created through transitional relationships.
- The policy system checks all constraints for validity when handling resource requests. Because it can locate constraints that are being

violated, the request specification can be checked for "correctness" with respect to those constraints. Similarly, during the instantiation of new resource types, the policy system can aid the designer by validating that at least one valid instance of the new type can be created.

- The system can also fill in attribute values based on correctness of constraints. This means that the requestor has the freedom to only specify the attributes that are meaningful, and let the system fill in the gaps. This simplifies the requests.
- The requestor can add additional constraints for the policy system as part of the request. Because the policy system forms the union of all constraints when constructing the system, request-specific policy can be easily incorporated during construction.

Many additional constraints can be added to this simple example to account for items such as licenses, software versions, or other attributes such as memory and CPU speed. In addition, higher level resources can be constructed in a similar manner. For example, a Server may take the role of a webServer, an applicationServer, or a databaseServer if the corresponding images are installed on it. Furthermore, by adding topology constraints (e.g. web server tier precedes an app server tier which in turn precedes a data base tier), one can construct much more complex resource types like a three-tier architecture or a highly available server and treat them as higher-level resources. These complex resource types may then be instantiated and deployed.

**Component Selection.**

In a resource utility, multiple components that offer the same base capability are often available in the resource pool. Examples may be different types of servers, firewalls, or network switches. However, these components frequently differ in the capabilities (e.g., security, availability, throughput) offered by them. Such capabilities can be captured by the attributes of the components, and depending upon the capabilities desired by the requestor, the appropriate components can be selected to meet the users' requests. In this section, we provide four examples to highlight how policies can be used to provide construction choices for components.

## Advertising Resource Constructions

One of the issues remaining is how “possible construction capabilities” can be advertised. Possible combinations may be large. In Grids, resources are described by their property sets, which are registered in a directory service or a registry, from where descriptions can be retrieved by co-allocators for making decisions where resource requests can be deployed. However, in enterprise environments resource constructions must be addressed opening a theoretical unbound space of possible combinations that cannot all be registered or advertised.

One approach addressing the issue is advertising only the base resources (resource atoms) and leaving possible constructions to the requestor. Some constructions may depend on the availability of certain resources such as virtual machine software or licenses and associated cost. For advertising those capabilities, construction enabler resources (e.g. VM software and licenses) can be registered along with their dependencies on resource atoms.

A third choice is to register capabilities rather than resources. Resource atoms such as hardware servers would be registered as capabilities, as resource constructions would be registered as capabilities, along with a set of dependent capabilities (as sets of AND or OR constraints), which recursively can be traversed generating the tree.

## Related Work

Introducing constraints in UML specification of systems for configuration purposes is discussed in [18]. They define a set of construction rules at one place termed a domain. In that sense the approach is similar to expert systems. In our approach, we embed constraints hierarchically thus distributing constraints on to various resource types, and taking into account these constraints as the construction happens as opposed to creating a large number of constraints (rules) a priori. Our approach enables flexibility and extensibility in specification of constraint and in automatic construction depending on the user requirements. The differing user requirements may result in one construction being different from another. We have also applied the concept to CIM [12], which is de-facto standard for management of IT infrastructure in the industry.

The ClassAds MatchMaking work [19] assumes that the match-maker matches the requestor entity’s request against the provider entity’s ClassAds (which are specifications in a semi-structured language). The assumption is that all the resources (like machines) exist a-priori and have been advertised. In a resource-utility environment however, some of the resource instances may not even exist a-priori (as is the case with transient/virtual resources) or may be logically constructed resources that have to be instantiated on-demand (e.g. appserver/tier/farm/e-commerce site). This causes a problem for approaches that undertake match-making only on instances. We enable construction on-the-fly by embedding constraints hierarchically in the resource types as described in this paper. The same concepts are extensible to resource instances as well. It is also not clear whether the ClassAds language supports first-order logic and linear arithmetic. As we have shown in the examples, it is important to have notions of quantifiers, implications, equivalences and other first order-logic expressions for reasoning.

There is lot of work that has been done in terms of specifying, and associating events, conditions and actions for policies, namely IETF [20], CIM [21], PARLAY [22], PONDER [26] etc. Additional work relates to using policy for SLA management [23]. These bodies of work, to the best of our knowledge, have not looked at incorporating first-order logic and linear arithmetic based constraints in resource types for automatic constructions of resources and have not used a constraint satisfaction approach for arriving at a constructed resource specification. The WS-Policy [24] work at OASIS has focused on generic schemas for specifying arbitrary policy assertions on web services. The constraints as specified in this article may be embedded inside these assertions.

## References:

[1] Ian Foster, Carl Kesselman, Jeff Nick, Steve Tuecke: *The Physiology of the Grid – An Open Grid Services Architecture for Distributed System Integration*, http://www.globus.org/ogsa, June 2002.

[2] Shane Robison: *Grids for the Enterprise*, Grid.Middleware Spectra, 3rd Edition, Spring 2004.

[3] GGF: Open Grid Services Infrastructure (OGSI), http://www.gridforum.org/ogsi-wg, February 2003.

[4] Globus: *The Globus Toolkit*, http://www.globus.org.

[5] Sven Graupner, Nigel Cook, Jean-Marc Chevrot: *HP Adaptive Control Specification v1.1*, October 2003.

[6] Hewlett-Packard: *Utility Data Center*,.

[7] *Portable Batch System*, http://www.openpbs.org.

[8] *Condor*, http://www.cs.wisc.edu/condor.

[9] Platform: Load Sharing Facility (LSF), http://www.platform.com/products/LSF/.

[10] Hewlett-Packard: *Farm Markup Language FML*, internal specification, 2002.

[11] Globus: The Globus Resource Specification Language RSL, http://www.globus.org/gram/rsl_spec1.html.

[12] DMTF: *CIM Modeling*, http://www.dmtf.org/standards/standard_cim.php.

[13] Hewlett-Packard: *SmartFrog*, http://www.smartfrog.org.

[14] Pruyne, J., Machiraju, V.: *Quartermaster: Grid Services for Data Center Resource Reservation*, Global Grid Forum Workshop on Designing and Building Grid Services, Chicago, October 8, 2003.

[15] Foster, I., Kesselman C.: *The GRID – Blueprint for a New Computing Infrastructure*, Chapter 11.2 Resource Management, pages 262-265, Morgan Kaufman Publishers, 1999.

[16] Singhal, S., et.al: *Quartermaster – A Resource Utility*, HP Tech Con, 2004.

[17] Sahai, A., Singhal, S., Machiraju, V., Joshi, R.: *Automated Policy-Based Resource Construction in Utility Computing Environments*, to appear in IEEE/IFIP Network Operations and Management Symposium (NOMS), Seoul, Korea, April 2004.

[18] Felfernig A, Friedrich G. E et al. *UML as a domain specific knowledge for the construction of knowledge based configuration systems*. In the Proceedings of SEKE'99 Eleventh International Conference on Software Engineering and Knowledge Engineering, 1999.

[19] Raman R, Livny M, Solomon M. *MatchMaking: Distributed Resource Management for High Throughput Computing*. In the proceedings of HPDC 98.

[20] *IETF Policy*. http://www.ietf.org/html.charters/policy-charter.html

[21] *DMTF-CIM Policy* http://www.dmtf.org/standards/documents/CIM/CIM_Schema26/CIM_Policy26.pdf

[22] *PARLAY Policy Management*, http://www.parlay.org/specs

[23] Sloman, M.J.: *Policy Conflict Analysis in Distributed Systems*, In the proceedings of Journal of Organizational Computing,, 1993

[24] *OASIS WS-Policy WG*, http://www.oasis-open.org

[25] Verma, D., Beigi, M., Jennings, R.: *Policy based SLA Management in Enterprise Networks*, Workshop on Policy, POLICY 2001.

[26] Damianou, N., Dulay, N., Lupu, E., Sloman, M.: *The Ponder Policy Specification Language*, 18-38, POLICY 2001.

[27] OASIS TC and Global Grid Forum, *The Web Services Resource Framework (WSRF)*, April 2004, http://www.globus.org/wsrf.

[28] OASIS, WSDM: *Management Using Web Services (MUWS 1.0)*, Working Draft September 29 2004.

[29] *Maui Scheduler*, http://www.supercluster.org/maui.

[30] Sahai, A., Graupner, S.,: *Web Services in the Enterprise: Concepts, Standards, Solutions and Management*, Network and Systems Management Series, 310 p., ISBN 0-387-23374-1, Springer Verlag, 2004.