Koon Leai Larry Tan and Kenneth J. Turner. Automated Analysis and Implementation of Composed Grid Services.
In Dimitrios Dranidis and Ilias Sakellariou, editors,
Proc. 3rd South-East European Workshop on Formal Methods
pages 51-64, Thessaloniki, Greece, November 2007.

Automated Analysis and Implementation of Composed Grid Services

Koon Leai Larry Tan and Kenneth J. Turner

Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, UK
klt | kjt @cs.stir.ac.uk

Abstract. Service composition allows web services to be combined into new ones. Web service composition is increasingly common in mission-critical applications. It has therefore become important to verify the correctness of web service composition using formal methods. The composition of grid services is a similar but new goal. We have previously developed an abstract graphical notation called CRESS for describing composite grid services. We have demonstrated that it is feasible to automatically generate service implementations as well as formal specifications from CRESS descriptions. The automated service implementations use orchestration code in BPEL, along with the service interfaces and data types in WSDL and XSD respectively for all services. CRESS-generated BPEL implementations currently do not use WSRF features such as implicit endpoint references for WS-Resources and interfacing to standard WSRF port types. CRESS-generated formal models use the standardised process algebra LOTOS. Service behaviour is modelled by processes, while service data types are modelled as abstract data types. Simulation and validation of the generated LOTOS specifications can be performed. In this paper, we illustrate how CRESS can be further extended to improve its generation of service compositions, specifically for WSRF services implemented using Globus Toolkit 4. We also show how to facilitate use of the generated LOTOS specifications with the CADP toolbox.

1 Introduction

Grid computing is one of the leading forms of distributed computing. It enables interoperability between disparately owned and heterogeneous resources in a standardised manner. The SOA (Service-Oriented Architecture) nature of Grid Services encourages creation of new and composite services. This allows existing services to be combined into new ones. This activity, commonly known as service orchestration, requires complex behaviour which usually involve interactions between two or more services. There is much interest in the practical realisation of grid service orchestration. In contrast, however, the rigorous analysis of such complex and usually critical behaviour has not received much attention. A major issue is that formal models (where they are used at all) are developed separately from the implementation. This paper reports on work that aims to bridge the model-implementation gap to allow for automated (formal) analysis as well as implementation of composed grid services. The approach also supports orchestration of grid and web services. The work reported here uses CRESS (Communication Representation Employing Systematic Specifications). It is an extension of work reported in [16, 17], with a new emphasis on formal analysis.

2 Background

The scope of this paper covers a number of different technical areas and tools. This section provides a high-level background for the general reader.

2.1 Grid Services

OGSA (Open Grid Services Architecture) defines the characteristics and capabilities of the Grid. This includes virtualised resource access, building virtual organisations, distributed and parallel computing, and security.

Grid applications are usually implemented as services to exploit the benefits of service-oriented architecture. One of the widely adopted approaches is to treat grid services as extensions of web services. This means that they interact via message exchanged using SOAP (Simple Object Access Protocol). The requirement for grid services to be stateful led to the initial work on OGSI (Open Grid Services Infrastructure); this is not very compatible with web services. Further developments led to WSRF (Web Service Resource Framework). This is a collection of interrelated standards defined by OASIS that are compatible with web services. In this paper, we use WSRF-based services as grid services.

Although there are numerous solutions to building grid applications and systems, the WSRF specifications are the most popular open standards. Globus Toolkit 4 (GT4) is a widely used toolkit that implements WSRF. In the work reported here, GT4 was used for supporting development of grid services.

2.2 Service Orchestration

Service composition or orchestration was a significant factor in the design of the SOA paradigm. Numerous solutions were originally developed for orchestrating web services. BPEL4WS (Business Process Execution Language for Web Services) was the combined results of several organisations working to standardise service orchestration. BPEL4WS was later renamed WS-BPEL [1] by OASIS, and is established as *the* way of composing web services.

Because of similarities with web services, *grid* service composition with WS-BPEL has also been considered [2, 10]. WS-BPEL was originally designed for web services, and was little influenced by the various evolutions of grid service standards for OGSI and WSRF. Our preliminary investigations [12] demonstrated that it was possible to orchestrate grid services using ActiveBPEL version 2, an implementation of BPEL. Technical issues, workarounds and limitations were highlighted in our work.

The timely convergence and compatibility of WSRF and web services encouraged us to study further the feasibility of composing grid services using BPEL. WS-BPEL 2 incorporated specifications that are compatible with WSRF, such as WS-Addressing 2004/08, thereby making interoperability with grid services easier. WS-BPEL 2 has been implemented by several open-source BPEL engines, such as ActiveBPEL version 3 and WEEP (Workflow Engine Enactment Project [10]). The work in this paper used ActiveBPEL 3 to support grid service orchestration.

2.3 Occupational Data Matching

Social scientists often make use of occupational information. Several well-known occupational classification schemes are used to code occupational information. Most occupational analyses use these standard classifications. The choice of classification scheme usually differs, as different classifications may favour particular type of analysis.

Most datasets capture occupational information using a single classification scheme. Occupational data researchers usually have to translate occupational variables to the classifications required by the analysis process in order to analyse their data. Aggregate occupational datasets, which contain summary data for occupational position, are often linked by social scientists to microsocial survey datasets. There can be several intermediate translations involved when no direct mapping exist. Hence there exist many translation procedures for dealing with various formats. Despite having occupational data and mapping resources, social scientists rarely achieve effective resource sharing due to lack of a standardised framework for resource discovery, dissemination and data formats.

The authors have worked on the GEODE project (Grid-Enabled Occupational Data Environment) whose primary objective is to use grid computing to vitalise occupational matching procedures and make them discoverable and accessible in a uniform manner. Researchers can link mimicrosocialurvey datasets to aggregate occupational resources, and they can easily map between classification schemes. GEODEwas the source of the challenges addressed in this paper.

2.4 LOTOS

LOTOS (Language Of Temporal Ordering Specification [7]) is a standardised formal technique used to specify concurrent and distributed systems. In contrast to other formal techniques, LOTOS supports the integrated formalisation of both behaviour and data types. This offers the advantage of modelling systems where behaviour can be influenced by the state of data. Rigorous analysis can then be used to validate and verify a LOTOS specification, while simulation can be used for rapid prototyping.

There is a wide range of tools for performing formal analysis of LOTOS specifications. TOPO/LOLA can be used to execute system behaviour and data type operations. CADP (Construction and Analysis of Distributed Processes [6]) is a well-known toolset that supports a wide range of formal analyses (for LOTOS and other formal languages). CADP allows desired temporal logic properties to be verified. These properties can be expressed using the regular alternation-free modal μ-calculus, and are verified against designated LOTOS specifications by various CADP tools such as *SVL* and *Evaluator*. Properties such as deadlock freedom, safety, liveness and fairness can be verified. The toolset also has other useful features such as comparing the behaviour of specifications, and reducing the model state space of specifications.

2.5 CRESS

CRESS is a domain-independent graphical notation that was developed for describing services. CRESS supports the specification of web and grid service composition. It takes

an abstract approach in which a high-level service description in CRESS is used to automatically generated a formal specification as well as an actual implementation. The implementation of a composite grid service is generated in BPEL. Services that are part of the composition have their service interfaces and data types generated in WSDL (Web Service Description Language [18]) and XSD (XML Schema Definition) respectively. These interfaces are used in the execution of the BPEL specification. Since CRESS focuses on service *composition*, the implementation and configuration of individual grid services is provided manually. The individual services are automatically incorporated along with the generated code. The formalisation of a composite grid service in LOTOS is fully automated. Service behaviour is represented by interacting LOTOS processes. Data types, including complex types (as in XSD), are generated as abstract data types. We have previously validated the LOTOS specifications generated from CRESS service diagrams [14]. The emphasis in this paper is new work on verification of the generated specifications.

Specifications are drawn graphically in CRESS using nodes, arcs and labels. This appeals to both technical and non-technical users: the focus is on high-level description, abstracting away the technical details required in the actual implementation. CRESS is designed as an extensible framework where support for new domains and target languages can be added like plug-ins.

2.6 Relation to Other Work

Web service orchestration has been actively studied and supported by pragmatic developments. There are numerous implementations for modelling and executing of workflows. We cite some of the well-known ones that also support grid applications below.

JOpera [9] is a service composition tool for building new services through combining existing services. It provides a visual composition language and also a run-time platform to execute services. JOpera claims to offer greater flexibility and expressive constructs than BPEL. Although initially focused on web services, support for grid service composition has also been investigated.

Taverna [8] was developed to model web service workflows specifically for the bioinformatics domain. It introduced SCUFL (Simple Conceptual Unified Flow Language) to allow grid applications to be modelled in a specialised workflow language.

OMII-BPEL (Open Middleware Infrastructure Institute BPEL [2]) aims to support the orchestration of scientific workflows which can involve a multitude of service processes and long-duration process execution. It provides a customised ActiveBPEL engine, and supports a set of constructs that are desirable for the specification of scientific workflows.

CRESS was designed for modelling many kinds of services, and has applications in many domains. For grid and web services, CRESS can be viewed as a workflow language for the specification of composite behaviour. In contrast to other approaches, CRESS generates implementation code in standard languages (BPEL, WSDL, XSD, LOTOS) which are already widely adopted and implemented. This also means that the generated code can be deployed in various vendor implementations that conform to these standards.

Prior to the standardisation of WS-BPEL 2 there have been investigations into using BPEL to compose grid services. [11] developed BPEL extensibility mechanisms to orchestrate OGSI and WSRF services. [19] makes use of proprietary constructs to achieve interoperability with WSRF services. Work based on CRESS demonstrated the possibility of using standard WS-BPEL to orchestrate grid services, though some workarounds were necessary [12]. Further empirical work has been carried out with CRESS and WS-BPEL 2, leading to improvements in CRESS for specifying grid services (and also stateful web services).

Formalisation of *web* services has received considerable attention. LTSA-WS (Labelled Transition System Analyser [5]) is a finite state method. Abstract service scenarios and actual service implementations are generated through two behavioural models in the form of state transition systems. Verification and validation are performed by comparing the two systems. The limitation of this formal approach is that it can handle data types but not their values. This impacts on the formal analysis of service composition as data values can be used to model conditions that influence the behaviour of a system. CRESS differs by generating the formal model and service implementation from an abstract description. CRESS uses LOTOS to model service compositions, and can therefore model data types as well as their values.

[3, 4] use a process algebraic approach in automated translation between BPEL and LOTOS. CRESS differs in that there is no specification of BPEL or LOTOS required. Instead a graphical notation, comprehensible to the non-specialist, supports abstract service descriptions that are translated into BPEL and LOTOS automatically. This is an advantage as service development may well involve personnel who are not trained in either BPEL or LOTOS.

3 Specifying Grid Service Composition

A full description of the CRESS notation can be found in [15]. Not all CRESS constructs are used in this paper. An example of the interactions between BPEL and grid services is given to illustrate the technical aspects of communicating with grid (WSRF) services.

3.1 CRESS Notation

A high-level overview of the CRESS notation is given here. A CRESS diagram shows the flow among activities, drawn as ellipses. Each activity has a number, one or more actions, and some parameters. The arcs between ellipses represents the flow of behaviour. Note that CRESS defines flows and not state machines; state is implicit.

Choice of flow can be indicated by label on an arc. An arc may be labelled with a value guard or an event guard to determine if it will be traversed. If a value guard holds, then the behaviour may follow the path designated by the arc. An event guard defines a possible path that is enabled only once the corresponding event occurs. Assignments can also be specified as labels on the arcs.

CRESS activities in grid service composition deal, among other things, with grid service inputs (**Receive**), responses (**Reply**), and invocations of external partner services (**Invoke**). Grid service operations are named **partner**.**port**.**operation**, e.g. *dou-*

blemap.dmap.xtoz. As required, operations are followed by an input parameter, an output parameter, and one or more faults.

A CRESS rule-box is drawn as a rounded-edge rectangle. It defines variables, but also aspects that do not explanation here. CRESS supports a range of types for web and grid services. Simple variables have types like **Natural** *n*, **Integer** *i*, and **String** *s*. CRESS supports a **Reference** type for endpoint references that bind a service instance to a particular resource (known as a WS-Resource).

Structured types can be defined in a rule box using '[...]' for arrays and '{...}' for records (i.e. structures). The following defines variables *mapData*, *mapXData*, *mapYData*, and *mapZData*. Their type is a record with fields: an integer *occPos*, and an array of elements with type *dataset*. The latter is an array of elements with type *recordData*, which is an array of *variableData* strings.

```
{ Integer occPos
  [ [String variableData] recordData] dataset
} mapData, mapXData, mapYData, mapZData
```

Array elements are accessed by index, e.g. *mapXData.dataset[1]*.

3.2 Occupational Data Matching using Grid Services

The example used in this paper (figure 1) illustrates a simplified yet still typical activity of occupational matching required by social scientists. It uses a subset of CRESS constructs for orchestrating grid services to combine the behaviour of occupational matching grid service partners.

The scenario assumes two existing grid service partners that perform occupational matching, but differ in requirements and outputs. The first occupational matching grid service *xmapy* requires occupation information to be in the *ClassOccX* scheme; it outputs mapped occupation results according to the *ClassOccY* scheme. The value of *occPos* in *mapXData* indicates the index location of *recordData* in the *ClassOccX* scheme. It outputs variable *mapYData* whose *occPos* value indicates its location in the *ClassOccY* scheme.

The second service *ymapz* is similar to *xmapy*. It requires input in the *ClassOccY* scheme, and outputs results in the *ClassOccZ* scheme. The service interface of the *ymapz* is purposely implemented differently to demonstrate interaction with stateful grid services. Invoking the *mapData* operation (node 3) returns an endpoint reference (EPR) that is used to retrieve the matching results (node 4).

The orchestrated service *doublemap* combines these two services to map occupational information from *ClassOccX* to *ClassOccZ*. It performs a check for an invalid occupational position in the scheme. (In this case, simply a negative number is invalid.) The deployment of *doublemap* behaves as a new service with the capability of mapping between occupational classifications. The partner services it uses are hidden from the clients of *doublemap*. Nodes 1 and 5 define the input and output when *doublemap.dmap.xtoz* is invoked. Note that the *mapData* variable is not explicitly used in the orchestration. Its use is explained in section 4.

A CRESS configuration diagram (figure 2) is required for defining deployment details such as service namespace, service endpoints, relations with service partners, and

possible definitions of resource properties for grid services. The **Deploys** clause lists the CRESS translator options. Service partners are specified prior to the '/'; the services to be deployed appear after this. CRESS generates WSDL files for all the given services, and includes the WSDL of the service partners (*XMAPY* and *YMAPZ*). All services, such as *XMAPY*, have a namespace prefix ('xmapy'), a namespace URI (Uniform Resource Name 'urn:XMapY'), and a base URI where they are deployed ('localhost:8880/wsrf'). In this example, all the services were deployed on the local computer on port 8880 (for grid services) and port 8080 (for the orchestrated service). However, they can be deployed anywhere in the network. As it happens, this example does not use resource properties (indicated by '-').

3.3 Translating CRESS Service Descriptions

The CRESS diagrams (*doublemap*, configuration) hold all the information needed to automatically generate a LOTOS specification and a BPEL implementation. Figure 3 compares the translations of this example. The figure shows non-commented lines of code for data types, behaviour, service stubs, and the number of generated files.

- The fixed code is the framework common to all grid applications. This is substantial in LOTOS as it contains many pre-defined complex data types from the CRESS library.
- Translation to LOTOS results as a single specification file. Translation to BPEL yields many files: one BPEL file per service, one WSDL per service/partner, one Java file per data type, and several deployment files.
- Code for external partners (*xmapy*, *ymapz*) has to be manually written.

In other grid service examples, the LOTOS specification is rather smaller than the corresponding BPEL/WSDL implementation. This is because of the very expressive constructs in LOTOSṠection 5.2 explains the reason for the rather larger LOTOS specification in this example.

The LOTOS specification and BPEL implementation have been made available at *http://www.cs.stir.ac.uk/~klt/doublemap/doublemap.zip*.

4 Translating Service Compositions to BPEL

Translating CRESS descriptions of web and grid services to BPEL is described in [15, 16]. BPEL and WSDL are specialised languages, resulting in complex and large implementations, so only a high-level description is given here.

The CRESS philosophy is that the service designer should not *need* to know about the target languages(BPEL and WSDL) in order to implement a composite service. However implementations for grid service partners (in Java) have to be provided.

4.1 Implementation Structure

The automatically-generated implementation consists of a set of service files for three services: two grid service partners and a BPEL process. For the composite service,

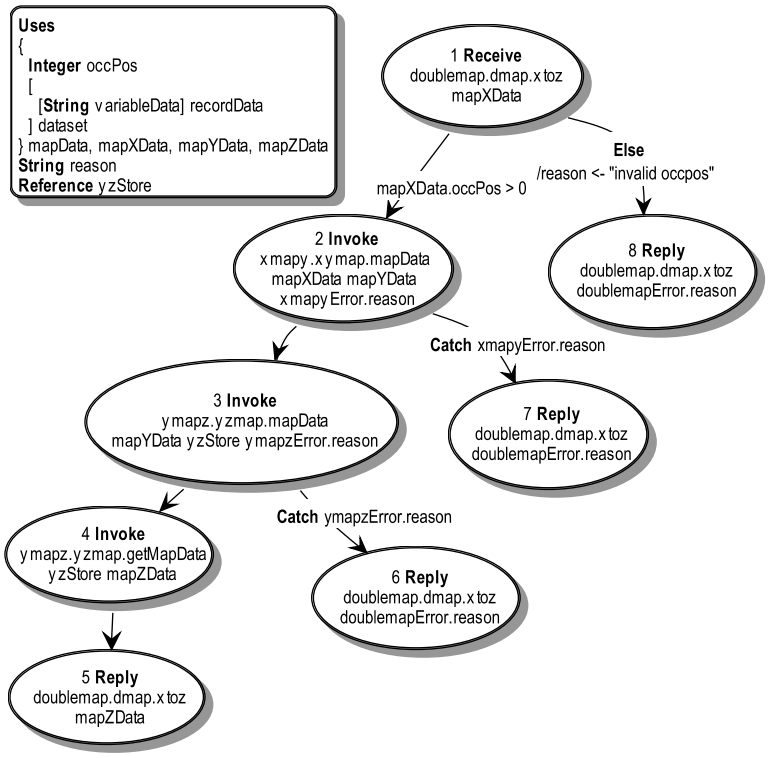

Fig. 1. CRESS Description of The *Doublemap* Service

Deploys XMAPY, YMAPZ / DOUBLEMAP

DOUBLEMAP	doublemap	urn:DoubleMap	localhost:8080/active-bpel	
XMAPY	xmapx	urn:XMapY	localhost:8880/wsrf	-
YMAPZ	ymapz	urn:YMapZ	localhost:8880/wsrf	-

Fig. 2. CRESS Service Configuration

Target	Fixed Code	Generated Code			Partner Code		Total
		Files	Types	Behaviour	Files	Behaviour	
LOTOS	1925	1	1617	113	2	76	**3768**
BPEL/WSDL	15	25	1438	148	10	537	**2138**

Fig. 3. Comparison of LOTOS and BPEL/WSDL Translations (files, lines of code)

its BPEL specification, WSDL (inclusive of service partners) and deployment files are created and bundled for deployment. For grid services, a WSDL service interface, Java implementation, and deployment descriptor are created per partner. CRESS will use a Java implementation if one has already been written (otherwise a simple default implementation is generated). These files are bundled as grid service archives. As illustrated in figure 4, the BPEL process *doublemap* is deployed to the ActiveBPEL platform for service orchestration, while grid service partners *xmapy* and *ymapz* are deployed to the GT4 platform for execution. GT4 and ActiveBPEL communicate to allow the BPEL process to invoke the grid service partners. In principle, it should be possible to run GT4 and ActiveBPEL in the same Tomcat container. However this is not currently possible due to incompatibilities between the versions of the Axis SOAP engine they use. This problem will be resolved as the two evolve to use compatible versions. In practice, the scenario of BPEL orchestrating services in disparate systems is common anyway.

Translation of CRESS data types into BPEL, WSDL and XSD is relatively straightforward. Simple CRESS types become XSD simple types, and structured CRESS types become XSD complex types. In BPEL, access to complex data types is via XPath queries. The assignment and guard constructs in CRESS are translated to XPath expressions. However the way BPEL uses variables differs, depending on whether they are used in messages or in expressions. CRESS handles this automatically, creating different definitions and code depending on the way variables are employed.

A grid service (WSRF) uses the 'document/literal' wrapped SOAP style which complies with the WS-Interoperability standard. Unlike RPC style, using 'document/literal' loses the operation names and thus relies on the data message structure to determine which operation should be invoked. As a consequence operations that uses the same message/parameter type cannot be distinguished. This is illustrated in [17]. This was the reason why *mapDatata* was defined in our example with a 'document/literal' wrapped style, the data structure having the same name as the operation.

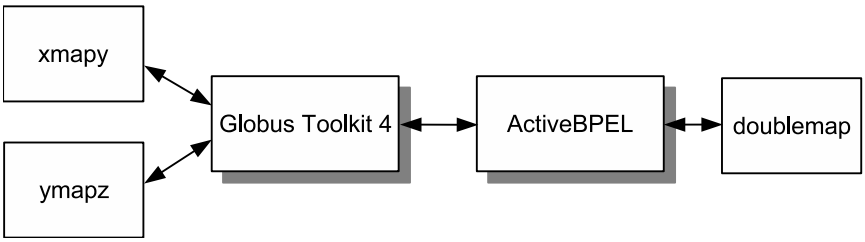

Fig. 4. Occupational Data Matching Service Deployment

5 Formal Analysis of Service Composition

The work reported in this paper has added the capability of verifying CRESS-generated LOTOS specifications using the CADP toolset. [13] describes how CRESS for web and grid services is translated into LOTOS.

CRESS has a set of predefined abstract data types for the specification of web and grid service composition. CRESS data types are mapped onto the base BPEL, WSDL and XSD types such as *xsd:integer* and *xsd:string*. The translation supports the definition of structures and arrays. These map onto complex types and arrays (*occurs*) in the implementation. BPEL simple types correspond to a limited range of LOTOS types. BPEL boolean corresponds to LOTOS Bool, BPEL natural to LOTOS Nat, and variations of BPEL string to LOTOS Text. Other numeric types in BPEL correspond to LOTOS type Number.

5.1 Validation

In principle, the LOTOS specification can be verified. However the complex data types and infinite data values make model checking rather impracticable. The authors have instead previously made use of rigorous validation (i.e. formally-based testing). Prior to the work presented here, scenario-based validation was the only means of analysing service specifications. This made use of MUSTARD (Multiple-Use Scenario Test and Refusal Description [14]) as a language-independent and tool-independent approach. This expresses use case scenarios that are translated into an appropriate target language (LOTOS here), and are automatically validated against the specification. Though validation is practicable, there are properties such as deadlock freedom that are better expressed and checked using verification. This was the motivation behind the new work, aiming to benefit from both formal verification and formal validation.

5.2 Tool Support for LOTOS Verification

CADP (Construction and Analysis of Distributed Processes [6]) has been used as to verify the generated LOTOS specifications. A LOTOS specification has to respect several restrictions for use with CADP. For example, definitions of abstract data types may not use formal sorts, operations or equations; these are incompletely specified types known parameterised types in LOTOS. Instead, CADP requires all types to be specified concretely. Constructor operations have to be identified for CADP. Verifying specifications is achieved through model checking. Hence, data types that are infinite have to be made finite in the external (C code) implementations. CADP-specific annotations are required on the LOTOS specification. We have automated as far as possible the task of making a CRESS-generated specification ready for verification.

CADP supports pragmas (specialised LOTOS comments) to annotate sort names, constructor operations, and data function names. These annotations are used by CADP in its translation of LOTOS to executable form.

The LOTOS specifications generated by CRESS contain parameterised types and actualisations (i.e. instantiations) of these; for example, this is how specific arrays are handled. To make the specification compatible with CADP, flattening of the actualised

data types is necessary to remove the specification of parameterised types. We have developed a tool that uses the TOPO/LOLA toolset to produce a flattened and annotated version of the data types.

Data types in web and grid services have finite ranges (e.g. *xsd:integer*); this depends on the programming language and platform used. This restriction does not apply to LOTOS types such as *Number*, which have an infinite range. Finite constraints have to be imposed on these data types for verification to be feasible. Though it can be achieved in LOTOS types by specifying each and every value, it is more practical to do so via constraints on the external C implementations. At the same time, it is possible to simplify types such as *String* and *Number* by representing them with simple C types like *char** and *double* respectively. As a result, the implementation of various operations is much simpler than its formalisation. As an example with *Number*, the value '3.142' can be handled directly in the C implementation. The LOTOS equivalent is *Number(+, t(3), t(1)∼4∼2)*, which is less readable. These simplifications make it easier to express verification properties, and these properties are also closer to the intended implementation. Our new work has created C implementations of the LOTOS types *Bool*, *Char*, *Nat*, *Number* and *Text*. These C implementations comprise macros and functions that define the sorts, their constructor operations, and operations where the equations are complicated. (The implementations created by CADP for simple operations can be used as automatically generated.)

Data types in CRESS-generated LOTOS are automatically annotated. This does not imply that the resultant specification is ready for verification. User-defined data structures still have to be annotated by hand as they are dynamic, and C implementations may still have to be manually generated if required. However, the effort required in annotation is greatly reduced by the tool, requiring limited effort to achieve a fully annotated specification.

Figure 5 outline the structure of the flattened and annotated LOTOS specification. The flattening process results in a single type with the same name as the specification. This data type contains all sorts, operations and equations that were specified in the CRESS-generated version. All the process definitions are preserved by the flattening. The mapping of CRESS names in figure 1 to LOTOS names should obvious.

5.3 Verification

Now that the fully annotated LOTOS specification has been created, various tools in CADP can be used to verify desirable properties. The *Caesar.adt* and *Caesar* tools were used along with the external implementations to generate a full C implementation for model checking and state space exploration. The *SVL* and *Evaluator* tools were used to verify properties against the specification. An SVL script was written to generate the state space from the LOTOS specification. Desirable properties of the system were then expressed using the regular alternation-free μ-calculus, and were verified against the state space:

- A web/grid service should have no deadlock in its behaviour. This can be specified as 'every state must have a successor state'. More specifically, the system must arrive at a state to accept incoming requests after a finite number of steps.

```
Specification GSSystem                                   (* orchestrated grid service *)

  Library ...                                            (* imported library types *)

  Type GSSystem Is                                       (* specific types *)
    Sorts                                                (* sorts of values)
      operation (*! implementedby ... *) ...            (* grid operation values *)
      port (*! implementedby ... *) ...                 (* grid port values *)
      mapdata (*! implementedby ... *) ...              (* mapdata values *)
      number (*! implementedby ... *) ...               (* number values *)
    Opns                                                 (* operations on values *)
      mapdata(*! implementedby ... constructor *) ...   (* mapdata constructor *)
      number(*! implementedby ... constructor external *) ...  (* number constructor *)
      ...
    Eqns                                                 (* equations defining operations *)
      ...

  Behaviour                                              (* overall specification behaviour *)
    Hide xmapy,ymapz In                                  (* hide partner gates *)
      (
        XMAPY [xmapy]                                    (* xmapy partner *)
      |||
        YMAPZ [ymapz]                                    (* ymapz partner *)
      )
    |[xmapy,ymapz]|                                      (* synchronised with partners *)
      DOUBLEMAP [doublemap,xmapy,ymapz]                  (* orchestration process *)

  Process XMAPY ...                                      (* xmapy partner *)
  Process YMAPZ ...                                      (* ymapz partner *)
  Process DOUBLEMAP_1 ...                                (* doublemap node 1 *)
  ...
  Process DOUBLEMAP_event ...                            (* doublemap event handler *)
```

Fig. 5. Flattened and annotated LOTOS specification structure

- Every request received by *doublemap* should result in a result or a fault. This was expressed by saying there no behaviour sequence should violate this property.
- Inputs containing a negative occupation position value must not result in a successful result.
- Each occupational value must result in the correct mapped value.

Initial verification results showed that the system did not meet the fourth condition. By examining the counterexample generated by *Evaluator*, the manually written specification of *ymapz* was found to be incorrect. There was an error in specifying the mapping of one particular occupational value, resulting in an incorrect mapped value. Though in practice this might have been found through validation, this would have required an exhaustive set of tests that explored the entire state space.

6 Conclusion

It has been seen that grid service orchestration can be achieved using CRESS. Occupational data matching activities have been used as a realistic example to illustrate the approach.

A single CRESS description of a composite grid service is automatically translated into a formal specification and a deployable implementation. This bridges the gap between formal specification and implementation, and encourages verification of system behaviour prior to implementation. It can help reduce the cost of development in two ways: new composite services can be quickly developed and readily deployed, and the system behaviour can be checked to detect errors at design time.

Rigorous validation of CRESS descriptions was already possible using MUSTARD. This is done by specifying test cases and checking them against the specification. A new ttechniquehas now been developed to automatically convert a LOTOS specification for use with CADP. Minimal manual effort is required to achieve this. Temporal properties can now be specified and verified against the specification. Though the verification works by constraining the state space of the system, global properties such as deadlock can now be checked. Important properties can be verified to establish confidence in the service description. Although validation cannot be used to establish desirable system properties, it is still useful for dealing with infinite state spaces.

CRESS currently supports static service partner endpoints. It is planned extend CRESS to support dynamic service endpoints. Interactions with WS-Resources (usually dynamic) will have a direct impact on the configuration of service endpoints in CRESS. The appropriate interaction with WS-Resources must be considered. It is also desirable to extend the CRESS coverage to the rest of the WSRF interfaces such as WS-ResourceLifeTime, WS-ServiceGroups and WS-BaseFaults.

Techniques for verifying CRESS-generated specifications have yielded favourable results. C implementations have been developed for CRESS library types. Coupled with automated annotation for CADP, formal verification of CRESS descriptions is now a practical task. There is still some limited user involvement in annotating user-defined data types. However, these user-defined data types are based on the CRESS library data types. It will be possible to fully annotate their specifications.

Suggestions have been made as to how CRESS can be enhanced to improve its support of orchestrated grid services, and to further automate the process of making a LOTOS specification ready for verification. For grid services, it has hopefully been demonstrated that CRESS is a useful approach for service orchestration and formal analysis.

References

1. Asaaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Yaron Goland, Neelakantan Kartha, Canyang Kevin Lie, Satish Thatte, Prasad Yendluri, and Alex Yiu, editors. *Web Services Business Process Execution Language*. Version 2.0. Organization for The Advancement of Structured Information Standards, Billerica, Massachusetts, USA, April 2007.
2. B. Butchart N. Cameron L. Chen B. Wassermann, W. Emmerich and J. Patel (2007). Sedna: A BPEL-based environment for visual scientific workflow modelling.

3. Antonella Chirichiello and Gwen 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, December 2005.
4. Andrea Ferrara. Web services: A process algebra approach. In *Proc. 2nd. International Conference on Service-Oriented Computing*, pages 242–251. ACM Press, New York, USA, November 2004.
5. Howard Foster. *A Rigorous Approach to Engineering Web Service Compositions*. PhD thesis, Imperial College, London, January 2006.
6. Radu Mateescu Hubert Garavel, Frédéric Lang. An overview of CADP 2001. In *European Association for Software Science and Technology (EASST) Newsletter*, volume 4, pages 13–24. August 2002.
7. 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.
8. Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
9. Cesare 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, November 2005.
10. Globus Alliance Project. Workflow Enactment Engine Project. *http://weep.gridminer.org/index.php/About_WEEP*, July 2007.
11. Aleksander Slomiski. On using BPEL extensibility to implement OGSI and WSRF grid workflows. In *Proc. Global Grid Forum 10*, Berlin, Germany, March 2005. Humboldt University.
12. 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 Omar Abuelma'atti, editors, *Proc. 7th PGNet Symposium*, pages 31–36. School of Computing, Liverpool John Moores University, Liverpool, UK, June 2006.
13. Kenneth J. Turner. Formalising web services. In Farn 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, October 2005.
14. Kenneth J. Turner. Validating feature-based specifications. *Software Practice and Experience*, 36(10):999–1027, August 2006.
15. Kenneth J. Turner. Representing and analysing composed web services using CRESS. *Network and Computer Applications*, 30(2):541–562, April 2007.
16. Kenneth J. Turner and Koon Leai Larry Tan. Graphical composition of grid services. In Didier Buchs and Nicolas Guelfi, editors, *Rapid Introduction of Software Engineering Techniques*, pages 1–16, Switzerland, September 2006. University of Geneva.
17. Kenneth J. Turner and Koon Leai Larry Tan. A rigorous approach to orchestrating grid services. *Computer Networks*, 51(15):4421–4441, October 2007.
18. World Wide Web Consortium. *Web Services Description Language (WSDL)*. Version 1.1. World Wide Web Consortium, Geneva, Switzerland, March 2001.
19. Matthew Zager. SOA/Web Services - Business Process Orchestration with BPEL. *http://webservices.sys-con.com/read/155631_1.htm*, December 2005.