Available online at www.sciencedirect.com
ScienceDirect
Computers in Industry 57 (2006) 778–786
COMPUTERS IN INDUSTRY
www.elsevier.com/locate/compind

BPEL4WS-based coordination of Grid Services in design

Kuo-Ming Chao [a,*], Muhammad Younas [b], Nathan Griffiths [c]

[a] Department of Computer and Network Systems, Coventry University, UK
[b] Department of Computing, Oxford Brookes University, UK
[c] Department of Computer Science, University of Warwick, UK

Accepted 27 April 2006
Available online 21 June 2006

Abstract

Engineers and scientists often do not have access to workflow control or sufficient computational resources, either due to limited resource availability and lack of process controls, or because of the quantity of data produced. The availability of Grid Services (e.g. GT3) presents a potentially feasible method to overcome the barriers of sharing heterogeneous computational resources. However, a lack of sophisticated coordination mechanisms for specifying workflow deters the widespread adoption of Grid Services. There is a need for a seamless, flexible, demand-based service to enable engineers and scientists to submit jobs to a computational Grid from remote sources in a manner that ensures that jobs are executed in an efficient, controlled method. In this paper, we propose a system that adopts Business Process Execution Language for Web Services (BPEL4WS) as a coordination language to define and manage 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 feasibility of the proposed system.

Keywords: Grid Services; Workflow; Cooperative design

1. Introduction

Engineering design in the automotive industry is a complex task that typically involves several multi-disciplinary design teams each using specialised software systems. Such systems often require significant computational resources and consume/produce large amounts of data when generating designs and effective predictive models. Design teams not only have to comply with design requirements (e.g. design specifications and finishing time) but must also ensure that a consistent overall product model is produced. Interactions among multiple design teams are inevitable for sharing information and knowledge.

Consider a scenario of an automotive design process in which a particular design team specialises in crash worthiness. This is concerned with protecting vehicle occupants during various modes of impact, such as high impact head-on collisions, side-on impacts and low impact shunts. Such a design process requires large computational resources and the effective coordination of various activities.

In order to ensure that the design process is efficient and effective, a workflow is introduced to coordinate the design activities. The exploitation of idle computational resources in the organisation is desirable to shorten the process and to improve the efficiency of design activities. The concept of a Computational Grid is employed to integrate 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 utilising Open Grid Service Infrastructure (OGSI) [1] and Business Process Execution Language for Web Services (BPEL4WS) [2]. It is to be noted this work does not consider the overhead that may incur as a consequence of message communication or interaction among design teams.

The following section introduces a realistic automotive design scenario that illustrates the issues we address. In Section 3 we give an overview of Grid Services. A recent industry standard for composing Web Services and workflow management, BPEL4WS, is introduced in Section 4, and a comparison between it and Grid Services is given in Section 5.

* Corresponding author. Tel.: +44 24 76888908.
E-mail addresses: k.chao@coventry.ac.uk (K.-M. Chao), m.younas@brookes.ac.uk (M. Younas), nathan@dcs.warwick.ac.uk (N. Griffiths).

0166-3615/$ – see front matter
doi:10.1016/j.compind.2006.04.012

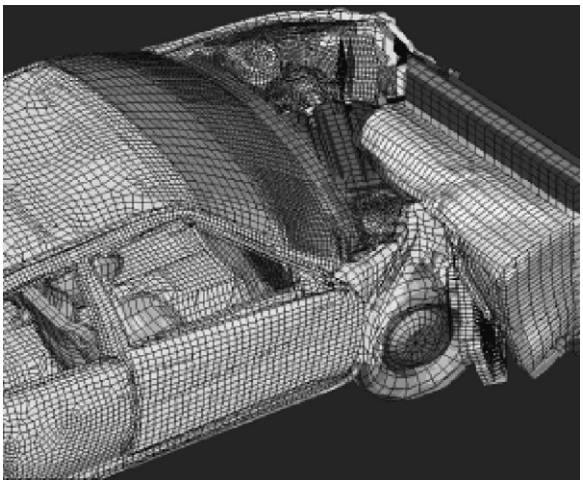

Fig. 1. Illustrates a Finite Element Analysis of a high speed frontal impact using an offset deformable barrier.

Our proposed architecture for integrating Grid Services and BPEL4WS is described in Section 6. The use of this architecture is illustrated by a case study in Section 7. In Section 8 we discuss related work and, finally, in Section 9 we give our conclusions.

2. Automotive simulation

In automotive design mathematical models are created to represent the structural integrity, dynamic response and durability of vehicles. This methodology uses Finite Element Analysis (FEA), the solution of simultaneous equations on geometry and materials represented by elements/nodes, to produce definitive results for a given problem (illustrated in Fig. 1). Engineers typically submit multiple FEA impact simulations, optimisation or design of experiments analyses. Full vehicle analysis produces gigabytes of data for a single crash mode with the latest safety models taking up to 2 days to complete on high-end multi-CPU machines. The aim of this type of analysis is to show performance trends 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 is produced. Fig. 2 illustrates the high-level workflow for how a number of design teams are involved in the design process.

Meeting the computational resource requirements and processing the vast amount of data produced is a fundamental difficulty. Dedicated computer servers are expensive, workstation scavenging does not ensure resource availability (since they are not dedicated), and nightly analysis windows are too short. In this paper, we address this issue by using Grid and Web Services technology to control information flow and processing. In the following sections we introduce Grid Services and workflow, and then describe our proposed architecture.

3. Grid Services

The OGSI specification utilises the WSDL [3] and XML schema definition languages from Web Services to define an extended component model [4]. The specification addresses common issues that occur in sophisticated distributed applications, including the management of distributed long-lived states. In order to achieve this, OGSI defines the notion of a *Grid Service instance* [5]. "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*: The OGSI approach to stateful Web Services is the notion of *serviceData*, which exposes a service instance's state data to service requestors for queries, updates

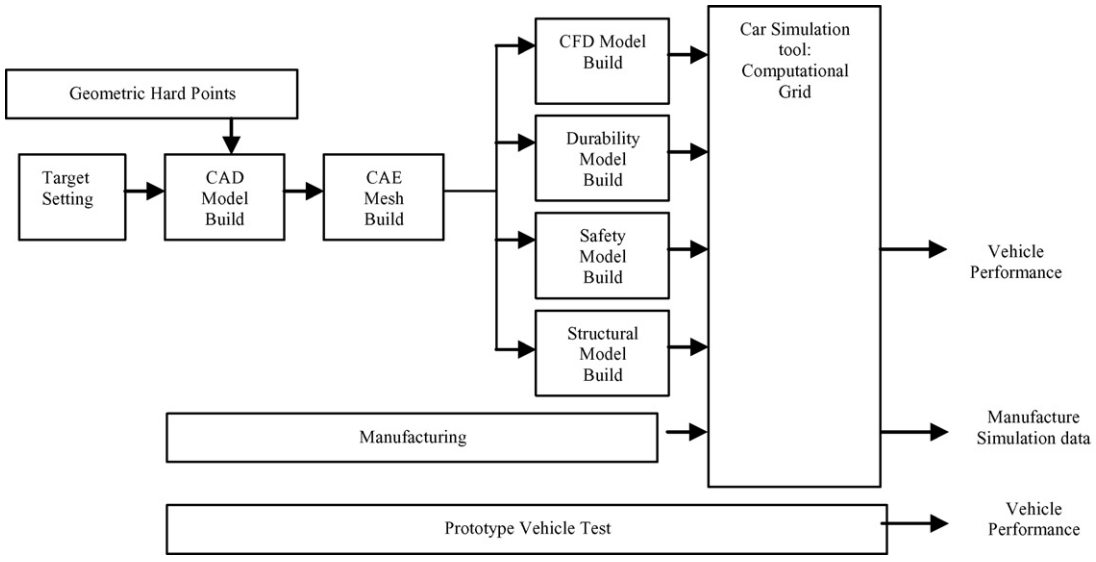

Fig. 2. Multi-disciplinary tasking within automotive engineering design.

and change notifications [5]. In the *serviceData* approach each item of data is associated with a set of methods (e.g. *get* and *set*) to access the state of data, in a similar manner to JavaBeans.

- *References*: OGSI uses Grid Service Handles (GSH) [6] 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, the information that a client requires to communicate with the service is contained in a GSR.
- *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 its relationship with other 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 its 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 attempt to address the issue of workflow management in Grid Services. The composition of Web Services, however, can be described by the workflow definition language BPEL4WS, as we discuss in the following section.

4. Workflow definition

BPEL4WS is an industry standard specification for defining the workflow between Web Services [2]. It provides a language for modelling complex and non-deterministic business processes. The characteristics of correlating business processes often depend on data and BPEL4WS provides a set of activities to model data-dependent behaviours, and conditional and time-out constructs to address non-deterministic behaviour. BPEL4WS also provides developers with the ability to specify exception conditions and their consequences, including recovery sequences. The most important feature of BPEL4WS is its support of business process coordination among multiple parties, enabling the definition of units of work at various levels of granularity of the business process. 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 can communicate with other services. Each partner has an associated WSDL document describing the information that a service contains. The process model allows developers to specify 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 takes a client request and triggers the process as a whole. The *reply* activity is the end of the process that responds to the triggering request. An *invoke* activity allows invocation of an operator associated with *portTypes* (as defined in a partner Web Service). The message state of the 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* and *switch*. The *sequence* activity defines blocks that contain one or more activities that are performed sequentially, and the *flow* activity allows activities within a block to be performed concurrently. Finally, the *correlation* construct specifies that only correlated instances can be invoked.

BPEL4WS, and its engine BPWS4J, offers predicable behaviour and performance. However, BPEL4WS is limited by inflexibility in the composition process, centralised workflow enactment, and the need for Web Services to be known and defined *a priori* [7]. The use of concrete Web Services, available beforehand, may lead to inefficiencies through sub-optimal selection of Web Services, and to potential service discontinuity.

5. Comparing Grid Services and BPEL4WS

There are two significant differences in approach between Grid Services and BPEL4WS. Specifically, the mechanisms for achieving coordination and for serialisation differ, as described below.

Firstly, coordination between Web Service instances is driven by data in BPEL4WS. Developers must define correlation sets from *portTypes* in WSDL and use them to correlate instances (in a similar manner to database systems handling tables through index keys). Alternatively, OGSI uses Grid Service instance references for coordination, where each instance has a unique reference (similar to an object reference). Since GT3 is mainly implemented through the JAXRPC specification (a Web Service specification based on Java RMI), the management of a collection of instances is similar to handling multiple instances in Java. Therefore, GT3 cannot export its instance references to BPEL4WS, and BPEL4WS cannot hold references of the Grid Service instances. Consequently, the additional functions in GT3 such as the grouping of Grid Services and life cycle management, pre-call and post-call Grid Services cannot be utilised by BPEL4WS

directly. Furthermore, BPEL4WS does not allow Web Service instances to be destroyed, instead it provides termination of the whole process.

Secondly, Grid Services allow serialisation of Grid Service instances, whereas BPEL4WS only serialises the variables in a process. Thus, BPEL4WS cannot instruct Grid Services to serialise the instances. Both BPEL4WS and GT3 refer to WSDL to obtain information about services, using this information to initiate and respond to requests. However, the versions they build upon are different. GT3 adopts WSDL1.2, renaming it as GWSDL, and BPEL4WS is based on version 1.1. BPEL4WS cannot parse the extra features proposed in the later version, although this may be easily resolved when WSDL1.2 becomes an official WWW specification. An important feature that GT3 supports is the notification mechanism that allows services to push or pull information when the state changes (similar to Java's observer/observable). However, there is no equivalent mechanism in BPEL4WS.

6. The proposed approach

In this section, we present our proposed approach. We first highlight the motivating technological issues, and then present the proposed architecture, and finally we describe how the architecture enables the coordination of Grid Services.

6.1. Motivation

The issue of Grid Service composition is not well addressed by OGSI. The aim of OGSI is to provide a platform that makes Grid Resources interoperable and transparent, but leaves orchestration and coordination of disparate resources to developers. Although OGSI is built upon Web Service technologies, additional features have been introduced to meet new requirements, and these lead to incompatibility between Grid Services and standard Web Services. The workflow language for composing Web Services, BPEL4WS, fits the OGSI idea of interoperability and transparency with the possibility of enhancing OGSI with composition ability. A Grid Service could present its volatile nature in the process, but to accommodate and exploit this, a flexible composition mechanism is needed. However, as described in Section 4, BPEL4WS is relatively static, requiring pre-scripting of workflow, and predefinition of participating services' roles. It was, however, designed to orchestrate Web Services rather than Grid Services, and the incompatibility between them complicates the application of BPEL4WS to orchestrate Grid Services.

6.2. Components of the proposed architecture

In order to enable Grid Service composition via BPEL4WS we propose a novel architecture, as shown in Fig. 3. The architecture introduces the notion of a Virtual Grid Service (VGS) to bridge the gap between Web Services and Grid Services. A VGS is a manufactured standard Web Service that contains the functionality to support communication between Grid and Web Services and to maintain the additional functions supported by Grid Services. The aim of this approach is to minimise or eliminate efforts on modifying existing BPEL4WS scripts when participating Grid Services are replaced with others. Additionally, this approach provides a late binding mechanism for BPEL4WS to invoke non-deterministic Grid Services.

VGS contain three key components: Proxy Grid Service, template WSDL, and a library containing a number of abstract functions that correspond to the OGSI component model. A Proxy Grid Service is an abstract class that includes an abstract function containing the necessary steps and information for

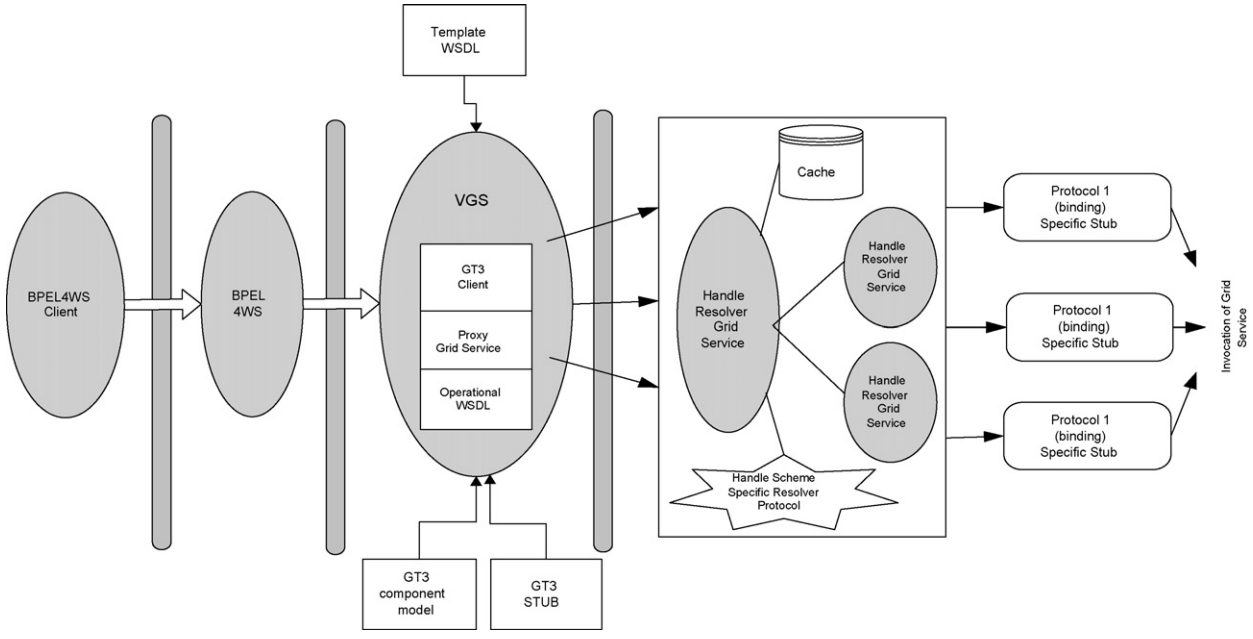

Fig. 3. Proposed Grid-based computer supported cooperative work framework.

locating an actual Grid Service. The stub code of the Grid Service is imported and accessible to Proxy Grid Service. The component model defined by OGSI is represented as a set of abstract functions and when the actual Grid Service is identified, these functions are inherited to produce a concrete class. The concrete class acts as a client-side program to invoke the implemented Grid Service. For example, the *startGService* function is defined in the Proxy Grid Service and implemented in its subclass for creating a new Grid Service instance. As the corresponding variable for the Grid Service instance is declared as a public instance, it makes the GSR visible to the whole class. In addition, the input and output of a service are explicitly defined as separate functions.

A template WSDL is an interface to a Proxy Grid Service that contains essential elements for BPEL4WS, namely: name space, role, *partnerlinktype*, and *portType*. The template WSDL is populated and becomes operational WSDL after a subclass of Proxy Grid Service is generated. So, the WSDL template is instantiated to describe the Proxy Grid Service subclass and to make these functions available to other applications.

The component model proposed by OGSI such as notification, serialisation, and logging, etc. are redefined as an interface class (AGridService), which is extended by the Proxy Grid Service. The actual functions or services are realised by a concrete grid service and become operators in the WSDL, allowing BPEL4WS to invoke them.

6.3. Coordinating design process

The proposed architecture is a bottom-up approach for the design process that assumes a number of existing Grid Services are available and must be coordinated to achieve a specific design activity (e.g. impact analysis). The type of services involved in the workflow is known, but no specific Grid Service has been identified. The users make shared operations available via JavaBeans and each operator has a corresponding method in a JavaBean. The operational WSDL, based on the template WSDL, describes these operations. The operational WSDL includes a ''start'' *portType*, which accommodates the Proxy Grid Service (*startGService* operator). Related methods in the JavaBean are defined in order to provide appropriate functions. The stub files are imported and included in the library for compilation. The Grid Service component model, defined as an interface class, is also inherited by the JavaBean. These interfacing methods are overloaded by the actual Grid Services defined in the JavaBean. After the files have been compiled, they are zipped and deployed in the web container. These integrated components form the VGS and are ready for use. The user writes BPEL4WS script to manipulate the VGS.

A Grid Service can use alternative Grid Services. In this case the alternative service must have the same stubs as the replaced service, but with a different URL or Grid Service instance. The only change needed to the program is the Proxy Grid Service; no modification of the BPEL4WS script or other components is required. The actual Grid Services to be deployed in the script do not need to be specified. This decoupling of the Grid Services and BPEL4WS via VGS provides a flexible way for late binding. No change to Grid Services is required. The proposed architecture does not intend to automate the design process, but it provides a scheme to alleviate the gap between Grid Services and BPEL4WS. At run time, the major steps involved in the interactions between these components are as follows:

1. A BPEL4WS script contains several participating VGS and the workflow. The user initiates the client to start the BPEL4WS engine and run the script. The operational WSDL in VGS are consulted according to the workflow descriptions in BPEL4WS.
2. The BPEL engine then invokes the Proxy Grid Service implementation. The Proxy Grid Service (represented by operational WSDL) triggers 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.
3. When a Grid service instance is successfully created, the *startGService* operator obtains a reference and stores it in the pre-defined public Grid Service instance attribute as a global variable. Therefore, it is visible to the whole instance.
4. When BPEL4WS wishes to call individual Grid Service operators it calls a BPEL4WS engine, such as BPWS4J or Collaxa, to activate VGS stored in the service container.
5. The VGS then uses the Grid Service Reference to tell the grid service container to invoke the corresponding Grid Services. The grid service replies with the results to the client making the request. The client resides in the Proxy Grid Service which in turn is contained in the VGS.
6. The Proxy Grid Service associated with its output operator passes the response to its VGS which invokes the corresponding output method. The BPEL4WS engine can obtain the result and pass it to the next service.

A number of tools were used to implement the proposed architecture, namely, GT3, JAXRPC, JWSDL, BPWS4J, and Java native method. GT3 provides a platform for enabling Grid Services, JAXRPC supports Web Services, the modification of WSDL is through JWSDL, and BPWS4J is the BPEL4WS engine. Java native method provides the communication interface between the applications and JAXRPC when the applications only support C based APIs.

7. Case study and expermential results

In order to evaluate the feasibility and effectivenes of the proposed system we have implemented a case study, involving impacting a tube of steel into a rigid wall with a given mass and velocity. The model is decomposed into separate domains so that calculations can be parallelised. The domains are represented by the boxes labeled A–D, shown in Fig. 4.

Each processor is responsible for solving a specific domain and communication is required for information across domain boundary conditions for each timestep. For each domain the contact forces, constraints and update node positions are

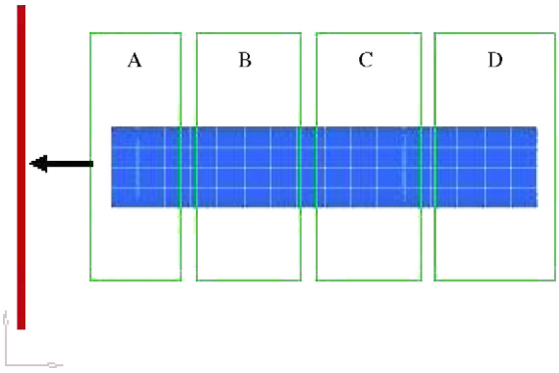

Fig. 4. An impact simulation calculating nodal displacements.

calculated in parallel. In our example, B and C require two inputs and outputs which are derived from their neighbouring reaction forces. A has the first impact and passes the force to B. D receives the force from C and bounces it back to C. Displacement simulation software is used and located at four different processors (nodes) in the Grid. There are four VGS and Grid Services being designed to accommodate simulation program. The interface of VGS to the simulation program is also included via Javabeans and Java native methods, since the simulation progam was implemented in C. The software has domain boundary information and maintains it's own state. The interaction flow among the VGSs is specified in the BPEL4WS. A snipet of BPEL4WS for modelling the interactions is shown in Fig. 5.

The relationship between the nodes are defined as partners which are specified in WSDL. Variables are used to hold data from one Grid Service and pass it to another. The *start* operation receives the user request and initiates the whole process. It takes one input object, *impact1*, containing information about mass, velocity and timestep. Variables *impact1* and *impact2* are the same as the *portType* but used for different purposes. The generated displacement of nodes is stored in the system as internal states.

The A-Impact activity is invoked to simulate the impact, and the generated results are passed to the B-Impact process. The C-Impact activity only starts when the B-Impact activity produces results. The D-Impact is the last activity, and waits for input from the C-Impact activity. The *flow* is used to allow activities to run concurrently. Modelling the dependency of the concurrent activities is the *link*.

We use two key criteria in evaluating the proposed system: *consistency* and *efficiency*. To examine consistency, we perform two experiements in different environments. First, we run the case study on a standalone machine with the original single system. Second, we run a modified system using the proposed system in a distributed environment. The force used is approximately 21 kN a beam, which has a mass of 25 kg. The experiments demonstrate the consistency of the proposed system by producing the same approximated result as the standalone version in terms of disposition of nodes in the tube.

To examine system efficiency four different scenarios were configured with different numbers of machines (each machine having 2.66 GHz CPU and 1Gb RAM). Fig. 6 shows the

```
<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>
```

relation between processing time and the number of machines. It can be seen that process time shortens as the number of machines increases. On a standone machine, the processing time is approximately 45 s for a job of this order. When the system runs on two machines the job is completed in 24.75 s, and when distributed over four different PCs it takes about 15.04 s. Our experiments show that the efficiency of the system does not increase linearly with the number of machines, due to communication and synchronisation overheads. The proposed system is, however, more efficient than the standalone version.

8. Related work and discussion

In this paper, we have investigated coordinating Grid Services using BPEL4WS. Our approach differs from other research by developing a flexible Grid Services coordination mechanism for cooperative design, which utilises the complementary functionality of Grid Services and BPEL4WS.

Various approaches have been proposed for coordinating and composing Grid Services or Web Services. For instance, Ref. [8] proposes a framework for automatically discovering and composing Grid Services using Semantic Web technologies. This framework is claimed to support dynamic workflow composition, and to enable users to share workflows of different granularities. However, this framework does not take into account the issue of Grid Service coordination with BPEL4WS.

Existing research mainly employs BPEL4WS to coordinate and compose Web Services. Hull and Su [9] report on the current manual and automated composition of Web Services. The authors analyse various modelling and compositional aspects of Web Services. The analysis is based on classifying current research and standards along various dimensions including message passing, action processing, and behaviour modelling. Another useful survey is conducted in Ref. [10], which illustrates various existing methods of Web Services composition within the context of *workflow composition* and *AI planning*. The composition process is illustrated in terms of: Web Services presentation, translation of Web Services languages, the main composition process, evaluation and the execution of composite Web Services. Refs. [9,10] identify that a fully automated and precise composition of Web Services has not been achieved through the existing techniques.

An agent-based approach is presented in Ref. [11] and the LARKS language is proposed to enable service matchmaking among Internet agents. LARKS supports both syntactic and semantic matchmaking of services. This matchmaking is based on parameters (or filters) including context matching, profile comparison, similarity matching and constraint matching. LARKS resolve service mismatch to some extent but is unable to ensure full matchmaking of heterogeneous services. Ref. [12] proposes an approach to automatically compose Web Services transactional workflow ontology. Such ontology is used to describe Web Services workflows such that agents can automatically find a composed workflow of Web Services. Moreover, a Case-Based Reasoning technique is applied to discover and compose different Web Services [13], allowing proactive and reactive composition. The claimed advantages of this approach are reduced composition cost, service collaboration and client satisfaction, and efficient service discovery and composition. Despite these advantages this approach fails to achieve precise composition of Web Services.

The above approaches incorporate various technologies to implement the composition process of Web Services, such as BPEL4WS, BMPL, OWL-S, and so on. The industry proposed Business Process Modelling Language (BPML) aims at modelling business data through a meta-language [14], but as yet there is no supported run-time environment. The Semantic Web community has proposed OWL-S [15] for describing the semantics of Web Services and composition mechanisms. However, there is no sophisticated engine like BPWS4J or Collaxa to support the OWL-S specification. Ref. [16] defines the semantics of Web Services via OWL-S and

```
<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>
```

Fig. 5. A snipet of BLEP4WS for design coordination.

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 mechanism, such as GoLog [17], to compose Web services.

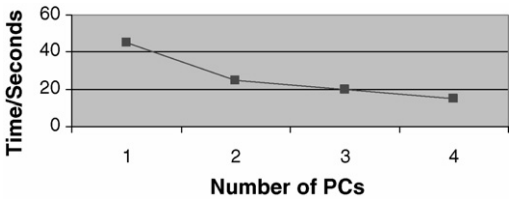

Fig. 6. Processing time with different PCs.

Based on our analysis of current research we observe that current approaches do not focus on Grid Service coordination with BPEL4WS. This is mainly due to the issues associated with using BPEL4WS in Grid Services. Our approach aimed at tackling those issues so as to ensure the coordination of Grid Services with BPEL4WS.

9. Conclusion

Large and complex engineering designs (e.g. in the automotive industry) often require significant computational resources and involve complicated design activities. Increasing utilisation of existing idle resources is the key to improving the efficiency and effectiveness of the design. The improvement of computational efficiency without a certain degree of relability is not a satisfactory solution. Ensuring consistency of design activities through a coordination protocol is important. In this paper, we have proposed 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 ensuring that the product model is consistent. The proposed architcture overcomes the incompatibility between BPEL4WS and Grid services by introducing the VGS, which is a web service, to support their seamless integration. The simplified case study adopted from automotive design demonstrates the feasibility and effectiveness of the proposed system.

The BPEL4WS is a standard scripting language requiring a prescribed workflow and known Web Services before the engine can be used. It can model static workflow, but it lacks flexibility and ability of modelling dynamic behaviours. A system being able to dynamically generate and change the workflow according to run-time awareness is desirable, since the design often requires flexible interactions, dynamic coordination mechanism, and intensive communication among design teams [18]. We are currently planning larger scale experiments involving a number of design teams and software systems, with the introduction of intelligent agent technology. The release of GT4 (an WS-Resource Framework implementation) is immement. Further investigation into GT4, which aims to provide coherent interfaces with standard Web Service technologies, is required for future development.

Acknowledgements

We would like to thank Rob Mahon from Jaguar Cars Ltd. for providing us with case study and valueable information. We also would like to acknowledge the technical assistances received from C.-J. Chen. We also would like to express our gratitudes to the anonymous reviewers for their valuable comments.

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] K.-M. Chao, M. Younas, N. Griffiths, I. Awan, R. Anane, Analysis of Grid Service composition with BPEL4WS, in: Conference Proceedings of 18th Advanced Information and Network Applications, IEEE CS, 2004.
[5] Service-Oriented Architecture (SOA) Definition: http://www.servicearchitecture.com/web-services/articles/serviceoriented_architecture_soa_definition.html.
[6] Globus Toolkit 3, http://www-unix.globus.org/toolkit/documentation.html.
[7] J.M. Vidal, P. Buhler, C. Stahl, Multiagent systems with workflows, IEEE Internet Computing January/February (2004) 76–82.
[8] S. Majithia, D.W. Walker, W.A. Gray, Automated composition of semantic Grid Services, in: Proceedings of the UK e-Science All Hands Meeting (AHM), 31st August–3rd September, Nottingham, UK, 2004.
[9] R. Hull, J. Su, Tools for design of composite Web Services Tutorial: ACM SIGMOD, June 2004, Paris, France.
[10] J. Rao, X. Su, A survey of automated Web Service composition methods, in: First International Workshop on Semantic Web Services and Web Process Composition, San Diego, CA, USA, July, 2004.
[11] K. Sycara, S. Widoff, LARKS: dynamic matchmaking among heterogeneous software agents in cyberspace, Autonomous and Multi-Agent Systems, 5, Kluwer Academic Publishers, 2002 173–203.
[12] J. Korhonen, L. Pajunen, J. Puustjärvi, Automatic composition of Web Service workflows using a semantic agent, in: IEEE/WIC International Conference on Web Intelligence (WI'03), October, Halifax, Canada, 2003.
[13] B. Limthanmaphon, Y. Zhang, Web Service composition with case-based reasoning, in: 14th Australasian Database Conference (ADC 2003), Adelaide, South Australia, February, (2003), pp. 201–208.
[14] BMPI.org, Business Process Modelling Language Specifications, http://www.bpmi.org/bpml.esp.
[15] 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, September 2003.
[16] S. McIlraith, T. Son, Adapting Golog for composition of semantic Web Services, in: Conference Proceedings on Knowledge Representation and Reasoning, April, 2002.
[17] E. Sirin, J. Hendler, B. Parsia, Semi-automatic composition of Web Services using semantic descriptions, in: Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, 2002.
[18] W.D. Li, J.Y.H. Fuh, Y.S. Wang, Collaborative computer-aided design—research and development trend, in: Proceedings of CAD'04 Conference, May 24–29, Pattaya, Thailand, 2004.

Kuo-Ming Chao received the MSc degree and the PhD degree in computer science from Sunderland University UK, in 1993 and 1997. He was a research associate in the Engineering Design Centre at University of Newcastle-upon-Tyne in 1997–2000. In late 2000, he joined the School of Mathematical and Information Sciences at Coventry University as a senior lecturer. His research interest includes intelligent agent technologies, game theory, engineering design, supply chain management, grid computing and web services.

Muhammad Younas is a senior lecturer in computer science in the Department of Computing, Oxford Brookes University, Oxford, UK. He received his PhD degree in computer science from the University of Sheffield, UK. His research interests include Web and database technologies, transaction processing, agent technology, and mobile computing. He has published more than 40 research papers in international journals, conferences, and workshops. He has also edited three books. He is the guest editor for various international journals. He has been involved in the steering, organizing and program committees on a number of international conferences and workshops.

Nathan Griffiths is a lecturer in computer science at the University of Warwick, Coventry, UK. He received his BSc and PhD in computer science from the University of Warwick in 1996 and 2000. Prior to his current post he ran a software engineering company. His research interests are distributed artificial intelligence, multi-agent systems and Grid computing. Recent work has involved the application of coopertive agents to Grid and peer-to-peer computing.