# Ontology-based Resource Matching in the Grid — The Grid meets the Semantic Web *

Hongsuda Tangmunarunkit, Stefan Decker, Carl Kesselman

Information Sciences Institute
University of Southern California
{Hongsuda, Stefan, Carl}@isi.edu

**Abstract.** The Grid is an emerging technology for enabling resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations. In the Grid environment, shared resources and users typically span different organizations. The resource matching problem in the Grid involves assigning resources to tasks in order to satisfy task requirements and resource policies. These requirements and policies are often expressed in disjoint application and resource models, forcing a resource selector to perform semantic matching between the two. In this paper, we propose a flexible and extensible approach for solving resource matching in the Grid using semantic web technologies. We have designed and prototyped an ontology-based resource selector that exploits ontologies, background knowledge, and rules for solving resource matching in the Grid.

## 1 Introduction

The Grid is an emerging technology for enabling resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations [12, 11]. Grids are used to join various geographically distributed computational and data resources, and deliver these resources to heterogeneous user communities [30, 9, 13]. These resources may belong to different institutions, have different usage policies and pose different requirements on acceptable requests. Grid applications, at the same time, may have different constraints that can only be satisfied by certain types of resources with specific capabilities. Before a resource (or a set of resources) can be allocated to run an application, the user or an agent must select resources appropriate to the requirements of the application [7]. We call this process of selecting resources based on application requirements *resource matching*. In a Grid environment, where resources may come and go, it is desirable and sometimes necessary to automate the resource matching to robustly meet specific application requirements.

Existing resource description and resource selection in the Grid is highly constrained. Traditional resource matching, as exemplified by the Condor Matchmaker [26] or Portable Batch System [23], is done based on symmetric, attribute-based matching. In these systems, the values of attributes advertised by resources are compared with those required by

* This work was supported by National Science Foundation under grant EAR-0122464. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

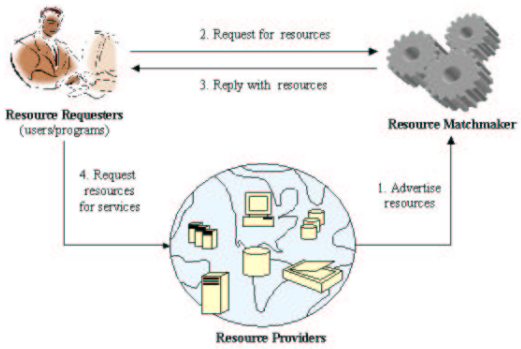



**Fig. 1.** Resource Matching: the matcher gathers resource information from resource providers. When a requestor submits a request to the matcher for a resource match, the matcher searches for the appropriate resource and returns the best result to the requestor. The requestor can then request the resource for its services or execution.

jobs. For the comparison to be meaningful and effective, the resource providers and consumers have to agree upon attribute names and values. The exact matching and coordination between providers and consumers make such systems inflexible and difficult to extend to new characteristics or concepts. Moreover, in a heterogeneous multi-institutional environment such as the Grid, it is difficult to enforce the syntax and semantics of resource descriptions.

To illustrate, consider if a machine's operating system is described as "SunOS" or "Linux." To query for a machine that is "Unix" compatible, a user either has to:

1. explicitly incorporate the Unix compatibility concept into the request requirements by requesting a disjunction of all Unix-variant operating systems, *e.g.*, (OpSys="SunOS" || OpSys="Linux"), or
2. wait for all interesting resources to advertise their operating system as Unix as well as either Linux or SunOS, *e.g.*, (OpSys={"SunOS," "Unix"}), and then express a match as set-membership of the desired Unix value in the OpSys value set, *e.g.*, hasMember(OpSys, "Unix").

In the former case, the disjunctive requirements become unwieldy as more abstract concepts are developed. In the latter, the advertisements become more complex and all resources must be updated before a match can occur.

In this paper, we propose a flexible and extensible approach for performing Grid resource selection using an ontology-based matchmaker. Unlike the traditional Grid resource selectors that describe resource/request properties based on symmetric flat attributes (which might become unmanageable as the number of attributes grows), separate ontologies (*i.e.*, semantic descriptions of domain models) are created to declaratively describe resources and job requests using an expressive ontology language. Instead of exact syntax matching, our ontology-based matchmaker performs semantic matching using terms defined in those ontologies. The loose coupling between resource and request

descriptions remove the tight coordination requirement between resource providers and consumers. In addition, our matchmaker can be easily extended, by adding vocabularies and inference rules, to include new concepts (*e.g.,* Unix compatibility) about resources and applications and adapted the resource selection to changing policies. These ontologies can also be distributed and shared with other tools and applications.

We have designed and prototyped our matchmaker using existing semantic web technologies to exploit ontologies and rules (based on Horn logic and F-Logic) for resource matching. In our approach, resource and request descriptions are asymmetric. Resource descriptions, request descriptions, and usage policies are all independently modeled and syntactically and semantically described using a semantic markup language; RDF schema. Domain background knowledge (*e.g.,* "SunOS and Linux are types of Unix operating system") captured in terms of rules are added for conducting further deduction (*e.g.,* a machine with "Linux" operating system is a candidate for a request of a "Unix" machine). Finally, matchmaking procedures written in terms of inference rules are used to reason about the characteristics of a request, available resources and usage policies to appropriately find a resource that satisfies the request requirements. Additional rules can also be added to automatically infer resource requirements from the characteristics of domain-specific applications (*e.g.,* 3D finite difference wave propagation simulation) without explicit statements from the user.

The rest of the paper is organized as follows. In Section 2, we summarize the semantic web technologies that we use in our work. Section 3 reviews related work to the resource selection problem. Section 4 describes the features and architecture of our ontology-based matchmaker and provides a matching example. Finally, we conclude our paper in Section 5.

## 2 Semantic Web Technologies

The Semantic Web [2] is the next generation of the web which evolves toward semantic knowledge representations and intelligent services (*e.g.,* information brokers, search agents) where information can be processed by machines. To fully realize this goal, standards for exchanging machine-understandable information have to be established. These standards define not only the syntactic representation of information, but also their semantic content. A technology stack, suggested by the W3C, that we use in our work consists of Resource Description Framework (RDF), which provides data model specification and an XML-based serialization syntax; ontologies, which enable the definition and sharing of domain vocabularies; and rules, which allow declarative processing of data.

### 2.1 The Resource Description Framework

At present, services on the Web are single islands. Common data models and data exchange standards are required in order to enable the fast integration of different datasources and to bridge semantic differences. The Web community has proposed the Resource Description Framework (RDF) [17] as a data model suitable for information integration tasks. The data model serves as a foundation for *ontology* languages.

### 2.2 Ontologies

An ontology is a specification of a conceptualization [14]. In this context, specification refers to an explicit representation by some syntactic means. In contrast to schema languages (like XML Schema or DTDs) ontologies try to capture the semantics of a domain by deploying knowledge representation primitives, enabling a machine to (partially) understand the relationships between concepts in a domain. Additional knowledge can be captured by axioms or rules. In the Web context, RDF-Schema [3] and OWL[1] are recommendations from the W3C for ontology modeling languages.

### 2.3 Rules

Rules, in combination with RDF and ontologies, are an active field of research. Rules can be used to capture domain knowledge. We have chosen TRIPLE [28] as the rule language for our approach. TRIPLE is based on Horn logic [21] and borrows many basic features from F-Logic [16]. It is especially designed for querying, transforming, and reasoning with RDF data. TRIPLE has no built-in support for knowledge representation languages, but can be configured by axioms to support arbitrary modeling languages (*i.e.,* RDF-Schema).

## 3 Related Work

### 3.1 Information Systems

Related to the resource selection problem are information systems for discovering, aggregating, publishing and querying against information about resources and services. Globus MDS [6] and UDDI [31] are two such examples; MDS has been widely used in the Grid community for resource discovery while UDDI has been used in the web community for business service discovery.

Both MDS and UDDI support simple query languages. However, they do not offer expressive description facilities, nor provide sophisticated matchmaking capabilities. In this environment, the usage scenario involves resource providers publishing descriptions of their properties to an information service/registry. A resource consumer then queries the registry to identify candidate resources prior to generating actual requests. Based on the returned queries, resource selection can be done either by a user or procedural algorithm. In this scenario, policy enforcement happens when a request is submitted to the resource/service providers. It is possible that a user request will fail, and hence prior resource selection effort is wasted.

### 3.2 Resource Matching in the Grid

We do not know of existing applications of ontological reasoning to matchmaking in the Grid. In the following, we review several variations on symmetric, attribute-based matchmaking technologies. These variations provide increasing levels of expressiveness but still require symmetric attribute models in their descriptive and constraint languages.

[1] See http://www.w3.org/2001/sw/WebOnt/ for more information.

*Symmetric attribute-based matching.* As part of Condor [5], Rajesh Raman *et. al* developed the classified advertisement (ClassAd) matchmaking framework for solving resource allocation problem in a distributed environment with decentralized ownership of resources [26]. This framework provides a bi-lateral match, allowing both resource consumers and providers to specify their matching constraints, *e.g.*, requirements and policy.

In this framework, properties of requests and resources are characterized in a form of arbitrary but common syntax (*e.g.*, attribute-value pairs) capable of representing both characteristics and policies (as shown in Figure 2). A symmetric requirement (expressed as a *constraint* statement) is then evaluated to determine, for each request-resource pair, whether there is a match. For the matching to work, it is crucial that both requests and resources use the same attribute names and agree upon attribute values. When multiple resources match a job requirement, a function (expressed by a *Rank* expression) can be used to assign an order to resources and the highest ranked resource is returned as a match.

Request ClassAd:

```
[ Type = "Job"; Owner = "user1";
  Constraint = other.Type == "Machine" && Arch == "INTEL"
    && OpSys == "SOLARIS251" && Disk >= 10000;
  Rank = other.Memory; ]
```

Resource ClassAd:

```
[ Type = "Machine"; Name = "m1"; Disk = 30000; Arch = "INTEL";
  OpSys = "SOLARIS251"; ResearchGrp = "user1", "user2";
  Constraint = member(other.Owner,ResearchGrp) && DayTime > 18*60*60;
  Rank = member(other.Owner,ResearchGrp) ]
```

**Fig. 2.** Two examples of Condor ClassAds. For each resource-request pair, constraint clauses are checked for compatibility against the other's properties. Rank is used to select among multiple matches.

*Gang-Matching.* To overcome the binary matching limitation of Condor Matchmaker, Raman *et. al* later proposed the gang-matching extension [24, 25], allowing a request ClassAd to specify a list of required bilateral matches. For example, a request may require one or more resources each of which must satisfy its described requirements, as well as, the inter-resource constraints. However, this extension does not support *set-matching* where resources are defined by their aggregate characteristics, *e.g.*, a set of computers with aggregate memory greater than 10 GB. Chuang Liu *et. al* proposed the set-extended ClassAd language and a technique for specifying and solving set-matching problem [20]. Although their set-matching system can be extended to solve the gang-matching problem, their system does not currently support this capability.

*Constraint-satisfaction-based matching.* Chuang Liu *et. al* recently proposed the Redline matching system: an alternative approach for doing resource selection in the Grid [19]. In this framework, the matching problem is first transformed into a constraint satisfaction problem [2] , the set of constraints are then checked to make sure that no conflicts occur,

[2] A constraint satisfaction problem (CSP) consists of a constraint C over variables $x_1, ..., x_n$ and a domain $D$ that maps each variable $x_i$ to a finite set of values, $D(x_i)$, that it is al-

and finally existing constraint solving technologies [22] (such as integer programming) are used to solve the transformed problem. Similar to Condor matchmaker, the Redline matching system is based on symmetric description of resource and request (*i.e.,* the same description syntax is used to describe both resources and requests). However, comparing to ClassAd, the Redline language is more expressive. It supports both gang-matching and set-matching capabilities.

A common requirement among these systems is the symmetric syntactic description of resources and requests properties. As illustrated in the previous example in Section 1, it is difficult to introduce new concepts or characteristics into the system. Moreover, in the Grid environment, where resources and users span multiple organizations, it may be difficult to guarantee that resources and requests will use the same attribute names, and that the semantics of the same attributes are interpreted the same way by both resource providers and consumers.

Our ontology-based matchmaker, on the other hand, is based on an asymmetric description. The system uses ontologies to semantically describe requests and resources. Matching between request specification to resource capabilities is done in terms of rules. Different request description models, along with the mapping rules, can be easily added to our matchmaker. Similar to these matching systems, our matchmaker provides the ability to describe properties and matching preference. Our matchmaker also supports a binary matching and gang-matching. We plan to support set-matching in the future.

### 3.3 Matchmaking in Other Domains

We summarize existing work in other domains that, similar to our work, have developed matchmakers based on ontologies.

*DAML+OIL based Matchmaking.* DAML+OIL based Matchmaking [18] describes a matchmaking system based on Semantic Web Technology, using a Description Logic reasoner to compare ontology-based service descriptions. Service advertisements are expressed as class expressions. The elements of the class expressions are taken from a domain ontology and a specific service ontology. During the matchmaking process the advertisements are classified into a hierarchy. The next step is to classify the request's service profile and the complement of the service profiles. Classifying the service profile and its complement allows the matchmaker to determine which service advertisements are compatible with the requests service profile. Our approach is different from their approach in that instead of using classification, we write rules to both capture background knowledge and explicitly determine when a request (*i.e.,* advertisement) matches resources (*i.e.,* services).

*InfoSleuth.* InfoSleuth [1] is an agent-based information discovery and retrieval system. The system adopts broker agents for syntactic and semantic matchmaking. The broker matches agents that require services with other agents that can provide those services. Agent capabilities and services are described using a common shared ontology of attributes and constraints which all agents can use to specify advertisements and requests

lowed to take. The CSP represents the constraint $C.x_1D(x_1).....x_nD(x_1)$. For example $C = x_1 > 1, x_1 + x_2 < 4, D(x_1) = [1, 2, 3], D(x_2) = [1, 2, 3]$.

to the broker. The matchmaking is then performed by a deductive database system, allowing rules to evaluate whether an expression of requirements matches a set of advertised capabilities. This approach is similar to ours. We extend the InfoSleuth approach in several directions: first we use RDF based Semantic Web technology, and second we provide more detail for how ontology-based reasoning interacts with the matchmaking rules. Furthermore we introduce background knowledge to the matchmaking process, allowing for a more flexible matchmaking procedure.

*LARKS/RETSINA.* LARKS/RETSINA [29] is a multiagent infrastructure. LARKS is an Agent Capability Description Language (ACDL). LARKS offers the possibility to use domain knowledge by using an ontology written in the concept language ITL. Unlike our approach, LARKS does not use a declarative rules for matchmaking.

## 4 Ontology-based Resource Matching

Our ontology-based matchmaker is developed based on semantic web technologies (described in section 2). In this section, we first summarize the desired features of our matchmaker, describe its architecture and finally the methodology.

Desired features of the ontology-based matchmaker are:

- *Asymmetric description of resource and request.* In our framework, the description of resources and requests are modeled and described separately. A semantic match between the two models will be provided. Due to the asymmetric description, no coordination between resource providers and consumers is required before new description vocabulary is added. This is not true for the symmetric attribute-based matching described in Section 3.
- *Sharing and Maintainability.* The ontologies are sharable and easier to maintain and to understand than flat attribute lists.
- *Bilateral Constraints.* A request description allows the request to specify its resource constraints in terms of requirements. At the same time, each resource can also independently express its usage policies (*e.g.,* identifying who is allowed the access) restricting matches to applications/requests. The matchmaker takes the policies of each resource and request constraints into accounts when searching for a match.
- *Ability to describe matching preference.* Both request and resource can specify its preference when multiple matches are found.
- *Multi-lateral matching.* A user can submit a request that requires multiple simultaneous resources, each matching its requirement clause. Set-matching [20] capability will also be added in the future.
- *Integrity Checking.* The matchmaker can use the domain knowledge to help identify inconsistencies in the resource descriptions before accepting it as an available resource. The integrity check can also be done for the request to make sure that there are no conflicts in the resource requirements. For example, a resource or request advertisement with `OperatingSystem="Windows2000"` and `CPUFamily="Sparc"` should be rejected.
- *Expressiveness.* Due to the asymmetric description, the request can be modeled specifically for domain specific applications. The high-level application characteristics can

be provided by the user. Furthermore, high-level characteristics can be automatically mapped to specific resource requirement configurations by the matchmaker.

– *Flexibility and Extensibility.* New concepts can be easily added into the ontology, *e.g.*, tightly-coupled machines or an MPI application. In addition, new constraints, *e.g.*, an MPI application requires tightly coupled machines, can be easily added in terms of rules.

### 4.1 Matchmaker Architecture

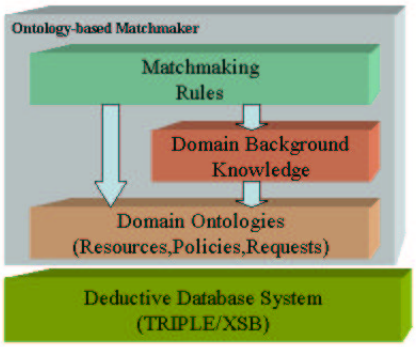



**Fig. 3.** Ontology-based Matchmaker

The ontology-based matchmaker consists of three components:

1. the *ontologies*, capturing the domain model and vocabulary for expressing resource advertisements and job requests,
2. *domain background knowledge*, capturing additional knowledge about the domain, and
3. *matchmaking rules*, defining when a resource matches a job description.

Figure 3 shows the relationship between these components. An arrow indicates the dependency between different components. For example, the background knowledge uses the vocabulary from the ontologies to capture background information. Matchmaking rules use both ontologies and background knowledge to match a request to resources. Our ontology-based matchmaker is built on top of TRIPLE/XSB deductive database system.

The matchmaking problem can be formally defined as follows. Let $\alpha$ be the set of all resource advertisements in a repository; $\mathcal{O}$ be a domain ontology; $\mathcal{B}$ be a set of domain background knowledge; $\mathcal{R}$ be a set of matchmaking rules defining a binary predicate $match$; and $\models_{match}$ be a consequence operator (such as the usual consequence operator from classical logic). Then for a given query or advertisement $q$, matchmaking is computing the set $\{\alpha \mid \mathcal{O} \cup \mathcal{B} \cup \mathcal{R} \models_{match} match(\alpha, q)\}$. In the following we describe the components in more details.

**Ontologies** We developed three initial ontologies. Using RDF-Schema, each ontology defines objects, properties of objects, and relationships among objects. These three ontologies are:

- *Resource ontology*. The resource ontology provides an abstract model for describing resources (*e.g.*, `ComputerSystem, OperatingSystem`), their capabilities (*e.g.*, `OperatingSystem.TotalPhysicalMemory=5000MB`) and their relationships (*e.g.*, `RunningOS(ComputerSystem` $X$`,OperatingSystem` $Y$`)`). Our initial model focuses on describing the capabilities of computational resources. The majority of our resource vocabularies are taken from the Common Information Model (CIM)—a conceptual information model for describing resource management that is neutral to vendors and implementation [4]. However, CIM focuses more on describing the physical elements of a system, not on abstract capability description. We started our resource model with a subset of CIM schema, modified and extended it to fit our requirements.
- *Resource Request ontology*. This ontology captures a request, properties of the request (*e.g.*, Request.Owner), characteristics of the request (*e.g.*, `JobType=`"`MPI`") and the resource requirements (*e.g.*, `MinPhysicalMemory=1G, NumberOfCPUs=16`). The ontology supports a request of multiple independent resources.
- *Policy ontology*. A model that capture the resource authorization and usage policies. For example, `AuthorizedAccounts=(ComputerSystem` $X$`, {user1,user2})` indicates a set of accounts that are authorized to access ComputerSystem $X$. Our model currently supports a simple authorization policy. We will expand the model to include usage policies in the future.

*Ontology Creation.* We use Protégé, an ontology editor which supports the RDF Schema, to develop our ontologies. Protégé [8] provides an integrated environment for editing ontology and instances. It hides the ontology language from the ontology developers allowing developers to work with high level concepts which as a result leads to rapid ontology development.

The Grid resources and users normally span across different organizations. The ability to share and exchange resource information is necessary for the creation of resource advertisements and job requests. Since our ontologies are represented by a semantic web standard, they can be easily exchanged and shared by other tools such as other rule-based engines or knowledge-based systems.

**Domain Background Knowledge** The background knowledge captures additional knowledge about the domain (usually at the instance level) which is not captured by the ontology. This knowledge is used during the matchmaking process. We use TRIPLE, a rule system based on deductive database techniques, as an effective and expressive representation mechanism for implementing the background knowledge. The knowledge is captured in terms of rules. These rules use the vocabulary defined by the ontology to add additional axioms which cannot be expressed by the Ontology language and which typically influence the reasoning with instances. Figure 4 shows an example of typical background rules. These rules define which operating systems are compatible with each other and define `compatible` as transitive, reflexive, and symmetric. They also define

```
@gridBackground { // specifies grid background knowledge
  Linux[rdfs:subClassOf->GR:OperatingSystem].
  Unix[rdfs:subClassOf->GR:OperatingSystem].
  Debian[rdf:type->Linux].  Redhat[rdf:type->Linux].
  SunOS[rdf:type->Unix].    Linux[rdf:type->Unix].

  // transitivity axiom
  FORALL X,Y,Z X[compatibleWith->Z]<- X[compatibleWith->Y] AND Y[compatibleWith->Z].

  // identity axiom
  FORALL X X[compatibleWith->X].

  //symmetry axiom
  FORALL X,Y X[compatibleWith->Y]<- Y[compatibleWith->X].

  FORALL X,Y,Z X[substitutes->Z] <- (Y[rdf:type->Z] and
     X[substitutes->Y]) or X[compatibleWith->Z].
}
```

**Fig. 4.** Part of Grid Background Knowledge

`substitutes` in terms of `compatible`, to determine which operating systems can be substituted by each other.

**Matchmaking Rules** The matchmaking rules define the matching constraints between requests and resources. These rules are implemented using TRIPLE rule language (Section 2.3). In additon to syntactic string/numeric equality and group membership rules which are primitive constraint expressions in existing attributed-based matchmakers, TRIPLE can reason about constraints in terms of object properties and their relationships specified in RDF data and background knowledge.

Figure 5 depicts part of the matchmaking effort. The rules require inputs—the set of advertisements `Data`, background knowledge `Background`, and domain ontology `Ontology`. The first rule defines the `match` property, which states when a `JobRequest` matches an advertisement of a ComputerSystem. This rule is defined in terms of other rules, *e.g.*, `matchesOS` (*match OperatingSystem*) and `matchesFS` (*match FileSystem*). The rule defining `matchOS` uses the Grid background knowledge by asking if the operating systems requested by the job can be substituted by the operating system provided by the `ComputerSystem` resource. The `matchesFS` rule checks whether the filesystem associated with the resource can satisfy the requested filesystem requirement. In the resource ontology, there are two classes of file systems (*i.e.*, LocalFileSystem and NetworkFileSystem) which are subclasses of the class `FileSystem`. The `matchesFS` rule invokes the reasoning with the ontology to check if its argument `Y` (associated with the resource) is an instance of the class `FileSystem`, and then performs simple arithmetic comparison to ensure that there is enough diskspace available for the request.

### 4.2 Deductive Database Engine for Ontology-based Matchmaker

We use TRIPLE/XSB as a deductive database system [27]. TRIPLE/XSB supports RDF-Schema and TRIPLE rule language. It is implemented on top of the XSB deductive database system [32]. TRIPLE rules are first compiled into XSB rules, which are then further compiled into instructions for the XSB virtual machine. TRIPLE/XSB evaluates

```
FORALL Data, Background @match(Data,Background,Ontology){
 FORALL X,Y  X[matches->Y] <-
        X[rdf:type->GR:JobRequest]@Data
    and Y[rdf:type->GR:ComputerSystem]@rdfschema(Data,Ontology)
    and ((X.GR:RequestResource.GR:RequiredMemory)@Data)[matchesMEM->(Y.GR:RunningOS)@Data]
    and ((X.GR:RequestResource.GR:RequiredOS)@Data)[matchesOS->(Y.GR:RunningOS)@Data]
    and ((X.GR:RequestResource.GR:RequiredFS)@Data)[matchesFS->(Y.GR:HostedFileSystem)@Data]
    and ((X.GR:RequestResource.GR:RequiredCPU)@Data)[matchesCPU->Y].

// checking OperatingSystem requirement
 FORALL X,Y  X[matchesOS->Y] <-
        X[rdf:type->GR:OSRequirement]@Data
    and Y[rdf:type->GR:OperatingSystem]@Data
    and ((X.GR:OSType)@Data)[substitutes->(Y.GR:OSType)@Data]@Background.

// checking FileSystem Requirement
 FORALL X,Y  X[matchesFS->Y] <-
        X[rdf:type->GR:FSRequirement]@Data
    and Y[rdf:type->GR:FileSystem]@rdfschema(Data,Ontology)
    and (X.GR:MinDiskSpace)@Data =< (Y.GR:AvailableSpace)@Data.
}
```

**Fig. 5.** Part of Matchmaking Rules

matchmaking rules, in combination with background knowledge and ontologies, to find the best match for the request.

**Performance.** The number and complexity of the rules determine the performance of the matchmaker. In the case of non-recursive rules, the matchmaking process is equivalent to computing database queries and views, which is known to be efficient using conventional indexing and join techniques. In the case of recursive rules (as shown in our example), the evaluation may be time consuming. However, there are evaluation techniques developed for deductive databases (which are deployed by XSB) that can be used to avoid unnecessary rule evaluations. In addition, careful rule development can further optimize the system performance.

### 4.3 Methodology

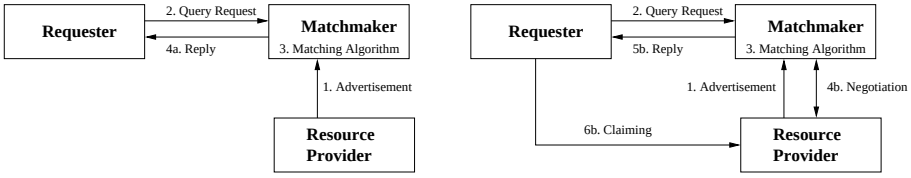
The first step in developing an ontology-based matchmaker is to create domain ontologies. We have modeled and prototyped three ontologies mentioned in Section 4.1. Once the ontologies are defined, the vocabularies can then be used to generate background knowledge and matching rules. Ontology modeling is an iterative process. When new types of resources become available or when existing resources offer new capabilities, the resource ontology has to be updated to reflect the new status. Similarly, when the vocabularies change, the background knowledge and rules have to be adjusted accordingly. The ontologies, background knowledge and matching rules can be incrementally extended and maintained as the Grid evolves.

### 4.4 Matchmaking Framework

**Resource Discovery.** The matchmaking framework consists of ontology-based matchmakers, resource providers and resource consumers or requesters. Resource providers periodically advertise their resources and capabilities to one or more matchmakers (Step 1 in Figure 6). The advertisement is generated either due to system configuration, or in

response to a query from the matchmaker. It is possible that resource providers may express their capabilities using a schema that is different from our ontology. In this case, we simply assume that a mapper between this schema to our ontology can be implemented. Upon receiving an advertisement, the matchmaker applies the appropriate mapper and then updates its list of available resources.

Each matchmaker maintains an aggregated list of available resources from the received advertisements. The list is based on soft-state updates from the providers. Each item on the list has an effective period associated with it. Whenever an advertisement is received, the effective period is appropriately set. When the effective period expires, the item is removed from the list. In the recently proposed Open Grid Services Architecture (OGSA) [10], the matchmaker could directly subscribe to service providers to get periodic XML-based status updates. The descriptive terms for services in that environment are extensible; this scenario is the central motivation for using extensible ontologies to map between evolving service descriptions and request models.




(a) Matchmaking Service (b) Brokering Service

**Fig. 6.** Actions involved in the two services provided by the matchmaker

**Matchmaker Services.** There are two services provided by the matchmaker; the matchmaking and brokering services. The brokering service is built on top of the matchmaking service. One of the two services is invoked when a requester submits a job request to the matchmaker (Step 2). A request is composed using vocabulary in the request ontology. Upon receiving a request, the matchmaker activates the matching rules to find a list of potential matches sorted according to the requester's preference criteria (Step 3). If the request is for a matchmaking service, the matchmaker simply returns the matched list (or *NoMatchFound*) to the requester (Step 4a).

In the case that the request is for a brokering service, there are two steps involved—negotiating and claiming. The matchmaker sends *negotiation* messages using the *negotiation protocol* to the highest-rank item in the list informing resources about a potential job request (Step 4b). A resource provider can accept or deny the request. If it accepts the request, appropriate resources will be allocated for the future job. If the request is denied, the matchmaker then attempts to negotiate with the next highest-ranked item in the list until the matched list is exhausted. If no match is found, the matchmaker returns to the requester a *NoMatchFound* message, else it returns a list of matched resources and associated handles, and appropriately updates its list of available resources and their

status (Step 5b). The requester can then contact the resource providers directly for their services using the *claiming protocol* (Step 6b).

### 4.5 Prototype Implementation

We have prototyped three initial ontologies (mentioned in Section 4.1) and an ontology-based matchmaker. Currently, the matchmaker reads instances of requests, available resources and authorization policies (described by vocabularies in the request, resource and policy ontologies, respectively) from an RDF file. Ontology instances are manually created using Protégé (which saves instances in the RDF format).

Our matchmaker currently offers only the matchmaking service. We plan to include the brokering service in the future. The user can activate the matchmaking service by submitting an instance file (in RDF) and a query asking for resources that satisfy the request specification. The query is then processed by the TRIPLE/XSB deductive database system using matchmaking rules, in combination with background knowledge and ontologies, to find the best match for the request.

Ongoing and future work is to implement the matchmaker as a Grid service. Once completed, a user will be able to submit a request specification (using our request ontology) from any client machine through a web service mechanism. The matchmaker service will notify the results to the user through similar mechanism.

### 4.6 Matching Example

We show a matching example by our matchmaker is this section. This matching example cannot be done easily by syntax-based matchmakers. Figure 7 shows examples of two instances of resources: a 64-CPU SunOS shared memory machine and a Linux cluster with 640 CPUs available. Due to space limitation, we are only showing a subset of resource properties that are relevant to the example. In this example, both resources belong to USC and only allow users who belong to "`rcf@usc.edu`" group to access the resources.

Figure 8 shows an example of a job request. The job request specifies that it wants one `ComputerSystem` resource for an MPI application. The resource requirements are specified with the prefix JobRequest.RequestResource. Since our background knowledge indicates that an MPI application can run on a tightly-coupled machine and both Linux cluster and shared memory are considered tightly-coupled machines, they both are candidate resources for an MPI application. Assuming that User1 has an account that belong to the "`rcf@usc.edu`" group, User1 is authorized to access both machines. The matchmaker then checks the capabilities of both resources against the resource requirements. Again, since our background knowledge specifies that both "`Linux`" and "`SunOS`" are types of "`Unix`", both resources pass the `OSType` requirement criteria. Because both resources are compatible with the resource requirements, the "`RankBy`" is used to select the best match. Finally, since the `MinClockSpeed` of "`Almaak.usc.edu`" is higher than that of "`Hpc.usc.edu`", the matchmaker returns "`Almaak.usc.edu`" as a match.

`RequestResource` is a relationship between `JobRequest` and `ResourceDescription` classes. We use this relationship to describe as many `ResourceDescription` instances as we want. For example, using the above request example, we can extend the above `JobRequest` to accommodate two resources by updating `NumberOfResources` to 2 and specifying another set of `JobRequest.RequestResource.*`.

| Property Names | Property Values |
|---|---|
| UnitaryComputer.Name | “Almaak.usc.edu” |
| UnitaryComputer.AuthorizedGroup | “rcf@usc.edu” |
| UnitaryComputer.NumberOfAvailableCPUs | 64 |
| UnitaryComputer.ComputerSystemProcessor.MinClockSpeed | 900 |
| UnitaryComputer.HostedFileSystem.AvailableSpace | 500 |
| UnitaryComputer.RunningOS.OSType | “SunOS” |
| UnitaryComputer.RunningOS.Version | “5.8” |
| UnitaryComputer.RunningOS.FreeVirtualMemory | 4000 |
| UnitaryComputer.RunningOS.FreePhysicalMemory | 4000 |
| UnitaryComputer.RunningOS.MaxProcessCPUs | 64 |
| UnitaryComputer.RunningOS.MaxProcessMemorySize | 2000 |

(a) A SunOS shared memory machine with 64 CPUs

| Property Names | Property Values |
|---|---|
| LinuxCluster.Name | “Hpc.usc.edu” |
| LinuxCluster.AuthorizedGroup | “rcf@usc.edu” |
| LinuxCluster.NumberOfAvailableCPUs | 640 |
| LinuxCluster.MinClockSpeed | 733 |
| LinuxCluster.HostedFileSystem.AvailableSpace | 5000 |
| LinuxCluster.RunningOS.OSType | “Linux” |
| LinuxCluster.RunningOS.Version | “7.2” |
| LinuxCluster.RunningOS.FreeVirtualMemory | 2000 |
| LinuxCluster.RunningOS.FreePhysicalMemory | 1000 |
| LinuxCluster.RunningOS.MaxProcessCPUs | 320 |
| LinuxCluster.RunningOS.MaxProcessMemorySize | 1500 |

(b) A linux cluster with 640 CPUs

**Fig. 7.** Available Resources

| Property Names | Property Values |
|---|---|
| JobRequest.Name | “Request1” |
| JobRequest.Owner | “User1” |
| JobRequest.JobType | “MPI” |
| JobRequest.NumberOfResources | 1 |
| JobRequest.RequestResource.ResourceType | “ComputerSystem” |
| JobRequest.RequestResource.RankBy | “CPUClockSpeed” |
| JobRequest.RequestResource.RequiredOS.OSType | “Unix” |
| JobRequest.RequestResource.RequiredCPU.MinNumberCPUs | 32 |
| JobRequest.RequestResource.RequiredMemory.MinPhysicalMemory | 1000 |
| JobRequest.RequestResource.RequiredMemory.MinVirtualMemory | 1000 |
| JobRequest.RequestResource.RequiredFS.MinDiskSpace | 200 |

**Fig. 8.** Job Request

## 5 Conclusion and Future Work

We have presented a prototype of an ontology-based resource matchmaker that exploits existing semantic web technologies. We have shown that Semantic Web technologies like RDF and RDF Schema can be used to build such a rule-based matchmaker. Since our matchmaker is built based on existing components, the effort to create and maintain the matchmaker is drastically reduced. So far, our experience with the ontology-

based matchmaker is promising. We plan to expand the three ontologies and enhance the matchmaking capability.

For example, the resource model will be extended to cover other kinds of physical resources (*e.g.,* database and storage systems, network connections) and abstract services (*e.g.,* a specialized finite difference solver). We envision a consensus process possibly organized as a GGF working group which standardizes the vocabulary for expressing resource description in the Grid. We also plan to extend the resource request model and our matchmaker capability to support set-matching.

As mentioned in Section 4, one of the desired features of the matchmaker is to allow users to submit a request in terms of high-level application characteristics, which will in turn be mapped to specific resource requirement configurations by the matchmaker. We will start our investigation with earthquake applications (*e.g.,* finite different inelastic wave propagation simulations) in the SCEC/ITR project [15].

To show that our resource matchmaker can be efficiently used in the Grid environment, a practical performance evaluation of our matchmaker needs to be conducted. We plan to investigate the scalability and performance of our ontology matchmaker in terms of number of rules and number of resources in the ontologies. An evaluation comparison with the existing resource matchmaker in the Grid such as Condor will also be included.

## References


1. R. J. Bayardo, Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 26,2, pages 195–206, New York, 13–15 1997. ACM Press.
2. Tim Berners-Lee. *Weaving the Web*. Texere Publishing, NA, 2000.
3. Dan Brickley and R. V. Guha. Resource description framework (rdf) schema specification 1.0.
4. Common information model (cim) standards. http://www.dmtf.org/standards/standard_cim.php.
5. The condor project. http://www.cs.wisc.edu/condor.
6. 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.
7. K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Proceedings of 8th Workshop on Job Scheduling Strategies for Parallel Processing*, Edinburgh, Scotland, July 2002. Lecture Notes in Computer Science, 2537:153-183, 2002.
8. H. Eriksson, R. W. Fergerson, Y. Shahar, and M. A. Musen. Automatic generation of ontology editors. In *Twelfth Banff Knowledge Acquisition for Knowledge-based systems Workshop*, Banff, Alberta, Canada, 1999.
9. Eurogrid: Application testbed for european grid computing. http://www.eurogrid.org.
10. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002. Extended version of Grid Services for Distributed System Integration.
11. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling sdalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.

12. Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for A New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, 1999.
13. Griphyn—grid physics network. http://www.griphyn.org/index.php.
14. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
15. T. H. Jordan and C. Kesselman *et al.* The scec community modeling environment—an information infrastructure for system-level earthquake research. http://www.scec.org/cme.
16. Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, July 1995.
17. O. Lassila and R. R. Swick. Resource description framework (rdf) model and syntax specification. In *W3C Recommendation, World Wide Web Consortium*. February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.
18. Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, 2003.
19. C. Liu and I. Foster. A constraint language approach to grid resource selection. Unpublished menuscripts.
20. C. Liu, L. Yang, I Foster, and D. Angulo. Design and evaluation of a resource selection framework. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, Edinburgh, Scottland, 2002.
21. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 2 edition, 1987.
22. Kim Marriott and J. S. Peter. *Programming with Constraints: An Introduction*. The MIT Press, Cambridge, Massachusetts, 1998.
23. The portable batch system. http://pbs.mrj.com.
24. R. Raman, M. Linvy, and M. solomon. Resource management through multilateral matchmaking. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 290–291, Pittsburgh, Pennsylvania, August 2000.
25. R. Raman, M. Linvy, and M. solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *Proceedings of the twelfth IEEE Symposium on High Performance Distributed Computing (HPDC12)*, pages 80–89, Seattle, Washington, June 2003.
26. R. Raman, M. Livny, and M. Solomon. Matchmaking distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, July 1998.
27. Michael Sintek and Stefan Decker. Triple - a query, inference, and transformation language for the semantic web. In Ian Horrocks and James Hendler, editors, *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, pages 364–378. Springer-Verlag, 2002.
28. Michael Sintek and Stefan Decker. Triple - an rdf query, inference, and transformation language. In Ian Horrocks and James Hendler, editors, *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2002.
29. K. Sycara, S. Wido, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace, 2002.
30. The teragrid project. http://www.teragrid.org.
31. Universal description, discovery and integration of web services. http://www.uddi.org.
32. The xsb research group. http://xsb.sourceforge.net/.