Mathematical Service Discovery

Julian Padget[1] and Omer Rana[2]

[1] Department of Computer Science
University of Bath, Bath, UK jap@cs.bath.ac.uk
[2] Department of Computer Science
Cardiff University, Cardiff, UK o.f.rana@cs.cardiff.ac.uk

Abstract. Matchmaking has been a subject of research for many years, but the increasing uptake of service-oriented computing, of which the Grid can be seen as a particular instance, has made effective and flexible matchmaking a necessity. Early approaches to matchmaking and current schemes in the Grid community, like ClassAds, take a syntactic point of view, essentially matching up literals or satisfying some simple constraints for the purpose of identifying computational resources. The increasing availability of web services shifts attention to the function of the service, but WSDL can only publish (limited) information about the signature of the operation which tells the client little about what the service actually does. The focus in the MONET (www.monet.nag.co.uk) and GENSS (genss.cs.bath.ac.uk) projects has been on describing the semantics of mathematical services and developing the means to search for suitable services given a problem description. In this paper we discuss (i) the schema extending WSDL that we call Mathematical Service Description Language (MSDL), (ii) a number of ontologies for describing various properties of mathematical services, (iii) an approach to describing pre- and post-conditions in OpenMath (www.openmath.org) and (iv) an extensible, generic matchmaking framework along with a suite of match plug-ins that are themselves web services.

1 Relevance to Computational Science

A long term vision for computational science is the realization of a desktop environment for scientific research, where the scientist is as easily able to find data sets, the algorithms to manipulate them and the means to display them — in silico experiments — as they currently do with physical materials in the laboratory — in vivo experiments.

The ability to solve large computational science by the coordinated use of distributed resources has been advocated by a number of researchers. Work in this area has primarily focused on the development of "Problem Solving Environments" (PSEs). A PSE is a complete, integrated computing environment for composing, compiling, and running applications in a specific area [10]. In many ways, a PSE is seen as a mechanism to integrate different software construction and management tools, and application specific libraries, within a particular problem domain. One can therefore have a PSE for financial markets [4], for gas turbine engines [8], etc. Focus on implementing PSEs is based on the observation that previously scientists using computational

methods wrote and managed all of their own computer programs – however now computational scientists must use libraries and packages from a variety of sources, and those packages might be written in many different computer languages. Engineers and scientists now have a wide choice of computational modules and systems available, enough so that navigating this large design space has become its own challenge. A survey of 28 different PSEs by Fox, Gannon and Thomas (as part of the Grid Computing Environments WG) can be found in [9], and practical considerations in implementing PSEs can be found in Li et al. [14]. Both of these indicate that such environments generally provide "some back-end computational resources, and convenient access to their capabilities". Furthermore, work-flow features significantly in both of these descriptions. In many cases, access to data resources is also provided in a similar way to computational ones.

In [7] the authors identify how the original multiphysics problem – in this case a gas turbine engine simulation – may be considered as a set of smaller simulation problems on simple geometries that need to be solved simultaneously while satisfying a set of interface conditions. These simpler problems may be chosen to reflect the underlying structure/geometry/physics of the system to be simulated, or artificially created by scientific computing techniques such as domain decomposition. For physical systems and devices, these sub-problems are usually modelled by partial differential equations. The next step is to create a network of collaborating solver agents in which each such agent deals with one of the sub-problems defined earlier. This work therefore also can be considered as an aspect of PSEs, where a larger problem is decomposed and handed off to independent agents which can then aggregate their results.

Looking at these two aspects of PSEs together, we can see the need for a "matchmaking" process, which is able to: (i) decompose a larger problem into smaller components, based on very specific domain dependent information; (ii) map each of these smaller problem components to particular solvers that can be found in a registry. The granularity of the decomposition process and the capability inherent within each problem solver provides two constraints on the usefulness of this approach.

2 Technical Background

The work reported here stems from a series of projects, each focusing on different contributions to the goal of building a computational environment for scientific research:

- **OpenMath** provides an extensible framework for the authoring of mathematical ontologies
- **MONET** demonstrates feasibility of semantic processing from user query to service invocation [5]
- **GENSS** generalizes the matchmaking/brokerage component [16] and extends matching to conditions and effects [18]
- **KNOOGLE** implements an open architecture for matchmaking and brokerage [12]

We will now discuss each of these and their contribution in some more detail.

2.1 OpenMath

The objective of OpenMath is to provide both a framework for authoring mathematical ontologies and to provide some fundamental ontologies. We write of ontologies in the plural because OpenMath supports a structured collection of ontological information built from components called content dictionaries — referred to as CDs. OpenMath does not attempt to be a complete ontology of mathematics, but rather provides a comprehensive core, including the basic mathematical structures (group, ring, field etc.), key constants and common operations/functions (trigonometric, hyperbolic, integration, differentiation etc.). New specialized mathematical ontologies can be added as the need arises and thus contribute to the broader corpus of mathematical ontologies, including private CDs (subject to a validation process defined and implemented by the OpenMath Foundation).

Many browsers support the presentation of mathematical markup through a plug-in for the W3C recommendation MathML, which takes two aspects depending usage: (i) MathML-P is for *presentation* and (ii) MathML-C is for *content*. The purpose of the latter is similar to OpenMath, namely to provide a neutral format for the communication of mathematical information between software components. However, MathML-C is a fixed ontology that only handles a subset of mathematics. OpenMath complements MathML-C by being extensible and by being the defined extension language for MathML-C.

OpenMath markup to some extent still reflects the period of its inception, when XML was a developing language. Consequently, there is little use of the more recent more sophisticated features of XML. OpenMath is intended to be as lightweight as possible so there are relatively few markup tags (see The OpenMath Standard v2.0 [26] for more detail: what follows is an abstracted summary from the standard for the sake of making this article self-contained):

- OMS: denotes a symbol, where the string name of the symbol and the CD in which it is defined are attributes of the tag: `<OMS cd="arith" name="minus">`
- OMV: denotes variable, where the string name of the variable is given in the `name` attribute of the tag
- OMI: denotes integer, for example `<OMI>2</OMI>`
- OMB: denotes a byte array and wraps a base64-encoded XML string
- OMSTR: denotes a string value
- OMF: denotes IEEE floating point number and the attributes may indicate size and even a value represented as a hexadecimal string
- OMA: denotes application, where its first child is the operator and the remaining children are the operands.
- OMBIND: denotes the binding constructor which has three children, a *binder*, a variable (specified by OMBVAR) and a body
- OMBVAR: variables used in binding constructor as above
- OME: error constructor, which has an arbitrary number of children, the first denoting the error and the remainder being OpenMath object relating to the error.

```
<om:OMOBJ><om:OMA>
  <om:OMS cd="arith1" name="minus"/>
  <om:OMA>
    <om:OMS cd="arith1" name="power"/>
    <om:OMV name="x"/>
    <om:OMI>2</om:OMI>
  </om:OMA>
  <om:OMA>
    <om:OMS cd="arith1" name="power"/>
    <om:OMV name="y"/>
    <om:OMI>2</om:OMI>
  </om:OMA></om:OMA>
</om:OMOBJ>
```

Fig. 1. OpenMath representation of $x^2 - y^2$

- OMATTR: the attribution constructor is used to wrap a sequence of attribute pairs which is how additional textual and semantic annotations of objects are constructed.
- OMATP: the attribute pair constructor is used in conjunction with OMATTR above.
- OMFOREIGN: the foreign constructor, which allows the inclusion of arbitrarily encoded data, such as:

  ```
  <OMFOREIGN encoding="text/x-latex">\sin(x)</OMFOREIGN>
  ```

By way of illustration the OpenMath representation of x^2-y^2 is shown in Figure 1. Detailed information about OpenMath, including the OpenMath 2.0 standard (June 2004) are available from the OpenMath website at `www.openmath.org`.

By providing the means to structure, author and publish markup for any aspect of mathematics, OpenMath establishes a way to describe both the functionality of any piece of mathematical software and the data that it inputs and outputs in an application and network neutral format. Thus it contributes to the goal of enabling the inter-operation of mathematical software components wherever they may be deployed.

2.2 MONET

MONET — Mathematics On the NET — had the objective of demonstrating the potential of semantic web techniques in service discovery, and given the partners' previous work with OpenMath, specifically mathematical service discovery, composition and invocation. Project details are available via `http://monet.nag.co.uk`, but we now summarize the main contributions of the project.

An important problem to solve at the outset was how to publish information about mathematical web services in a way that could be used to help achieve the project goals. Consequently, an embedding of WSDL [28], called Mathematical Service De-

scription Language (MSDL)[1] was defined to incorporate the necessary semantic information to support the discovery process. As with some OpenMath design decisions, MSDL is a product of its time: OWL-S was not completed and the tools for DAML-S were relatively experimental, so while the structure of MSDL documents mimicked these more general approaches to service description, they were under project control and allowed demonstration of principle. A complementary Mathematical Query Description Language (MQDL) was defined for posing queries. The structure of the two schemas is necessarily very similar, comprising the following elements:

Classification: The specification of what the service does (a more detailed description appears in [5] and [16]):
- Reference to a problem description library — terms supplied by the Mathematical Problem Description Language (MPDL).
- Reference to taxonomies, e.g. to GAMS (Guide to Available Mathematical Software) [3].
- Supported Semantics, such as which OpenMath CDs the application can process.
- Supported Directives, such as *solve*, *prove* and *decide*.

Implementation Details: information about the specific service
- A reference to an Algorithm Description, using entities from the OpenMath CD containing symbols for describing algorithmic complexity.
- Software Details – information about the hosting software package.
- Hardware Details – self explanatory.
- Algorithmic Properties, including attributes such as accuracy and resource usage.
- Descriptions of actions needed to solve a problem.

It should be noted that not all of the Classification or Implementation details are mandatory.

Service Interface Description: Typically a WSDL document.

Service Binding Description: Map from abstract problem components and actions to elements of the service interface.

Broker Interface: The API exposed to the broker. Typically, this is a service URI and an interface description.

The demonstrator architecture is shown in Figure 2 and allows us to trace out two scenarios:

- Service registration: the provider registering a service submits a MSDL document, elements of which refer to the MONET and the OpenMath ontologies. The registry manager then processes elements of the mathematical service description into OWL because that is the representation over which the Instance Store operates. That description is then entered into the repository and the process is complete.
- Service discovery and invocation: the client seeking a service submits a Mathematical Query Description Language (MQDL) document to the Plan Manager.

[1] The XSD schema for MSDL is available from `http://monet.nag.co.uk/cocoon/monet/publicdocs/index.html`

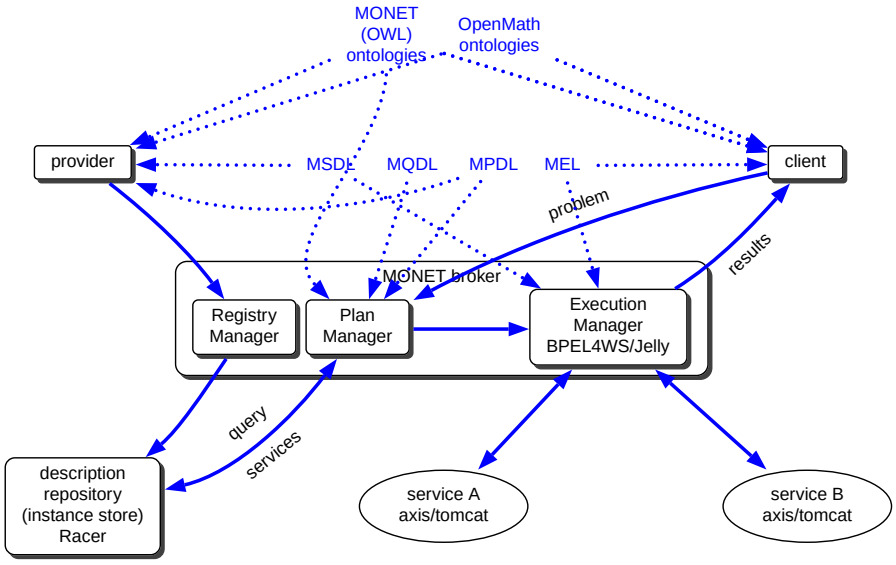

Fig. 2. The MONET Architecture

The mathematical elements of the problem description are then translated into OWL and passed to the Instance Store to find any candidate services according to a description logic querying process. The plan manager selects one of the candidate services and then invokes the execution manager to handle the actual calling of the web service. The results are then returned as a Mathematical Explanation Language (MEL) document to the client.

From a practical point of view the MONET architecture demonstrated the feasibility of semantic match making, but the demonstrator was very limited in that there is only one repository, one matching technique and one selection policy. Furthermore the matching technique was essentially using signature information and computing the equivalent of a multi-method look-up. Nevertheless, by demonstrating end-to-end support for computational problems using ontological information and web services it had established the viability of the approach and raised awareness of the direction future semantic grids research might take.

2.3 GENSS

The purpose of the GENSS (Grid-enabled Numerical and Symbolic Services) was to build on the outputs of MONET and, in particular to tackle the more complex problems

inherent in reasoning about the conditions and effects of services, while building links with the UK's e-Science research program, arguing that semantic discovery of (mathematical) services was an important enabling activity for e-Science. The main outputs of the project are more sophisticated approaches to mathematical service matching, including mathematical analysis of the conditions and effects, the use of multiple matching techniques and access to multiple registries. A detailed report on the matching techniques appears in [16] while the revised architecture is discussed in [12]. We now summarize the main contributions of the project.

GENSS Matchmaking Strategy The information source for the matchmaking process remains as for MONET: the MQDL describing the query and the MSDL document describing the service, but now attention was focused on working with the information about the conditions and effects in each case. One of several difficulties is that the expressions marked up with OpenMath in the condition and effect fields of the problem and service description may be equivalent semantically, but be written very differently. To begin to tackle this problem the expressions are normalized — not in the sense that there is any absolute normal form for mathematics, just the right one for the current purpose. Thus a fairly standard set of transformation is carried out dealing with:

- Logical equivalences — using standard rewrites
- Associative operators — are flattened, so for example the OpenMath equivalent of `(+ a (+ b c))` becomes `(+ a b c)`.
- Context dependent equivalences — for example $i+1>0 \Rightarrow i >= 1$ if $i \in \mathbb{Z}$, but not if $i \in \mathbb{Q}$.
- Alpha conversions — consistent naming of variables in problem and service, so that name comparison is meaningful
- Commutative operators — reorder arguments to bring constants towards the operator (and subsequently evaluate constant combinations) and so that the left hand side is less than the right hand side.
- Conversion to disjunctive normal form to capture, if present, the alternatives between pre- and post-conditions.

As a result, the conditions and effects take on the form $Q(L(R))$ where:

- Q is a quantifier block e.g. $\forall x \exists y$ s.t. $\cdots$
- L is a block of logical connectives e.g. $\wedge, \vee, \Rightarrow, \cdots$
- R is a block of relations. e.g. $=, \leq, \geq, \neq, \cdots$

With a summary of the normalization process in place, the two scenarios of registration and discovery become relatively straightforward: in the first case, the service description is normalized and stored in the registry; in the second the query is normalized and the registry is traversed calculating a similarity value between the query and each service. This latter results in a list of URIs ordered by similarity value.

Matching techniques A major development in GENSS was the idea of a matchmaker shell within which several match modes could be applied to the service and aggregate match scores computed. Thus, several matchers were deployed for use in the GENSS matchmaker:

- Structural: used to determine whether task and capability match exactly; cheap if not often very successful
- Syntax+Ontology: used to compare elements and attributes in task and capability using the taxonomic structure of types to test for inclusion relations
- Ontological reasoning: as had previously been demonstrated in MONET (described in section 2.2)
- Function: use conditions and effects expression to establish whether: $T_{cond} \Rightarrow C_{cond} \wedge C_{eff} \Rightarrow T_{eff}$; in other words do the conditions of the capability subsume those of the task and do the effects of the task subsume those of the condition. In fact, the tests applied aim to establish the *equivalence* of the given expressions:
  - *Algebraic equivalence*: where we wish to show that $Q - S = 0$ algebraically by translating the expression into the input syntax of a computer algebra system and calculating the difference. In general undecidable, but the approach outlined may be useful in practice. For example: $x^2 - y^2$ and $(x + y)(x - y)$. The idea stems from the work of Richardson [21] on the identification of zero.
  - *Value substitution*: where we wish to show that $Q - S = 0$ by substituting random values for the variables in the relation sub-expressions in the $Q(L(R))$ structure outlined above. This must be done with care: variables must be renamed consistently and the same random value substituted for a given variable in each expression. If the result is zero, it is only *evidence* not *proof* of equivalence. This relies on later work also by Richardson [22] on the so-called Uniformity Conjecture

The computation of the similarity score is quite detailed and not easily summarized, so the interested reader is referred to [16].

GENSS Architecture and Critique The GENSS architecture is shown in Figure 2.3 in which we can identify the various differences with MONET. The most obvious is that not only are there multiple matcher mechanisms, but those matchers are deployed as web services that are accessed from the matchmaker. The second key difference is the adoption of a standard registry component, namely a UDDI registry. However, there are still several aspects of this design that can be criticized: (i) the matcher work-flow is fixed (ii) UDDI registry searching is based on textual information (iii) the selection policy is fixed as the service with the highest similarity score

Work-flow enactment In a further development from MONET where we had posited a stand-alone broker, in GENSS we also demonstrated how the matchmaker shell could be turned into a web service and then built into a work-flow. The enactment system used was Triana[2]. Like other such systems, it works by scouring specified resources, such as UDDI registries for services and then displays them in a selection palette on the side. In Figure 4, the broker has simply been interposed between a widget to read input from the user and another for the display of results, while in Figure 5 the output from running the query are displayed. Although this only demonstrates the functionality, it also suggests the capability, given sufficient suitable services in available registries, of

[2] Details of Triana can be found via `www.trianacode.org`

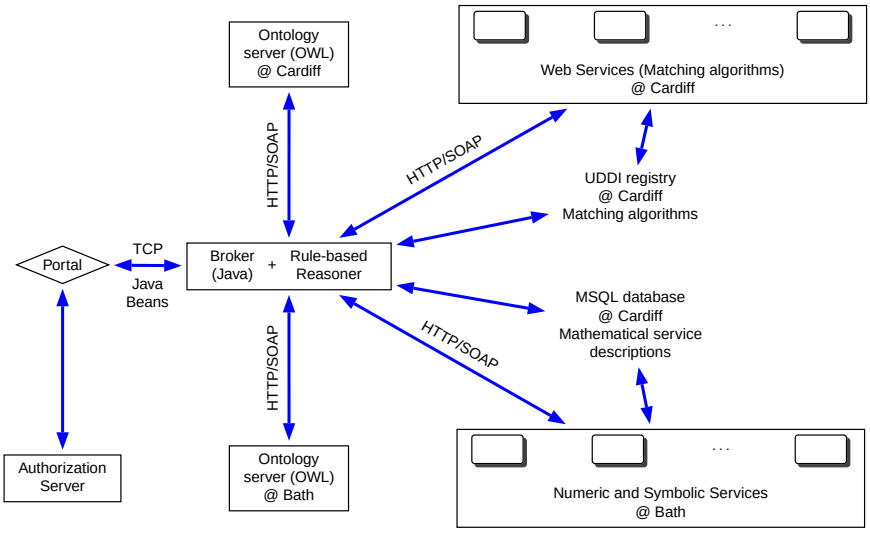

Fig. 3. The GENSS Architecture

constructing work-flows including proxy services. By "proxy" service, we mean that some work-flow elements are instances of the broker that when invoked can discover and call the appropriate service to meet the requirements, which may themselves have only been established dynamically earlier in the work-flow.

Generation of Semantic Descriptions The final contribution from GENSS has been an initial investigation of how mathematical information may be used to help generate service descriptions automatically. It is widely appreciated that authoring WSDL is a tedious and error-prone process, so that several Java IDEs will now generate it automatically for the user. However, we need to generate MSDL, including pre- and post-condition information.

We had been working with the Aldor language[3] and its algebra package as a means to remove the GENSS broker's dependency on Maple and thus on licensed software. The Aldor type system derives from that of Axiom/Scratchpad and provides a two-level categorical-style polymorphic dependent-type structure that has been established is adequate, while still remaining decidable for checking, to capture correctly the many mathematical relations required in building a strongly-typed computer algebra system. The consequence of this expressive power is that because the type system actually captures the necessary mathematical knowledge about the function it implements, it can be used for:

– Automatic wrapper generation

[3] The detailed history of Aldor is quite complicated, but it is probably sufficient to say that it inherits from the computer algebra system Axiom (market by Numerical Algorithms Group for some years) and Scratchpad (developed by IBM over many years) and is BSD-license software

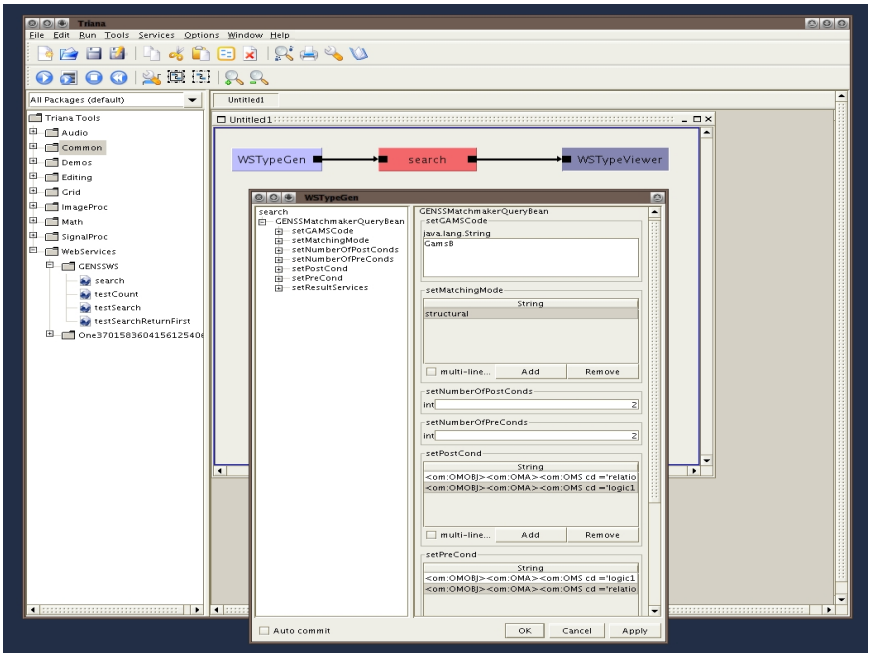

Fig. 4. A Triana work-flow with matchmaker (Screenshot provided by Tom Goodale)

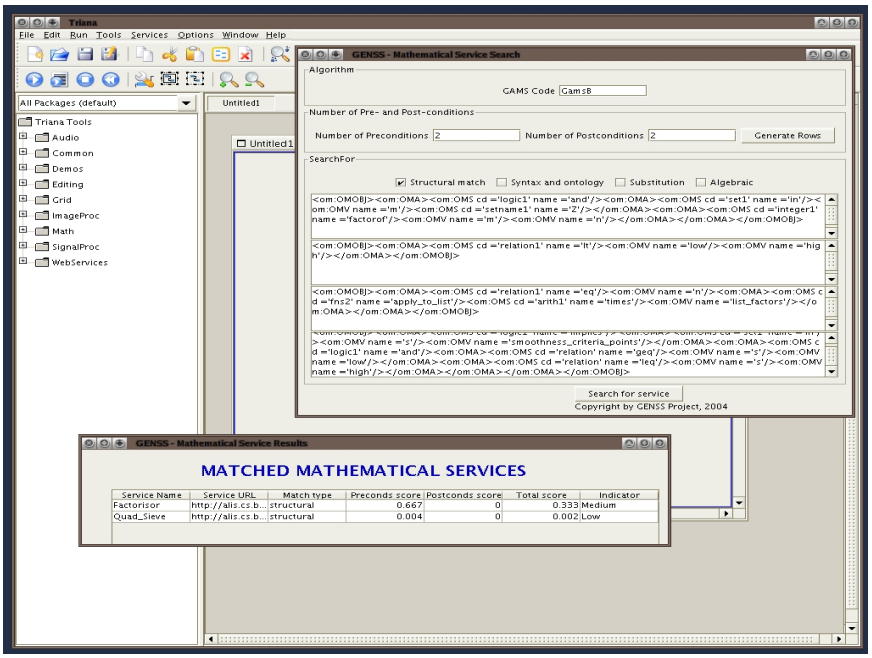

Fig. 5. Results from the matchmaker in Triana (Screenshot provided by Tom Goodale)

– Automatic generation of OpenMath for service description signatures and for (part of the) conditions and effects

However, it should be emphasized that the OpenMath generation depends on the availability of the appropriate CDs in OpenMath to reflect the corresponding types in Aldor. There are several other delicate technical issues that are covered in detail in [18].

GENSS Outcomes The contributions of the project are outlined above, but to put them in the context of this article, the project demonstrated that it was possible to go further on the problem of service discovery to examine conditions and effects and that it was practical to deploy and invoke multiple matchers using the basic technology of the semantic web — web services — and even to integrate the broker with work-flow enactment, thus making the broker work with the primary components for computational science.

3 A generic matchmaker/broker

Reflection on the design and limitation of the GENSS architecture have fed into a further project called KNOOGLE (pronounced noo-gl), in which the previous system is being re-factored to produce robust generic tools, and demonstrated in the context of other current UK e-Science projects.

From a client point of view, the brokerage function can be parameterized by three requirements:

– Where to find descriptions of entities to match against
– How to match the query against a description
– How to choose between the matched descriptions

Some clients may like to have each of these fixed, whereas others might like the opportunity to control some or even all of these at the point of calling the broker. Thus there is a complete spectrum ranging from no fixed actions to all three being fixed. For each case, we specify what information the client must provide:

– A set of registries identifies the places to look for candidate services
– A set of matchers, deployed as web services, identifies how to calculate a similarity score between the query and a service (note: each matcher web service must take a query description and a service description as arguments and deliver a score in the range $[0, 1]$)
– A selection policy, defined as a query over the match results, identifies one or no service to invoke. More details about specifying selection policy appear later.

These three issues lead to a refinement of the GENSS architecture of Figure 2.3 resulting in the KNOOGLE architecture of Figure 6, where:

– The registries have been replaced by the UDDI-compliant Grimoires registry, which supports semantic querying for services and the annotation of services with various forms of metadata (string, URL and RDF). The broker now accepts a list of such registries to search for candidate services

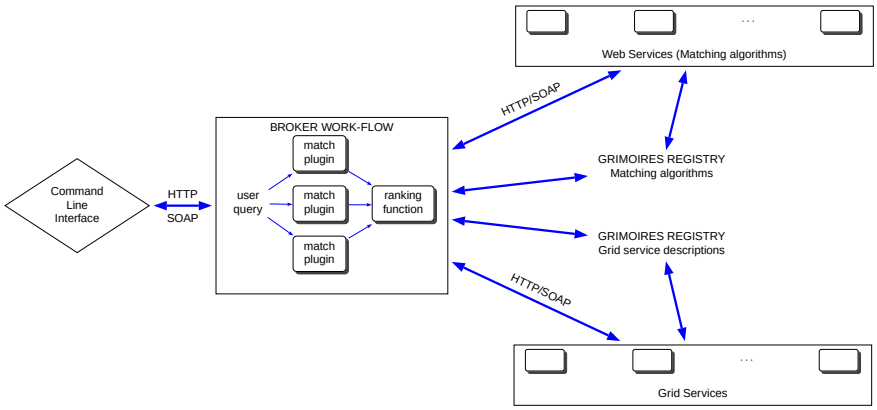

Fig. 6. The KNOOGLE Architecture

– The matchers are specified as a list of URLs identifying WSDL service descriptions. These matchers are treated as alternatives in that each matcher is invoked with the query and the service description, resulting in a match-score. This score is then asserted into a RDF store associating matcher, service and score for subsequent assessment by the selection policy.
– The selection policy is specified as a query in the RDQL language over the set of RDF triples that resulted from the matching process. This query should result in the identification of either no service, in the case that none satisfies the selection policy, or one service that does.

Current Developments The KNOOGLE project [20] is currently running and has the target of delivering tools for end-user construction and deployment of brokers by the middle of 2007. The broker functionality is planned for demonstration in the context of two related current projects (also in the UK): (i) GridSAM [6] which provides a client interface for the submission and monitoring of jobs using the JSDL [1] framework and (ii) Taverna, which is a work-flow enactment system that has seen much use in the bio-informatics domain. We now describe these two demonstrators in more detail.

GridSAM: The aim of the GridSAM project is to provide an executable submission and monitoring service. Each GridSAM instance has a set of processors on which it may execute programs. The clients of the GridSAM instances must provide their input data and the executable they wish to run, they must also provide a JSDL document which gives constraints on where and how their program is to be run. The GridSAM instance creates a DRMConnector, which translates the JSDL to processor-specific JCL and executes the job on the distributed resources. The GridSAM instance also provides tools to monitor the job's progress.

One problem with this architecture is the requirement that the client is in possession of the executable of the code they wish to execute. The client could be relieved of this task by keeping the code in a repository, accessible to a broker. Our first use case addresses this issue by placing the broker between the client and the GridSAM

instance, so the client sends a description of the problem to the broker, which retrieves the executable from a repository, sends this to the GridSAM instance, then receives the results back from the GridSAM instance and forwards these back to the client.

A second problem is that the client may not be aware of the resources available, but would like to search for suitable resources based on some description of needs. This too can be addressed by brokerage. Thus in the second use case, the client provides the executable, which the GridSAM instance sends to a DRMConnector augmented by a brokerage function. The DRM-broker determines what resources are available to be used, communicates this to the DRMConnector which then deploys the job, receives the results and passes these back to the client.

Taverna: The Taverna workbench [19] is a graphical environment with which users can construct, edit, browse and share work-flows. The application provides three views: one on available services found either locally or in specified registries, another depicting the work-flow diagram and a third called the model explorer. This last allows the user to specify input sources and output locations for the work-flow along with the actual services that will do the work.

There are numerous components associated with the Taverna system including the FreeFluo work-flow enactor, the KAVE metadata store and the FETA service discovery component. FETA uses the myGrid service ontology [29], which provides descriptions of bioinformatics tasks and data types, to express descriptions of the kind of bioinformatics services sought. However, functional descriptions of bioinformatics services are hard to formulate because the datatypes involved are usually not defined in any formal type language[15]. Consequently that service information is typically limited to the name of the service — which might in itself be descriptive, but such information is hard to recover — the names of the operations supported and the names of the input and output parameters. Thus, the lack of type information means there is even less than a type signature available, so queries are most likely to be stated in terms of the name of the service sought. Although the query technology is based on JENA and RDQL, myGrid users apparently do not normally construct their own RDQL queries: instead they are provided with some prepared queries from which to choose. Taverna then presents the results in a similar way to that in which other services which are listed and they can then be dragged into the work flow like any other service.

The objectives of integrating the KNOOGLE broker with Taverna are

- To demonstrate the integration of the broker with a work-flow enactment system. This has already been achieved with Triana, but Taverna is the approved work-flow environment for OMII projects.
- To enhance Taverna through the provision of a flexible external matchmaking facility to complement the built-in FETA system described above, as well as providing access to external registries and the adding options for (i) the creation of bespoke matcher/broker components (ii) access to a range of different matching technologies (iii) the means to embed proxy services in work-flows that can be resolved into actual services through the function of the broker, as described earlier in section 2.3.

It is perhaps paradoxical that the result of the evolution of the brokerage architecture described here is now little more than a shell, with almost no function in itself: an instance of the KNOOGLE broker has no registries, no matching function and no selection function; it can do nothing except wait to be supplied with these three pieces of information. The result of re-factoring has been to hollow out the original architecture and delegate the functions either to client-specified parameters or externally supplied web services. Yet the result is potentially more useful, more flexible and more customizable than the original. Perhaps evidence of the validity of the statement "less is more".

4 Related Work

A variety of matchmaking systems have been reported in the literature over the last couple of decades, although it seems that almost all are relatively special-purpose or domain-specific in some way or another and there is little to indicate success in re-application outside the domain of their original development. Nevertheless, there are some valuable ideas to be found as we attempt to show in our review below. From a computational science perspective, we observe that much of the prior art in matchmaking has focused on AI or free text and it is only in the last five years or so, with the advent of widely accepted ontological frameworks, that there has been rising interest in services and service matching. Parts of this survey have appeared in earlier publications [16, 12].

The SHADE (SHAred Dependency Engineering) matchmaker [13] operates over logic-based and structured text languages. The aim is to connect information sources dynamically. The matchmaking process is based on KQML (Knowledge Query and Manipulation Language) communication [25]. The content languages of SHADE are a subset of KIF (Knowledge Interchange Format) [11] as well as a structured logic representation called MAX (Meta-reasoning Architecture for "X"). Matchmaking is carried out solely by matching the content of advertisements and requests. There is no knowledge base and no inference performed, however rules may be added dynamically making MAX flexible and adaptable.

COINS (COmmon INterest Seeker) [13] is a matchmaker over free text descriptions.. The motivation behind COINS is given as a need for matchmaking over large volumes of unstructured text on the Web and the unsuitability of existing matchmaking technology for such an application domain. Initially the free text matchmaker was implemented as the central part of the COINS system but it turned out that it was also useful as a general purpose facility. As in SHADE the access language is KQML. The System for the Mechanical Analysis and Retrieval of Text (SMART) [23] information retrieval system is used to process the free text, producing a document vector using SMART's stemming and "noise" word removal, after which document vectors are compared using inverse document frequency. Such technology could usefully be redeployed now as a web service and straightforwardly incorporated into a broker using the architecture outlined above.

LARKS (Language for Advertisement and Request for Knowledge Sharing) [24] was developed to enable interoperability between heterogeneous software agents and had a strong influence on the DAML-S specification. The system uses ontologies defined by a concept language ITL (Information Terminology Language). The technique used to calculate the similarity of ontological concepts involves the construction of a weighted associative network, where the weights indicate the belief in relationships. While it is argued that the weights can be set automatically by default, it is apparent that the construction of realistically weighted relationships requires human input and thus impacts the general deployment of such technology.

InfoSleuth [17] is described as a system for discovery and retrieval of information in open and dynamically changing environments. The brokering function provides reasoning over the advertised syntax and the semantics. InfoSleuth aims to support cooperation among several software agents for information discovery, where agents have roles as core, resource or ontology agents. There is a distinguished broker agent encapsulating a matching function, which serves to bring agents requiring services together with those offering. The matching operations are a mixture of syntactic, structural and ontological proximity, inspiring the similar mechanisms developed in GENSS.

The GRAPPA [27] (Generic Request Architecture for Passive Provider Agents) system allows multiple types of matchmaking mechanisms to be employed within a system. It is based on receiving arbitrary matchmaking offers and requests, where each offer and request consist of multiple criteria. Matching is achieved by applying distance functions which compute the similarities between the individual dimensions of an offer and a request. Using specialized aggregation functions, the similarities are projected to a single value to constitute a match score. There is a clear link between the ideas in GRAPPA and our final KNOOGLE architecture.

MathBroker [2] is a project at RISC-Linz with some elements in common with those described here, including providing semantic descriptions of mathematical services. It too uses MSDL, however it seems that most of the matchmaking is achieved through traversing taxonomies, while actual understanding of the pre- and post-conditions is still regarded as an open problem.

In the main, matchmaking research projects have tried to deliver generic results, capable of being adapted subsequently for particular domains. However, the motivation for many such projects has primarily been e-commerce (as a means to match buyers with sellers, for instance), where it is hard to describe accurately the actual function of a service, compared to the case of mathematical functions. In other cases, the work has been driven by a specific language, notably KQML in some cases above, which although powerful, does not enjoy widespread appeal.

In contrast, we believe that the approach we have outlined here, has attempted to learn from this history, by putting as little as possible in the matchmaker/broker itself and building on the power of web services and work-flow enactment, technologies that at present appear to have good prospects for the medium-term, to provide a "late-binding" of whatever functionality is desired, while also offering a degree of future-proofing through the means to publish new matching techniques and rapidly deploy new brokers using them.

5 Conclusions

We have presented a short history of a selection of closely related projects that have fortuitously led one into another to deliver a series of outputs that could be very beneficial for the provision and up-take of computational science services. From OpenMath, we obtain a general framework for mathematical semantic annotation of services, (as well as a *lingua franca* for communication between mathematical services). From MONET we get the confirmation that ontological reasoning can help in service discovery. From GENSS, we see how mathematical reasoning over the ontological description of conditions and effects can make that discovery process more precise. And finally in KNOOGLE, with the benefit of hindsight, we see better how to engineer an architecture to deliver past and future matching technology using both mathematical and a wide range of other techniques.

Acknowledgements

The work reported here has been partially supported by the European Commission and the Engineering and Physical Sciences Research Council of the UK. Specifically: (i) OpenMath I and II were funded by the CEC (ESPRIT Project 42969 and project IST-2000-29719); partners: Numerical Algorithms Group Ltd (United Kingdom), University of Bath (United Kingdom), Stilo Technology Ltd (United Kingdom), INRIA Sophia-Antipolis (France), University of St Andrews (United Kingdom), Technical University of Eindhoven (Netherlands), Springer Verlag (Germany), The University of Nice (France), Konrad-Zuse Zentrum für Informationstechnik (Germany), Nibbles.it (Italy), Research Institute for Symbolic Computation (Austria), German Research Centre for Artificial Intelligence (Germany), University of Helsinki (Finland), International University of Bremen (Germany), University of Köln (Germany) (ii) MONET was funded by the CEC (project IST-2001-34145); partners: NAG Ltd., Stilo Ltd., University of Eindhoven, Université de Nice/INRIA Sophia Antipolis, University of Bath, University of Manchester, University of Western Ontario (iii) GENSS was funded by the EPSRC (UK) under the Semantic Grids call of the e-Science program (project GR/S44723/01); partners: University of Bath and Cardiff University (iv) KNOOGLE is funded by the Open Middleware Infrastructure Institute managed program, which in turn is funded by the EPSRC (UK); partners University of Bath and Cardiff University. Finally, particular thanks are due to Omer Rana for input on the computational science context and to Bill Naylor for commenting on drafts of this article.

References

1. Stephen McGough Ali Anjomshoaa, Darren Pulsipher. Job Submission Description Language WG (JSDL-WG), 2003. Available from `https://forge.gridforum.org/projects/jsdl-wg/`.
2. Rebhi Baraka, Olga Caprotti, and Wolfgang Schreiner. A Web Registry for Publishing and Discovering Mathematical Services. In *EEE*, pages 190–193. IEEE Computer Society, 2005.

3. R.F. Boisvert, S.E. Howe, and D.K. Kahaner. GAMS: A Framework for the Management of Scientific Software. *ACM Transactions on Mathematical Software*, 11(4):313–355, December 1985.
4. O. Bunin, Y. Guo, and J. Darlington. Design of problem-solving environment for contingent claim valuation. In *Proceedings of EuroPar 2001*, volume 2150 of *LNCS*. Springer Verlag, 2001.
5. Olga Caprotti, Michael Dewar, James Davenport, and Julian Padget. Mathematics on the (Semantic) Net. In Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *Proceedings of the European Symposium on the Semantic Web*, volume 3053 of *LNCS*, pages 213–224. Springer Verlag, 2004. ISBN 3-540-21999-4.
6. John Darlington. Gridsam grid job submission and monitoring web service. `http://www.omii.ac.uk/projects/`, 2004. Last visited September 2006. See also `http://www.lesc.ic.ac.uk/gridsam/`.
7. Tzvetan Drashansky, Elias N. Houstis, Naren Ramakrishnan, and John R. Rice. Networked agents for scientific computing. *Communications of the ACM*, 42(3):48–54, 1999.
8. S. Fleeter, E. Houstis, J. Rice, C. Zhou, and A. Catlin. A problem solving environment for simulating gas turbines. In *Proceedings of* 16^{th} *IMACS World Congress*, pages 104–105, 2000.
9. G. Fox, D. Gannon, and M. Thomas. A summary of grid computing environments. *Concurrency and Computation: Practice and Experience (Special Issue)*, 2003.
10. E. Gallopoulos, E. N. Houstis, and J. R. Rice. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 1(2), 1994.
11. M. Genesereth and R. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical report, Computer Science Department, Stanford University, 1992. Available from `http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps`.
12. Tom Goodale, Simone A. Ludwig, William Naylor, Julian Padget, and Omer F. Rana. Service-oriented matchmaking and brokerage. In Paul Watson, editor, *Proceedings of UK e-Science All Hands conference*. EPSRC, 2006.
13. D. Kuokka and L. Harada. Integrating information *via* matchmaking. *Intelligent Information Systems 6(2-3), pp. 261-279*, 1996.
14. M. Li, O. F. Rana, D. W. Walker, M. Shields, and Y. Huang. *Component-based Software Development*, chapter Component-based Problem Solving Environments for Computational Science. World Scientific Publishing, 2003.
15. Phillip Lord, Pinar Alper, Chris Wroe, and Carole Goble. Feta: A light-weight architecture for user oriented semantic service discovery. In A. Gómez-Pérez and J. Euzenat, editors, *European Semantic Web Conference*, pages 17–31. Springer-Verlag, 2005.
16. Simone Ludwig, Omer Rana, William Naylor, and Julian Padget. Matchmaking Framework for Mathematical Web Services. *Journal of Grid Computing*, 4(1):33–48, March 2006. Available via `http://dx.doi.org/10.1007/s10723-005-9019-z`. ISSN: 1570-7873 (Paper) 1572-9814 (Online).
17. W. Bohrer M. Nodine and A.H. Ngu. Semantic brokering over dynamic heterogenous data sources in InfoSleuth. In *Proceedings of the 15th International Conference on Data Engineering, pp. 358-365*, 1999.
18. William Naylor and Julian Padget. From untyped to polymorphically typed objects in mathematical web services. In William Farmer, editor, *Proceedings of MKM2006*. To appear in Springer LNCS, 2006.
19. Tom Oinn, Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord,

Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Chris Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1067–1100, 2006.
20. Julian Padget. Knoogle matchmaking and brokerage framework. `http://www.omii.ac.uk/projects/`, 2006. Last visited September 2006.
21. D. Richardson. Some Unsolvable Problems Involving Elementary Functions of a Real Variable. *Journal of Computational Logic*, 33:514–520, 1968.
22. Daniel Richardson. The uniformity conjecture. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *CCA*, volume 2064 of *Lecture Notes in Computer Science*, pages 253–272. Springer, 2000.
23. G. Salton. *Automatic Text Processing*. Addison-Wesley, 1989.
24. K. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Journal of Autonomous Agents and Multi Agent Systems*, 5(2):173–203, June 2002.
25. D. McKay T. Finin, R. Fritzson and R. McEntire. KQML as an agent communication language. In *Proceedings of 3rd International Conference on Information and Knowledge Management, pp. 456-463*, 1994.
26. The OpenMath Society. The OpenMath Standard, June 2004. Available from `http://www.openmath.org/standard/om20-2004-06-30/omstd20.pdf`.
27. D. Veit. *Matchmaking in Electronic Markets*, volume 2882 of *LNCS*. Springer, 2003. Hot Topics.
28. W3C. *Web Services Description Language (WSDL) Version 1.2 W3C Working Draft*. W3C, 2002-2003. Available from `http://www.w3.org/TR/wsdl12`.
29. Chris Wroe, Robert Stevens, Carole Goble, Angus Roberts, and Mark Greenwood. A suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *The International Journal of Cooperative Information Systems*, 12(2):597–624, 2003.