Kenneth J. Turner and Koon Leai Larry Tan. A Rigorous Approach to orchestrating Grid Services. Computer Networks, 51(15):4421-4441,

A Rigorous Approach to Orchestrating Grid Services

Kenneth J. Turner, Koon Leai Larry Tan

Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, UK

Abstract

Although conceived for web services, it is shown how BPEL (Business Process Execution Language) can be used to orchestrate a collection of grid services. This is achieved using the technique of CRESS (Communication Representation Employing Systematic Specification) to describe the composition of grid services. CRESS descriptions are automatically translated into LOTOS (Language Of Temporal Ordering Specification), allowing systematic checks for interoperability and logical errors prior to implementation. MUSTARD (Multiple-Use Scenario Test and Refusal Description) is used to validate the generated specification against use case scenarios. The same CRESS descriptions are then automatically converted into BPEL/WSDL code for practical realisation of the composed services. Grid services are executed by Globus Toolkit 4, while their orchestration is supported by the ActiveBPEL engine. The MUSTARD scenarios are used again to evaluate the implementation. The overall approach therefore supports rigorous development and automated creation of orchestrated grid services.

Key words: BPEL (Business Process Execution Language), Grid Service, LOTOS (Language Of Temporal Ordering Specification), Service Orchestration

1 Introduction

Grid computing has emerged as a leading form of distributed computing. Service-oriented architecture provides a framework for combining grid services into new ones. This paper reports on work to orchestrate grid services. In particular, it illustrates a method that allows the orchestration to be validated rigorously prior to automated implementation.

CRESS (Communication Representation Employing Structured Specification) was developed as a general-purpose graphical notation for services. Essentially, CRESS

Email addresses: kjt@cs.stir.ac.uk (Kenneth J. Turner), klt@cs.stir.ac.uk (Koon Leai Larry Tan).

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 analyse services from the Intelligent Network [23], Internet Telephony [24], Interactive Voice Response [25], Call Processing, and web services [26]. In a new development, CRESS has now been extended to grid services. Since grid services are similar to web services, but certainly not the same, this paper focuses on the advances that have been necessary for grid computing. The differences between web and grid computing are discussed in section 2.1.

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.

Grid computing has been enthusiastically adopted by many disciplines. Early uses were in the physical sciences (e.g. particle physics, astrophysics). However, grid computing has rapidly spread into other areas such as the life sciences (e.g. bioinformatics, genomics), engineering (e.g. electronics, photonics), the earth sciences (e.g. environmental science, geoscience), and medicine (e.g. pharmacology, medical imaging).

Grid computing has now moved beyond scientific disciplines into 'softer' areas such as the arts, business, economics and the humanities. As a concrete illustration of this, the paper uses an example that is taken from occupational data analysis in the field of social science. Social scientists often work with large datasets such as surveys that are evaluated with respect to occupational trends. There is a lack of good technical means for accessing and manipulating such datasets in a distributed manner. Grid services offer many advantages for this, but a rigorous approach is desirable.

2 Background

The work reported here draws on a number of different technical areas. This section provides some background information for the general reader.

2.1 Grid Computing

Grid computing is named by analogy with the electrical power grid. Just as power stations are linked into a universal electrical supply, so computational resources can be linked into a computing grid. Distributed computing is hardly a new area.

But the architecture and software technologies behind the grid have captured the attention of those who perform large-scale computing, e.g. those who work in the sciences. Grid computing offers a number of distinctive advantages that include:

- support for virtual organisations that transcend conventional boundaries, and may come together only for a particular task
- portals that provide ready access to grid-enabled resources
- single sign-on, whereby an authenticated user can make use of distributed resources such as data repositories or computational servers
- security, including flexible mechanisms for delegating credentials to third parties to act on behalf of the user
- distributed and parallel computing.

Open standards for the grid are being created by the GGF (Global Grid Forum, *www.gridforum.org*) and the OGF (Open Grid Forum, *www.ogf.org*). Grid services are governed by OGSA (Open Grid Services Architecture [10]).

Grid applications often make themselves available via services that are comparable to web services – another area of vigorous development. For a time, grid services and web services were not fully compatible. The major issue was the need for stateful services that have persistent state. A grid-specific solution to this was developed initially, but this was clearly something that web services could also benefit from.

Web and grid developers therefore cooperated on a solution that exploited existing web service standards but met the needs of grid computing for resource separation. A harmonised solution was defined by WSRF (Web Services Resource Framework [11]). This is a collection of interrelated standards such as WS-Resource and WS-ResourceProperties. WSRF is implemented by several toolsets, including Globus Toolkit version 4 (GT4 [22], *www.globus.org*) that is widely used in grid computing.

With WSRF, web and grid services now share a common technical basis. However, web services have a more commercial focus while grid services have a more scientific focus. As a result, different sets of supporting facilities are provided. OGSA provides a rich superstructure of facilities that include security and authorisation, job management and scheduling, data abstraction and indexing, and data access and transfer. The work reported in this paper focuses on the basic concepts of grid services and their orchestration. It does not address these higher-level facilities.

2.2 Service Orchestration

This paper emphasises the *composition* of grid services, not the description of *isolated* grid services. Composing services has attracted considerable industrial interest. This is achieved by defining a *business process* that captures the logic of how the individual services are combined. The term *orchestration* is also used for this.

Historically, competing solutions were developed for orchestrating services. A major advance was the multi-company specification for BPEL4WS (Business Process Execution Language for Web Services [1]), which is being standardised as WS-BPEL (Web Services Business Process Execution Language [2]). BPEL is now well established as a way of composing web services. However, its use for composing grid services has received only limited attention.

Orchestration defines the logic for combining a number of separate partner services. A nice feature of the approach is that the composition is a service in its own right. BPEL was originally defined for web services, and is currently unaware of the special characteristics of grid services. For example, resource and security aspects of grid services are not recognised. As a widely adopted standard, BPEL is nonetheless likely to find increasing use for orchestrating grid services. The work reported here demonstrates its potential.

To avoid the reader having to be expert in BPEL and WSDL, only a high-level description is given in this paper. However, the CRESS representation of orchestration is intentionally similar to BPEL. This allows a service designer to make use of CRESS with minimal training.

2.3 Occupational Analysis

Many forms of social analysis use occupation as a significant factor. For example, Governments are interested in questions like the following. Is the number of school teachers rising? How well are doctors being paid compared to firefighters? Is the gender imbalance in computing being addressed? Are better paid people moving to rural areas?

Social scientists make wide use of aggregate occupational information databases that contain summary data on particular occupational positions. Many approaches are used, for example alternative taxonomies for occupations. However the distribution and use of such classifications is rather limited in social science [16]. Aggregate occupational information datasets are often just published via web pages with informal instructions. The datasets are seldom annotated with metadata about their structure and provenance. They are seldom available via repositories, which inhibits their use by a wider community.

Social scientists often wish to link aggregate occupational information with micro-social survey datasets. The latter may be very large, and subject to strict controls on access. Moreover their analysis through statistical models is often computationally intensive. Although occupation is an important measure in many social science analyses, social scientists rarely obtain the benefits of effective resource sharing. This applies to aggregate occupational information, and also to analysis of micro-social data using occupations. The overall result is that it is often difficult for social

scientists to collaborate. This is unfortunate because many advantages could be obtained by effective sharing of data and computing resources.

The authors are working on the GEODE project (Grid-Enabled Occupational Data Environment, *www.geode.stir.ac.uk*). The premise of this project is that grid computing is an effective solution to many of the problems faced by social scientists in using occupational data. GEODE has been the source of the research challenges addressed in this paper. Occupationally-related services are therefore used here for illustration.

2.4 LOTOS

LOTOS (Language Of Temporal Ordering Specification [13]) was standardised for the specification of communications systems. It is, however, a general-purpose language that has been used in other applications such as conformance testing, embedded systems, hardware design, and safety-critical systems. LOTOS uses a process algebra to specify behaviour, and algebraic data types to specify data.

To avoid the reader having to be expert in LOTOS, only the high-level structure of generated specifications is given in this paper. A data type and its associated operations are specified by a *type*. Behaviour is described by a *process*. LOTOS processes are parameterised by *gates* at which communication takes place. Typically, behaviour is specified in a modular fashion as the composition of a number of processes. Process may synchronise on specified gates, or may be interleaved (i.e. operate independently in parallel). The whole specification has a top-level *behaviour*.

2.5 CRESS

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 specialised characteristics that require corresponding support in CRESS:

- A wider range of data types is now supported, including arrays and arbitrarily nested structured types. Specialised types have been added for dealing with grid services, such as certificates and endpoint references.
- Additional BPEL-compatible constructs have been included to make grid service orchestration more convenient.
- Support has been introduced for external partners shared amongst a number of services. Special treatment is needed to merge their descriptions from different

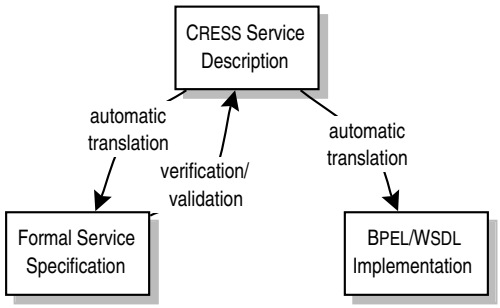

Fig. 1. A Rigorous Approach to Composing Grid Services (from [28])

services.

- External partners may need to communicate with other partners not visible to the orchestrating process. Support has been created for such 'phantom partners'.
- CRESS now has support for declaring and translating grid service resources.

As illustrated in figure 1, CRESS offers a formally-based approach for orchestrating grid services. In the context of grid computing, the steps in development are as follows:

- The desired composition of grid services is first described using CRESS. This gives a high-level overview of the service interrelationships. Because the description is graphical, it is relatively accessible even to non-specialists.
- The CRESS descriptions are then automatically translated into a formal language. In general, CRESS supports two formal approaches that have been standardised for communications systems: LOTOS and SDL (Specification and Description Language [15]), though only LOTOS is used in this paper. Obtaining a formal specification of a composite grid service is useful in its own right: it gives precise meaning to the services and their combination.
- Although CRESS creates an outline formal specification for each partner service, this defines just its interface. This is sufficient to check basic properties such as interoperability. However for a fuller check of functionality, a more realistic specification must be written manually for each partner. The automatically generated interface helps to avoid simple but common misunderstandings. The automated specification allows a rigorous analysis to be performed prior to implementation.
- A competent designer can be expected to produce a satisfactory service implementation. However, *combining* services often leads to unexpected problems. The services may not have been designed to work together, and may not interoperate properly. The issues may range from the coarse (e.g. a disagreement over the interface) to the subtle (e.g. interference due to resource competition). This is akin to the feature interaction problem in telephony, whereby independently designed features may conflict with each other. CRESS supports the rigorous evaluation of composite services. Problems may need to be corrected in either the CRESS descriptions or in the partner specifications. Several iterations may be required before the designer is satisfied that the composite grid service meets

its requirements.
- The CRESS description is then automatically translated into an implementation. The interface to each service is defined by the generated WSDL (Web Services Description Language [31]). The orchestration of partner services is defined by the generated BPEL. The partner implementations must be created independently, using their formal specifications as a guide. Finally, the scenarios used to validate the specification can be re-used to check the implementation.

Translating grid service orchestration is completely different for LOTOS and for BPEL/WSDL. The CRESS toolset has a common front-end but separate back-end compilers. In theory, a properly validated and verified specification should lead to a dependable implementation. However, practical issues such as performance might require the implementation to be tuned.

2.6 Relationship to Other Work

A number of techniques have been developed for formalising *web* services. Most approaches use finite state methods or process algebras.

As an example of finite state methods for web services, [12] gives a timed semantics for XLANG that allows web services to be checked for interoperability, and also to be implemented via transition systems. LTSA-WS (Labelled Transition System Analyzer for Web Services [9]) is a well-developed approach that allows composed web services to be described in a BPEL-like manner. Service compositions and workflow descriptions are automatically checked for safety and liveness properties. LTSA-WS models activities between a business process and its partners. It defines a translation of BPEL4WS to FSP (Finite State Processes) that is then used for verification and validation. In contrast, CRESS uses a higher-level representation for automatic generation of a formal specification (LOTOS) and of a deployable implementation (BPEL, WSDL). CRESS is also a multi-purpose approach that works with many different kinds of services and with different target languages.

[8] describes an elaboration of the original LTSA-WS work. Service compositions and workflow descriptions are described using MSCs (Message Sequence Charts [14]). These are synthesised into state transition systems and verified for safety and liveness properties. Service implementations specified with BPEL4WS are used to generate a second behavioural model (also a transition system) which is analysed in comparison to the MSC model. CRESS differs in using a more abstract and language-independent notation for services of any variety. CRESS also automatically integrates the specifications and implementations of partner services.

As an example of process algebraic methods for web services, automated translation between BPEL and LOTOS has been developed [5,7]. This has been used to specify, analyse and implement a stock management system and also negotiation

through web services. CRESS differs from this work in using more abstract descriptions that are translated into BPEL and LOTOS. CRESS descriptions are language-independent, and can thus be used to create specifications in other formal languages (e.g. SDL). CRESS also offers a graphical notation that is more comprehensible to the non-specialist. This is important since service development often involves business and marketing staff as well as technical experts.

Orchestration of web services has been well received in industry. Scientific workflow modelling has also been studied by a number of research projects. The MyGrid project has prepared an overview of these (*http://phoebus.cs.man.ac.uk/twiki/bin/view/Mygrid*). Only some of the better known workflow languages are mentioned below.

JOpera [19] was conceived mainly for orchestrating web services, though its applicability for grid services has also been investigated. JOpera claims greater flexibility and convenience than BPEL. Taverna [18] was also developed for web services, particularly for coordinating workflows in bioinformatics research. The underlying language SCUFL (Simple Conceptual Unified Flow Language) is intended to be multi-purpose, including applications in grid computing. A philosophical difference is that CRESS focuses on use of standards like BPEL rather than research languages.

[6] describes support for BPEL being undertaken for the OMII (Open Middleware Infrastructure Institute). Like the present work, this is also investigating the feasibility of using BPEL to orchestrate grid services. Support has been developed for the modelling, enactment and monitoring of business and scientific workflows. A plug-in for the ECLIPSE IDE provides design tools for workflow definition and system configuration. The OMII-BPEL environment incorporates a customised version of ActiveBPEL to address the needs of large-scale scientific workflows. In terms of implementation capability, this work has similar coverage to CRESS but has been applied to much larger problems. However, CRESS aims at a higher-level description of service composition and is not exclusively focused on grid services. In addition, CRESS offers the advantage that formal analysis of workflows can be performed using the same service descriptions that define the implementation. Unlike CRESS, OMII-BPEL does not appear to deal with WS-Addressing (Web Services Addressing [32]) for binding to grid service resources.

Formal methods have seen little use for composing *grid* services. [33] uses pi-calculus to define composition signatures for grid services. This allows precise models to be developed of grid service compositions – particularly of concurrency aspects. CRESS similarly supports formal specification of grid service orchestration, but the specifications are automatically generated and the focus is on standardised orchestration mechanisms. An interesting technical point is whether process mobility (in the sense of the pi-calculus) needs to be supported. From a careful study of BPEL, the authors concluded that this capability is not required. LOTOS is

therefore sufficiently powerful for composite grid services.

CRESS is designed for modelling composite services, but was not conceived as a workflow language. CRESS serves this role only when orchestrating grid or web services; its use in other domains is rather different. An important point is that CRESS focuses on generating code in standard languages. For service orchestration, this means BPEL/WSDL. This allows CRESS to exploit industrially relevant developments.

Several researchers have used BPEL to compose grid services. [4] discusses programmatic ways in which BPEL can support grid computing. [21] examines how extensibility mechanisms in BPEL can be used to orchestrate grid services. However, the focus of such work is pragmatic. For example, grid services may be given a web service wrapping for compatibility. (Semi-)automated methods of composing grid services have also been investigated, e.g. work on adapting ideas from the semantic web [17]. However, the formal aspects of this work are rather limited.

In summary, CRESS is distinguished in a number of important respects from other approaches:

- CRESS is a general-purpose notation for describing services and their notations. It is not restricted to one domain (e.g. web or grid services), and has in fact been proven in six different areas.
- CRESS is a compact, graphical and versatile notation. Its generality makes it extensible to new applications. Its relative simplicity allows many details of specification and implementation to be hidden. Indeed, it is possible to specify, implement and validate services without any knowledge of the underlying languages. Other service development approaches often expose the underlying details, and require more technical knowledge.
- CRESS supports automated formal specification and automated implementation, allied to rigorous validation and verification. This allows CRESS to be used for systematic service development. Few other approaches deal with both specification and implementation, and certainly not in the range of domains supported by CRESS.
- CRESS supports multiple target languages, diagram editors, supporting toolsets and platforms. Other approaches are usually tied to their own native languages and tools.
- CRESS is focused on standardised languages and standardised communications. Other approaches may follow their own approach, which can be a barrier to wider uptake. With reference to standards for service orchestration, web services have been the main focus of others. Formal approaches often leave out the more intricate aspects of orchestration, such as full support for data typing, handling of faults, and treatment of compensation. CRESS aims to provide comprehensive (though not complete) coverage of these aspects.

3 Describing Composite Grid Services with CRESS

Figure 2 shows the subset of CRESS constructs needed in this paper to orchestrate grid services; CRESS supports more than is described here. Look ahead to figures 3 and 4 for examples of CRESS.

3.1 CRESS Notation for Grid Services

A CRESS diagram shows the flow among activities, drawn as ellipses. Each activity has a number, one or more actions, 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. Activity nodes and guards may have associated assignments.

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).

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 *fieldCount*. This in turn is a record with string *field* and natural *count* as fields.

[{ **String** field **Natural** count } fieldCount] hits, misses

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

3.2 Occupational Data Analysis using Grid Services

The extended example used in this paper typifies the kinds of services being developed for occupational analysis on the GEODE project. Although it is simplified for illustrative purposes, it shows many of the key ideas behind grid services. The example also shows most of the CRESS constructs used to orchestrate services. It

CRESS	Meaning
partner.port.operation	A fully qualified operation name.
name (*.variable*)? | *.variable*	A fault with name and optional variable value, or with variable value only.
/ variable <– value	An assignment associated with a node or arc.
Catch *fault*	A handler for the specified fault. A fault name or fault value must match the **Catch** name or variable type. A fault is considered by the current 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 enabling.
Compensation	A handler that defines how to undo work after a fault. Compensation 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.

Fig. 2. CRESS Notation (? optional, * zero or more, | alternative)

makes use of grid service partners to perform commonly required tasks such as data conversion and statistical calculations. As grid services, these benefit from distribution, parallel execution, and security. These partners are combined into two orchestrated services: an analyser to convert and analyse occupational data, and a splitter to perform a conditional frequency analysis according to some criterion. The orchestration supplies the logic to create a new high-level service for occupational data analysis.

Suppose a social scientist collects data in an occupational survey that records job, address, age and gender of each person. A common requirement is to perform a conditional frequency analysis on such data. For example, it might be useful to know the percentage of people aged over 50 for each type of job. There is often a need to split the data on some criterion, e.g. the percentage of female employees or the percentage of plumbers in Scotland.

Survey data is often in many different formats. An external converter service is therefore used to convert data to a standard format. This makes use of embedded metadata describing the format. As well as this, the converter stores the converted data as a *resource* and makes it available via what is called an *endpoint reference* in grid computing. This identifies a resource that can be passed to another service for retrieval and processing.

Data security is often a major issue. The converter therefore authorises data use. This is achieved by what grid computing calls a *certificate* – an unforgeable digital document that identifies the requester. This is how *single sign-on* is achieved in a grid environment. The same mechanism also supports *delegation*, whereby a user (or application) passes authority to another to perform certain actions. The ease of securely sharing and using data also makes it feasible for *virtual organisations* to be established. These are collaborative groups of users that typically cross organisational boundaries. Virtual organisations may be set up for particular purposes, or may be long-lived.

Survey data naturally requires considerable statistical analysis. A separate statistics service is therefore used. The only statistical function needed in this paper is a frequency analysis, using a criterion like *'age>50'* or *'gender<>male'* (not male).

With these external services, an analyser service can now be defined. This orchestrates the converter and statistics services to provide a completely new service. The analyser accepts a reference to survey data and, if authorised, returns an analysis based on a single criterion.

In turn, a more complex splitter service can be defined. This uses the converter to normalise and store the data. It then calls the analyser for the given criterion and its inverse. For speed, both analyses are performed in parallel. These results are then combined as percentages satisfying the criterion.

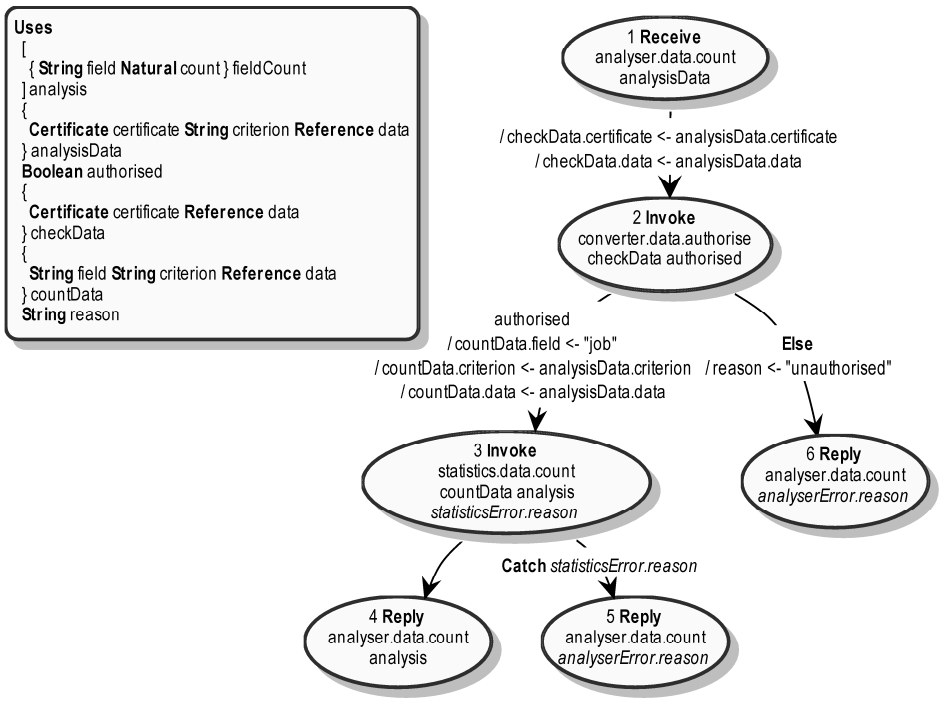

Fig. 3. CRESS Description of The Analyser Service

3.3 CRESS Description of The Analyser Service

The analyser is an auxiliary service that supports the main application. Its CRESS description appears in figure 3. The rule-box at the top-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. *'address=Stirling'* or *'∼gender=male'*).

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 (arc to node 3), 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 service can tally any field in the data, though only the *job* field is used in this example.

The statistics operation *count* is then invoked to make a conditional frequency analysis (node 3). Normally, this will lead to an analysis being returned 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 (arc to node 6), the fault reason 'unautho-

rised' is set. The analyser then returns a fault to the requester (node 6).

3.4 CRESS Description of The Splitter Service

The splitter offers the primary service to the user. Its CRESS description appears in figure 4. The rule-box at the top-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. For this reason, the splitter can also make use of the analyser's variables.

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 '$\sim$'. The analyser service is executed twice in parallel with the corresponding 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 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 $\frac{60}{60+20}$ or 75% as a percentage. This is repeated for every distinct job in the dataset, storing the percentages in *hits*.

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 if there is a fault.

Faults caught by the splitter have a *reason* value but no specific fault name (**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

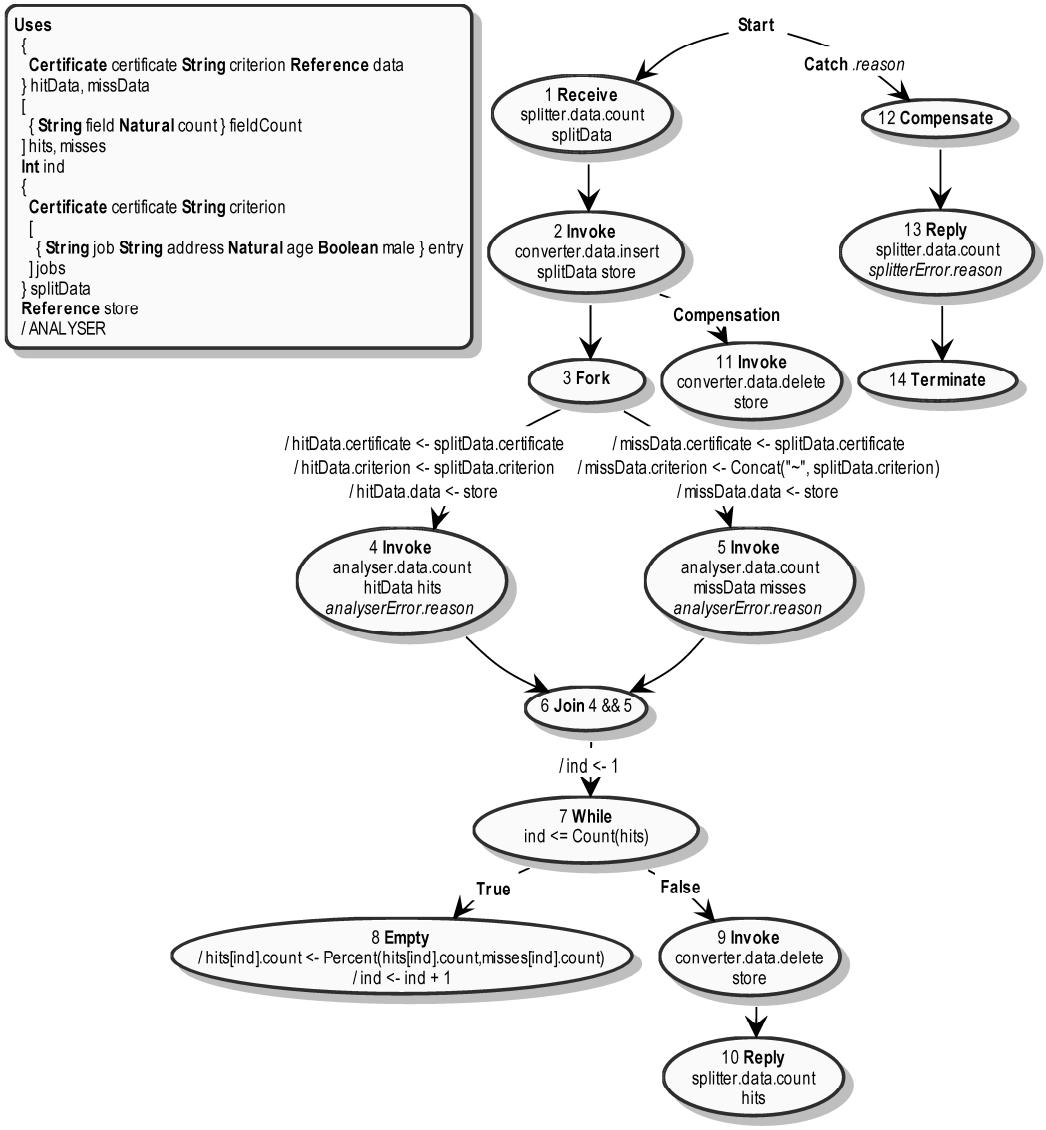

Fig. 4. CRESS Description of The Splitter service

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.

A business process may require *correlation* to relate a partner response to the invocation that triggered it. This needs some relationship between the two, typically a shared field such as a reference number. However, correlation is not required if the underlying communication mechanism (e.g. a socket connection) automatically relates responses to invocations. The splitter may simultaneously invoke two instances of the analyser (nodes 4 and 5 in figure 4), so correlation could be required. However the addition of correlation significantly complicates the description, and

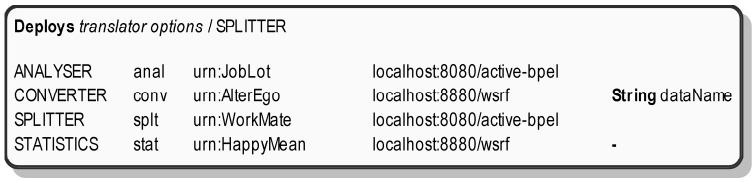

Fig. 5. CRESS Description of The Service Configuration

is not actually needed due to the nature of the specification and implementation in this case. For simplicity it has therefore been omitted.

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 floating point number 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, formalisation and rigorous analysis are highly desirable.

3.5 The CRESS Service Configuration

Now that the various services have been introduced, the CRESS configuration diagram can be shown. Figure 5 shows how the services here are described. The **Deploys** clause lists the CRESS translator options and, following '/', the services to be deployed. Although only *SPLITTER* is named, this implicitly includes all of the other services because of the inferred dependencies. The parameters of each service then follow in the configuration diagram. All services, such as *CONVERTER*, have a namespace prefix ('conv'), a namespace URI (Uniform Resource Name 'AlterEgo'), and a base URI where they are deployed ('localhost:8880/wsrf'). As can be seen, all the services were deployed on the local computer: the grid services on port 8880, and the orchestrated services on port 8080. However, they can be deployed anywhere in the network.

Grid services may have resources. Declarations of these may follow the other parameters. Only the converter here has a resource: the *dataName* that identifies the data it has stored. Every instance of the converter has a unique resource value, identified by its *resource key* in grid terminology. As a grid service, statistics might also have a resource but in this case does not have one (shown as '-' in figure 5). A composite service may also have resources. For example, if the analyser service were

Target	Fixed Code	Generated Code			Partner Code		Total
		Files	Types	Behaviour	Files	Behaviour	
LOTOS	850	1	560	360	2	190	**1960**
BPEL/WSDL	15	27	2810	390	12	790	**4010**

Fig. 6. Comparison of LOTOS and BPEL/WSDL Translations (files, lines of code)

stateful then it too would have resource declarations.

3.6 Translation of The CRESS Diagrams

The CRESS diagrams (analyser, splitter, configuration) hold all the information needed to automatically generate a LOTOS specification and a BPEL implementation. Figure 6 compares the translations of the occupational data analysis example. The figure shows non-comment lines of code for data types and behaviour, and the number of generated files:

- The fixed code is the framework common to all grid applications. This is substantial in the case of LOTOS because it contains many complex data types – 17 in total that cover numbers, strings, states, values and the like.
- The automatically generated code is shown for data types and behaviour. The LOTOS translation is a single file. The BPEL translation yields many files: one BPEL file per service, one WSDL file per service/partner, one Java file per data type, and several deployment files.
- The code for the external partners (converter, statistics) has to be manually written. The Java coding conventions for grid services require several files per partner.
- Figure 6 does not include the files used to validate the specification and the implementation.

As would be expected, the LOTOS specification is rather smaller than the corresponding BPEL/WSDL implementation. LOTOS is more noticeably compact when used with larger examples, since there is a significant overhead in LOTOS data types. These support functions on numbers, strings, etc. that are standard in any programming language. The most striking difference is in the large number of files that are required to support BPEL/WSDL.

4 Translating Grid Services to LOTOS

The translation of CRESS into LOTOS for *web* services is described in [26]. LOTOS is a specialised language and the resulting specifications are complex, so the details

are unlikely to be of interest to the general reader. The description here is therefore high level. A selection of grid service examples (including the one in this paper) has been made available for download from *www.cs.stir.ac.uk/~kjt/software/download/gs-examples.tar.gz*.

In fact, CRESS is designed so that the service designer does not *need* to know anything about the target language (LOTOS here). A specification can be generated and analysed without any knowledge of this. However, someone (perhaps an external party) must provide specifications of the partner services in LOTOS.

4.1 Specification Structure

A simple command creates a service specification automatically. Figure 7 outlines the generated specification structure for the occupational data analysis example. The fixed specification framework includes a variety of data types required for all grid services (e.g. *Array* is a generic array, *Value* is a generic value). Some of the types are very complex. For example, LOTOS does not have built-in types for numbers or strings. These, and their associated operations, have to be specified from scratch.

Partner services are often used by only one business process. In the LOTOS translation, this means they are hidden inside the definition of the corresponding processes. However, figures 3 and 4 raises two interesting problems. The converter service is used by both the analyser and the splitter, and so is shared. Translation options in the configuration diagram identify the converter as a 'merge partner' that is shared by the services.

A more subtle problem arises because the converter and the statistics services must communicate: the former stores the data that the latter must access. In a real implementation, this would be achieved by means of shared access to (say) a database. In LOTOS terms, this communication has to be made visible. This is done by defining implicit partners in the translation options: here a database. Of course, the function of this 'phantom partner' is just to store data, so it need not be specified as an actual database. Phantom partners exist at the top level of the specification.

The top-level behaviour shows the analyser and splitter operating independently in parallel because each is a service in its own right. However, they synchronise with their shared partners. The converter is explicitly shared by the analyser and the splitter, while the database is implicitly shared by all services. The statistics partner is used only by the splitter and so appears lower in the structure.

The top-level behaviour is followed by data types that are generated from the particular services in use (e.g. *Event* describes service events, *Port* describes the service ports). Other generated types reflect the specification variables (e.g. *Analysis* from

```
Specification GSSystem...                                   (* Grid Service system *)
  Library...                                                (* library type imports *)
  Type Array...                                             (* array *)
  ...
  Type Value...                                             (* value *)

  Behaviour                                                 (* overall behaviour *)
    Hide converter,database In                              (* hide internal gates *)
      DATABASE...                                           (* DATABASE partner *)
    |[database]|                                            (* synchronised with services *)
        CONVERTER...                                        (* CONVERTER partner *)
      |[converter]|                                         (* synchronised with services *)
          ANALYSER...                                       (* ANALYSER service *)
        |||                                                 (* interleaved with *)
          SPLITTER...                                       (* SPLITTER service *)

    Type Event...                                           (* event *)
    ...
    Type Port...                                            (* port name *)

    Type Analysis...                                        (* analysis array *)
    ...
    Type SplitData...                                       (* splitData record *)

    Process ANALYSER...                                     (* ANALYSER service *)
      Hide statistics In                                    (* hide internal gates *)
        STATISTICS...                                       (* statistics partner *)
      |[statistics]|                                        (* synchronised with *)
        ANALYSER_1...                                       (* ANALYSER main process *)

      Type ANALYSER_RESULT...                               (* ANALYSER result values *)
      Process ANALYSER_N...                                 (* ANALYSER processes *)

    Process CONVERTER...                                    (* CONVERTER partner *)

    Process DATABASE...                                     (* DATABASE partner *)

    Process SPLITTER...                                     (* SPLITTER service *)
      Hide analyser In                                      (* hide internal gates *)
        ANALYSER...                                         (* ANALYSER service *)
      |[analyser]|                                          (* synchronised with *)
        SPLITTER_1...                                       (* SPLITTER main process *)

      Type SPLITTER_RESULT...                               (* SPLITTER result values *)
      Process SPLITTER_N...                                 (* SPLITTER processes *)

    Process STATISTICS...                                   (* STATISTICS partner *)
```

Fig. 7. LOTOS Specification Structure

figure 3, *SplitData* from figure 4).

The process definitions for partners and services now follow. The analyser and splitter specifications are generated automatically from the CRESS diagrams. The converter, database and statistics partners are normally pre-specified by hand; their definitions are automatically included in the complete specification.

The analyser specification includes a local statistics instance, while the splitter specification includes a local analyser instance. A number of analyser and splitter processes are generated, numbered *N* according to the corresponding diagram node.

4.2 Data

CRESS has been extended for grid services to handle arbitrarily complex structured types, including arrays and records. As part of the translation, these types are flattened so that only a single level of structuring is specified by a type. The flattening process also identifies common types that are shared among variables.

Apart from service-specific types, CRESS supports standard XML types and the specialised types needed for grid services. A certificate is an XML document, and is treated as a string. A qualified name is also a string. Endpoint references are more complex. In grid terms, they are XML documents with the URI of a service and a unique resource key. This could be problematic for a LOTOS translation because an endpoint reference can be used by any service to access the corresponding resource. From a formal point of view, this would seem to need mobile processes in the style of pi-calculus. However, the usage of endpoint references in BPEL means that the service being invoked is always explicit. An endpoint reference can therefore be translated as an opaque value that uniquely identifies the resource; the current translation uses a natural number.

Some additional XPATH functions are now supported by the CRESS translators (e.g. *count*, *position*, *percent*) due to the particular requirements of grid services.

4.3 Behaviour

An activity sequence in a CRESS diagram becomes a sequence in LOTOS. However, parts of a CRESS diagram often have to be translated as separate LOTOS processes. This happens, for example, when part of a diagram can be reached by different paths or is invoked as an event handler.

Although not visible in the outline of figure 7, processes are parameterised by the

variables introduced in the service definitions (e.g. *analysis*, *analysisData* from figure 3). CRESS also adds internal parameters to keep track of the business process state and the termination status of partner calls. Grid service resources are represented as process parameters too.

A BPEL activity results in successful termination or failure. LOTOS behaviours therefore exit with true or false. For simple behaviours, this is the only result of a process. A fuller description of state is required when dealing with compensation handling or with concurrency. Forks and compensation, as used by the splitter, are particularly complex to translate into LOTOS. Essentially, they require the LOTOS processes to carry state along parallel paths and to record the state history for compensation. (This is what the *ANALYSER_RESULT* and *SPLITTER_RESULT* types in figure 7 support.) For space reasons, the reader is referred to [26] for details of the translation.

For each grid service, the CRESS translator statically discovers where event handlers are defined and the scopes where these apply (global, or associated with an **Invoke**). An event dispatcher process is then generated with reference to these handlers according to their scopes. If a fault handler does not exist for the current scope, the global handler (if any) is tried. Faults have to be matched against handlers in a particular order: **Catch** with a matching fault name, **Catch** with a matching fault name and value type, **Catch** with a matching value type, **CatchAll**.

A **Compensate** action, a **Throw** action or a fault invokes the event dispatcher with information about the scope, event name and value type. The fault handling rules of BPEL require fault values to be coerced into a single LOTOS type *Value*. This is needed so that the kind of value can be matched against **Catch**. For example, a fault handler expecting a string must check if the value is indeed a string; another handler for the same fault name might deal with floating point fault values.

4.4 Rigorous Analysis of Grid Services

Translating orchestrated grid services into LOTOS is valuable in its own right. For example, a number of errors, omissions and ambiguities were found in the the BPEL standards (mainly in complex areas such as event handling and data handling).

In implementation practice, grid services are manually debugged. The generated LOTOS can, of course, be manually simulated as well. However, a major advantage of the formalisation is that it supports a wide variety of rigorous analyses. An important issue in orchestrating grid services is to ensure their interoperability. Problems arise from simple misinterpretation of interfaces or from more subtle semantic incompatibility. Such problems often lead to deadlock in LOTOS terms.

In principle, the LOTOS specification can be formally verified. For example, the

splitter service must not fault (a safety property), and an invocation of it must eventually receive a response (a liveness property). Unfortunately the complex data types and infinite data values make model checking rather impracticable. For this reason, the authors favour the use of rigorous validation (i.e. testing) instead of verification (e.g. model checking).

MUSTARD (Multiple-Use Scenario Test and Refusal Description [27]) has been developed as a language-independent and tool-independent approach for expressing use case scenarios. These are automatically translated into the chosen language (LOTOS here) and automatically validated against the specification. This is useful for initial validation of a specification, and also for later regression testing following a change in the service description.

Validation checks only some aspects of a specification and so is far from complete. However, carefully chosen scenarios will exercise the most critical characteristics. In fact, the same criticism could be made of model checking. At best, it can verify only the properties it is given for a specification. The major difference is that validation is practicable: a specification is validated by MUSTARD in a matter of minutes, whereas verification is typically much lengthier or cannot be performed in practice. Another advantage of validation is that the same scenarios can be used to check both the specification and the implementation.

Scenario-based validation is also a useful way of checking for interference among supposedly independent services. This is the well-known feature interaction problem (e.g. [3]) that has been extensively studied in telephony. This has received little attention in other domains, though [30] discusses feature interactions in web services. Interactions may arise for technical reasons (e.g. conflicting services are activated by the same input) or for resource reasons (e.g. services have a shared resource or external partner). One way of interpreting service interaction is that a service behaves differently in the presence of some other service.

Grid services are formally validated by MUSTARD scenarios that check critical aspects of their behaviour. It then becomes possible to check services in isolation as well as in combination. This can effectively and efficiently detect service interactions, though failure to detect interactions does not mean the services are interaction-free.

MUSTARD supports scenarios with sequences, alternatives, non-determinism, concurrency and service dependencies. In addition, both acceptance tests and refusal tests may be formulated. There is insufficient space here to explain MUSTARD fully, so reference to [27] and a few examples must suffice.

A MUSTARD scenario names a test and a behaviour that must succeed. Scenarios for occupational data analysis provide data to be split on some criterion; a digital certificate (not shown here) is also defined. The data includes entries about individuals that state their job, address, age and gender (type *splitData* in figure 4). The

splitter normally produces an analysis as a list of job-percentage pairs (type *hits* or *misses* in figure 4). For convenience the split data and the expected analysis have been defined separately, though these may also be used directly as parameters.

The example in figure 8 provides data on one plumber aged 23. (Strings in MUSTARD are preceded by a single quote.) Analysing this data for plumbers under 20 should determine that 0% of plumbers satisfy this criterion.

MUSTARD can also check that faults are reported when expected. In the scenario of figure 9, the analysis criterion 'Age!30' is wrong and must yield a *splitterError* fault with reason 'malformed'. ('<>' in this example is an empty list.)

It is also desirable to check scenarios that require concurrent execution. Figure 10 shows two instances of the splitter being called concurrently for different datasets (not shown) and different criteria.

All the scenarios so far are acceptance tests: the specification must perform as described. It is also important to include refusal tests that check the specification does not have extra behaviour. A refusal sequence in MUSTARD starts with legitimate behaviour and ends with behaviour that must not happen. (Both the initial behaviour and the refused behaviour may be composite.) Figure 11 shows a simple example. Since a criterion may be arbitrarily negated, '$\sim\sim\sim$Age$>$30' is in fact valid. The specification must therefore not throw a 'malformed' fault.

5 Translating Web Services to BPEL

The translation of CRESS into BPEL/WSDL for grid services is outlined in [29]. BPEL and WSDL are specialised languages and the resulting implementations are complex, so they are unlikely to be of interest to the general reader. The description here is therefore high level. However, a selection of grid service examples (including the one in this paper) has been made available for download from *www.cs.stir.ac.uk/~kjt/software/download/gs-examples.tar.gz*.

In fact, CRESS is designed so that the service designer does not *need* to know anything about the target languages (BPEL/WSDL here). An implementation can be generated without any knowledge of these. However, someone (perhaps an external party) must provide implementations of the partner services in Java. It is also worth emphasising that the *same* CRESS descriptions are translated into LOTOS and into BPEL/WSDL.

```
define(No_Plumbers_Data,                          % individual data to check
  list(                                           % list of values
    Entry('Plumber,'1 Rose Lane Stirling,23,True)))   % sample individual

define(No_Plumbers_Analysis,                      % analysis expected of individuals
  list(                                           % list of values
    FieldCount('Plumber,0)))                      % 0 percent are plumbers

test(No_Plumbers,                                 % no plumbers scenario
  succeed(                                        % test must succeed
    send(splitter.data.count,                     % ask for job percentages
      SplitData(Certificate,'Age<20,No_Plumbers_Data)),   % get under 20s
    read(splitter.data.count,No_Plumbers_Analysis)))      % expect this analysis
```

Fig. 8. Scenario for A Simple Dataset

```
test(Invalid_Query,                               % invalid query scenario
  succeed(                                        % test must succeed
    send(splitter.data.count,                     % ask for job percentages
      SplitData(Certificate,'Age!30,<>)),         % invalid criterion
    read(splitter.data.count,SplitterError,'Malformed)))   % expect this fault
```

Fig. 9. Scenario yielding A Fault

```
test(Concurrent,                                  % concurrency scenario
  succeed(                                        % test must succeed
    interleave(                                   % interleaved sequences
      sequence(                                   % sequence of behaviour
        send(splitter.data.count,                 % ask for job percentages
          SplitData(Certificate,'Age<20,Concurrent_Data1)),   % get under 20s
        read(splitter.data.count,Concurrent_Analysis1)),      % expect this analysis
      sequence(                                   % sequence of behaviour
        send(splitter.data.count,                 % ask for job percentages
          SplitData(Certificate,'Gender=Female,Concurrent_Data2)),   % get females
        read(splitter.data.count,Concurrent_Analysis2)))))    % expect this analysis
```

Fig. 10. Scenario with Concurrent Behaviour

```
test(Valid_Query,                                 % refusal scenario
  refuse(                                         % behaviour to be refused
    send(splitter.data.count,                     % ask for job percentages
      SplitData(Certificate,' ~~~Age>30,<>)),     % get not not not over 30s
    read(splitter.data.count,SplitterError,'Malformed)))   % this fault must not occur
```

Fig. 11. Scenario with Refused Behaviour

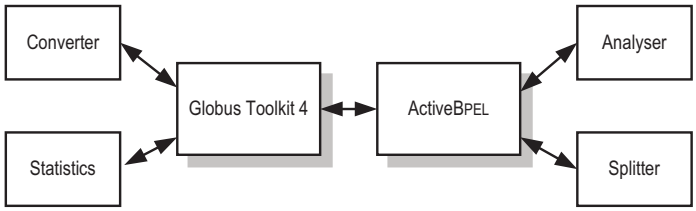

Fig. 12. Deployment of Occupational Data Analysis Services

5.1 Implementation Structure

A simple command creates a service implementation automatically. The normal GT4 build procedure is rather restrictive and requires particular naming conventions to be followed. Instead, CRESS uses its own simpler and more flexible build procedure (though some utilities from GT4 are part of this).

The end result of the automated implementation is a set of service files. Grid services are deployed to the GT4 platform for grid computing (*www.globus.org*). BPEL processes are deployed to the ActiveBPEL platform for service orchestration (*www.activebpel.org*). As shown in figure 12, these are distinct platforms that run their own kinds of services. GT4 and ActiveBPEL communicate to allow BPEL processes to call grid services. In principle, it should be possible to run GT4 within the same Tomcat container as ActiveBPEL. However this is currently not possible due to incompatibilities in the underlying packages (notably the AXIS package for SOAP messaging). As GT4 evolves, this problem will disappear. In practice anyway, BPEL is likely to coordinate services running on a number of systems over a network.

In comparison to the work on OMII-BPEL [6], CRESS has similar goals for practical grid service orchestration – though there are some differences. OMII-BPEL is focused on pragmatic support for modelling, enactment and monitoring large-scale scientific workflows. CRESS is more concerned with high-level models that can be verified and validated as well as implemented. So far, CRESS has not been evaluated on large problems. However, this is not a limitation of CRESS. Rather, it reflects on the support of process execution. In fact, it should be possible to combine the use of CRESS (particularly its formal support of analysis) with the OMII-BPEL work (particularly its run-time support).

Since ActiveBPEL was designed for web services, it is not surprising that web services can be run in the same container as BPEL. In fact it is possible for the same BPEL process to orchestrate a mixture of web and grid services.

As will have been seen from figure 6, implementing grid services requires many files in different languages. Figure 13 gives one example of what is involved. The designer creates CRESS diagrams for the services, usually with CRESS's own diagram editor though others can be used. Each service diagram is translated into a

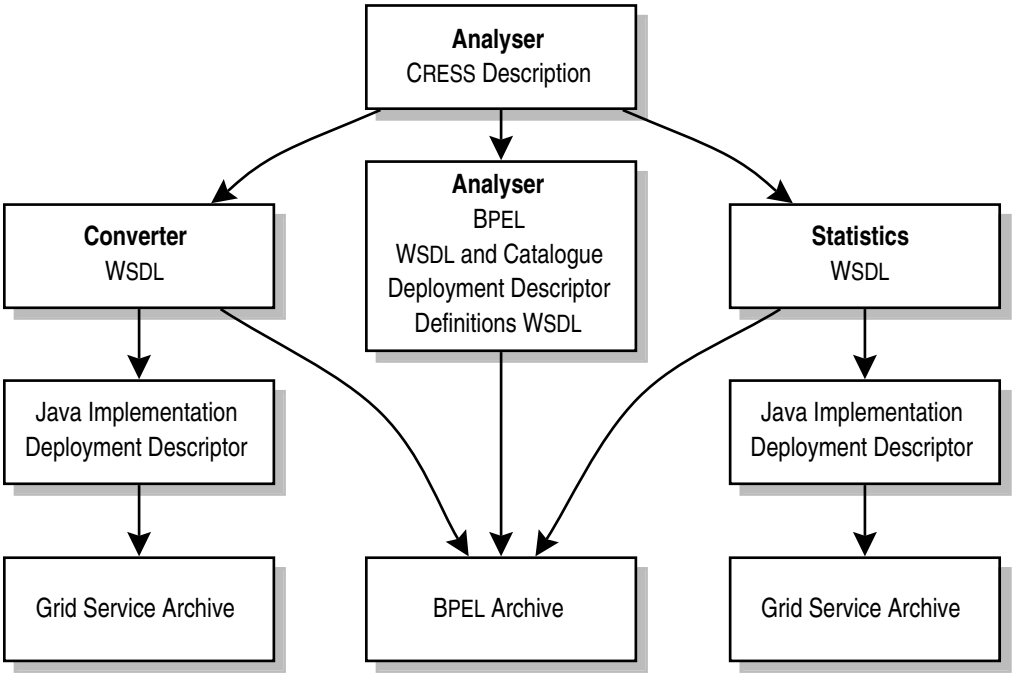

Fig. 13. Translation of The Analyser and Its Partner Services

BPEL description, the WSDL interface to this, and various ancillary files (WSDL catalogue, deployment descriptor, WSDL for shared definitions). All these files are bundled as a BPEL archive and deployed into ActiveBPEL.

The translation also generates code for the converter and statistics partners. CRESS creates a WSDL interface to each partner, a Java implementation, and a deployment descriptor. However, CRESS uses a Java implementation if one has already been written. All these files are bundled as a grid service archive and deployed into GT4.

5.2 Data

The translation of CRESS data types into BPEL/WSDL is intricate but reasonably straightforward. Simple CRESS types become XML simple types, while structured CRESS types become XML complex types. Accessing fields in a complex type requires XPATH queries.

Unfortunately, BPEL defines and uses variables differently depending on whether they appear in messages or just in expressions. CRESS handles this automatically, creating different definitions and code depending on how the variable is actually employed.

Assignments associated with CRESS arcs require careful treatment. Consider the arc between nodes 2 and 3 in figure 3. Since paths with different assignments may lead to the same node, such assignments have to be associated with the immediately

prior node. If this occurs in the context of a choice, the assignments must be made dependent on the condition. Thus the assignments following node 2 in figure 3 depend on the value of *authorised*. A different situation arises with the assignments following node 3 in figure 4. These lie on parallel paths, and so have to be associated with the following nodes (4 and 5).

5.3 Behaviour

CRESS deals with the differing uses of SOAP by service orchestration and by grid services. BPEL processes often use an 'rpc/encoded' SOAP style (though other styles are possible). Grid services always use a 'document/literal' SOAP style. An unfortunate consequence of this is that operations with the same parameter type cannot be distinguished. CRESS detects this as an error, and requires the designer to adopt a 'wrapped' SOAP style to distinguish them.

GT4 currently imposes a limitation on the orchestration of grid services. The most desirable form of security is the so-called WS-SecureConversation that allows *credential delegation* in grid terminology. Unfortunately the current implementation of GT4 requires all services to use the same container for delegation to work. As long as GT4 and ActiveBPEL cannot coexist in the same container, this is not feasible – but this problem will disappear in time.

The authors have also made use of Oracle BPEL Process Manager (*www.oracle.com/technology/products/ias/bpel*). Unlike ActiveBPEL, this does not have the possibility of coexistence in the same container as GT4 and is therefore even more remote from an integrated security solution.

An orchestrated process currently behaves like a web service. The authors are working to overcome limitations of ActiveBPEL that mean resources have to be treated outside the orchestration. Ideally grid service resources should be accessible to the orchestrating process, and this should be allowed to have resources as well.

ActiveBPEL version 2.0 and Globus Toolkit 4.0.0 were originally used by the authors for grid service orchestration. This version of ActiveBPEL conforms to the BPEL4WS standard, and therefore does not support the version of WS-Addressing (Web Services Addressing [32]) that is compatibile with WSRF (Web Services Resource Framework [11]). The absence of this support requires a work-around to orchestrate grid services. Instead of assigning an endpoint reference to a partner link, this information must be sent as a service invocation parameter. An additional problem was that GT4 support of endpoint addressing was incompatible with the version of WS-Addressing that it claimed to handle. As a result, endpoint references failed to deserialise. It was necessary to change the WS-Addressing schema used by GT4 for this to work properly.

With the recent introduction of ActiveBPEL version 3.0 and Globus Toolkit 4.0.4, these problems have been eased. ActiveBPEL now supports WS-BPEL [2], which uses an appropriate version of WS-Addressing. This has allowed CRESS to support endpoint references properly. These can now be assigned to partner services so that service resources can be used. However, this is not quite enough for orchestrating grid services with resources. When invoking a grid service using the assigned partner link endpoint reference, ActiveBPEL uses additional attributes (e.g. the service actor). This results in GT4 not being able to identify the service resource even though the required information is present. Even though the information complies with the standards, GT4 is unable to extract the correct WS-Addressing information. This has required the authors to modify the message handler chain used by GT4 so that resource information can be properly handled.

With the resolution of security and resource issues, orchestrated grid services will become true grid services in their own right. However, even the present limitations do not detract from the value of grid service orchestration.

6 Conclusions

It has been seen how CRESS has been adapted to support orchestration of grid services. This offers the advantage that new composite services can be constructed from existing ones. As a realistic example, occupational data analysis by social scientists has been used to explain how orchestrating grid services is achieved.

The CRESS description of composite grid services is automatically translated into LOTOS for rigorous validation with MUSTARD (and also, in principle, through verification). This allows a range of errors to be eliminated before committing to implementation. The authors did in fact discover some subtle problems by this means:

- Job percentages are rounded to whole numbers. The validation scenarios expected that this would be done by rounding to the nearest whole number, whereas the specification rounds by truncation. Although this was an error in the scenarios rather than the specification, it drew attention to something that needed attention in the implementation.
- A concurrency issue was discovered. Two instances of the analyser are created by the splitter (nodes 4 and 5 in figure 4). If one of these faults, the associated compensation deletes the corresponding data store (node 11 in figure 4). Depending on the exact sequence, it is possible for the data store to be deleted by one parallel branch while the other is still executing. This was corrected by changing the specification of the converter.
- An even more complex situation arises because deletion of a data store is not confirmed (nodes 9 and 11 in figure 4). As a result, deletion may be arbitrarily delayed. A new instance of the splitter may therefore encounter a ‘half deleted’

data store. The original version of the converter specification did not handle this properly.

As a result of the rigorous validation, the authors have reasonable confidence that the CRESS descriptions are correct. The final CRESS orchestration was automatically implemented and run using ActiveBPEL and GT4. The implementation was checked using JUnit tests based on the MUSTARD scenarios for validating the specification. There is thus a double re-use: the same CRESS descriptions are used for specification and implementation, and the same MUSTARD scenarios are also used for both.

The whole development process is highly automated. Only the external partner services have to be translated manually. The CRESS toolset is very portable, being written in Perl. It has been run on four different operating systems. The toolset also supports four target languages and five application domains. For grid services, it has hopefully been demonstrated that CRESS is a useful approach for orchestration.

Acknowledgements

The authors thank their co-workers on GEODE for their insights, particularly Paul Lambert (University of Stirling), Vernon Gayle (University of Stirling) and Richard Sinnott (University of Glasgow). Larry Tan's work was supported by the UK Economic and Social Research Council under grant RES-149-25-1015.

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, editors. *Business Process Execution Language for Web Services*. Version 1.1. BEA, IBM, Microsoft, SAP, Siebel, May 2003.

[2] A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C. K. Lie, S. Thatte, P. Yendluri, and A. Yiu, editors. *Web Services Business Process Execution Language*. Version 2.0. Organization for The Advancement of Structured Information Standards, Billerica, Massachusetts, USA, Apr. 2007.

[3] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen. A feature-interaction benchmark for IN and beyond. *IEEE Communications Magazine*, 31(8):18–23, Aug. 1993.

[4] K.-M. Chao, M. Younas, N. Griffiths, I. Awan, R. Anane, and C.-F. Tsai. 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. Institution of Electrical and Electronic Engineers Press, New York, USA, 2004.

[5] A. Chirichiello and G. Salaün. Encoding abstract descriptions into executable web services: Towards A formal development. In *Proc. Web Intelligence 2005*. Institution of Electrical and Electronic Engineers Press, New York, USA, Dec. 2005.

[6] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price. Grid service orchestration using the business process execution language (BPEL). *Grid Computing*, 3(3-4):283–304, Sept. 2005.

[7] A. Ferrara. Web services: A process algebra approach. In *Proc. 2nd. International Conference on Service-Oriented Computing*, pages 242–251. ACM Press, New York, USA, Nov. 2004.

[8] H. Foster. *A Rigorous Approach to Engineering Web Service Compositions*. PhD thesis, Imperial College, London, Jan. 2006.

[9] H. Foster, S. Uchitel, J. Kramer, and J. Magee. Compatibility verification for web service choreography. In M. Aiello, editor, *Proc. 2nd. International Conference on Service-Oriented Computing*, New York, USA, Nov. 2004. ACM Press.

[10] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Supercomputer Applications*, 35(6), 2002.

[11] S. Graham, A. Marmakar, J. Mischinsky, I. Robinson, and I. Sedukhin, editors. *Web Services Resource*. Version 1.2. Organization for The Advancement of Structured Information Standards, Billerica, Massachusetts, USA, Apr. 2006.

[12] S. Haddad, T. Melliti, P. Moreaux, and S. Rampacek. A dense time semantics for web services specification languages. In *First International Conference on Information and Communication Technologies*. Institution of Electrical and Electronic Engineers Press, New York, USA, Apr. 2004.

[13] ISO/IEC. *Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*. ISO/IEC 8807. International Organization for Standardization, Geneva, Switzerland, 1989.

[14] ITU. *SDL combined with ASN.1*. ITU-T Z.105. International Telecommunications Union, Geneva, Switzerland, 1995.

[15] ITU. *Specification and Description Language*. ITU-T Z.100. International Telecommunications Union, Geneva, Switzerland, Aug. 2002.

[16] P. S. Lambert. Handling occupational information. *Building Research Capacity*, 4:9–12, 2002.

[17] S. Majithia, D. W. Walker, and W. A. Gray. Automated composition of semantic grid services. In *Proc. 3rd. UK e-Science All Hands Meeting*. University of Nottingham, UK, Aug. 2004.

[18] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

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

[20] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *Proc. International Conference on Web Services*, pages 43–51. Institution of Electrical and Electronic Engineers Press, New York, USA, June 2004.

[21] A. Slomiski. On using BPEL extensibility to implement OGSI and WSRF grid workflows. In *Proc. Global Grid Forum 10*, Berlin, Germany, Mar. 2005. Humboldt University.

[22] B. Sotomayor and L. Childers. *Globus Toolkit 4: Programming Java Services*. Morgan Kaufmann, San Francisco, USA, Mar. 2006.

[23] K. J. Turner. Formalising the CHISEL feature notation. In M. H. Calder and E. H. Magill, editors, *Proc. 6th. Feature Interactions in Telecommunications and Software Systems*, pages 241–256. IOS Press, Amsterdam, Netherlands, May 2000.

[24] K. J. Turner. Modelling SIP services using CRESS. In D. A. Peled and M. Y. Vardi, editors, *Proc. Formal Techniques for Networked and Distributed Systems (FORTE XV)*, number 2529 in Lecture Notes in Computer Science, pages 162–177. Springer, Berlin, Germany, Nov. 2002.

[25] K. J. Turner. Analysing interactive voice services. *Computer Networks*, 45(5):665–685, Aug. 2004.

[26] K. J. Turner. Formalising web services. In F. Wang, editor, *Proc. Formal Techniques for Networked and Distributed Systems (FORTE XVIII)*, number 3731 in Lecture Notes in Computer Science, pages 473–488. Springer, Berlin, Germany, Oct. 2005.

[27] K. J. Turner. Validating feature-based specifications. *Software Practice and Experience*, 36(10):999–1027, Aug. 2006.

[28] K. J. Turner. Representing and analysing composed web services using CRESS. *Network and Computer Applications*, 30(2):541–562, Apr. 2007.

[29] K. J. Turner and K. L. L. Tan. Graphical composition of grid services. In D. Buchs and N. Guelfi, editors, *Rapid Introduction of Software Engineering Techniques*, pages 1–16, Switzerland, Sept. 2006. University of Geneva.

[30] M. Weiss and B. Esfandari. On feature interactions in web services. In *Proc. IEEE International Conference on Web Services*, pages 88–95, San Diego, California, July 2004.

[31] World Wide Web Consortium. *Web Services Description Language (WSDL)*. Version 1.1. World Wide Web Consortium, Geneva, Switzerland, Mar. 2001.

[32] World Wide Web Consortium. *Web Services Addressing (WS-Addressing)*. World Wide Web Consortium, Geneva, Switzerland, Aug. 2004.

[33] J. Zhou and G. Zeng. Describing and reasoning on the composition of grid services using pi-calculus. In S. An and D. Wei, editors, *Proc. 6th. International Conference on Computer and Information Technology*. Institution of Electrical and Electronic Engineers Press, New York, USA, Sept. 2006.