# Applying the Semantic Web to Manage Knowledge on the Grid

Feng Tao[*], Colin Puleston[^], Carole Goble[^], Nigel Shadbolt[*], Liming Chen[*], Graeme Pound[+], Fenglian Xu[+], Simon Cox[+]

[*] Department of Electronics and Computer Science, University of Southampton, Southampton, U.K.
{ft, nrs, lc}@ecs.soton.ac.uk
[2] Department of Computer Science, University of Manchester, Oxford Road, Manchester, U.K.
{carole, colin.puleston}@cs.man.ac.uk
[+] e-Science centre, Scholl of Engineering Science, University of Southampton, Southampton, U.K.
{gep, flx, s.j.cox}@soton.ac.uk

**Abstract:** Geodise [2] uses a toolbox of Grid enabled Matlab functions as building blocks on which higher-level problem solving workflows can be built. The aim is to help domain engineers utilize the Grid and engineering design search packages to yield optimized designs more efficiently. In order to capture the knowledge needed to describe the functions & workflows so that they may be best reused by other less experienced engineers we have developed a layered semantic infrastructure. A generic knowledge development and management environment (OntoView) that is used to develop an ontology encapsulating the semantics of the functions and workflows, and that underpins the domain specific components. These include: an annotation mechanism used to associate concepts with functions (Function Annotator); a semantic retrieval mechanism and GUI that allows engineers to locate suitable functions based on a list of ontology-driven searching criteria; and a GUI-based function advisor that uses the functions' semantic information in order to help function configuration and recommend semantically compatible candidates for function assembly and workflow composition (Domain Script Editor and Workflow Construction Advisor). This paper describes this infrastructure, which we plan to extend to include the semantic reuse of workflows as well as functions.

*Keywords: Ontology, Semantic Grid, Engineering, Knowledge Management*

## 1. Introduction

The Grid has provided an operational infrastructure that enables distributed scientific computing and resource sharing by those working in various domains. In Geodise, a toolbox of Grid enabled Matlab functions for *Engineering Design Search and Optimisation* (EDSO) have been developed [6] as building blocks on which higher level problem solving workflows can be built to help domain engineers utilize the Grid and engineering design search packages to yield optimized designs more efficiently.

To support seamless scientific collaboration, and intelligent process automation, it has become increasingly important that information and resources are consistently and semantically enriched using a shared vocabulary, and made machine understandable amongst distributed e-Science virtual communities. This is particularly required in the e-science vision of future large-scale science over the Internet where the sharing and coordinated use of diverse resources in dynamic, distributed virtual organization is commonplace [4].

In order to achieve this vision, Geodise has adopted Semantic Web based knowledge management. The aim is to help engineers working in the EDSO domain to utilize the Grid and engineering design search packages to yield optimized designs more efficiently. In order to capture the knowledge needed to describe the functions & workflows so that they may be best reused by less experienced engineers we had to develop a layered semantic infrastructure as illustrated in Figure 1

- **OntoView** [11]. A generic knowledge development and management environment that is used to develop the ontology and underpins the other more domain specific components.

- **Function Annotator**. A Matlab function annotation mechanism used to associate concepts with functions
- **Semantic Retrieval GUI**. A mechanism that allows engineers to locate suitable functions based on a list of ontology-driven searching criteria.
- **Workflow Construction Advisor**. A GUI-based component integrated with the Geodise workflow construction environment [9] and a domain script editor that use the functions' semantic information in order to recommend semantically compatible candidates for function assembly and workflow composition as well as to assist their configurations.
- **An Ontology** developed by an engineer expert in the Geodise EDSO functions in collaboration with knowledge workers. The annotation makes use of the ontologies to semantically describe available resources in the domain such as the Grid enabled functions, optimisation methods and their configurable parameters. The ontologies are represented using the RDF-based DAML+OIL language [12].

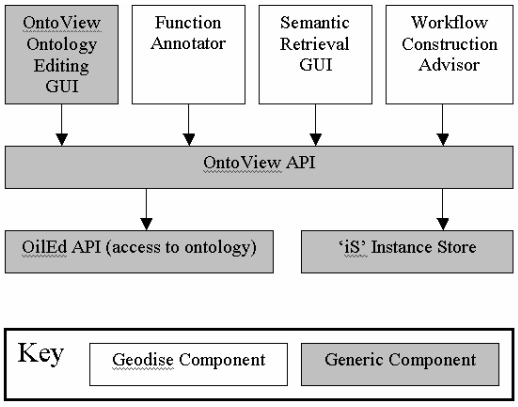



**Figure 1 Geodise Knowledge Architecture**

This paper describes this infrastructure, which are currently extending to include the semantic reuse of workflows as well as functions. The rest of the paper is organized as follows: Section 2 describes the knowledge *capture* by a specialist domain expert using the OntoView knowledge editor, working in collaboration with the Geodise knowledge team. Section 3 describes the *binding* of the knowledge to instances of Geodise functions, function signatures and other Geodise entities by an engineer using the annotator. Section 4 shows how the knowledge is *reused* for enhanced workflow construction. Section 5 gives related works and Section 6 draws some conclusions and points to future work.

## 2. Capturing Knowledge: Building Ontologies

In order to form a conceptualization of the domain in Engineering Design Search & Optimisation, we interviewed domain experts, domain system developers as well as studying the domain application manuals and domain source code for key EDSO concepts and their relationships.

The ontologies are represented in a machine-understandable language with formal semantics and reasoning capability, namely DAML+OIL. This language is based on Description Logics. Ontologies in this language can be elaborate and expressive, and the temptation is to over complicate the interface to them, rendering them daunting and incomprehensible to the user. Instead we adopted a simplified presentation interface that loses little of the expressivity of the language but hides it from the user. We call this OntoView – it provides a "domain expert-sympathetic" *view* over the ontology, configurable by the expert knowledge engineer in collaboration with the domain specialists.

The view consists of a set of relatively simple "view entities" that map to more complex constructs in the underlying ontology. As these entities are manipulated in the view, corresponding modifications will be produced in the ontology. The manner in which the entities in a particular ontology view map to the constructs in the underlying ontology, is determined by a "view configuration" (Figure 2), specifically created for that ontology, and stored in an XML-based format.

The knowledge used in Geodise falls into two categories:

- **Concepts**, represented by the ontologies, for example "Geodise_Function";
- **Instances** of the ontological concepts whose descriptions include concrete data values (e.g. integers and strings) as well as references to other instances and other concepts.

For example, the "OptionsMatlab" functon is represented by an instance of the "OptionsMatlabToolbox" concept (which in turn is a sub-concept of "GeodiseFunction", and so on up the function hierarchy). It also has other properties that can be represented by

references to other instances (e.g. the "invokesSoftwarePackage" property has value "Options") and some that can be represented as concrete data-values (e.g. the "author" property has value "Graeme Pound" which is a simple string value).

The instances, which naturally are greater in number than the concepts, are managed by OntoView using the Description Logic based *iS* instance store [7].

The instance descriptions maintained by OntoView can contain constructs that in DAML+OIL would be part of a class description, and others that would be part of an individual description. Therefore they are not simply equivalent to DAML+OIL individuals.

The 'iS' mechanism not only supports more expressive instance descriptions than DAML+OIL, it also supports reasoning over large numbers of individuals in realistic time, in contrast Racer [8] whose algorithms run in something more like exponential time, or FaCT [10] which can do no reasoning with individuals whatsoever.

Another factor to consider is that the OntoView instance descriptions may include instance-to-instance references, which are not permitted by 'iS'. To get round this restriction, OntoView stores each instance as a recursive description, with each instance-reference being replaced by a full instance description. The price paid for this extra expressiveness is that instance descriptions can become quite large, which has obvious effects on the efficiency of instance storage and query execution. We are currently investigating the various trade offs involved in using 'iS' in this fashion, within the specific context of Geodise.

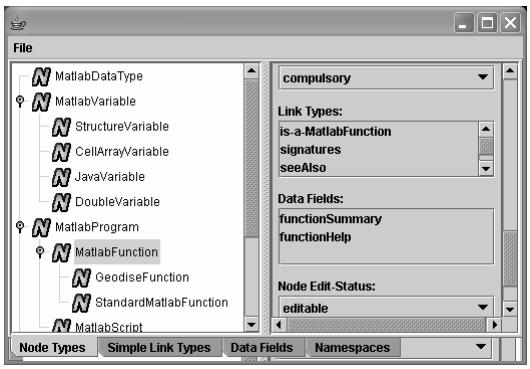



**Figure 2 View configuration GUI**

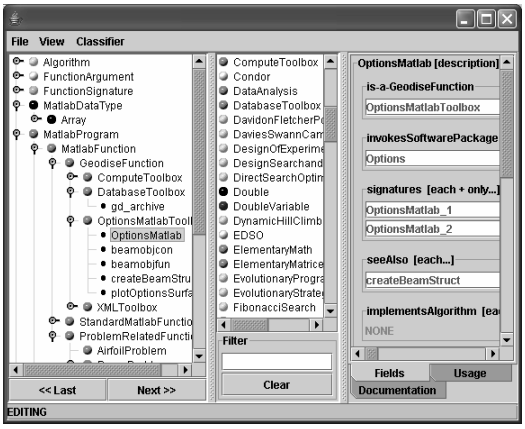



**Figure 3 OntoView editor GUI**

Figure 3 shows the Geodise function ontology loaded into the OntoView ontology editing GUI.

These tools collectively make it possible for the domain expert to create and manipulate complex DAML+OIL ontologies and instances.

OntoView also includes a Java API that allows client code to perform ontology reasoning and instance manipulation, as well as query formation and execution. The domain specific components - the function annotator, the function query GUI and the knowledge advisor, are all built on top of this API.

## 3. Binding Knowledge: Function Annotation

In addition to capturing the knowledge of the Geodise domain experts to form the basis of the Geodise knowledge framework, we also want to capture and reuse the domain knowledge of the ordinary Geodise user. Hence we have created a *Function Annotator* that allows end-users to semantically annotate their own Matlab functions, whilst incorporating them into the Geodise environment, making them available for use in building workflows, and equally importantly making them available for reuse by other end-users.

Figure 4 shows the GUI of the annotation tool, which consists of a Matlab Function Category, an Annotation Description Palette and a Function Browser. The left hand panel, i.e the Matlab Function Category contains a function hierarchy derived from the function ontology, and displaying available annotated functions under the various function categories.

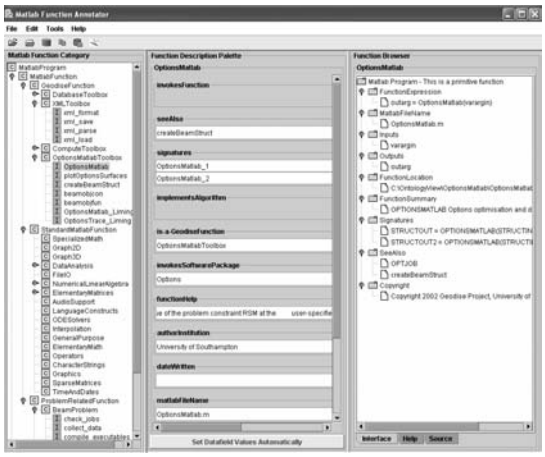


**Figure 4 Function annotator GUI**

The right hand panel is the Function Browser, which is used to load Grid resources for knowledge acquisition.

We have provided a parsing capability to facilitate automatic information extraction. As primitive Matlab functions have conventions for interface specification, we are able to obtain important information about a resource such as input, output parameters and location details directly from the Matlab code. The extracted information will be listed on one tab panel on the right-hand panel, and can be used directly by the function provider in creating the annotations. Other types of semantic information can be expressed manually, either as a result of viewing the code, or via the utilization of knowledge that is not expressed explicitly in the code.

In general the semantic annotations are specified using a mixture of drag-and-drop of formal ontological concepts, and the input of simple data-values such as strings and integers. The ontological concepts involved are presented in the engineers' own terminology as embodied by the function ontology.

Using the Function Annotator the function provider does the following:

1) Loads the Matlab function into Function Browser;
2) Selects an appropriate function type from Matlab Function Category by navigating the concept tree of the function category in the left hand panel.
3) Fills in ontology-driven forms, automatically generated in the Annotation Palette using direct input or drag and drop.

The resulting semantic descriptions of the functions are finally archived via OntoView, into the *iS* database.

The Function Annotator can operate recursively. For instance if the function provider wishes to specify that the function uses a particular algorithm, then s/he may select either an ontological concept representing a type of algorithm, or possibly an existing instance representing a specific algorithm, or else if there is no existing concept or instance that provides a suitable description of the algorithm, can create a suitable new instance. To do this s/he will be presented with a algorithm-definition panel that is similar to the main Function Annotator panel.

## 4. Reusing Knowledge

Knowledge reuse is based on the semantic information generated by the previous stages. We demonstrate here the function query mechanism and a higher-level knowledge advisor that assists function discovery, function configuration, function script assembly and workflow composition.

### 4.1 Ontology driven function query

As functions have been previously annotated with rich semantic information, they can be queried based on various criteria such as "invokeSoftwarePackage" and "author". The criteria are function properties defined in the ontology and used in function annotation activity therefore retaining the consistency through out the knowledge management life cycle.

As illustrated in Figure 5, in order to form a query, end users fill out a ontology driven query form by either selecting pre-defined ontological concepts or concrete instances in a specific category (as shown in the superimposed screenshot) or directly providing a concrete data-value. The query formed in this example is "retrieve all *Geodise functions* whose author is *Graeme Pound* and which invoke Software Package *JavaCog*". The query is then submitted to the OntoView query mechanism which in turn utilises the iS query mechanism

The query mechanism also allows the formation of recursive queries. Hence if a concept or instance required for the formation of the query does not exist, the user is can define it him/herself. For example rather than specifying that a function should invoke a specific named software package as represented by a current

instance, a query may just specify the characteristics of the required package, such as the tasks that it performs, or the methods it uses.

Description Logic based language such as DAML+OIL support queries that are unattainable by using standard database queries. OntoView not only provides a means by which complex Description Logic semantics can be represented within the ontology and the instance descriptions, it also takes advantage of these semantics to allow the formation of complex DL queries, which it then passes on to the underlying the iS query mechanism. This allows end-users who are totally unaware of the existence of Description Logics to formulate queries employing the following types of DL-based constructs:

**Concept Hierarchy:** Given the hierarchical fact that genetic algorithm is a kind of stochastic algorithm, which is a kind of optimisation algorithm, if A is a genetic algorithm with properties x, y and z, then it will be found by each of the following queries:

- Find all optimisation algorithms with properties x, y and z
- Find all stochasic optimisation algorithms with properties x, y and z
- Find all genetic algorithms with properties x, y and z

**Existential and universal quantification:** OntoView can use both existential and universal quantification in various ways to allow the following distinct types of queries to be framed:

- Find all function signatures with inputs x, y and z, and no others
- Find all function signatures with inputs x, y and z, and possibly others
- Find all function signatures that only require inputs x, y or z
- Find all function signatures that do not require inputs x, y or z

**Cardinalities:** It is possible to frame queries involving minimum, maximum and exact cardinality constraints, such as:

- Find all function signatures with exactly three inputs of type x and at least 2 inputs of type y.

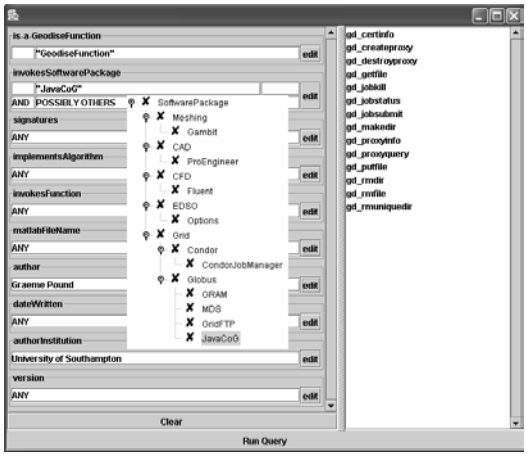



**Figure 5 Ontology driven query forming and execution**

## 4.2 Semantic matching based knowledge advisor

Once semantic instances are made available, it is possible to access and post-process these instances to deduce actionable knowledge.

### 4.2.1 Reusing semantic instances in advice giving

Functions can be assembled together only if their interfaces semantically match each other to some extents, i.e., a function's input semantically consumes the output of another function. Workflow builders, especially beginners are often not clear about the semantic interfaces of the functions. We argue in this paper that suggestions can be deduced through semantic interface matching. This is especially useful when the function repository is dynamically updated or the number of functions is large, which should be the case in the future e-Science community.

Each function can be viewed as a domain specific service that must be configured correctly and assembled with other services to form a problem solving workflow. The granularity of the services varies from low-level atomic functions (usually generic) to high-level workflow building blocks (often more problem specific), which are themselves made up of low-level functions.

There are two types of advice:

**1. Function configuration advice** - this provides automatically generated advice on

function configuration. We call it "horizontal advice" as it is triggered during function configuration, i.e., horizontal scripting.

Semantic decomposition is used when a function parameter is a complex type, e.g., a structure that contains a list of fields that are either primary types or complex types. In such cases, the semantic interface can be extended by recursively decomposing the relevant parameter and its subfields until there are no more complex types. This often yields richer semantic interfaces that contain more concepts and relationships for semantic matching.

**2. Function assembly advice** – functions that can be assembled together according to their interface compatibility. This is referred to as "vertical advice" which is triggered during the vertical assembly of configured function instances.

In addition to the primary data types such as "string" and "integer" used in function interfaces, semantic data types can be used to consider function compatibility when suggesting next step functions for a currently deployed function. This is demonstrated in Figure 6, which shows a semantically matched chain of "FunctionSignature" instances. Each such instance represents a valid set of input variables for some (potentially overloaded) function, and the associated set of output variables. The semantic matches are represented as links, indicating a valid function assembly. Note that each I/O element has an "ArgumentType", the values of which are used in comparison. The workflow at the bottom of Figure 6 is the advised function assembly.

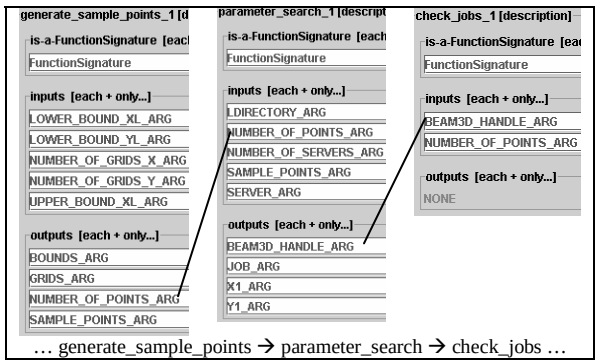



**Figure 6 Semantic matching for function assembly**

Initial results are shown in Figure 7 where advice is given in a console for a particular function, showing the "forward" and "backward" functions that are semantically compatible for function assembly.

```
Matching function <parameter_search> on its signature
"parameter_search_1" ...
                **Forward**
Matchmaking with OptionsMatlab_1 ...
Incompatible.
... ...
Matchmaking with check_jobs_1 ...
Compatible.
... ...
<check_jobs> signature "check_jobs_1" matched on
[beam3d_handle]
<collect_data> signature "collect_data_1" matched on [job]
                **BackWard**
... ...
<generate_sample_points> signature "generate_sample_points_1"
matched on  [number_of_points, sample_points]
```

**Figure 7 Console result of the advisor**

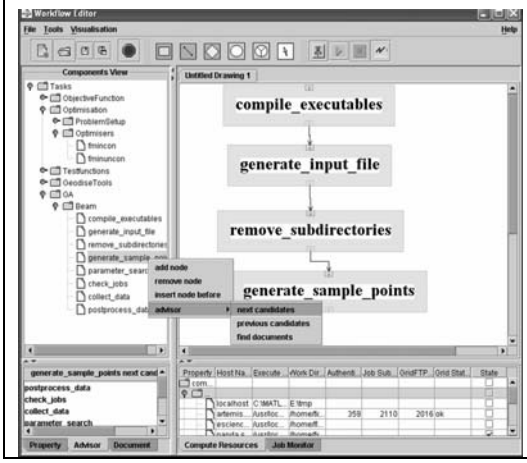
## 4.3 Using the knowledge advisor

There are two application scenarios in which the advisor can be integrated in Geodise. In both cases, semantic based knowledge can be reused in Geodise.

### a) Workflow Composition Environment (WCE)

The workflow composer in Geodise is a GUI based application that allows engineers to visually select tasks from a function hierarchy, configure and assemble them into a workflow for EDSO problem solving.

The purpose of integrating the semantic based advisor into the GUI based WCE is to make use of rich semantic annotations and help the users choose suitable functions and make appropriate configuration during workflow assembly.



```
……
% Compile and transfer the beam3d executable to the client
compile_executables( 'blue02.iridis.soton.ac.uk', server, number_of_servers, ldirectory )

% Generate the input file, and transfer it to the Globus servers
generate_input_file( server, number_of_servers, ldirectory )

% Clean-up. Remove all subdirectories starting with "job"
remove_subdirectories( server, number_of_servers )

% Generate sample points between lower and upper limits
[sample_point, number_of_points, bounds, grids] = generate_sample_points( 2.5, 3.5, 1.5, 2.5, 3, 3 )
……
```

**Figure 8 Advisor integrated in the WCE and the generated scripts**

As illustrated in Figure 8, each function (in the left hand side panel) that has been previously semantically enriched, the workflow advisor can be called to deduce its contextual functions (as listed in the left bottom panel in Figure 8) that can be deployed before/after it. This is achieved by semantically processing the instances generated. In this way, the users can focus on compatible functions that can be of use to further assemble the workflow without tediously investigating the semantic interface of all irrelevant functions. It then generates a Matlab script and submits it to a Matlab server for execution. It also takes care of the workflow management, monitoring and execution, but this is outside the scope of this paper, interested reader can refer to [9] for further information.

## b) Domain Script Editor (DSE)

Quite often, engineers need to edit domain related scripts in addition to GUI based design tools, such as the WCE. But manipulating plain texts is painful and tedious to some people. In Geodise, Matlab is the script language that glues EDSO and grid computing resources together. This motivated the design of a domain script editor together with the advisor integrated.

Key features include:

- **Horizontal advice** on component configuration – exposing the semantic interface, tool-tipping semantic annotations, auto-completions, etc, as shown in popping up windows in Figure 9.
- **Vertical advice** on components assembly – semantic interface matching and reasoning for contextual component recommendation as shown in left bottom panel in Figure 9, where the blue arrow represents for a pre-contextual candidate and the red one for a consequence candidate.
- **De-centralized** - Semantic instances are collected at the stage of knowledge acquisition, separately from their use.
- **Generic** - The DSE is Ontology/Semantic powered meaning that it can be used to advise on different domain scripts when loaded with corresponding semantic annotations. E.g., Gambit scripts, Geodise functions including computation toolbox and database toolbox, problem specific function scripts in Matlab, etc.
- **Component based** - It can be delivered as a java swing GUI component that can be used in any java application (e.g., in the GUI based workflow composer as an alternative view of the workflow).

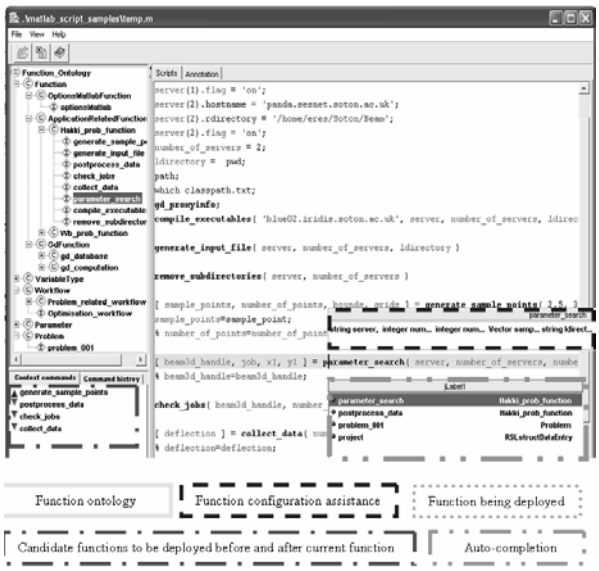



**Figure 9 Domain script editor integrated with the advisor**

## 5. Related work

We have been inspired by the on-going AKT (www.aktors.org) which demonstrates knowledge technologies addressing various stages of knowledge management life cycle from knowledge acquisition, modelling to publication and reuse. While this is demonstrated through list of different projects under AKT, Geodise endeavors to adopt them in the domain of Engineer design search and optimisation and demonstrate an end-to-end knowledge management life cycle within one project.

Previously in [5] we used OWL [3] in the knowledge management life cycle. Protégé2000 with OWL plug-in was used in building ontology and instance population. By using Jena, the advisor accesses the semantics, processes it and provides advice.

We also used pre-defined rules in a JESS rule base [1] to advise workflow assembly, the advantage of this approach is that domain experts can specify function assembly rules that are not consistent to the result of function semantic matching. The disadvantage is the limitation of scalability and the high overhead cost of a rule engine when there are only few rules.

## 6. Summary

In this paper, we have introduced various tools addressing three different stages of the semantic web based knowledge management life cycle – knowledge creation, knowledge capture and knowledge reuse - for assisting engineers using the Geodise toolkit. Accessing and reasoning with the ontology and instances is facilitated via the OntoView mechanism on top of which function annotation and reuse services are built. The reuse includes ontology driven queries over instances and the semantic matching based knowledge advisor for function configuration and assembly. We are currently extending this system to incorporate the semantic annotation and retrieval of the configured workflows.

# References:


1. Tao, F, Chen, L, Shadbolt, N,R. Pound, G, Cox, S.J. (2003) Towards The Semantic Grid: Putting Knowledge To Work In Design Optimisation Proceedings of I-KNOW '03, 3rd International Conference of Knowledge Management, pp. 555-566

2. The Geodise project. http://www.geodise.org

3. Web Ontology Language Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210/

4. The Semantic Grid website. http://www.semanticgrid.org

5. Feng Tao, Nigel Shadbolt, Liming Chen, Fenglian Xu, Simon Cox, "Semantic Web based Content Enrichment and Knowledge Reuse in e-Science", submitted to the 3nd International Conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems (ODBASE), Larnaca, Cyprus, 25-29 Oct, 2004.

6. G. Pound, H. Eres, J. Wason, Z. Jiao, A. J. Keane, and S. J. Cox, A Grid-enabled Problem Solving Environment (PSE) For Design Optimisation Within Matlab. To appear in - IPDPS-2003, April 22-26, 2003, Nice, France

7. Ian Horrocks, Lei Li, Daniele Turi, Sean Bechhofer, "The Instance Store: DL Reasoning with Large Numbers of Individuals", International Workshop on Description Logics - DL2004

8. Volker Haarslev, Ralf Möller , "RACER System Description", International Joint Conference on Automated Reasoning", IJCAR'2001, June 18-23, 2001, Siena, Italy, p.p. 701-706

9. Xu, F and Cox, S.J. “Workflow Tool for Engineers in a Grid-Enabled Matlab Environ-ment”, Proceedings of UK e-Science All Hands Meeting 2003, pp. 212-215

10. The FaCT reasoning system. http://www.cs.man.ac.uk/%7Ehorrocks/FaCT/

11. "The 'OntoView' Knowledge Management System": http://www.geodise.org/publications/papers.htm

12. The DARPA Agent Markup Language Homepage, http://www.daml.org/