Cooperative Design in Grid Services

Rob Mahon[1], Kuo-Ming Chao[2], Muhammad Younas[2], Nathan Griffiths[3]

[1]*Product Development IT, Jaguar Cars Ltd, UK*
rmahon@jaguar.com
[2]*School of MIS, Coventry University, UK*
{k.chao, m.younas}@coventry.ac.uk
[3]*Department of Computer Science, University of Warwick, UK*
nathan@dcs.warwick.ac.uk

Abstract

Quite often workflow control and computational power are not available for engineers and scientists either due to the availability of resources which can be exacerbated by the lack of process controls or the amount of data produced. The availability of grid services (e.g. GT3) presents a potential feasible method to overcome the barriers of sharing heterogeneous computational resources. However, lack of sophisticated coordination mechanisms to specify the workflow would deter the prevailing of the grid services. There is a need for a seamless, flexible, demand based service for engineers/scientists who are able to submit jobs to a computational grid from a remote source in a manner that would ensure the job is executed in an efficient, controlled method. In this paper, we propose a system that adopts BPEL4WS as a coordination language to define and manage the workflow among grid services to meet engineering requirements. A prototype system is built and evaluated with a case study from the automotive industry to demonstrate the feasibilities of the proposed system.

1. Introduction

Large and complex engineering design in the automotive industry often involves multiple disciplinary design teams with each design team using specialised software tools which often require exhaustive computational resources and produce large amount of data in order to produce designs and effective predictive models. Each design team does not only have to comply with its design requirements (e.g. design specifications and finishing time) but also has to ensure that a consistent product model is produced alongside other design teams. The interactions among multiple design teams are inevitable for sharing the information and knowledge. During the design process one particular design team specialises in crash worthiness. This is the study of protecting the occupants of vehicles during various modes of impacts and minimising the amount of damage such as high impact head on collisions, side on impacts and low impact shunts.

In order to certify design processes to be efficient and effective, a workflow is introduced to coordinate the design activities. The exploitation of idle computational resources in the organisation is needed to shorten the design process or to improve efficiency of design activities. The concept of computational grid is deployed to integrate the heterogeneous computational resources and provide a seamless and flexible environment for design simulations and job execution. This paper presents a novel Computer Supported Cooperative Work (CSCW) architecture that enables multiple disciplinary design teams to work seamlessly via grid computing such as Open Grid Service Architecture (OGSA) [1] and BPEL4WS (Business Process Exestuation Language for Web Services)[2].

This paper is structured as follows: The next section will describe a realistic automotive design scenario in order to illustrate the issues encountered by the industry. Section 3 describes current problems encountered and a possible solution to address the identified issues. Section 4 gives an overview of the specifications of (OGSA) and its main characteristics. A new industry standard for composing web services and managing the workflow among web services, BPEL4WS, will be introduced in Section 5. Section 6 describes a proposed grid enabling architecture that integrates OGSA and BPEL4WS to provide a seamless and manageable CSCW environment for engineering crashworthiness design. In Section 7, a case study will be used to demonstrate the proposed architecture. The discussion and conclusion will be presented in the last Section.

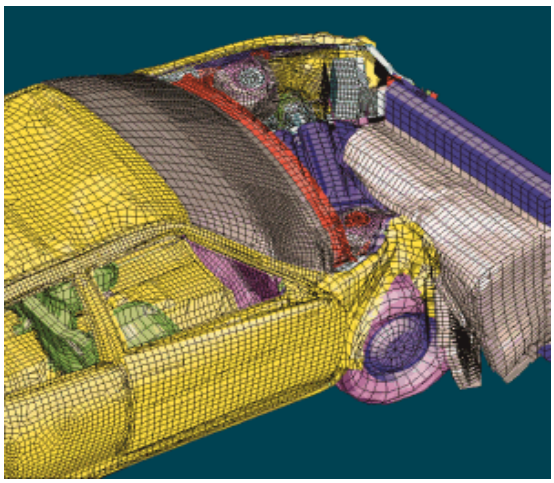

Figure 1. Illustrates an Finite Element analysis of a high speed frontal impact using an offset deformable barrier

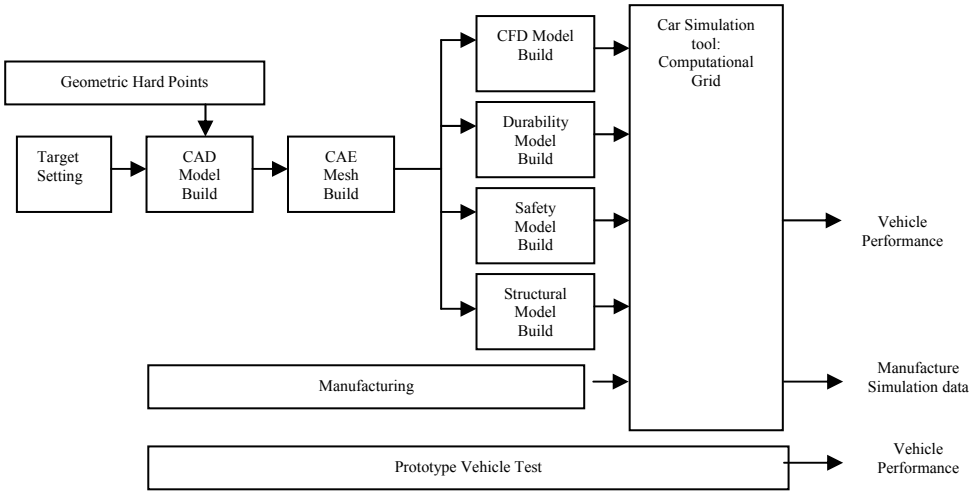

Figure 2. Shows multidisciplinary tasking within automotive engineering

2. Automotive Simulation

In automotive design, Safety engineers create mathematical models to represent the structural integrity, dynamic response and durability of vehicles to name just a few. This methodology uses techniques called Finite Element Analysis (FEA), the solution of simultaneous equations on geometry and materials represented by elements/nodes, producing definitive results for a given problem (see Fig. 1). The complexity of defined problems has increased dramatically with the advance of computational resources. For example, engineers submit multiple FEA impact simulations, optimisation or design of experiments analysis on multi-processor servers producing large quantities of data which takes time to process and analyse. Full vehicle analysis can produce gigabytes of data for a single crash mode with the latest safety models taking up to 2 days to complete on new multi-CPU high end servers. During Design of Experiments (DOE) or Optimisation analysis results can be collated statistically and results presented via charts and graphs to the analyst. This type of analysis is sometimes called Design or Experiments, Optimisation, Robustness Design or Sensitivity Analysis and generally involves larger amounts of computation and hence resources over regular single FE jobs. The aim of this type of analysis is to show performance trends rather than specific animations with an aim to provide a greater feeling of performance reliability and build quality. This involves a number of design teams and tools with a coherent coordination approach to ensure a consistent product model produced. Fig. 2 shows a number of design teams involved in creating a car model at various stages of design process and the high level workflow.

A common issue is identifying the funds to secure computational resources. Dedicated compute resources are generally too costly for large scale analysis and PC or workstation scavenging is not dedicated or does not ensure the compute resource for analysis work. Workstations are not usable in a grid whilst an engineer/scientist is using it in his/her general day-to-day design activities as it slows the operation to a crawl unless it is a high end multi-processor desktop workstation. Nightly analysis windows are not long enough for non-linear predictive analysis as once jobs are started it is difficult to transfer to another workstation.

Another large problem is processing the vast amounts of data created by the analysis. This is a problem from the model conceptual stage right through to the post processing and storage. It currently takes too much time to process individual jobs. More automation is needed to control the information flow and process this data. The compute grid and web technology are introduced in order to address the aforementioned issues.

3. Working Practices

The solution is to build a compute grid made up from various sources, dedicated high end servers and by clustering workstations and PC's into one large pool. MPP (Massively Parallel Processor) analysis utilising libraries such as MPI (Message Passing Interface) are used to distribute jobs and speed up analysis on workstations to ensure jobs could complete overnight whilst the dedicated resources run continuously. Comparable analysis can be a problem with certain types of jobs executed on different types of compute servers. Where this is the case efficient resource management associated with operating system and hardware set-up are critical.

The infrastructure of the overall system is based on client-server architecture and a centralised fileserver/database system for sharing information. Web interfaces on the client would be used to control and track analysis. A web page template written with standard web protocols is used for visualising job set-up and process to ensure correct practice. Physical inputs and job characteristics are stated or standard inputs like barrier models, input loads, velocities and material properties can be obtained from central FE (Finite Element) libraries.

This could be accessed on the client via a secure web server. Jobs can be tracked individually or job statistics can be obtained and displayed for grid capacity planning or historic data used for comparison or utilisation purposes.

The third party grid enabling software focuses on scheduling on the computational resources. LSF, PBS-Pro, and Codine are examples of commercial schedulers that distribute work packets to computational resources. Currently the LSF from Platform is adopted.

Figures 3 and 4. CAE simulation in initial positions, courtesy of TNO

Figures 5 and 6. Impact and airbag delivery for a belted female adult and an unbelted child.

With the latest scheduling software it is possible to intelligently schedule and guarantee completion of batch workloads across a distributed IT environment. Load balancing techniques help meet service levels agreements with customers and helps ensure the right resources are automatically allocated to the right users with maximum efficiency.

With the use of web technologies the steps involved in the analysis procedure can be controlled. Process control is a subject on its own but in the context of this work it is enough to say that each step through the analysis setup, submission and post processing is tightly controlled via the web interface content.

Once all the input data has been succesfully gathered in to the pre-analysis input files the user can define the output parameters and initiate the task. Analysis templates and wrapper scripts are used to control job submission with additional software codes such as Storm by Easi Engineering or ISight by Engenious Software to control the manner in which the optimisation analysis is executed. These type of codes specialise in optimisational algorithms used for this Design of Experiments (DoE) or Optimisation analysis. The results are gathered for each job and collated where post job scripts can automatically process the results and present the data to the shared areas where dynamic web pages are written with the results contained. Job tracking showing the state of jobs and servers are accessed via a web console from the remote location. Results are presented via animations, graphs and other statistical methods to gain a good understanding of design variables within the design envelope.
Figures 3-6 shows animation stills from TNO's Madymo occupant analysis software.

Current practices improve the usage of idle computational resources, but they are some drawbacks. In a crash simulation analysis, mutliple analysis instances and jobs can be created by various users for different cases but the correlation between the discipline instances and jobs is not well defined. An enabling tehcnology to manage multiple instances and jobs is required across the various disiplines to improve the coordination of results and input/output data.

The existing architecture is built upon Web technology. This leads to client side programs that cannot utilise the results produced by servers for further analysis. An investigation to Web Service technology is needed.

The MPP process lacks definition, flexibility and facility for modelling the workflow. In addition, the stateless transition between jobs increases the difficulty in maintaining systems when interruptions are occur. It is also important to look into emerging standards and open source grid specification in order to gain long term support from IT industry. The next section briefly describes the Open Grid Services Infrastructure (OGSI) specification.

4. Grid Services

The OGSI specification utilises the WSDL [3] and XML schema definition languages from Web services to define an extended component model [6]. The aim of the specification is to address the common issues that occur in sophisticated distributed applications, such as the management of distributed long-lived states. In order to achieve this aim, OGSI defines the notion of a *Grid service instance* [4]. "A *Grid service instance* is a (potentially transient) service that conforms to a set of conventions (expressed as WSDL interfaces, extensions, and behaviours) for such purposes as lifetime management, discovery of characteristics, notification, and so forth." [4]. The OGSI specification not only inherits the interoperability features from Web services, but also includes the following features.

- *Stateful interactions*: *serviceData* is the OGSI approach to stateful Web services. It exposes a service instance's state data to service requestors for queries, updates and change notifications [4]. The concept of *serviceData* is similar to a JavaBean. Thus, each item of data is associated with a set of methods (e.g., *get* and *set*) to access the state of data (attributes).
- *References*: OGSI uses Grid Service Handles (GSH) [5] to name and manage Grid service instances. A client wishing to communicate with a service instance

must map the GSH to a Grid Service Reference (GSR). This is because a GSH only contains a minimal set of information, such as a URI and it does not carry sufficient information to allow a client to communicate directly with the service instance. Instead, a GSR contains all the information that a client requires to communicate with the service.

- *Collection of service instances:* OGSI allows a number of services to be grouped together so that they can be easily maintained by clients. A Grid service can define it's relationship with other member services in the group. Services can join or leave a service group.
- *Life Cycle management*: This gives a client the ability to create and destroy a service instance according to it's requirements.
- *Inheritance:* OGSI adopts some of the features from the WSDL 1.2 such as *portType* inheritance which allows one *portType* to extend from other *portTypes*. To distinguish between WSDL 1.1 and 1.2, OGSI uses GWSDL to name the WSDL 1.2 *portTypes*.
- *Asynchronous notification*: OGSI provides a facility for asynchronous notification of state change using a pull/push mechanism.

In summary, the OGSI specification is an attempt to provide an environment for Grid services to be more manageable within large and complex distributed applications and to provide a platform for higher-level mechanisms to compose services. However, it does not address the issue of workflow management in the grid services. The following section summarises one of the new industry standards in composing web services.

5. Workflow (BPEL4WS)

BPEL4WS is an industry standard specification for defining the workflow between Web services [2]. It is intended to provide a workflow language to model complex and non-deterministic business processes. The characteristics of correlating business processes often depend on the data and BPEL4WS provides a set of activities to model data-dependent behaviours. BPEL4WS provides conditional and time-out constructs in order to address non-deterministic situations which often occur in business processes. BPEL4WS also provides developers with the ability to specify exception conditions and their consequences, including recovery sequences. The most important feature of BPEL4WS is to support business process coordination among multiple parties. This enables the outcome (success or failure) of units of work at various levels of granularity of the business processes. BPEL4WS enables modelling of long-running interactions between business processes with nested units of work between them and each with its own data requirements.

BPEL4WS is built upon three XML-based specifications: WSDL 1.1, XML Schema 1.0 and XPath 1.0. Partners are used by BPEL4WS to model interacting services in business processes. Each partner has a unique name and other services can interact with the partner through it's name. Each partner is associated with a WSDL document, which describes the information that a service contains. The process model allows developers to specify the relationships between partners through a set of pre-defined activities in order to orchestrate Web services.

In BPEL4WS, the business process begins with a *receive* activity that receives a request from the client which triggers the process as a whole. The *reply* activity is the end of the process that responds to the request associated with a *receive* activity. The *invoke* activity allows invocation of an operator associated with *portTypes* (which is defined in a partner Web service). The state of messages related to business process is temporarily stored in variables.

Developers can handle known and unexpected exceptions with *throw* and *compensate* activities. The response to external events can be specified through event handlers. Control flow in BPEL4WS is similar to traditional structured process control containing constructs such as *while*, *switch*, and *sequence*. A *sequence* activity defines blocks that contain one or more activities that are performed sequentially. The *flow* activity allows the activities within the block to be performed concurrently. Finally, the *correlation* construct specifies that only correlated instances can be invoked.

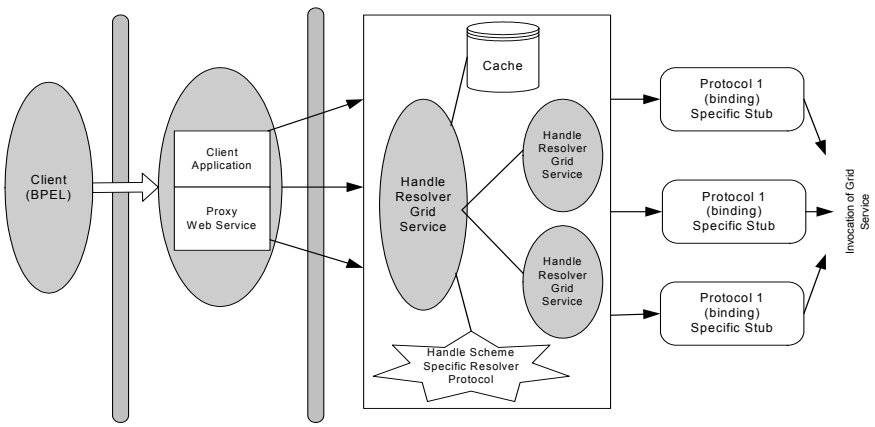

Figure 7 Proposed Grid-Based Computer Supported Cooperative Work Framework

6. The Proposed Architecture

BPEL4WS was originally designed to orchestrate standard Web services, but not for Grid services. The incompatibility between grid and web services complicates the application of BPEL4WS to orchestrate the grid services. Detailed analysis on the differences between grid and web services can be found in [6]. In order to alleviate this problem we propose an architecture (as shown in Fig. 7) to enable Grid service composition via BPEL4WS. In the proposed architecture we wrap Grid service clients as Web services called Proxy Web Services. All of the interfaces defined for the Grid services are re-defined in Java Beans as an XML complex type (in WSDL) with a public Grid service instance attribute. An additional operator, *startGService*, is defined and implemented in the Proxy Web Service. This operator is to create new Grid service instances. The process of a series of activities being carried out is described as follows:

The BPEL4WS user initiates the client. The Proxy Web Services are invoked according to the workflow descriptions in BPEL4WS. The Proxy Web Services will trigger corresponding Grid services through an embedded *startGService* operator. The *startGService* operator is the standard procedure for creating a Grid service instance by calling a GSH, holding its returned value and mapping it to a GSR. When a Grid service instance is created, the *startGService* operator obtains a reference and stores it in the predefined public Grid service instance attribute. Thus, the GRS reference is stored as a global variable and is visible to the whole instance. When BPEL4WS wishes to call individual operators in the Grid service it calls a BPEL4WS engine, such as BPWS4J or Collaxa, to activate Proxy Web Services stored in the Web service container. The Proxy Web Service then uses the Grid service reference, which is stored in the public attribute, to tell the Grid service container to invoke the corresponding Grid services. The Grid service replies with the results to the Grid service client that made the request. The Grid service client passes the response to its Proxy Web Service. The BPEL4WS engine can obtain the result and pass it on to the next service. This architecture is illustrated in Figure 7.

This principle can be used to design a Grid service from existing BPEL4WS descriptions. The advantage of this approach is that the impact on the BPEL4WS descriptions and the associated WSDL can be minimised when the Grid service is re-deployed to different locations.

7. Case Study

In order to evaluate the feasibility and effectivenes of the proposed system a relative small scale system was adopted. This simple case study involves impacting a tube of steel into a rigid wall with a given mass and velocity. The model in decomposed into domains so that calculations can be parrallelised. The domains are represented by the boxes labeled A, B, C and D (see Figure 8).

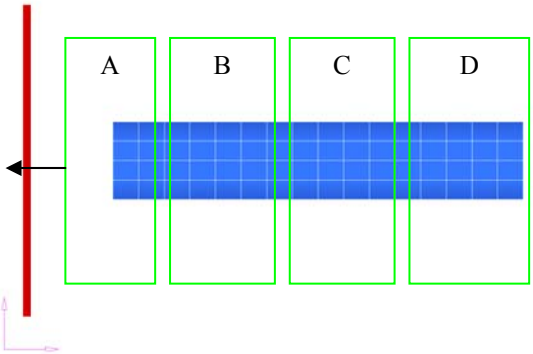

Figure 8. An impact simulation calculating nodal displacements.

Each processor(s) is responsible for solving it's own domain. Communication is required between the processors for information across domain boundary conditions for each timestep. For each domain the following is calculated in parallel: contact forces, constraints and finally update node positions. In this case, B and C require two inputs and outputs which are derived from their negbouring reaction forces. A has first impact and passes the force to B. D receives the force from C and bounces the force back to C. Displacement simulation software is used and located at 4 different processors (nodes) in the grid. The software has domain boundary information and maintains it's own state. The following shows the snipet of BPEL4WS for modelling their interactions:

```
<process name="Displacement Simulation"
----
 <variables>
  <variable name="impact1"
        messageType="tns:impactinfo"/>
  <variable name="impact2"
        messageType="tns:impactinfo"/>
----------
 </variables>
 <partners>
  <partner name="A-displacement"
       serviceLinkType="lns:CarADisplacementLinkType"
       myRole="A-Boundary"/>
  <partner name="B-displacement"
       serviceLinkType="lns:CarBDisplacementLinkType"
       myRole="B-Boundary "/>
  <partner name="C-displacement"
       serviceLinkType="lns:CarCDisplacementLinkType"
       partnerRole=" C-Boundary "/>
  <partner name="D-displacement"
       serviceLinkType="lns:CarDDisplacementLinkType"
       partnerRole="D-Boundary "/>
 </partners>

 <Sequence>
  <receive name="initial" partner="A-displacement"
       portType="DisplaceA:CarAPT"
       operation="start" variable="impact"
       createInstance="yes">
   </receive>
  <flow>
  <links>
<link name = "AtoB" />
<link name = "BtoC"/>
--------------
</links>
```

```
  <invoke name="A-impact" partner="A-displacement"
        portType="DisplaceA:CarBPT"
        operation="ImpactAnalysis"
        inputVariable="impact1"
        outputVariable="impact2">
       < Source linkname=”AtoB”>
  </invoke>
  <assign name="assign">
   <copy>
            <from variable = "impact2"
             portType="tns:displacementinfo" />
            <to variable="impact1" PortType = "tns:impactinfo/>
   </copy>
  </assign>
  <invoke name="B-impact" partner="B-displacement"
        portType="DisplaceB:CarBPT"
        operation="ImpactAnalysis"
        inputVariable="impact1"
        outputVariable="impact2">
      < Target linkname=”AtoB”>
      < Source linkname=”BtoC”>
 </invoke>
  </flow>
 </Sequence>
</Process>
```

The relationship between four nodes are defined as partners which are specified in WSDLs. The variables are used to hold the data from one grid service and pass to another. The operation *start* receives the user request and starts the whole process. It takes one input object *impact1* which contains the information about mass, velocity and timestep. Variables *impact1*and *impact2* are the same as the port type but used for different purposes. The generated displacement of nodes is stored in the system as internal states.

A-Impact activity is invoked to simulate the impact and generates the result that passes to B-Impact process. The C-Impact activity does not start until B-Impact activity produces the result. The D-Impact is the last activity waiting for the input from activity C-impact. The *flow* is used to allow them to run concurrently. Modelling the dependency of the concurrent activities is the *link*. The exmperimental results show that the proposed approach produces the same result as the standalone one.

8. Conclusions

Large and complex engineering designs (car industry) often requires significant computational resources and involve complicated design activities. To increase utilisation of the existing idle resources is the key to improve efficiency and effectiveness of the design. The improvement of compuational effeciency without a certain degree of relability is not a promising solution. To ensure the consistency of design activities through a coordination protocol is an important issue. In this paper we propose a novel architecture for integrating grid services with BPEL4WS to allow designers to specify workflow between design activities with a high level language as well as to ensure that the product model is consistent. This simplified case study adopted from car design industry demonstrates the feasibility and effectiveness of the proposed system.

BPEL4WS is not the only specification for orchestrating Web services. BPML (Business Process Modelling Language) proposed by an industry consortium aims at modelling business data through a meta-language [7], but there is no supported run-time environment yet. The Semantic Web community has proposed OWL-S [8] for describing the semantics of Web services and composition mechanisms for Web services. However, there is no sophisticated engine like BPWS4J or Collaxa to support the OWL-S specification. [9] defines the semantics of Web services via OWL-S and translates the descriptions to BPEL4WS. Thus, BPWS4J can provide a run-time environment to execute Web services accordingly. Other on-going research is to use agents with a specific reasoning mechanisms such as GoLog [10], to compose Web services.

Currently we are planning to run on larger scale experiments which will involve a number of design teams and software systems with the introduction of intelligent agent technology. Piror to that, a number of existing software tools need to be re-engineered in order to run on a distributed environment.

References

[1]. OGSI, Open Grid Services Infrastructure (OGSI) Version 1.0, http://www-unix.globus.org/toolkit/documentation.html

[2]. Business Process Execution Language for Web Services Version 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[3]. W3C Note "Web Services Definition Language (WSDL) 1.1", http://www.w3.org/TR/WSDL

[4]. Service-*Oriented Architecture (SOA) Definition*: http://www.servicearchitecture.com/web-services/articles/serviceoriented_architecture_soa_definition.html.

[5]. Globus toolkits 3, http://www-unix.globus.org/toolkit/documentation.html

[6]. Chao, Kuo-Ming, Younas, Muhammad, Griffiths, Nathan, Awan, Irfan, and Anane, R. “Analysis of Grid Service Composition with BPEL4WS”, *to appear in Conference Proceedings of 18th Advanced Information and Network Applications*, IEEE CS, 2004.

[7]. BMPI.org, Business Process Modelling Language Specifications, *http://www.bpmi.org/bpml.esp*

[8]. OWL Services Coalition. OWL-S: Semantic Markup for Web Services. OWL-S v. 1.0 White Paper, http://www.daml.org/services/daml-s/0.7/daml-s-wsdl.html, Sept 2003.

[9]. S. McIlraith and T. Son. Adapting Golog for Composition of Semantic Web Services. *Conference Proceedings on Knowledge Representation and Reasoning*, April 2002.

[10]. Evren Sirin, James Hendler, Bijan Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions." *Proceedings of "Web Services: Modeling, Architecture and Infrastructure" workshop in conjunction with ICEIS2003*, 2002

.