Koon Leai Larry Tan and Kenneth J. Turner. Orchestrating Grid Services using BPEL and Globus Toolkit 4.
In Madjid Merabti, Rubem Pereira, Carol Oliver and OmarAbuelma'atti, editors, Proc. 7th PGNet Symposium, pages 31-36,
ISBN 1-902560-13-9, School of Computing, Liverpool John Moores University, Liverpool, UK, June 2006.

# Orchestrating Grid Services using BPEL and Globus Toolkit 4

Koon Leai Larry Tan, Kenneth J. Turner, *University of Stirling*

***Abstract*—Grid services exploit an emerging distributed computing technology that offers possibilities for distributed resource sharing and collaboration. The standards for WSRF (Web Service Resource Framework) have allowed grid services to converge more closely on web services. Composing web services has attracted significant effort and commercial interest. This has resulted in BPEL (Business Process Execution Logic) as a standard way of orchestrating web services. Because of the similarities with web services, there is a natural question of whether grid services can be orchestrated in like manner. It is explained how CRESS (Chisel Representation Employing Systematic Specification) has been extended to describe grid service composition. It will be seen how BPEL has been adapted for this purpose, using ActiveBPEL as the orchestration engine and Globus Toolkit 4 as the grid service container. The problems arising with orchestrating grid services are discussed, along with possible workarounds.**

***Index Terms*— BPEL (Business Process Execution Logic), CRESS (Chisel Representation Employing Systematic Specification), Grid Service, Service Orchestration, WSRF (Web Service Resource Framework)**

## I. Introduction

THIS paper deals with several aspects: distributed computing, service orchestration and occupational data in social science. Grid computing has recently emerged as the leading form of distributed computing that paves a way to share resources such as computational power, data storage and services amongst different organizations. Based on the service-oriented architecture, it is then possible to combine existing grid services and hence derive new and complex services. The composition of services is known as service orchestration. Many social scientists often collaborate and perform analyses on large datasets such as occupational data surveys. There has not been much emphasis on creating infrastructures to improve data access, sharing and collaboration in a distributed manner.

The paper discusses the investigations conducted to achieve orchestration of a collection of grid services using Globus Toolkit 4 and ActiveBPEL.

Manuscript received May 2, 2006.

K. L. L. Tan is with the University of Stirling, Stirling FK9 4LA SCOTLAND (phone: +44 1786 467421; fax: +44 1786 464551; e-mail: klt@cs.stir.ac.uk).

Kenneth J. Turner, is with University of Stirling, Stirling FK9 4LA SCOTLAND (e-mail: kjt@cs.stir.ac.uk).

### A. Grid Computing

Grid computing is analogous to an electrical power grid. As electric power resources are linked together into a universal grid and provided with standard sockets, so computational resources can be linked into a computational grid and accessed via standard protocols. A computational grid is defined as a system that coordinates resources that are not subject to centralized control, using standard protocols and interfaces to deliver non-trivial qualities of service [1]. Grid computing offers a number of distinctive advantages that include:

- resource virtualization, where standard interfaces are used to access resources in a heterogeneous environment
- creation of virtual communities that can span multiple organizations for collaboration and resource sharing
- resource discovery, whereby queries can be made to locate resources to work with
- resource management, whereby resource owners can impose policies on different users in the virtual organization
- security, including flexible mechanisms for delegating credentials to third parties to act on behalf of the user
- single sign-on, whereby a user once authenticated can access authorized resources within the virtual organization
- distributed and parallel computing.

OGSA (Open Grid Services Architecture [2]) is being developed by the GGF (Global Grid Forum [3]) to define a common, standard and open architecture for grid-based applications. The goal of OGSA is to standardize the services that are commonly required in a grid system (job management services, resource management services, security services, etc.) by specifying a set of standard interfaces for these services.

Web services were chosen as the underlying technology for grid services because they are open-standard and do not require particular platforms or language implementations. However OGSA needs the underlying middleware to be able to manage state. This is a requirement that web services have not defined a standard for, though in principle it can be achieved. During early development work, OGSI (Open Grid Services Infrastructure) was implemented to meet the requirement of "stateful" services. For this reason, grid

services were incompatible with web services although in principle using the same technology.

As grid computing evolved, WSRF (Web Services Resource Framework) replaced OGSI as a specification that is compatible with web service architecture. It provides the stateful services that OGSA requires. WSRF 1.2 has recently been accepted as an OASIS standard [4]. GT4 (Globus Toolkit 4) is one of the available software toolkits that implements WSRF. It was developed by the Globus Alliance [5] and is widely used to create grid services.

### B. Service Orchestration

The trend of distributed computing is towards a service-oriented architecture that represents computing functions as services. This opens up the possibilities create new services by combining existing services. As a result there is reusability of current resources, and new operations can come into effect quicker.

Service orchestration (or composition) has attracted significant interests from both industrial and academic organizations. This is achieved by a defining 'business process' that captures the logic of how individual services are combined.

In the context of web services, a major advance came from the development of BPEL4WS (Business Process Execution Logic for Web Services) as a result of the combined efforts of many companies. This specification is being standardized as WS-BPEL (Web Service Business Process Execution Language), which is now established as the way to compose web services. Though compatible with web service architecture, limited attention has been given to composing grid services using BPEL. This paper is an effort to use the ActiveBPEL execution environment to orchestrate grid services deployed with GT4.

### C. Occupational Data

Census data is often used in many social analyses. The results help to identify trends in society and provide valuable inputs to social planning and policy making. Many of the analyses involve data with occupational information variables which then are required to be linked to occupation classifications. There are numerous occupational classifications that have their distinct characteristics and advantages targeted at certain form of analysis.

Datasets in social science are often incompatible (differing in file and content formats) and lack good documentation. They can be very large and subject to tight security measures for access. There are no current standards to govern formats, data access and sharing. Analyses performed on social science datasets can be computationally intensive. All these factors contribute to the difficulties of achieving effective and productive collaborations amongst social science researchers.

Grid computing offers features that are very beneficial in occupational data analysis. The authors are working on the GEODE project (Grid-Enabled Occupational Data Environment, www.geode.stir.ac.uk). GEODE is creating a grid environment to specifically address the challenges faced by social scientists using occupational data. GEODE has been the source of the research challenges addressed in this paper. Occupationally-related services are used as illustrative examples.

### D. CRESS

CRESS (Chisel Representation Employing Structured Specification) was developed as a general-purpose graphical notation for services. Essentially, CRESS describes the flow of actions in a service. It thus lends itself to describing flows that combine grid services. CRESS has been used to specify and analyze voice services from the Intelligent Network, Internet Telephony, Interactive Voice Response, and web services [7]. For the work reported here, CRESS was extended to deal with composition of grid services.

Service descriptions in CRESS are graphical and accessible to non-specialists. A major advantage of CRESS descriptions is that they are automatically translated into formal languages for analysis, as well as into implementation languages for deployment. CRESS offers benefits of comprehensibility, portability, rigorous analysis and automated implementation.

CRESS is extensible, with plug-in modules for each application domain and each target language. Although support for web services had already been developed, it has been necessary to extend this significantly for use with grid services. In addition, grid services have specialized characteristics that require corresponding support in CRESS.

### E. Related Work

There are several tools available for describing web service composition in the style of BPEL. CRESS differs in being a multi-purpose approach that works with many different kinds of services and with many different target languages.

JOpera [6] defines a visual notation for composing grid services and provides a kernel to run and monitor the services in operations. However it does not use a standard language such as BPEL for service composition. CRESS translates descriptions of grid service composition into BPEL and performs rigorous analysis prior to automated deployment.

Several efforts are using BPEL as the means of composing grid services [8,9]. [8] differs in that the focus is on programming solutions. [9] is different in that it manipulates message headers to achieve communication between a grid service and its associated resources (known as WS-Resource, or Web Service Resource).

The current paper is complementary to [10], which is focused on the formal aspects of grid service composition.

## II. Describing Composite Grid Services with CRESS

Figure 1 shows the subset of CRESS constructs needed in this paper for grid services.

### A. CRESS Notation for Grid Services

External services are considered to be *partners*. They offer their services at *ports* where operations may be performed.

| CRESS | Meaning |
|---|---|
| **Catch** *fault* | A handler for the specified fault. A fault with name and value requires a matching Catch name and variable type. A fault with a value only requires a matching Catch variable type. A fault is considered by the current scope and progressively higher-level scopes until a matching handler is found. |
| **Compensate** *scope*? | Called after a fault to undo work. Giving no scope means compensation handlers execute in reverse order of being enabled. |
| **Compensate** | A handler that defines how to undo work after a fault. A compensation handler is enabled only once the corresponding activity completes successfully. When executed, it expects to see the same process state as when it was enabled. |
| **Empty** | No action, used as a place-holder. |
| **Fork** *strictness?* | Used to introduce parallel paths; further forks may be nested to any depth. Normally, failure to complete parallel paths as expected leads to a fault. This is strict parallelism (*strict*, the default). Matched by **Join**. |
| **Join** *condition*? | Ends parallel paths. An explicit join condition may be defined over the termination status of parallel activities. This gives the node numbers of immediately prior activities, e.g. 1&&2 means these (and the prior ones) must succeed. |
| **Invoke** *operation output* (*input faults**)? | An asynchronous (one-way) invocation for output only, or a synchronous (two-way) invocation for output-input with a partner service. Potential faults are declared statically, though their occurrence is dynamic. |
| **Receive** *operation input* | Typically used at the start to receive a request for service. An initial Receive creates a new instance, usually matching a Reply for the same operation. |
| **Reply** *operation output* \| *fault* | Typically used at the end to provide an output response. Alternatively, a fault may be thrown. |
| **Terminate** | Ends a business process abruptly. |
| **While** *condition* | Loops as long as the condition is true. |

Invoking a service may give rise to a *fault*.

A CRESS diagram shows the flow among activities, drawn as ellipses. Each activity has a number, an action and some parameters. Arcs between ellipses shown the flow of behaviour. Note that CRESS defines flows and not a state machine; state is implicit.

Normally a branch means an alternative choice, but following a fork activity it means a parallel path. An arc may be labelled with a value guard or an event guard to control whether it is traversed. If a value guard holds, behaviour may follow that path. An event guard defines a possible path that is enabled only once the corresponding event occurs. Activities and guards may have associated assignments. These are prefixed by '/' and written in the form *variable* <− *value*.

In CRESS, operation names have the form *partner.port.operation*. Fault names have the form *fault.variable*, the fault name or variable being optional.

A CRESS rule-box, drawn as a rounded rectangle, defines variables and subsidiary diagrams (among other things). Simple variables have types like **Natural** *n* or **String** *s*. CRESS also supports grid computing types such as **Certificate** (a digital security certificate), **Name** (a qualified name) and **Reference** (an endpoint reference that characterises a service instance and its associated resources). Though the example uses **Certificate,** grid security is not currently used as explained in the next section.

Structured types can also be defined, using '[...]' for arrays and '{...}' for records. For example, the following defines two variables *hits* and *misses*. Their type is an array of elements with type *jobCount*. This in turn is a record with string *job* and natural *count* as fields.

[ { **String** job **Natural** count } jobCount ] hits, misses

Since array elements are accessed by index rather than by element type, a typical value might be *hits[3].count*.

The CRESS descriptions are mapped naturally to relevant BPEL functions and syntax. Once analysed, the CRESS descriptions are automatically translated into BPEL/WSDL.

### *B. Occupational Data Analysis using Grid Services*

The example illustrated here typifies the kinds of services being developed for occupational analysis in the GEODE project. The scenario for the example is the following:

- data is collected in an occupational survey that records job, address, age and gender
- data is often in different formats, and has to be normalised and represented as a resource (using a converter grid service)
- data security is required (emulated in this example)
- a conditional frequency analysis (using a statistics grid service) can be performed on the on the data. An example of a condition is 'age>50'

### *C. CRESS Description of The Analyser Service*

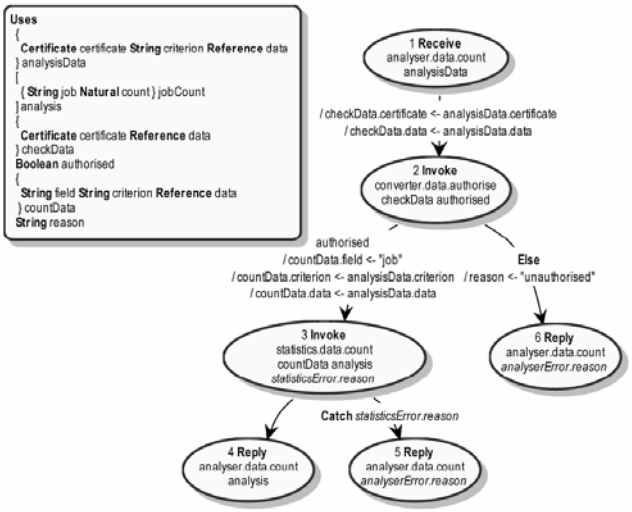


Fig. 2. CRESS Description of The Analyser Service

The analyser service is an auxiliary service that supports the main application. Its CRESS description appears in figure 2. The rule-box on the left of the figure defines types and variables. The raw data is *analysisData*: the requester's certificate, the analysis criterion, and a reference to data to be analysed. The result is an *analysis*: a list of job-count pairs.

For example, it might be determined that there are 60 plumbers, 40 electricians, etc. that meet the criterion (e.g. *gender* = *male* or *age*<20).

Initially the analyser receives a request to perform a *count* operation on the analysis data (node 1). The requester's certificate and a reference to the data are copied for checking authorisation (arc to node 2). The converter is then asked to *authorise* use of this data (node 2). If permitted, the information for the statistics service is set up. This defines the field to be counted ('job'), the analysis criterion, and a reference to the data.

The statistics operation *count* is then invoked to make a conditional frequency analysis (node 3). Normally, this will return an analysis to the requester (node 4). However if the statistics service faults (name *statisticsError*, value *reason*), this is caught (arc to node 5) and returned as a fault by the analyser (node 5).

If the converter does not authorise access, the fault reason 'unauthorised' is set (**Else** arc to node 6). The analyser then returns a fault to the requester (node 6).

### *D. CRESS Description of The Splitter Service*

The splitter offers the primary service to the user. Its CRESS description appears in figure 3. The rule-box on the left of the figure defines types and variables. The raw data is *splitData*: the requester's certificate, an analysis criterion, and a list of entries giving job, address, age and gender. The analysis yields *hits* (entries that match the given criterion) and *misses* (those that do not). The final entry in the rule-box, '/*ANALYSER*' indicates that the splitter depends on the analyser service.

Initially the splitter receives a request to perform the *count* operation on *splitData* (node 1). The converter service is invoked to normalise and store this data, returning a *store* reference to it (node 2). Now the splitter follows two parallel paths (node 3). On each path, the certificate, analysis criterion and store reference are set. The path leading to node 4 is for the given criterion (*hitData*), while that leading to node 5 is for its inverse (*missData*). A criterion is negated by prefixing it with '~'. The analyser service is executed in parallel with both sets of parameters (nodes 4 and 5), resulting in *hits* and *misses* for these paths. These paths join at node 6, where it is required that both paths have led to a successful result (4&&5).

Now the results of the two analyses have to be combined. The splitter loops through the data (node 7). For each value in *hits* and *misses*, their relative percentage is calculated (node

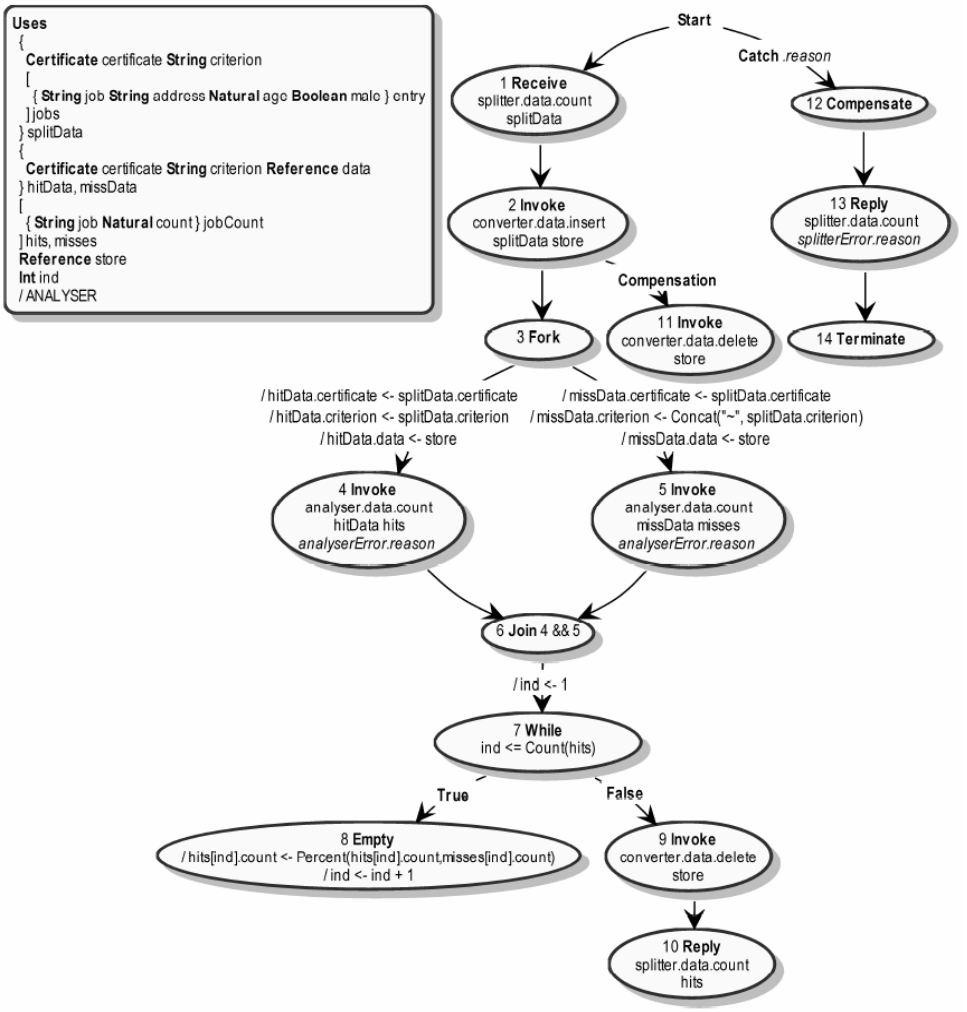


Fig. 3. CRESS Description of The Splitter Service

8): *Percent* is just a CRESS convenience function to make the intention clearer. Suppose the splitter was called to analyse the male percentage for jobs. If there are 60 male plumbers and 20 female plumbers in the survey, *hits* will be set to 60 / (60+20) or 75% as a percentage. This is repeated for every distinct job in the data.

At the end of the loop, the converted data has served its purpose and is deleted (node 9). Finally, job percentages are returned as the result of the analysis (node 10). The splitter has to take into account that its external partners may fault due to some error. For example, the converter service might fault because the data is improperly formatted. The analyser service might fault because access to the data is unauthorised or because an invalid criterion has been given. The service designer must carefully consider the consequences of faults. In particular, any changes that arose during execution of the service must be undone. In this example, any data created and stored by the converter must be deleted.

Faults caught by the splitter have a reason value (**Catch** prior to node 12). This invokes compensation to undo any actions that have been taken (node 12). The splitter then reports the fault to the requester (node 13) and terminates abruptly (node 14). Compensation may be needed after invoking an external partner, as this is where work often needs to be undone after a fault. The converter invocation to store data (node 2) has associated compensation. A fault leading to compensation will call this compensation handler (node 11). This deletes the associated data and returns.

As has been seen, the splitter service orchestrates the actions of two partner services: converter and analyser. In turn, the analyser service orchestrates the converter and statistics services. Although four services now have to cooperate, the user of the splitter service sees it as a whole. This is a major advantage, because the detailed design of the service does not then need to be visible. The major issue is whether the services work together smoothly, or whether there are interoperability problems. Even though this is a comparatively small example, it will be appreciated that there are many possibilities for error. It is very easy to make a mistake when calling a service, for example supplying a double where an integer is expected. Deadlocks are also a risk. Many more subtle problems can arise from semantic incompatibilities among the services. For these reasons, formalization and rigorous analysis are highly desirable.

## III. Using ActiveBPEL to Orchestrate Grid Services

CRESS mainly uses ActiveBPEL as the orchestration engine for web services [11]. ActiveBPEL has also been used in the work reported here for grid service orchestration.

Many issues have arisen during the practical development of orchestrated grid services. Not all issues are currently resolved, but a majority of them have been addressed to achieve the goal of grid service orchestration.

### A. Deployment of ActiveBPEL and GT4

Both ActiveBPEL and GT4 can be deployed to run within a container that uses AXIS (the Apache SOAP engine). Both are independently shipped with AXIS libraries (web services libraries) and scripts that can be deployed into an Apache Tomcat container and run immediately. In principle, ActiveBPEL and GT4 can be deployed within the same Tomcat container. In practice, this is not feasible with the current versions. GT4 uses an older version of AXIS that is incompatible with that used by ActiveBPEL. This means they cannot both be used within Tomcat if they are to start up and function correctly.

The right solution is for the ActiveBPEL and GT4 developers to converge on the same AXIS version. In the meantime, the authors deployed ActiveBPEL and GT4 in separate Tomcat containers. This is not unreasonable since BPEL can coordinate grid services running on different locations. This is very likely anyway in a realistic deployment, e.g. the occupational data analysis described in this paper.

The drawback of not being able run both within the same container is that the GSI (Grid Security Infrastructure) provided by GT4 cannot used for more comprehensive security. This is further discussed in the following sub-section

### B. Using Grid Security

GSI [13], developed by the Globus Alliance [5], provides well-defined security mechanisms for operations and collaborations in a grid. It uses public-key cryptography for authentication, authorization and message exchanges. It thus establishes trust among entities within a computing grid. A distinct feature is single sign-on whereby, through delegation of credentials, a user need authenticate to the grid only once to use authorized resources in a virtual organization that may be owned by different entities. Credential delegation is made possible by the standard for WS-SecureConversation, which will not be elaborated here.

It is desirable in this research to illustrate the use of credential delegation. However, the current GT4 implementation requires the authentication service (AuthenticationService) to be in the same container as the destination service to establish a security context. This is currently not possible as ActiveBPEL and GT4 must be deployed in separate containers due to AXIS incompatibilities. Credential delegation requires a security context to be established first, hence it is currently not used.

### C. Web Service Resource Addressing

This issue was already identified prior to commencing the research. In WSRF, identification of a WS-Resource pair (service + state) is achieved by an endpoint reference that specifies the service address and the resource key. This is done implicitly, meaning that the ports in use are already bound to a service and a resource. BPEL and Web Services do not interpret or use resource identifications, though they use endpoint references. Furthermore, in BPEL the service addresses for grid services are explicitly and statically defined at deployment time. There is no way to tell the BPEL engine (ActiveBPEL) to use endpoint reference implicitly to retrieve

a grid service's ports. There has to be a way to tell grid services (analyser, splitter) which WS-Resource are to be used. One way is to have ActiveBPEL explicitly forward the endpoint reference passed by the client to the grid services as an operation parameter. The grid services then have to identify the resource using the endpoint reference received in the operation call. In principle, this is feasible and can be achieved. GT4 uses the WS-Addressing implementation by Apache [11] for endpoint references.

Endpoint references need the WS-Addressing schema. It was discovered that the schemas used by ActiveBPEL and GT4 are incompatible, although the XML namespaces are identical. The BPEL processes (analyser, splitter) yield errors when they receive endpoint reference created by the WS-Addressing implementation in GT4. It was also found that the endpoint references generated by WS-Addressing in by GT4 do not conform to their own XML schema definition! In addition, ActiveBPEL imposes custom requirements on endpoint references received: the sub-elements must contain valid values, which is not required in GT4. The workaround is to define a corresponding WS-Addressing schema that matches the endpoint reference definition in the Apache approach (used by ActiveBPEL). Valid dummy values must be set by clients in endpoint references prior to invoking BPEL processes.

This successfully orchestrates grid services using GT4 and ActiveBPEL, although not currently using WS-Resource in the preferred manner. It is hoped that ActiveBPEL can be extended to handle endpoint reference implicitly.

## IV. Conclusion

It has been seen how BPEL can be used to support orchestration of grid services. Issues that cause obstacles for grid service orchestration have been identified, and workaround solutions have been implemented where feasible. CRESS features and support for description of composite grid services have contributed largely to the achievement of efficient grid service orchestration.

It has been demonstrated that BPEL can also orchestrate grid services in addition to web services. The issues identified can serve as useful research references for relevant and future work with respect to orchestrating grid services using BPEL.

## Acknowledgment

The authors thank their co-workers on GEODE for their insights, especially Paul Lambert (Stirling) and Richard Sinnott (Glasgow). Larry Tan was responsible for creating and realising grid service orchestration, while Ken Turner undertook the CRESS developments. Larry Tan's work on GEODE was supported by the UK Economic and Social Research Council under grant RES-149-25-1015.

## References

[1] I. Foster, "What is the Grid? A Three Point Checklist", GRIDToday, July 20, 2002.

[2] I. Foster et al.., "Grid services for distributed system integration", *Supercomputer Applications*, 35(6), 2002.

[3] Global Grid Forum (GGF). http://www.ggf.org/, Apr. 2006.

[4] OASIS. WSRF specification, http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf , Apr. 2006.

[5] The Globus Alliance. http://www.globus.org/, Apr. 2006.

[6] C. Pautasso. JOpera: An agile environment for web service composition with visual unit testing and refactoring. In *Proc. Symposium on Visual Languages and Human Centric Computing*. IEEE Press, New York, USA, Nov. 2005.

[7] K. J. Turner. Formalising web services. In F. Wang, editor, *Proc. Formal Techniques for Networked and Distributed Systems*, LNCS 3731, pages 473–488. Springer, Oct. 2005.

[8] K.-M. Chao et al.. Analysis of grid service composition with BPEL4WS. In Y. Shibata and J. Ma, editors, *Proc. 18th. Advanced Information Networking and Applications*, volume 1, pages 284–289. IEEE Press, New York, 2004.

[9] Matthew Zager, Business Process Orchestration with BPEL: BPEL supports time-critical decision making, *SOA/Web Services*, http://webservices.sys-con.com/read/155631_1.htm, Dec. 2005.

[10] K. J. Turner, K. L. L. Tan, A formal basis for orchestrating grid services, *FORTE'06*, Apr. 2006. Under review.

[11] K. J. Turner. Representing and analysing web services. *Network and Computer Applications*, Mar. 2006. In press.

[12] Apache Addressing. http://ws.apache.org/addressing/, Apr. 2006

[13] The Globus Alliance. Grid Security Infrastructure, http://www.globus.org/toolkit/docs/4.0/security/key-index.html, Apr. 2006.

[14] World Wide Web Consortium. WS-Secure Conversation, http://specs.xmlsoap.org/ws/2005/02/sc/WS-SecureConversation.pdf, Apr. 2006.