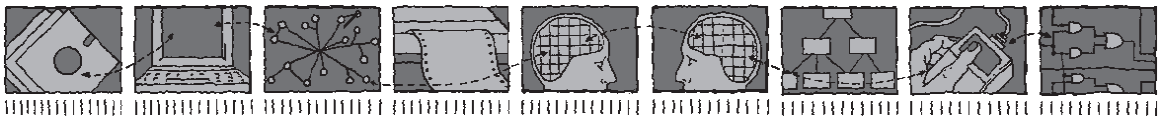
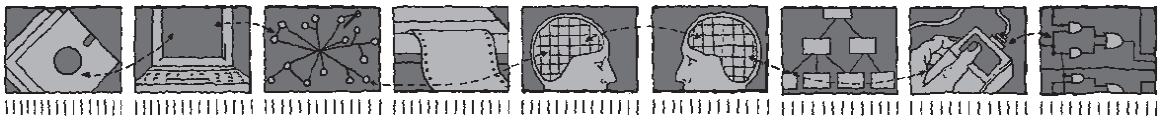


Department of Computing Science and Mathematics
University of Stirling



Case Studies Using CRESS to develop Web and Grid Services

Koon Leai Larry Tan

Technical Report CSM-183

ISSN 1460-9673

December 2009

Department of Computing Science and Mathematics
University of Stirling

Case Studies Using CRESS to develop Web and Grid Services

Koon Leai Larry Tan

Department of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland

Telephone +44 1786 467 421, Facsimile +44 1786 464 551
Email klt@cs.stir.ac.uk

Technical Report CSM-183

ISSN 1460-9673

December 2009

Abstract

Web and grid services are forms of distributed computing based on the SOA (Service-Oriented Architecture) paradigm which offers loose-coupling of functionality and interoperability in heterogeneous environments, achieved by using standards such as SOAP (Simple Object Access Protocol) and WSDL (Web Services Description Language). Web and grid services are widely used in businesses and research, where functionality is usually mission-critical. Services may be combined to create new services, an activity generally known as service composition. There are standards such as WS-BPEL (Web Services – Business Process Execution Language) that implement service composition. There is increasing implementation of web/grid services and their compositions, with more complex behaviour being developed.

Formal techniques support mathematical reasoning about the behaviour of systems which can detect errors and identify issues prior to implementation, and therefore can minimise the potential cost of errors as early as possible. Formal techniques can be applied to web and grid service composition. However, the attention given to their use is, by contrast, far less than the practical realisation of service implementation. Two reasons for this are the lack of formal expertise, and methodological differences between the practices of formal methods and service implementation.

CRESS is an abstract graphical notation and toolset for describing services at a high level, whereby specifications and implementations are automatically generated. CRESS has been extended for web and grid service compositions with automated support for specification, analysis, implementation and testing that appeals to non-specialists. Service behaviour is automatically specified in LOTOS (Language of Temporal Ordering Specification). Formal analysis is supported in a manner appealing to non-specialists by abstracting formal aspects using high-level languages; analysis is carried out automatically. These formal analyses, in the forms of validation and verification, are supported by MUSTARD (Multiple–Use Scenario Testing and Refusal Description) and CLOVE (CRESS Language-Oriented Verification Environment). Implementations of (composed) web and grid services are generated as Java and BPEL services, for deployment in Axis (web services), Globus Toolkit (grid services), and ActiveBPEL (BPEL services). Implementation analysis in the form of validation is supported by MINT (MUSTARD Interpreter), which is a high-level notation and tool that automates testing and performance evaluation on implemented services. These are integrated with CRESS, providing an integrated environment for service development in a rigorous way.

This report demonstrates the integrated methodology using web and grid service compositions as case studies.

Contents

Abstract **i**

1 Methodology Overview **1**

2 Case Studies **3**
2.1 Web Service Case Study . . . 3
2.2 Grid Service Case Study . . . 3

3 Development Of The Lender Composed Web Service **3**
3.1 Introduction . . . 3
3.2 Design . . . 4
3.2.1 Service Diagram . . . 4
3.2.2 Service Configuration . . . 6
3.2.3 Checking Diagrams . . . 6
3.3 Formal Specification . . . 6
3.3.1 Approver Specification . . . 6
3.3.2 Assessor Specification . . . 7
3.3.3 Lender Specification . . . 7
3.3.4 Summary . . . 8
3.4 Formal Analysis . . . 8
3.4.1 Formal Validation . . . 8
3.4.2 Formal Verification . . . 12

3.5 Implementation . . . 14
3.5.1 Approver Implementation . . . 15
3.5.2 Assessor Implementation . . . 15
3.5.3 Lender Implementation . . . 15
3.5.4 Summary . . . 15
3.6 Implementation Validation . . . 15
3.6.1 Approver Implementation Validation . . . 16
3.6.2 Assessor Implementation Validation . . . 16
3.6.3 Lender Implementation Validation . . . 17
3.6.4 Performance Evaluation . . . 17
3.7 Summary . . . 18

4 Development Of The Supplier Composed Web Service 18
4.1 Introduction . . . 18
4.2 Design . . . 18
4.2.1 Service Diagram . . . 19
4.2.2 Service Configuration . . . 21
4.2.3 Checking Diagrams . . . 21
4.3 Formal Specification . . . 21
4.3.1 Dealer1 Specification . . . 21
4.3.2 Dealer2 Specification . . . 22
4.3.3 Supplier Specification . . . 22
4.3.4 Summary . . . 22
4.4 Formal Analysis . . . 22
4.4.1 Formal Validation . . . 22
4.4.2 Formal Verification . . . 24
4.5 Implementation . . . 25
4.5.1 Dealer1 Implementation . . . 26
4.5.2 Dealer2 Implementation . . . 26
4.5.3 Supplier Implementation . . . 26
4.5.4 Summary . . . 26
4.6 Implementation Validation . . . 27
4.6.1 Implementation Validation of Dealer1 . . . 27
4.6.2 Implementation Validation of Dealer2 . . . 27
4.6.3 Implementation Validation of Supplier . . . 28
4.6.4 Performance Evaluation . . . 28

5 Development Of The Broker Composed Web Service 29
5.1 Introduction . . . 29
5.2 Design . . . 29
5.2.1 Service Diagram . . . 29
5.2.2 Service Configuration . . . 30
5.2.3 Checking Diagrams . . . 30
5.3 Specification . . . 30
5.4 Formal Analysis . . . 30
5.4.1 Formal Validation . . . 30
5.4.2 Formal Verification . . . 33
5.5 Implementation . . . 34
5.5.1 Summary . . . 34
5.6 Implementation Validation . . . 35
5.6.1 Broker Implementation Validation . . . 35
5.6.2 Performance Evaluation . . . 35

6 Development of Allocator Composed Grid Service **36**
6.1 Introduction . . . 36
6.2 Design . . . 36
6.2.1 Service Diagram . . . 36
6.2.2 Service Configuration . . . 37
6.2.3 Checking Diagrams . . . 37
6.3 Formal Specification . . . 37
6.3.1 Resource Specification . . . 39
6.3.2 Factory Specification . . . 39
6.3.3 Mapper Specification . . . 40
6.3.4 Allocator Specification . . . 40
6.3.5 Summary . . . 40
6.4 Formal Analysis . . . 40
6.4.1 Formal Verification . . . 40
6.4.2 Formal Validation . . . 41
6.5 Implementation . . . 42
6.5.1 Implementation of Factory Service . . . 43
6.5.2 Implementation of The Mapper Service . . . 43
6.5.3 Implementation of The Allocator Service . . . 44
6.6 Implementation Validation . . . 44
6.6.1 Allocator Implementation Validation . . . 44
6.6.2 Performance Evaluation . . . 44

7 Evaluation **45**

A Composed Web Services **47**
A.1 Partner Specification . . . 47
A.1.1 Approver Completed LOTOS Specification . . . 47
A.1.2 Assessor Interface LOTOS Specification . . . 47
A.1.3 Assessor Completed LOTOS Specification . . . 47
A.1.4 Dealer1 Interface LOTOS Specification . . . 48
A.1.5 Dealer1 Completed LOTOS Specification . . . 48
A.2 Partner Implemenetation . . . 49
A.2.1 Approver Completed Implementation . . . 49
A.2.2 Assessor Code Skeleton . . . 50
A.2.3 Assessor Completed Implementation . . . 50
A.2.4 Dealer1 Code Skeleton . . . 50
A.2.5 Dealer1 Completed Implementation . . . 51
A.3 Formal Specification . . . 51
A.3.1 Lender LOTOS Specification . . . 51
A.3.2 Supplier LOTOS Specification . . . 52
A.3.3 Broker LOTOS Specification . . . 53
A.4 Formal Validation . . . 54
A.4.1 Approver Translated LOTOS Scenarios . . . 54
A.4.2 Assessor Translated LOTOS Scenarios . . . 54
A.4.3 Lender Translated LOTOS Scenarios . . . 55
A.4.4 Dealer1 Translated LOTOS Scenarios . . . 56
A.4.5 Dealer2 Translated LOTOS Scenarios . . . 57
A.4.6 Supplier Translated LOTOS Scenarios . . . 58
A.4.7 Broker Translated LOTOS Scenarios . . . 59
A.5 Formal Verification . . . 60
A.5.1 Annotated Lender LOTOS Specification . . . 60
A.5.2 Translated μ-calculus Properties for Lender . . . 61
A.5.3 Generated C Code of User Types for Lender . . . 61
A.5.4 Annotated Supplier LOTOS Specification . . . 62
A.5.5 Automated Adaptation of Supplier LOTOS Specification for Compositional Verification . . . 62

A.5.6 Constrained Data Types and Operations for Supplier . . . 63
A.5.7 Translated μ-calculus Properties for Supplier . . . 63
A.5.8 Generated C Code of User Types for Supplier . . . 63
A.5.9 Generated SVL Script for Supplier . . . 64
A.5.10 Annotated Broker LOTOS Specification . . . 64
A.5.11 Manual Adaptation of Annotated Broker LOTOS Specification . . . 66
A.5.12 Translated μ-calculus Properties for Broker . . . 68
A.5.13 Generated C Code of User Types for Broker . . . 69
A.5.14 SVL Script for Broker . . . 70
A.6 Implementation . . . 71
A.6.1 File Structure of Generated Code in Lender . . . 71
A.6.2 Approver WSDL . . . 73
A.6.3 Assessor WSDL . . . 73
A.6.4 Lender WSDL . . . 74
A.6.5 Common Definitions - lender_defs.wsdl . . . 74
A.6.6 Lender BPEL . . . 75
A.6.7 File Structure of Generated Code in Supplier . . . 76
A.6.8 Dealer1 WSDL . . . 78
A.6.9 Supplier WSDL . . . 78
A.6.10 Common Definitions - supplier_defs.wsdl . . . 79
A.6.11 Supplier BPEL . . . 80
A.6.12 File Structure of Generated Code in Broker . . . 81
A.6.13 Broker WSDL . . . 82
A.6.14 WSDL Common Definitions . . . 83
A.6.15 Broker BPEL . . . 83
A.7 Implementation Validation . . . 84
A.7.1 Approver MINT Properties . . . 84
A.7.2 Approver Translated MINT Scenarios . . . 85
A.7.3 Assessor MINT Properties . . . 85
A.7.4 Assessor Translated MINT Scenarios . . . 85
A.7.5 Lender MINT Properties . . . 86
A.7.6 Lender Translated MINT Scenarios . . . 86
A.7.7 Dealer1 MINT Properties . . . 86
A.7.8 Dealer1 Translated MINT Scenarios . . . 87
A.7.9 Dealer2 MINT Properties . . . 87
A.7.10 Dealer2 Translated MINT Scenarios . . . 87
A.7.11 Supplier MINT Properties . . . 87
A.7.12 Supplier Translated MINT Scenarios . . . 88
A.7.13 Broker MINT Properties . . . 88
A.7.14 Broker Translated MINT Scenarios . . . 88

B Allocator Composed Service **89**
B.1 Partner Specification . . . 89
B.1.1 Resource Completed LOTOS Specification . . . 89
B.1.2 Factory Completed LOTOS Specification . . . 90
B.1.3 Mapper Completed LOTOS Specification . . . 90
B.2 Formal Specification . . . 90
B.2.1 Allocator LOTOS Specification . . . 90
B.3 Formal Verification . . . 91
B.3.1 Annotated LOTOS Specification . . . 91
B.3.2 Translated μ-calculus . . . 91
B.3.3 Generated C Code of User Types . . . 92
B.4 Formal Validation . . . 92
B.4.1 Allocator Translated LOTOS Scenarios . . . 92
B.5 Implementation . . . 93
B.5.1 File Structure of Generated Code . . . 93

B.5.2 Factory Implementation . . . 95
B.5.3 Factory WSDD . . . 96
B.5.4 Factory JNDI Configuration . . . 96
B.5.5 Factory WSDL . . . 97
B.5.6 Mapper JNDI Configuration . . . 97
B.5.7 Mapper WSDL . . . 97
B.5.8 Allocator WSDL . . . 98
B.5.9 Common Definitions - allocator_defs.wsdl . . . 99
B.5.10 Allocator BPEL . . . 99
B.5.11 Allocator PDD . . . 100
B.5.12 Allocator MINT Properties . . . 101
B.5.13 Allocator Translated MINT Scenarios . . . 101

List of Figures

1 Methodology Lifecycle . . . 2
2 LoanStar Composite Service Development . . . 4
3 Lender CRESS Diagram . . . 5
4 Web Service Configuration Diagram for Lender . . . 6
5 Specifying Approver and Assessor . . . 7
6 Implementation of Approver and Assessor . . . 14
7 DoubleQuote Composite Service Development . . . 19
8 Supplier CRESS Diagram . . . 20
9 Web Service Configuration Diagram for Supplier . . . 21
10 Specifying Dealer1 and Dealer2 . . . 21
11 Implementation of Dealer1 and Dealer2 . . . 25
12 CarMen Composite Service Development . . . 29
13 Broker CRESS Diagram . . . 31
14 Web Service Configuration Diagram for Broker . . . 31
15 Allocator Composite Service Lifecycle . . . 37
16 Allocator CRESS Diagram . . . 38
17 Grid Service Configuration . . . 38
18 Specifying Factory, Mapper and Resource . . . 39
19 Implementation of Factory and Mapper . . . 43

1 Methodology Overview

Figure 1 illustrates how a typical service development uses the integrated methodology. The methodology exhibits the following iterative phases: high-level design, abstract model specification, design-phase analysis, implementation, and post-implementation analysis. From an overall perspective, the methodology supports development iteration. Transitions from design to analysis to implementation to testing are supported in a convenient manner. This automates a lot of work on behalf of the developers and analysts, enabling them to focus on important tasks such as analysis, re-design, implementation and testing.

The developer starts with the high-level design of the composed services and their configuration as CRESS service diagrams and configuration diagram. This is where the developer defines application data, partner involvement, service behaviour flow, and deployment configuration. These high-level descriptions are the foundation for the rest of the methodology – specifically seen in the automated formal specification and implementation on which further activities are based.

Formal specification of service behaviour is required in order to perform design analysis using validation (scenario tests) and verification (property assertions). This is automatically generated by CRESS tools which translate the high-level CRESS diagrams into a LOTOS specification. The generated specification fully specifies the behaviour of all CRESS services, and provides outline behaviour for non-CRESS partner services. The developer then adds handcrafted specifications for the non-CRESS partner services to their generated outline behaviour, which is usually done for thorough analysis. The range of analyses that can be performed depends on the depends on the details of the specification – more thorough analysis through more detailed specification. Validation and verification of the services can be performed in either order as they are independent of each other.

The developer defines high-level validation scenarios for the services to be validated using MUSTARD. The MUSTARD scenarios can be defined any time and are used only when validation is performed. The developer executes automated formal validation via CRESS. The outcomes of validation, specifically diagnostic traces of validation failures, provide the developer with feedback on the high-level design and manual partner specification. The developer can iterate the design, specification, and validation phases until satisfactory; the choice is entirely up to the developer's preference. For example, a developer who prefers a progressive spiral approach may add more details to the specification of non-CRESS partner services and define more validation scenarios to be validated at the next iteration.

The developer performs automated formal verification in a similar way to formal validation. The developer specifies high-level CLOVE properties to be verified. Usually CLOVE's predefined property templates are used as they are commonly verified properties, which also simplifies the property specification for the developer. The CLOVE properties may be defined any time and are used only when verification is performed. The developer executes the formal verification to achieve efficient verification via CRESS. By default, deadlock and livelock freedom are checked automatically. The outcomes of verification, in particular for compromised properties, provide counter-examples that the developer can use to diagnose and address the problem. The developer can iterate design, specification, and verification phases similarly to formal validation.

Once satisfied with the formal analysis, the developer should be more confident of the service design, particularly for the composed service. The developer then configures the services for implementation, and generates implementation of the services. CRESS composed services are generated as BPEL services, and non-CRESS partner services are generated for Java implementation. The developer then provides the Java implementation of web or grid partner services, usually by adding code to the code skeletons that are generated. Once implementation is completed, the services are then deployed into their respective service hosting containers (BPEL in ActiveBPEL, partner web services in Axis, and partner grid services in Globus Toolkit 4). CRESS automates the service deployment if specified by the developer.

Once implemented services are deployed, the developer can validate the functionality and evaluate the performance in that order as there is little or no benefit evaluating performance if the service functionality is not operating as desired. Validation of service implementations is similar to that of formal specifications, reusing the same sets of MUSTARD scenarios but executing them against actual services using the MINT tool. This allows developers to be confident that what worked in the design (validation) is also exhibited by the service implementation (the ultimate product). The developer executes functional validation which is similar to the formal validation. The testing outcomes, specifically diagnostic traces of failure, provide feedback to the developer who can use them to diagnose and address problems. The design, implementation, and testing are iterated until satisfactory; the choice of iteration is up to the developer's preference as described earlier for the formal analysis, except usually at this point all the MUSTARD scenarios are already specified. Once satisfied with the functional validation, the developer can proceed to performance evaluation, putting the target service under load, executing multiple tests either

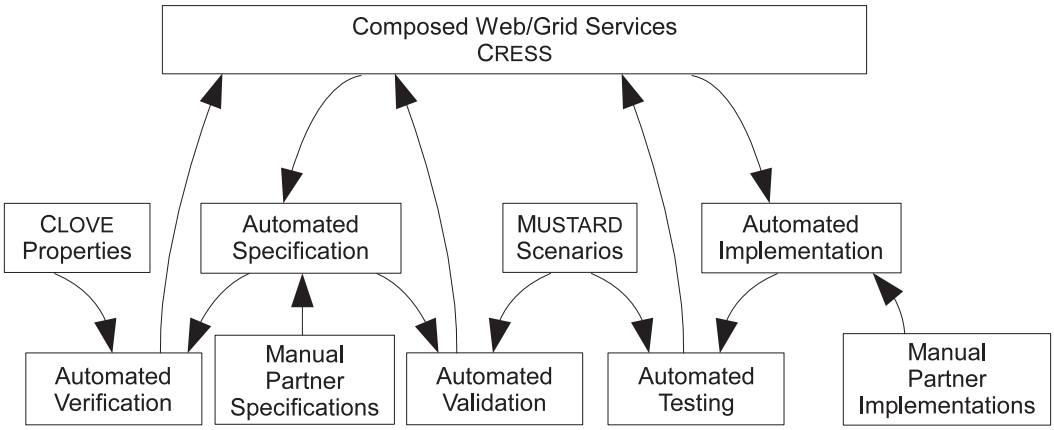


Figure 1: Methodology Lifecycle

sequentially or simultaneously as specified by the developer. This evaluates the service behavour consistency as well as obtaining insights to configuration issues such as resource allocation on hosting environments, in order to meet the non-functional requirements of the services.

Although the methodology does not mandate that all phases be followed or in a strict pattern, the convenient and automated support for most of the phases will encourage and motivate their application. For example, developers have the choice not to validate designs. However, design validation is simple, high-level and automated; the analysis contributes to service quality. It is therefore advantageous that all phases be applied, as direct benefits such as improved service quality can be achieved with limited effort.

The methodology is based on the CRESS toolset and therefore conforms to its framework, including directory structure, filename conventions, and framework-specific files.

CRESS 4.3, MUSTARD 1.7, CHIVE 1.8, CLOVE 1.0, and MINT 1.0 are used in this report.

2 Case Studies

A total of four web and grid service compositions are used in combination as case studies to demonstrate each individual aspect of the integrated methodology from the perspective of development lifecycle. These case studies have been realised using an instance of the Apache Tomcat servlet container for the deployment of web partners (Axis) and BPEL services (ActiveBPEL), and an instance of the GT4 container for grid partner services.

2.1 Web Service Case Study

The web service composition example comprises three composite business web services: LoanStar, DoubleQuote, and CarMen [6] that organise car deals and financial loans, with the capability of identifying car deals based on the customer's needs. Each of these composite web services and their partner services are useful in their own right, as the web service architecture obeys the SOA paradigm of loosely-coupled autonomous functionality.

LoanStar is a lender business that provides financial loans. These are subject to approval, with interest rates determined by the approval evaluation and/or risk assessment. LoanStar's lender business involves two partners FirstRate and RiskTaker. FirstRate has an existing and possible manual approver service which evaluates loan proposal requests, and rejects or approves loans with varying interest rates depending on its evaluation. RiskTaker provides the assessor service which carries out risk assessments of loan proposals.

DoubleQuote is a car supplier conglomerate service that facilitates car orders to match customer needs, and also supports cancellation of orders. Its business process involves two car dealers, BigDeal and WheelerDealer, each having their own car quotation, order and order cancellation operations.

The CarMen service is a broker solution that organises a car purchase and its financial loan, whereby only a successful car order and then its corresponding loan approval will constitute a successful transaction. CarMen will not proceed to make a loan proposal request should a car order be unsuccessful.

2.2 Grid Service Case Study

Occupational research is one specific sub-discipline of the research in social science. Often there are data management activities performed on the datasets, to prepare them for analysis. With regard to occupational data, one such data management task is the translation of occupational unit codes [2, 3]. There are many occupational classification schemes available, each usually favouring or the basis for a specific occupational analysis. Occupational datasets are commonly mapped to a desired classification in order to support a specific type of analysis.

The Allocator service is an example of a social science workflow underpinning data management in occupational research, preparing the data to be compatible for analysis. The Allocator composition involves two partner services: Factory that allocates resources containing occupational scheme information, and Mapper that performs the actual mapping of occupations using the allocated resources.

The following subsections presents the development lifecycle of each composite business process using the integrated methodology prescribed figure 2. Code snippets of specifications and implementations are given both in the body of the case studies and in corresponding appendices.

3 Development Of The Lender Composed Web Service

3.1 Introduction

This section describes the development lifecycle of LoanStar and its business partners FirstRate and RiskTaker. The informal description of their business operations is first given to capture the requirements. This is then followed by the development lifecycle, of the services where the integrated methodology is applied in the order of design, formal specification and analysis (validation and verification), implementation and testing.

LoanStar is a lender service that is similar to the classic loan approval process used as an example in the BPEL4WS standard [1]. LoanStar is a business process that combines two individual partner services FirstRate and RiskTaker as part of its behaviour.

FirstRate is the *approver* that evaluates loans and returns a loan rate when loan is approved. FirstRate operates as follows (purely as a hypothetical example):

- if applicant's name starts with Ken, the loan rate is 3.7%

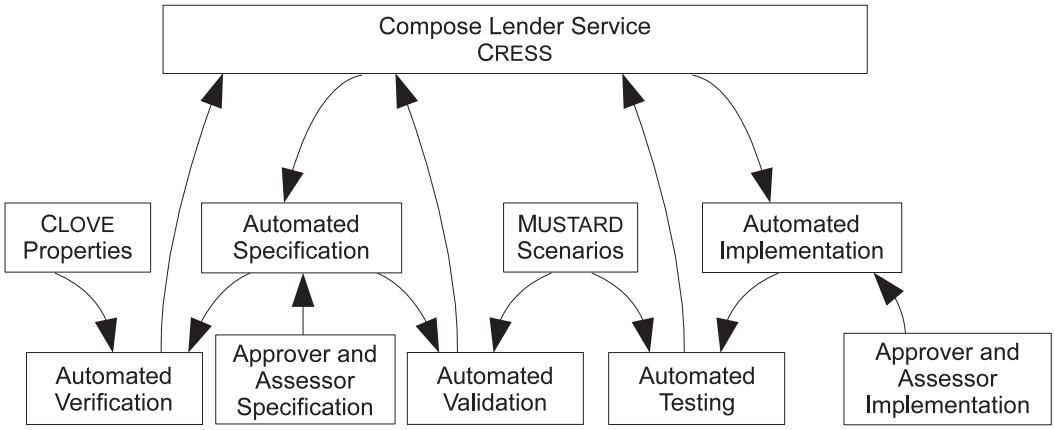


Figure 2: LoanStar Composite Service Development

- otherwise, the loan rate of 4.1% is given if the applicant's address includes Scotland
- otherwise, the loan rate of 4.4% is given if the proposal amount is less than 15000,
- otherwise the loan is refused with "loan unacceptable" as the reason.

RiskTaker is an *assessor* that assesses the risk of a proposal request, and returns the risk as a string. RiskTaker operates as follows (purely as a hypothetical example):

- if applicant's name ends with Turner, the risk is low
- otherwise, if applicant's address is a UK address, the risk is medium
- otherwise, the risk is high.

LoanStar combines FirstRate and RiskTaker in the following business logic. Loan approval evaluation is assumed to be costly, therefore an approval request is only made to FirstRate for proposals having an amount greater than 10000. For proposals seeking loans of 10000 or less, a risk assessment is made through the RiskTaker. If RiskTaker evaluates a low risk, then LoanStar immediately approves the loan proposal with a rate of 3.5%. All other risk levels will direct the proposal request to FirstRate for approval.

Figure 2 shows the process of the service development. The developer first describes the composed behaviour in CRESS. The composite service specification is then automatically generated, and the developer fills in the details of the outline behaviour for partner services FirstRate and RiskTaker. The developer also specifies the MUSTARD scenarios and CLOVE properties. The developer then executes the automated validation and verification, analyses the results, and addresses the issues identified by the analyses. Once satisfied with the analyses, the developer starts the automated implementation. The developer provides the actual implementation for the code skeletons generated for partner services FirstRate and RiskTaker. The developer now performs automated implementation validation and performance evaluation for the deployed services using the same MUSTARD scenarios as for the specification, addressing implementation and resource configuration issues that are discovered.

3.2 Design

The developer graphically defines the LoanStar business process and the configuration of its associated services. The services are named as lender (LoanStar), approver (FirstRate) and assessor (RiskTaker); these are the corresponding diagram names.

3.2.1 Service Diagram

The developer describes the behaviour of Lender as a CRESS diagram, shown in figure 3, with *lender* as its given name saved in the <CRESS>/ws/lender directory.

In the rule box, the data type and variable proposal2 are declared with Approver as owner, implying Approver's namespace, using the syntax 'proposal2:approver'. This is the input data structure for the approver.loan.approve

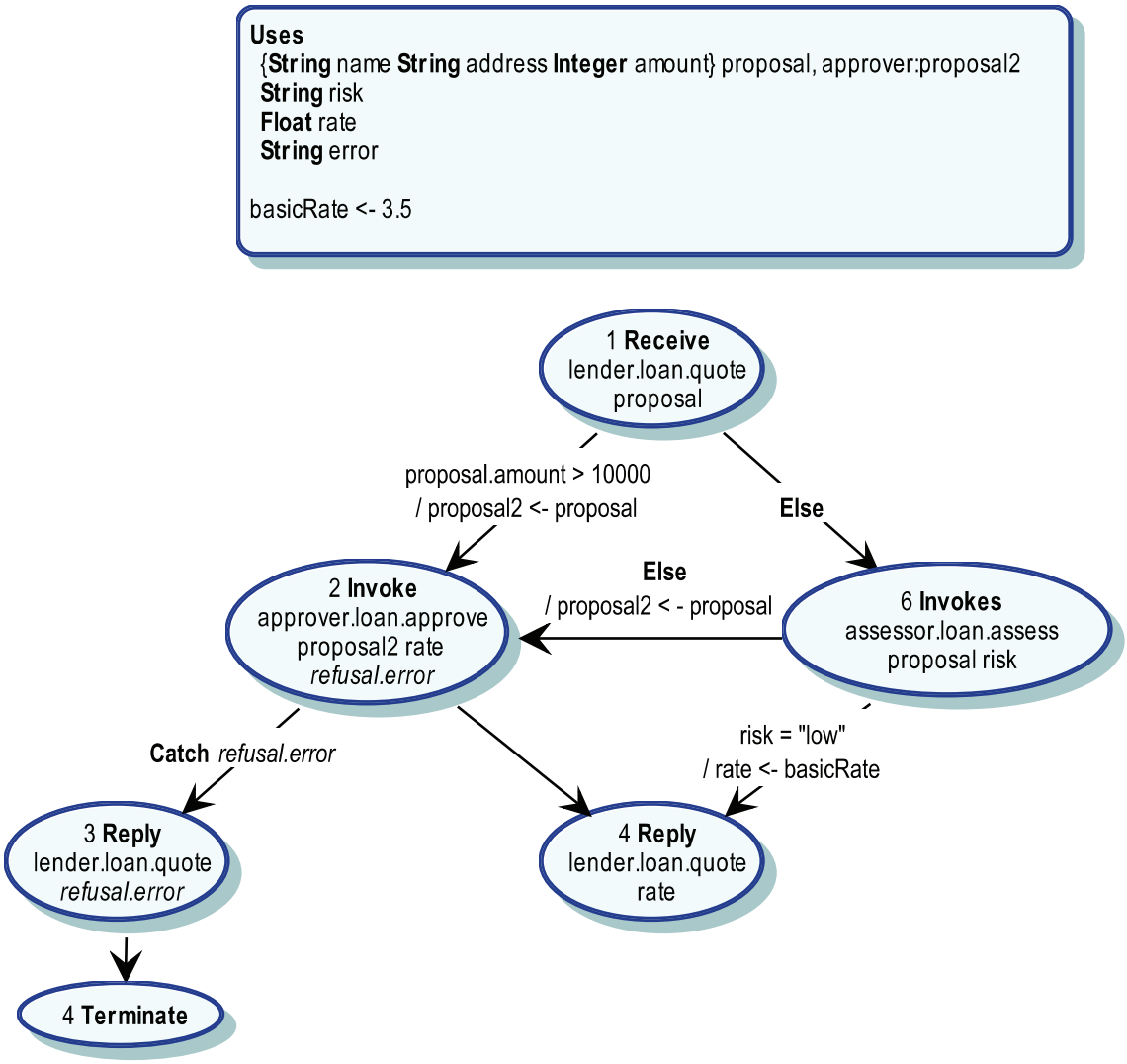


Figure 3: Lender CRESS Diagram

operation. Lender also defines its own Proposal type with identical structure but using the Lender namespace, the syntax omitting '*owner:*', which defaults to Lender as the current diagram. By doing so, Lender has 're-packaged' the entire service representation to its client, hiding away information about partners for the reason of trade secrets. Lender makes the 'proposal2' variable assignment from its 'proposal' data type in order to invoke Approver quote operation. The Assessor service has the same (LoanStar) owner and is developed with the Lender's namespace. Following these definitions are three variables named *risk*, *rate* and *error*, which are of the CRESS primitive types String and Float. A constant/macro *basicRate* is defined with the float value 3.5.

The process behaviour initiates from node 1, where Lender receives a loan quote request through the lender.loan.quote operation with a *proposal* data value. The value guard from node 1 to node 2 is followed if the proposal amount is over 10000 onwards. When this condition holds, an assignment is made to set the value of the 'proposal' variable to 'proposal2', which is used in the invocation of 'approver' at node 2. The '**Else**' on the right branch from node 1 caters for cases that do not meet the former condition linking to node 2. Two arcs emerges from node 2. If the invocation of approver.loan.approve throws a fault with name 'refusal' and a String type value, the arc labelled with '**Catch** *refusal.error*' is followed, leading to node 3. In this arc, the fault value of the String is set into variable *error*. The lender.loan.quote **Reply** returns the fault (node 3) and the process terminates (node 4). The arc to node 5 is followed if the invocation at node 2 is successful – the lender.loan.quote **Reply** returns the loan rate. At node 6, the Assessor is invoked for risk evaluation. A "low" risk satisfies the risk = "low" guarded arc, with an associated assignment of 3.5 to the rate: node 4 then returns. Otherwise the '**Else**' is followed, with an assignment made to 'proposal2' for the invocation of approver.loan.approve at node 2.

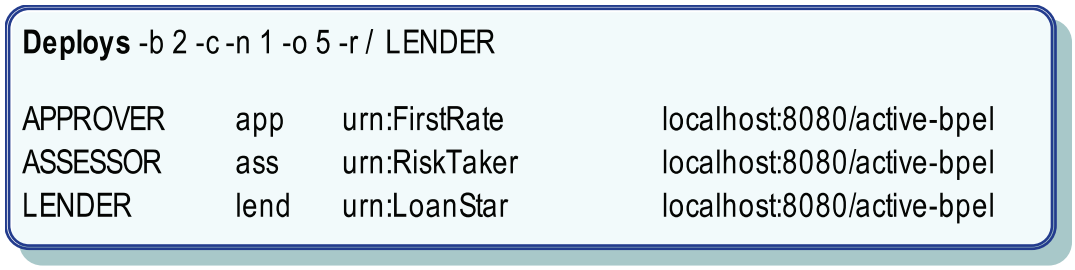


Figure 4: Web Service Configuration Diagram for Lender

3.2.2 Service Configuration

The developer now configures the Lender composite web service and its partners Approver and Assessor for formal specification and analysis, which is the immediate phase to follow. This is illustrated in the configuration diagram of figure 4. One instance of the Lender's composite behaviour is sufficient for formal validation analysis, specified with '-n 1' (number of instances: 1). As web services, these services should be always ongoing, and therefore the -r repeated option specifies that behaviour will repeat after the leaf node actions in the CRESS diagram. Comments will be generated in the specification, indicated by the '-c' comment option. The service to 'deploy' formally is the Lender composite service, which implies its partner services Approver and Assessor. Following the **Deploys** clause are the each service's parameters comprising the service name, namespace prefix, namespace, and deployment location.

3.2.3 Checking Diagrams

The diagrams are then checked by the CRESS tool for syntactic errors. This can be done directly within the CHIVE editor for CRESS diagrams. The initial checking of the Lender diagram reveals the following syntactic errors:

[Error] cress_check: LENDER − invalid declaration type for 'risk'
[Error] cress_check: LENDER − incomplete binding 'proposal2 < − proposal' guard node Else
[Error] cress_check: LENDER − invalid signal name 'Invokes' in node LENDER.6
[Error] cress_check: LENDER − node LENDER.4 is duplicated

The Lender CRESS diagram is then corrected accordingly by the developer. The primitive data type name for variable *risk* is corrected to String. Node 4 Reply is corrected to 5. Node 6 is corrected to use **Invoke**. The assignment expression '/ proposal2 < - proposal' (with a space between '<' and '-') is corrected to '/ proposal2 <- proposal'. No errors were found after running another check of the diagram.

3.3 Formal Specification

The formal specification is automatically generated. Figure 5 illustrates how the specification is obtained. The following two commands generates the Approver and Assessor LOTOS specification interfaces, and their formal specifications are then manually completed and automatically combined into the composed service specification of Lender. The following commands:

touch ws.base
tlotos ws
OR cress realise -t lotos -y ws

update the timestamp of the file ws.base to be more recent than ws.lot if it exists, meaning that the 'tlotos' (Topo LOTOS) command will (re-)generate the LOTOS specification automatically. The developer can manually specify the behaviour for Approver and Assessor as an extension of the default interface behaviour, described respectively in sections 3.3.1 and 3.3.2 as approver.lot and assessor.lot. The above two commands are executed again to achieve a complete specification for Lender that includes the manual specification of the partners.

3.3.1 Approver Specification

The following is the interface behaviour of Approver from the initial generation of the Lender specification. this is a self-contained LOTOS process:

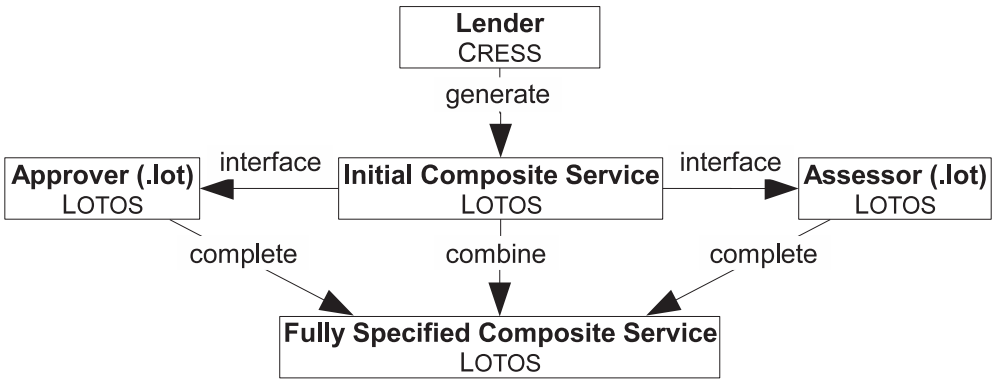


Figure 5: Specifying Approver and Assessor

```
Process APPROVER [approver] : Exit(States) :=      (* APPROVER partner *)          1
  approver !loan !approve ?proposal2:Proposal;     (* 'approve' input *)           2
  (                                                                                 3
    approver !loan !approve !AnyNumber;            (* 'approve' output *)          4
    APPROVER [approver]                            (* repeat behaviour *)          5
  []                                               (* or *)                        6
    approver !loan !approve !Refusal !AnyText;     (* 'Refusal' fault *)           7
    APPROVER [approver]                            (* repeat behaviour *)          8
  )                                                                                 9
EndProc                                            (* end APPROVER *)              10
```

As will be seen, the default behaviour of Approver contains no specific details but only operation requests (value passing) and responses (value offers) of *Any* values. The developer then manually adds the behaviourial details into this default behaviour. The specification is listed in Appendix A.1.1, and is saved as approver.lot. The Approver specification uses CRESS-defined LOTOS abstract data types and operations which are defined in the 'stir' (Stirling) library, and also the Proposal type which was automatically generated. These library and user-defined types are automatically declared and defined in the overall specification, i.e. Lender.

3.3.2 Assessor Specification

The specification of Assessor partner service is provided using the same approach as Approver. The following is the interface behaviour that is automatically generated for Assessor:

```
Process ASSESSOR [assessor] : Exit(States) :=      (* ASSESSOR partner *)          1
  assessor !loan !assess ?proposal:Proposal;       (* 'assess' input *)            2
  assessor !loan !assess !AnyText;                 (* 'assess' output *)           3
  ASSESSOR [assessor]                              (* repeat behaviour *)          4
EndProc                                            (* end ASSESSOR *)              5
```

The behaviour of the Assessor partner service is then manually specified by the developer using the above interface behaviour. The resultant specification, saved as assessor.lot, is listed in Appendix A.1.3, which models the RiskTaker operations.

3.3.3 Lender Specification

The Lender's service behaviour is fully and automatically generated. A high-level LOTOS specification is given in Appendix A.3.1 showing the Lender composite service that is generated. Since the specifications of Approver and Assessor are fully specified in their respective files, they are included into this overall specification.

The specification of the behaviour is centred on the Lender's composed behaviour as described in its CRESS diagram and service configuration that **Deploys** the Lender service. Therefore the specification has one gate (end-point) which is 'lender' that is externally visible, with Approver and Assessor hidden from the view of Lender's clients. The specification includes CRESS-defined LOTOS library data types (e.g. String), ports (e.g. loan), operations (e.g. assess), fault events (e.g. refusal), and user-defined complex data types (e.g. proposal).

Lender Automated	Approver Manual	Assessor Manual
412	31	19

Table 1: Specification of Lender, Approver and Assessor Services (no. of lines)

For the overall behaviour expression, the 'LENDER [lender]' LOTOS process defines the behaviour of Approver and Assessor synchronised with the behaviour of Lender. These are respectively specified as the APPROVER, ASSESSOR and LENDER_1 (i.e. Lender from node 1) processes. The services and their behaviour are autonomous and therefore independent from one another. By the behaviour synchronisation via gates '|[approver, assessor]|', service communication is established whereby LENDER_1 (the main entry to Lender behaviour) can communicate with APPROVER and ASSESSOR. The partners Approver and Assessor do not communicate with each other and therefore they are interleaved using the '|||' operator.

The Lender itself is viewed externally as a business process by its clients. The involvement of other services need not be made known to Lender's clients. This is modelled in the LOTOS process LENDER. The relationship amongst the behaviour of LENDER, APPROVER and ASSESSOR can be explained as the following: APPROVER and ASSESSOR are interleaved as a behaviour unit with their communication hidden in synchronisation with the LENDER behaviour represented by LENDER_1. APPROVER and ASSESSOR are autonomous services and have no communications between themselves (and are therefore fully interleaved using the '|||' operator); their communication endpoints are represented by their respective LOTOS gates. The behaviour of APPROVER and ASSESSOR is enclosed within round parenthesis, representing a logical unit of behaviour. As the Lender service employs both services, their grouped behaviour is synchronised via their respective named gates where interaction with the Lender can take place.

3.3.4 Summary

Apart from manual specification of partners Approver and Assessor, the formal specification is fully automated such that all data types and behaviour are specified. Even the partner specification itself is added to interface behaviour that is automatically generated. Using this specification, formal validation and verification can be performed.

Table 1 lists the code summary in terms of lines of code generated and manually provided.

3.4 Formal Analysis

This subsection describes the formal validation and verification of the three services and how the methodology is used.

3.4.1 Formal Validation

Validation is then performed on the composed and partner services. There is no strict order for which service is to be validated. However, given the relationship of these three services, it is simpler and logical to validate the partner services first before the composed service, which validates the components first then their integration. For this case study, the order of validation was Approver, Assessor and then Lender.

The developer defines MUSTARD scenarios for Approver, Assessor and Lender respectively as .mstd files of the same name. These defined scenarios are a form of testing on Approver, Assessor, and Lender, whereby various service invocations and reading of the correct responses are required to assert a pass in validation.

The manual specification of Approver was syntactically correct when included into the specification; no errors were reported by the 'tlotos' tool when it attempted to load the specification into Lola to parse its syntax. However the behaviour of Approver should be validated as to its behaviour. Below are the informal descriptions of the Approver scenarios for testing various aspects of the Approver behaviour.

- A client named 'Ken Smith' from 'Liverpool UK' seeking a loan amount of 6000 should receive a rate 3.7%
- A client named 'Angus Og' from 'Airth Scotland' seeking a loan amount of 20000 should receive a rate 4.1%

- A client named 'Nancy Turner' from 'Manchester England' seeking a loan amount of 14999 should receive a rate 4.4%
- A client named 'Ian Carey' from 'Croydon England' seeking a loan amount of 15000 should be rejected with a refusal fault containing a string message 'loan acceptable'.

The following are the corresponding MUSTARD scenarios specified in the file approver.mstd which resides in the same location as the Lender CRESS diagram and approver.lot.

```
test(Low_Rate,
  succeed(
    send(approver.loan.approve,Proposal2/Proposal('Ken Smith,'Liverpool UK,6000.)),
    read(approver.loan.approve,3.7)))

test(Medium_Rate,
  succeed(
    send(approver.loan.approve,Proposal2/Proposal('Angus Og,'Airth Scotland,20000.)),
    read(approver.loan.approve,4.1)))

test(High_Rate,
  succeed(
    send(approver.loan.approve,Proposal2/Proposal('Nancy Turner,'Manchester England,14999.)),
    read(approver.loan.approve,4.4)))

test(Loan_Unacceptable,
  succeed(
    send(approver.loan.approve,
         Proposal2/Proposal('Ian Carey,'Croydon England,15000.)),
    read(approver.loan.approve,refusal,'loan unacceptable)))
```

The above MUSTARD scenarios each comprise a **send** and **read** primitive. The service name (approver), port (loan), operation (approve), data types (proposal, string, float), and fault name (refusal) correspond to those defined in the Lender CRESS diagram. The Proposal2/Proposal syntax was specified as the data type definition for Proposal (i.e. type 'Proposal2' constructs 'Proposal'). As Proposal and Proposal2 have the same structure, the abstract data type collapsed as one data type using Proposal, even for the Approver service, which simplifies the specification (e.g. assignment) but does not affect the behaviour from the perspective of valdiation. 'Proposal2/' is still specified as the MUSTARD scenarios are reused in the validation of Approver's implementation where the actual data type has to be used in the web service interaction. MUSTARD scenarios are automatically translated into Lola-compatible tests which are self-contained LOTOS processes, merged together with the overall specification, and then validated. The translated LOTOS processes are listed in Appendix A.4.1.

The developer then executes the automated validation for Approver. The following are the MUSTARD interpreted results from the formal validation performed by Lola.

```
Test APPROVER Low Rate ...          Fail          0 succ   1 fail   1.3 secs

send(approver.loan.approve,Proposal2/Proposal('Ken Smith, 'Liverpool UK, 6000.))
<failure point>

Test APPROVER Medium Rate ...       Pass          1 succ   0 fail   1.4 secs
Test APPROVER High Rate ...         Pass          1 succ   0 fail   7.5 secs
Test APPROVER Loan Unacceptable ...Pass           1 succ   0 fail   1.9 secs
```

In the results, the Low_Rate scenario did not pass the validation. The diagnostic trace of Low_Rate demonstrates that the scenario is unable proceed after the first action that is **send**(approver.loan.approve, Proposal2/Proposal('Ken Smith, 'Liverpool UK, 6000.)). The next action is **read**(approver.loan.approve, 3.7) is unsuccessful, indicating that the value returned is not 3.7. As the validation is targetted at Approver, the specification in approver.lot is inspected. It is found that the behaviour at line 5 of approver.lot in Appendix A.1.1 specifies the event value offer approver !loan !approve !number(+,t(3), t(8)) when the validation scenario expects 3.7. The specification is corrected to approver !loan !approve !number(+,t(3),t(7)) as that is the intended rate to be returned by Approver as a requirement. The validation is executed again and all the Approver MUSTARD scenarios passed.

The developer now moves on to validate Assessor. The following are the informal validation scenarios to test the various possible response behaviours of Assessor:

- The assessment for client named 'Mark Turner' from 'Carlisle UK' for loan amount 20000 should be 'low' risk.
- The assessment for client named 'Fred Hoyle' from 'UK' for loan amount 5000 should be 'medium' risk.
- The assessment for client named 'Patrice Touvet' from 'Paris France' for loan amount 1000 should be 'high' risk.

The following are the corresponding MUSTARD scenarios specified in the file assessor.mstd. They are similar apart from values which validate the three possible responses of Assessor. Their corresponding LOTOS translations are listed in Appendix A.4.2.

```
test(Low_Risk,
  succeed(
    send(assessor.loan.assess,Proposal('Mike Turner,'Carlisle UK,20000.)),
    read(assessor.loan.assess,'low)))

test(Medium_Risk,
  succeed(
    send(assessor.loan.assess,Proposal('Fred Hoyle,'UK,5000.)),
    read(assessor.loan.assess,'medium)))

test(High_Risk,
  succeed(
    send(assessor.loan.assess,Proposal('Patrice Touvet,'Paris France,1000.)),
    read(assessor.loan.assess,'high)))
```

These three validation scenarios pass with the following MUSTARD results which are obtained within seconds.

Test ASSESSOR Low Risk ...	Pass	1 succ	0 fail	1.3 secs
Test ASSESSOR Medium Risk ...	Pass	1 succ	0 fail	0.4 secs
Test ASSESSOR High Risk ...	Pass	1 succ	0 fail	0.4 secs

The Lender behaviour is a logical composition comprising the Approver and Assessor. Although validation of Lender may imply these partner services, it does not directly validate their behaviour, which is why they also have their own validation performed apart from Lender. The following are informal descriptions of the validation scenarios for Lender.

- A client named 'Nancy Turner' from 'Manchester England' for an amount of 9999 should be at rate 3.5%.
- A client named 'Nancy Turner' from 'Manchester England' for an amount of 10000 should not require risk assessment. This client should therefore be offered the rate 3.5 as the client's name ends with "Turner".
- A client named 'Ken Boyle' from 'Dublin Ireland' for an amount of 10000 should be at rate 3.7%.
- A client named 'Mary Duncan' from 'Wick Scotland' for an amount of 20000 should be at rate 4.1%.
- A client named 'Sally Dean' from 'Cardiff Wales' for an amount of 14999 should be at rate 4.4%.
- A client named 'Ian Carey' from 'Croydon England' for an amount of 15000 should be be refused with a fault name refusal and message 'loan unacceptable'.

Their following MUSTARD scenarios are defined by the developer in the file lender.mstd, residing in the same location as the Lender CRESS diagram.

```
test(Little_Low_Risk,
  succeed(
    send(lender.loan.quote,Proposal('Nancy Turner,'Manchester England,9999.)),
    read(lender.loan.quote,3.5)))
```

```
test(No_Risk_Assess_Low,
  refuse(
    send(lender.loan.quote,Proposal('Nancy Turner,'Manchester England,10000.)),
    read(lender.loan.quote,3.5)))

test(Lots_Ken,
  succeed(
    send(lender.loan.quote,Proposal('Ken Boyle,'Dublin Ireland,10000.)),
    read(lender.loan.quote,3.7)))

test(Lots_Scotland,
  succeed(
    send(lender.loan.quote,Proposal('Mary Duncan,'Wick Scotland,20000.)),
    read(lender.loan.quote,4.1)))

test(Lots_Under_15000,
  succeed(
    send(lender.loan.quote,Proposal('Sally Dean,'Cardiff Wales,14999.)),
    read(lender.loan.quote,4.4)))

test(Lots_Exceeds_15000,
  succeed(
    send(lender.loan.quote,Proposal('Ian Carey,'Croydon England,15000.)),
    read(lender.loan.quote,refusal,'loan unacceptable)))
```

These MUSTARD scenarios are translated into the LOTOS test scenarios which are listed in Appendix A.4.3. They are similar and straightforward with the exception of No_Risk_Assess_Low, therefore its LOTOS specification is illustrated below. The MUSTARD scenario uses the **refuse** construct. This indicates that the last event which is **read**(lender.loan.quote,3.5) should not happen as the Lender behaviour logic should not lead to any risk assessment for amounts greater or equal to 10000. The **refuse** is translated in the Lola test process as a choice behaviour. If the event 'lender !loan !quote !Number(+,t(3),t(5))' should synchronise, then the behaviour will stop without an OK event, implying the validation did not pass. Otherwise the alternative behavioural path is followed which asserts the OK event indicating the scenario passes.

```
Process LENDER_Little_Low_Risk [lender,OK] : NoExit :=                                    1
  lender !loan !quote !proposal(t(N)~a~n~c~y~ ^ ~T~u~r~n~e~r,                              2
  t(M)~a~n~c~h~e~s~t~e~r~ ^ ~E~n~g~l~a~n~d,Number(+,t(9)~9~9~9,< >));                      3
  lender !loan !quote !Number(+,t(3),t(5));                                               4
  OK;                                                                                     5
  Stop                                                                                    6
EndProc (* LENDER_Little_Low_Risk *)                                                     7
```

The validation results below show that only the No_Risk_Assess_Low scenario does not pass, and there is a diagnostic trace for the validation scenario.

```
Test LENDER Little Low Risk ...      Pass          1 succ 0 fail    1.3 secs
Test LENDER No Risk Assess Low ...   Inconclusive  1 succ 1 fail    0.4 secs

send(lender.loan.quote,Proposal(nancy turner,manchester england,number(+,10000,)))
read(lender.loan.quote,Number(+,3~,5~))
<failure point>

Test LENDER Lots Ken ...             Pass          1 succ 0 fail    0.4 secs
Test LENDER Lots Scotland ...        Pass          1 succ 0 fail    1.0 secs
Test LENDER Lots Under 15000 ...     Pass          1 succ 0 fail    0.8 secs
Test LENDER Lots Exceeds 15000 ...   Pass          1 succ 0 fail    2.0 secs
```

The validation outcome for No_Risk_Assess_Low is Inconclusive, with one success path and one failure path. The diagnostic trace shows the failure path which indicated that the rate 3.5% returned by the Lender

behaviour was unexpected. As a result the path behaves as 'stop' (line 15 of Appendix A.4.3, which is the LENDER_No_Risk_Assess_Low LOTOS process). The reason for the one success path is due to the non-determinism specified in the internal event 'i' as the one of the first events in the choice operator. This event will be tried since all possible paths in the validation behaviour are tried. As 'i' events are always successful, this path leads to OK indicating a success. Should the read for rate 3.5 fail to synchronise, then this is only one path that is the successful one, therefore the validation scenario will pass.

The failure of No_Risk_Assess_Low indicates that the behaviour of Lender is not doing the right thing. Given the context of the problem, rate 3.5 is returned by Lender only if a risk assessment is made by Assessor, as there is no such rate returned by the Approver by inspecting the specification. The Lender CRESS diagram is inspected as Assessor was invoked as part of its logical behaviour. There are only two guards from node 1 in the Lender diagram where there is a condition of 'proposal.amount $>$ 10000' leading to node 2; otherwise it leads to node 6 via the **Else** guard. From the No_Risk_Assess_Low scenario description, the proposal amount is 10000 in the **send**. From the requirements of Lender, this should go to the Approver for a loan approval instead of the risk assessment. It was found that the guard condition 'proposal.amount $>$ 10000' should be 'proposal.amount $>=$ 10000' to implement this requirement correctly. After amending the Lender diagram, the validation is re-run and then all scenarios pass.

The automated formal validation executes with results in a few seconds, which is very effective for analysing the specification and detecting errors contradicting the scenarios specified.

3.4.2 Formal Verification

The same LOTOS specification is used for formal verification of the composed behaviour in the methodology. However, the specification has to be automatically annotated for CADP. Appendix A.5.1 lists a snippet of the annotated LOTOS, illustrating the annotation of the Proposal type. As the composed service Lender uses only non-CRESS partner services which are fairly simple, there is no need to perform verfication using compositional mode. The following are the informal descriptions of the verification properties desirable for the system. The developer then defines the CLOVE properties and executes the verification in non-compositional mode.

- Service should be free from deadlock.
- Service should be free from livelock.
- The service should start only with the signal lender.loan.quote accepting only Proposal values.
- All Proposal requests should receive either a reply of a rate (any number) or a refusal fault with message "loan unacceptable".
- A more specific response property is that valid responses for any Proposal values are either the rates 3.5, 3.7, 4.1, 4.4, or the refusal fault of message "loan unacceptable".
- Proposals having client names containing 'Ken' that request loan amounts of 9999 and below should be offered the rate of 3.5.

The following are the CLOVE enumeration and properties defined by the developer. The **initials** define the permitted signals during the start of the service, which will be used by the initial safety property that is built into CLOVE. The other explicitly specified properties were constructed using the global response verification template. Deadlock and livelock freedom are checked by default in CLOVE.

```
initials(signal(lender.loan.quote,?proposal))

enumerate_complex(
proposal\(\"(KEN TURNER|LARRY TAN)\",\"UK\",[5|7]0000\.0\)
proposal\(\"(KEN TURNER|LARRY TAN)\",\"UK\",[5|7]000\.0\)
...
)

property(General_Response,
   response(global,
      signal(lender.loan.quote,?proposal),
```

```
      choice_any(signal(lender.loan.quote,?number),
        signal(lender.loan.quote,refusal,'loan unacceptable'))))

property(Specific_Response,
  response(global,
    signal(lender.loan.quote,?proposal),
    choice_any(signal(lender.loan.quote,3.500000),
      signal(lender.loan.quote,3.700000),
      signal(lender.loan.quote,4.100000),
      signal(lender.loan.quote,4.400000),
      signal(lender.loan.quote,refusal,'loan unacceptable'))))

property(All_Ken_No_Risk_Approve,
    response(global,
      ''LENDER !LOAN !QUOTE'' # '' !''
      # 'PROPOSAL (''.*KEN.*'', ''.*'', [1-9][0-9][0-9][0-9][.].*)',
      signal(lender.loan.quote,3.500000)))
```

External observable inputs to the specification are provided for the generation of the state space for verification. The only data structure that is externally observable as input is the Proposal data type. The enumeration for type Proposal is provided where the values will be used in the explicit state space generation in the verification process. From the two patterns the enumeration of the Proposal values are as follows:

```
Proposal(''KEN TURNER'',''UK'',50000.0)
Proposal(''KEN TURNER'',''UK'',70000.0)
Proposal(''LARRY TAN'',''UK'',50000.0)
Proposal(''LARRY TAN'',''UK'',70000.0)
```

The translated μ-calculus and a high-level code snippet of the C implementation for values are listed in Appendix A.5.2 and A.5.3.

Initially, deadlock and livelock freedom are first checked prior to CLOVE properties, to get a general verification of the behaviour of the system prior to more detailed checks. The verification returns the following results:

```
Verifying             DEADLOCK FREEDOM ... FALSE
''LENDER !LOAN !QUOTE !PROPOSAL (''LARRY TAN'', ''UK'', 50000.000000)''
''i'' ''(i)''
''i'' ''(i)''
''LENDER !LOAN !QUOTE !REFUSAL !''LOAN UNACCEPTABLE''''
''LENDER !LOAN !QUOTE !PROPOSAL (''LARRY TAN'', ''UK'', 70000.000000)''
DEADLOCK

Verifying             LIVELOCK FREEDOM ...  TRUE
```

Model checking the behaviour produces a counter-example for the freedom of deadlock. The path trace shows that the first loan request results in a response, but a subsequent request which is denoted by ''LENDER !LOAN !QUOTE !PROPOSAL (''LARRY TAN'', ''UK'', 70000.000000)'' can proceed no further, nor even any internal events at all. By observing the CRESS diagram, all loan requests value greater or equals to 10000 will not be directed to the Assessor at all, which narrows down the behaviour to trace. A MUSTARD scenario was quickly defined to represent the same value as the subsequent request. Validation achieved a Pass within less than a second:

```
test(Lots_Larry,
  succeed(
    send(lender.loan.quote,Proposal('Larry Tan,'UK,70000.)),
    read(lender.loan.quote,refusal,'loan unacceptable)))

Test LENDER Lots Larry ...          Pass          1 succ   0 fail   0.5 secs
```

This result narrows down to the behaviour handling of a second request, where there is no way to proceed further, in this case with Approver. The Approver's specification was checked and it was found that Approver's behaviour (Appendix A.1.1) was specified with a **Stop** (line 23) instead of repeating the behaviour after responding with the first "loan unacceptable" fault. The specification, which was manually completed by the analyst, was

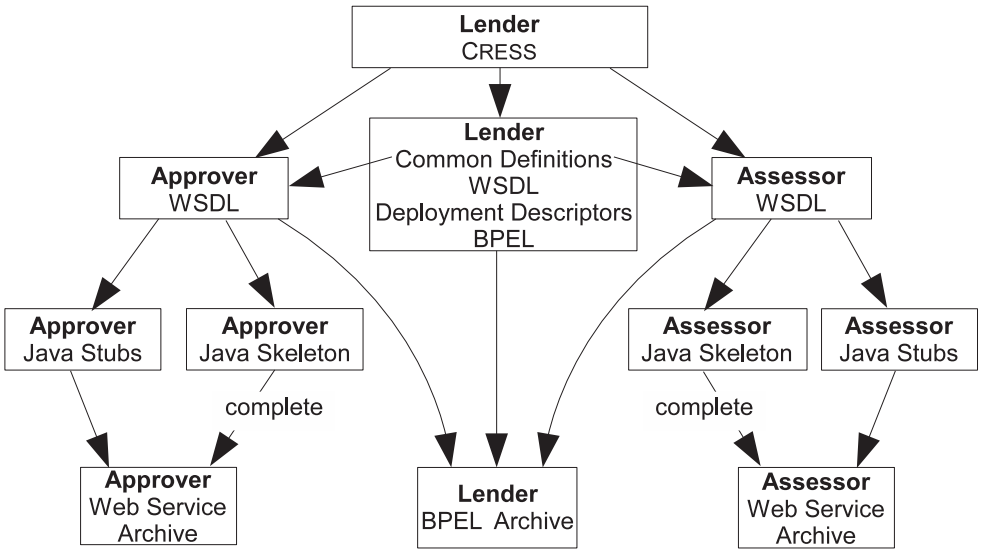


Figure 6: Implementation of Approver and Assessor

corrected to repeat the Approver's behaviour, as it is a web service whose functionality is supposed to be always alive. The verification for deadlock and livelock freedom is performed again, and the entire composite service behaviour is found to be free from deadlock.

The verification is performed by the developer, running the following command within the directory containing Lender CRESS diagram:

cress_verify -t lotos -v ws -i -o lender

The outcome from verification of the properties is listed below, for the properties verified in the service behaviour:

Verifying LENDER GENERAL RESPONSE ...	TRUE
Verifying LENDER SPECIFIC RESPONSE ...	TRUE
Verifying LENDER ALL KEN NO RISK APPROVE ...	TRUE
Verifying DEADLOCK FREEDOM ...	TRUE
Verifying LIVELOCK FREEDOM ...	TRUE
Verifying INITIALS SAFETY ...	TRUE

3.5 Implementation

This step corresponds to the implementation aspect of the methodology where there is automated generation of code and deployment. As Approver and Assessor partner services are non-CRESS, their implementations were provided (they are 'in-house' services and therefore not externally owned). Approver has its own Proposal2 complex data type. Assessor uses the data types defined in Lender. Figure 6 illustrates how this is achieved for Approver and Assessor. Service implementation code skeletons are automatically generated and manual code is then added. These implementation details of Approver and Assessor are respectively described in section 3.5.1 and 3.5.2. The implementation of Lender is fully automated.

To engage the implementation phase, the developer configures the service configuration diagram with the following parameters:

Deploys -c -b 2 -o 5 / LENDER

This configuration **Deploys** the Lender service which implies the Approver and Assessor as they are used in its behaviour. The BPEL implementation of all composed services, in this case only Lender, is described in the WS-BPEL 2.0 standard (-b 2). As there is intent to perform implementation validation, the timeout threshold is set to five seconds (-o 5) which generates the appropriate MINT validation configuration for all the services specified to be deployed (Approver, Assessor and Lender). The code will be generated with comments (-c).

The developer prepares empty directories 'approver' and 'assessor' in the same location as the Lender CRESS diagram. These directories will be used to hold the actual service implementation later. The developer executes the following command-line in the CRESS 'bpel' directory for the automated implementation:

cress_expand -v ws main.bpel

The above command automates the generation of the implementation for Lender and all its partners as configured. Service interfaces (WSDL), deployment configuration files (WSDD and PDD), code stubs translated from WSDL, and partner service code skeletons are generated in this process. In the current directory (normally 'bpel'), a 'lender' directory is created containing all the implementation code for the three services. Within this 'lender' directory there are the 'approver' and 'assessor' directories which respectively contain their service interface, deployment files, Java code stubs and service implementation code skeleton in the directory structure according to their namespace/package definition. Appendix A.6.1 lists the file structure and the code listing of all the files generated. The service interfaces and common definition WSDL are in Appendix A.6.2, A.6.3, and A.6.5.

The full implementation of Approver and Assessor are then provided by the developer, described respectively in section 3.5.1 and 3.5.2. The automated implementation is executed again with the -d option which automatically deploys all the services.

3.5.1 Approver Implementation

The Approver's code skeleton was generated in lender/approver/FirstRate/ LoanBindingImpl.java under CRESS's 'bpel/lender' directory. The developer will build upon this implementation, but as lender/approver/FirstRate/approver.java under CRESS's web service domain directory (ws). The following is the Java code skeleton generated for approver.java. This code skeleton is translated from the Approver's WSDL which was automatically generated:

```
package FirstRate;
...
public class LoanBindingImpl implements FirstRate.LoanPort{
  public float approve(FirstRateDefs.Proposal2 proposal2)
    throws java.rmi.RemoteException, LoanStarDefs.StringMessage {
    return -3;
  }
}
```

The code skeleton provides a dummy implementation in which the deployable web service archive approver.wsr is created as part of the initial implementation generation process. The developer adds the actual implementation into this code skeleton and runs the automated implementation, which then includes the actual functionality. The implementation by the developer, as approver.java, is listed in Appendix A.2.1.

3.5.2 Assessor Implementation

As illustrated in figure 2, the implementation of Assessor using its generated code skeleton is performed along with the implementation for Approver. This is similarly done by adding the implemention to its generated code skeleton. The completed implementation by the developer, as <CRESS>/ws/lender/assessor/assessor.java, is listed in Appendix A.2.3.

3.5.3 Lender Implementation

The implementation of Lender is fully automated in this phase, which generates the WSDL interface, BPEL code, and deployment descriptor files (WSDL catalogue and PDD). These are compiled together as a BPEL archive lender.bpr deployable into the ActiveBPEL container. The WSDL and BPEL code are respectively listed in Appendix A.6.4 and A.6.6. The descriptors are standard ActiveBPEL descriptions but are not listed for brevity.

3.5.4 Summary

Table 2 shows the number of lines of code and files that are automatically produced and that are manually provided in the implementation phase. The implementation for the three services is highly automated apart from the manual implementation of partners, which is added into the generated code skeletons. The automation of the implementation took less than a minute to produce the deployable service archives.

3.6 Implementation Validation

The implementation of the services can be validated/tested after deployment in the ActiveBPEL container. The MUSTARD scenarios (approver.mstd, assessor.mstd, and lender.mstd) for the three services have already been

Service	WSDL (incl. partners)	Java	Deployment	BPEL
Approver	2 files (136 lines)	8 files (738 lines)	34 lines	–
Assessor	2 files (129 lines)	8 files (657 lines)	34 lines	–
Lender	4 files (284 lines)	–	2 files (66 lines)	192 lines

Table 2: Code Generation Summary of Approver, Assessor and Lender Service

defined in the formal validation. The MINT validation configuration properties are generated as part of the implementation process, therefore the services can be immediately validated. The developer starts the automated validation of the implementation for the partner services (Approver and Assessor) and then the Lender composite service.

cress_validate -v ws -t bpel ws

The following subsubsections discuss the implementation validation setup for the three services, followed by their implementation validation. The performance evaluation of the Approver, Assessor and Lender service are carried out after functionality is validated.

3.6.1 Approver Implementation Validation

The Approver's MINT configuration file is automatically generated by the implementation generation, and is listed in Appendix A.7.1. The MUSTARD scenarios for the Approver service were already described in section 3.4.1, and their translation to MINT scenarios are listed in Appendix A.7.2.

The validation of each of the MINT scenarios for Approver is carried out. The results of the MINT validation are:

```
Test APPROVER Low Rate ...          Pass    1 succ   0 fail   1.1 secs
Test APPROVER Medium Rate ...       Pass    1 succ   0 fail   1.1 secs
Test APPROVER High Rate ...         Pass    1 succ   0 fail   1.1 secs
Test APPROVER Loan Unacceptable ...Fail     0 succ   1 fail   1.1 secs

send(approver.loan.approve,Proposal2(″Ian Carey″,″Croydon England″,15000.))
<failure point>
```

From the results, only the Loan_Unacceptable scenario did not pass the implementation validation. From the diagnostic trace provided, the invocation denoted by the **send** did not receive (**read**) a refusal fault with "loan unacceptable" message. As this operation is validating only the Approver, its implementation is inspected. It is found that the condition for a loan of 4.4% was 'amount ¡= 15000' (Appendix A.2.1 line 30) which the invocation of this scenario causes to be executed. This was not right considering the requirements of Approver. This condition is therefore followed by re-creation and redeployment of Approver. Thereafter the implementation validation is executed again and all the scenarios pass, showing consistency with the corresponding formal validation.

3.6.2 Assessor Implementation Validation

Similarly to Approver, the Assessor's MINT configuration file is generated automatically, as listed in Appendix A.7.3. Its translated MINT scenarios from the MUSTARD scenarios described 3.4.1 are listed in Appendix A.7.4. The results from automated validation of Approver are as follows:

```
Test ASSESSOR Low Risk ...              Pass 1 succ 0 fail 1.1 secs
Test ASSESSOR Medium Risk ...           Fail 0 succ 1 fail 1.1 secs

send(assessor.loan.assess,Proposal(″Fred Hoyle″,″UK″,5000.))
<failure point>

Test ASSESSOR High Risk ...             Pass 1 succ 0 fail 1.1 secs
```

The validation of the Assessor scenarios have successfully passed except for Medium_Risk which fails to receive the "medium" assessment. This pinpoints the Assessor implementation, where it is found that the return value for "UK" addresses is mispelled as "medum". This is corrected to "medium", and the Assessor is re-created and redeployed via the automated implementation. Thereafter, the validation of all scenarios is successful, showing consistency with the formal validation.

3.6.3 Lender Implementation Validation

Similarly to the Approver and Assessor, the implementation validation of the Lender service requires its MUSTARD scenarios and MINT configuration file. The MUSTARD scenarios for the Lender service, already described in section 3.4.1, are reused. The configuration and translated MINT scenarios are in Appendix A.7.5 and A.7.6.

The following are the implementation validation outcomes:

Test LENDER Little Low Risk ...	Pass	1 succ	0 fail	1.9 secs
Test LENDER No Risk Assess Low ...	Pass	1 succ	0 fail	1.2 secs
Test LENDER Lots Ken ...	Pass	1 succ	0 fail	1.2 secs
Test LENDER Lots Scotland ...	Pass	1 succ	0 fail	1.2 secs
Test LENDER Lots Under 15000 ...	Pass	1 succ	0 fail	1.2 secs
Test LENDER Lots Exceeds 15000 ...	Pass	1 succ	0 fail	1.2 secs

The validation scenarios of the Lender have all passed. This is attributed to the earlier formal validation which corrected in particular the No_Risk_Assess_Low scenario which led to correcting the service behaviour description in the Lender CRESS diagram. From the first round of implementation validation, it was noted that Assessor was not returning the correctly spelled "medium" risk, but that did not affect the behaviour of Lender. This was because the condition guards used in Lender were not "low".

3.6.4 Performance Evaluation

After the functional validation is satisfactory for all three services, the performance evaluation can be carried out to inspect the deployment configuration of the service environment and consistency of service behaviour under load. It is expected that the services should be able to handle up to 150 simulataneous requests and respond within five seconds. This is indicated by the service.timeout in the MINT configuration files, which was specified by the '-o 5' option in the service configuration diagram.

The performance evaluation is first carried out in sequential mode for 150 runs per scenario. This establishes as initial confidence that the service functionality is consistent under a series of invocation. The developer executes the following command:

cress_validate -v ws -t bpel -ps150 lender

The results of the sequential load testing below showed that the service implemented behaviour is consistent for a series of 150 invocations which are very similar for all the rest of the sequential performance evaluation.

Test APPROVER Low Rate ...	Pass		150 succ	0 fail	0.1 secs
	Sequential	0 inco	true	cons 0.0 secs .. 1.1 secs	
...					

The concurrent performance mode is then tried, implying the configuration of the parameters '-pc150' for concurrent performance mode. The concurrent performance evaluation for partner service Approver and Assessor pass with a load of 150. The concurrent performance evaluation for Lender has reported results of Inconclusive, meaning inconsistency of behaviour, and Fail, with reported Java exceptions of "java.net.SocketTimeoutException: Read timed out" and "java.net.ConnectException: Connection refused: connect". The former means invocation timeout and the latter means unable to connect at all. Overall, the results indicate that under the specified concurrent load of 150 the Lender service is not able to function consistently. The hosting server (Apache Tomcat) was observed to report "All threads are busy, waiting. Please increase maxThreads or check the servlet status" and "java.lang.OutOfMemoryError: Java heap space" errors when concurrent validation was the only activity in progress at the server. This reveals that the server is not configured adequate resources to host the web services with the expected performance requirements. The java.net.SocketTimeoutException was most likely due to the server not being able to process the request in time within the timeout threshold that was set. Over time, the Java heap space ran out for the engine, which led to the java.net.ConnectException as the container could not acquire memory resources for processing. The reason that the concurrent performance evaluation on Approver and Assessor have achieved Pass is because the resources configured was adequate enough (maxThreads at 150 and default

JVM heap size are sufficient) to execute them successfully. Lender, being a composite service, involves invoking Approver and also Assessor services and therefore will require more resources (in this case threads and memory heap). The server was then configured with adequate threads and Java heap size by respectively increasing the maxThreads attribute in Tomcat's server.xml and Java heap size for the server (e.g. via environment variable CATALINA_OPTS). The concurrent performance evaluation was executed again, and the service exhibited consistency under this performance load.

3.7 Summary

The formal verification analysis detected errors that were not covered by the validation. This is because the nature of certain analyses are easier with verification in that the properties are concise but cover a scope that is difficult or impossible with validation, such as deadlock and global response where there would be an infinite number of scenarios.

4 Development Of The Supplier Composed Web Service

4.1 Introduction

This section describes the development lifecycle of DoubleQuote and partners BigDeal and WheelerDealer, which are all newly developed services.

DoubleQuote (supplier) is a composed service that involves two car dealer partners: BigDeal (dealer1) and WheelerDealer (dealer2). All these three services are to be developed under the assumption that there is no specific service ownership. Both dealers have a variety of offers for cars, giving the price and the delivery as quotes for car needs. the DoubleQuote business process will initiate with a car need request which comprises the customer's name, address, and the car model. Upon receiving the need for a car, DoubleQuote will seek quotations from both dealers. A quote offer contains a reference number, dealer identifier, price and delivery (in days). The returned quotes are compared by price, favouring the cheaper. If the prices are identical, then the quote offering faster delivery is favoured. The dealer for the chosen quote is then invoked with an order by DoubleQuote. The offer is returned to DoubleQuote's customer as a reference. The DoubleQuote service allows its customers to cancel the car order, which leads to arranging the car order cancellation with the associated dealer.

BigDeal (*dealer1*) has the following offers for cars:

- Mondeo: price 20000, 15-day delivery
- A5: price 33000, 30-day delivery
- Megane: price 11000, 5 day delivery
- Others: price infinite (1000000), 0 day delivery

WheelerDealer (*dealer2*) operates as follows:

- Mondeo: price 20000, 10-day delivery
- A5: price 35000, 20-day delivery
- Astra: price 18000, 30 day delivery
- Others: price infinite (1000000), 0 day delivery

The development lifecycle of the services follows the integrated methodology in the order of design, formal specification, analysis (validation and verification), implementation, and implementation validation with performance evaluation. Figure 7 illustrates the development activities in this section with correspondence to the methodology sections.

4.2 Design

This section describes the activities with regard to the high-level design of the DoubleQuote composite service where the developer describes the CRESS diagram, performs service configuration, and checks the diagram for syntax errors.

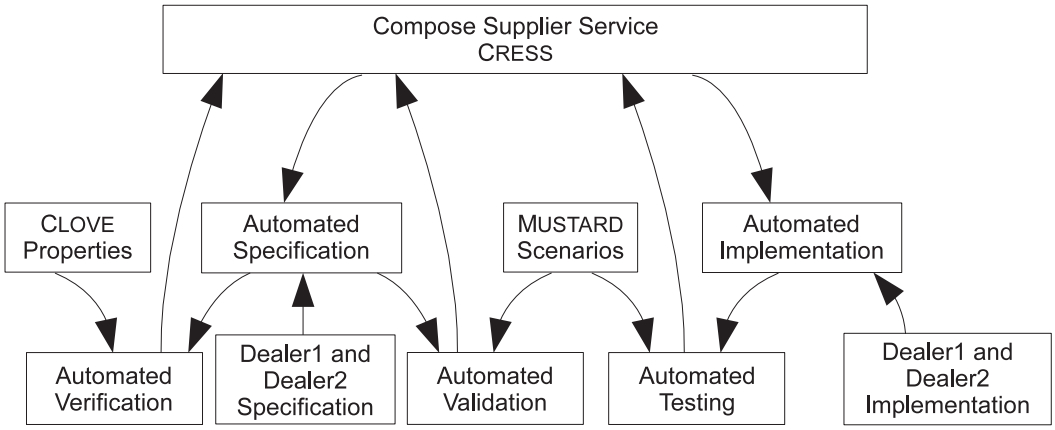


Figure 7: DoubleQuote Composite Service Development

4.2.1 Service Diagram

The requirements specification for DoubleQuote is described as a CRESS diagram using CHIVE, shown in figure 8, with *supplier* as its given name saved under the <CRESS>/ws/supplier directory. BigDeal and WheelerDealer are given names dealer1 and dealer2 respectively in the CRESS Supplier diagram.

The rule box specifies two user-defined data types *need* and *offer*, which are also named variables used in the service. The *need* is a complex data structure comprising name, address, and model, all of **String** type. The *offer* is also a complex data structure comprising: **Natural** *reference* as order reference number; **String** *dealer* which is the name of the dealer who offers the car deal; **Float** *price* as the cost of the car; and **Natural** *delivery* which is the number of days to deliver the car. The variable *offer2* also uses the same data type as *offer*.

Supplier has a *car* port with two operations named *order* and *cancel*, which are for executing the process of car ordering and order cancellation respectively. Both dealers each have a *car* port which has three operations named *quote*, *order*, and *cancel*, respectively for obtaining car quotations, ordering of cars, and cancelling car orders. The Supplier's order operation and both dealer's quote operation are synchronous. All other operations are asynchronous, having only input and no output. The use of the data definitions in these service operations is described in the nodes of the service behaviour description. Briefly the supplier.car.order, dealer1.car.quote and dealer2.car.quote operations each have the *need* and *offer* data types as respective input and output. The supplier.car.cancel, dealer1.car.order, dealer2.car.order, dealer1.car.cancel and dealer2.car.cancel operation have the *offer* data type as input.

The service behaviour is described as follows. Nodes 1 and 10 are the entry points of the Supplier's service behaviour, which are for its clients to order cars and cancel orders respectively. As there is more a one starting point, these nodes are therefore specified as branches from the **Start** node. Upon receiving a need for the car at node 1, simulataneous requests are made to both dealers for quotes described from nodes 2 to 5. Node 2 explicitly defines a **Fork** with parallel branches outgoing to nodes 3 and 4, indicating the simualtaneous invocation of dealer1.car.quote and dealer2.car.quote operations. These return offer-typed data values as variables *offer* and *offer2* respectively. The parallel invocations are synchronised back to the Supplier's business process as successful execution at this point, with the **Join** condition '3&&4' at node 5 thereby completing join (i.e. nodes 3 and 4 must complete successfully). The guarded expression '(offer.price $<$ offer2.price) || ((offer.price = offer2.price) && (offer.delivery $<$ offer2.delivery))' is the comparison of the two offers from the dealers, describing that offers are selected in the order of lower price followed by faster delivery. If the prices and delivery are the same then an order is placed with dealer2. This expression is specified from perspective of the offer by dealer1. Therefore if satisfied, the Supplier process will place an order with dealer1 using the *offer* information (node 6), followed by replying to its customer at node 7. Otherwise the offer of dealer2 is chosen (**Else** branch), with Supplier placing an order at node 8, and replying to the customer at node 9.

The *cancel* operation receives cancel requests at node 10, where the guarded expression is satisfied if the dealer of the offer parameter is dealer1 (a macro which refers to **String** value BigDeal) therefore leading to invoking its 'cancel' operation at node 11 using the same offer parameter. Otherwise the 'cancel' operation of dealer2 is invoked at node 12.

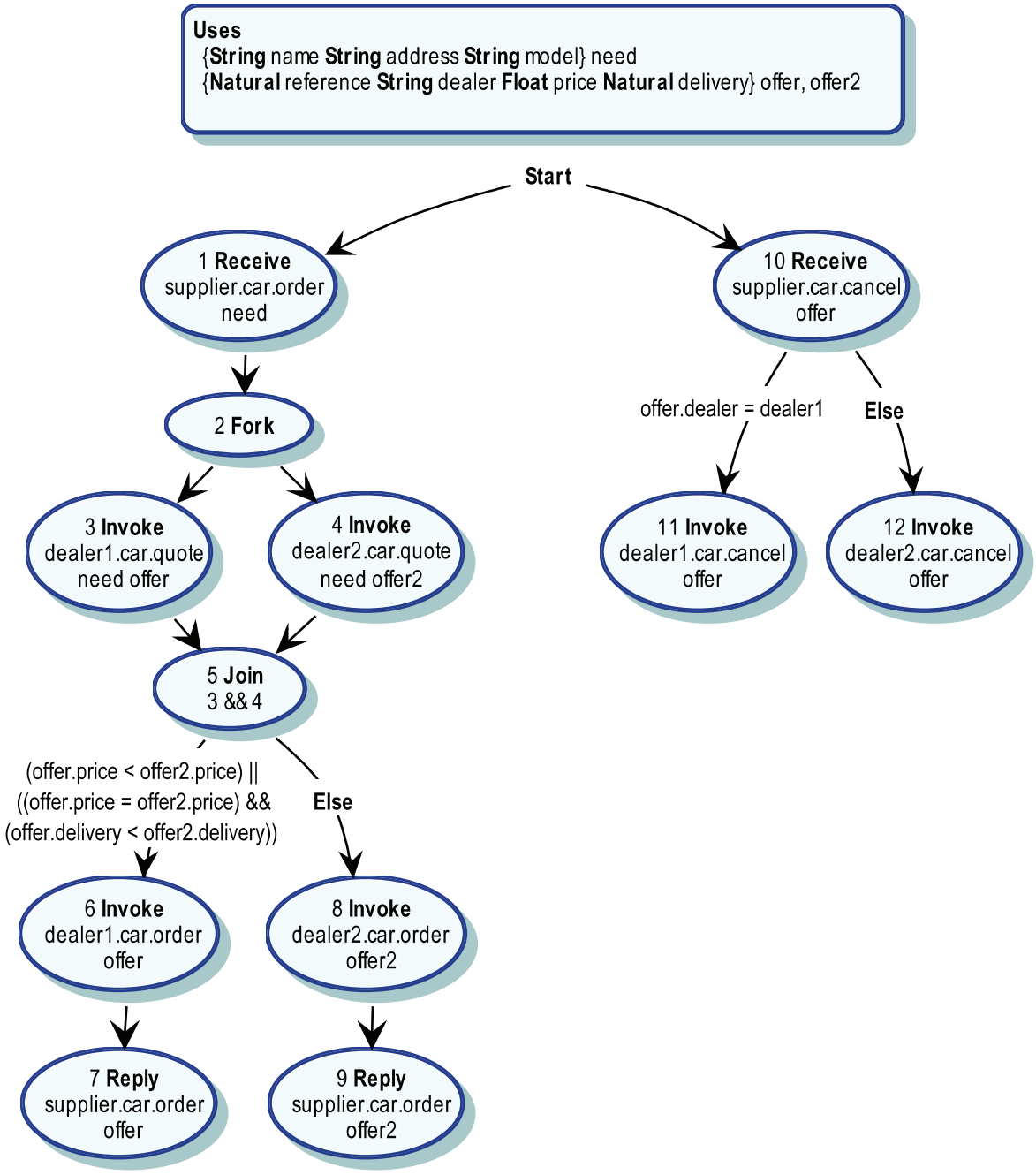


Figure 8: Supplier CRESS Diagram

```
Deploys -b 2 -c -n 1 -o 5 -r / SUPPLIER

DEALER1     deal1   urn:BigDeal          localhost:8080/active-bpel
DEALER2     deal2   urn:WheelerDealer    localhost:8080/active-bpel
SUPPLIER    supp    urn:DoubleQuote      localhost:8080/active-bpel
```

Figure 9: Web Service Configuration Diagram for Supplier

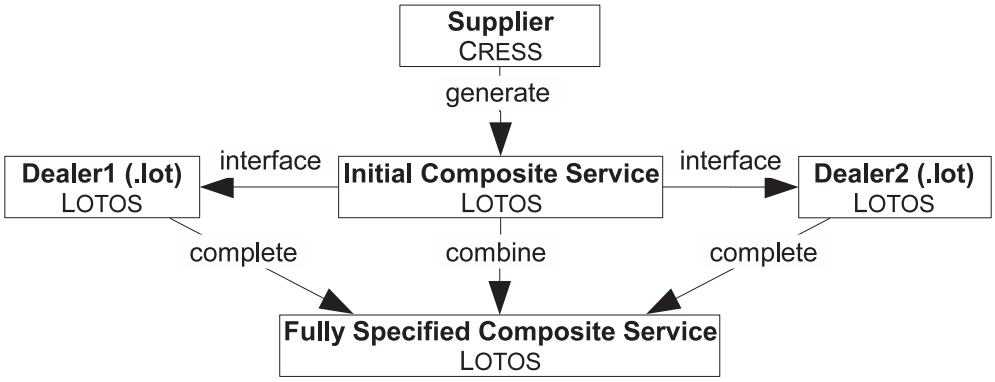


Figure 10: Specifying Dealer1 and Dealer2

4.2.2 Service Configuration

The developer configures the Supplier composite web service and its partners Dealer1 and Dealer2 for formal specification and analysis, which is the immediate phase to follow. This is illustrated in figure 9. This is very similar to that in the Lender, therefore the configuration description here is abbreviated.

4.2.3 Checking Diagrams

The developer checks the Supplier diagram with CRESS, and no syntactic errors are found.

4.3 Formal Specification

The specification is automatically generated. Figure 10 illustrates how the specification is achieved. Using the explicit approach to the generation of the Dealer1 and Dealer2 LOTOS specification interfaces, their formal behaviour is manually completed respectively as dealer1.lot and dealer2.lot, and then automatically combined with the composed service specification of Supplier.

4.3.1 Dealer1 Specification

The following generated LOTOS code is the interface behaviour of Dealer1, which is inferred from the three operation signatures (cancel, order, quote) from the Supplier diagram, specified according to the parameter types.

```
Process DEALER1 [dealer1] : Exit(States) :=   (* DEALER1 partner *)                1
  dealer1 !car !cancel ?offer:Offer;          (* 'cancel' input *)                2
  DEALER1 [dealer1]                           (* repeat behaviour *)              3
[]                                            (* or *)                            4
  dealer1 !car !order ?offer:Offer;           (* 'order' input *)                 5
  DEALER1 [dealer1]                           (* repeat behaviour *)              6
[]                                            (* or *)                            7
  dealer1 !car !quote ?need:Need;             (* 'quote' input *)                 8
  dealer1 !car !quote !AnyOffer;              (* 'quote' output *)                9
  DEALER1 [dealer1]                           (* repeat behaviour *)              10
```

Supplier Automated	Manual Dealer1	Manual Dealer2
724	159	159

Table 3: Specification Generation of Dealer1, Dealer2 and Supplier Service

EndProc (* end DEALER1 *) 11

This interface is then manually completed in dealer1.lot, and automatically combined as the formalisation is once more executed. The high level completed LOTOS is found in Appendix A.1.5. Dealer1 maintains a list of car makes (vehicles) along with their prices and delivery, along with lists of quotes and orders. Valid offers are based on the vehicles' information. The lists of quotes and orders are maintained for cancellation.

4.3.2 Dealer2 Specification

The specification for Dealer2 is also done in a similar way to Dealer1, by manual completion of the generated interface behaviour (which has the same operation signature as for Dealer1). Dealer2 behaves quite similarly to Dealer1, differing in the vehicles in stock and their quotations. Therefore the LOTOS specification is not shown here but is available by downloading the case study.

4.3.3 Supplier Specification

The formal specification of the Supplier is fully created as it is a CRESS diagram. Appendix A.3.2 lists the high-level structure of the generated specification.

4.3.4 Summary

Apart from manual specification of partners Dealer1 and Dealer2, the formal specification is fully automated with all data types and behaviour specified. Even the manual specification itself is added to the interface behaviour that is automatically generated. Using this specification, formal validation and verification can be performed. Table 3 lists the code summary in terms of lines of code generated and manually provided.

4.4 Formal Analysis

4.4.1 Formal Validation

To validate the three services, the developer defines their MUSTARD scenarios respectively in the .mstd files in their own names. The scenarios are quite similar to those described in the LoanStar (Lender) case study. They invoke and read values, with the exception of using variables. Likewise, the order of validation is logically on the partner services Dealer1 and Dealer2, followed by the Supplier composed service.

Dealer1's MUSTARD scenarios are defined below and are straightforward. The use of ?Natural in the **read** indicates that it accepts any value of Natural as is the reference number in the Offer data type. In this situation, the reference number of the Offer returned by Dealer1 is dynamic, which is reasonable in a business quotation. Therefore the use of the variable notation provides a convenient way to describe such scenarios in a compact way. It also supports a general form of validation in addition to specific scenarios. For example, the Mondeo test specifies: an invocation of the Need should receive an Offer of 20000 in value and 15 days from BigDeal, regardless of the reference number as long as it is a Natural. The translated LOTOS tests are listed in Appendix A.4.4.

```
test(Mondeo,
  succeed(
    send(dealer1.car.quote,Need('Mark Flowers,'Uist Scotland,'Mondeo)),
    read(dealer1.car.quote,Offer(?Natural,'BigDeal,20000.,15))))

test(A5,
  succeed(
    send(dealer1.car.quote,Need('Peter Gough,'Congleton UK,'A5)),
```

read(dealer1.car.quote,Offer(?Natural,′BigDeal,33000.,30))))

test(Megane,
succeed(
send(dealer1.car.quote,Need(′Jan Hiddink,′Hengelo Netherlands,′Megane)),
read(dealer1.car.quote,Offer(?Natural,′BigDeal,11000.,5))))

test(XJ6,
succeed(
send(dealer1.car.quote,Need(′Iain MacKay,′Throsk Scotland,′XJ6)),
read(dealer1.car.quote,Offer(?Natural,′BigDeal,1000000.,0))))

The developer executes the validation, and all but the Megane scenario pass. A diagnostic trace shows it did not pass at the **read** action after **send**. Inspection of the Dealer1 complete LOTOS specification reveals that it is returning a three-day delivery, whereas this scenario is expecting five. The Dealer1 specification is corrected as it was an error that Dealer1 should not quote for a Megane any delivery period except for five. The developer executes the formal validation again directly after the correction without having to regenerate the specification as the validation implicitly picks up the updates. All the scenarios pass the second round of validation.

The validation of Dealer2 is very similar to Dealer1 apart from the values, hence their informal descriptions are not described here but only the MUSTARD scenarios. Their translations into LOTOS are also listed in Appendix A.4.5, which is largely similar to that for Dealers but differing only in values.

test(Mondeo,
succeed(
send(dealer2.car.quote,Need(′Mark Flowers,′Uist Scotland,′Mondeo)),
read(dealer2.car.quote,Offer(?Natural,′WheelerDealer,20000.,10))))

test(A5,
succeed(
send(dealer2.car.quote,Need(′Peter Gough,′Congleton UK,′A5)),
read(dealer2.car.quote,Offer(?Natural,′WheelerDealer,35000.,20))))

test(Astra,
succeed(
send(dealer2.car.quote,Need(′Hywel Thomas,′Swansea Wales,′Astra)),
read(dealer2.car.quote,Offer(?Natural,′WheelerDealer,18000.,30))))

test(XJ6,
succeed(
send(dealer2.car.quote,Need(′Iain MacKay,′Throsk Scotland,′XJ6)),
read(dealer2.car.quote,Offer(?Natural,′WheelerDealer,1000000.,0))))

In the automated formal validation for Dealer2, all the scenarios can pass. this indicates that under these scenarios Dealer2 is behaving as expected from the observation of the values it accepts and returns.

The Supplier composite service involves both Dealer1 and Dealer2. With regard to car quotation offers, the Supplier business logic is to use the dealer with the lower offer price. If the offer prices are identical, then the dealer with faster delivery is chosen. Therefore in the MUSTARD scenarios specified by the developer listed below, there are scenarios that expect offers from Dealer1 (BigDeal) and from Dealer2 (WheelerDealer). The translated LOTOS tests are listed in Appendix A.4.6.

test(Mondeo,
succeed(
send(supplier.car.order,Need(′Mark Flowers,′Uist Scotland,′Mondeo)),
read(supplier.car.order,Offer(?Natural,′WheelerDealer,20000.,10))))

test(A5,
succeed(
send(supplier.car.order,Need(′Peter Gough,′Congleton UK,′A5)),
read(supplier.car.order,Offer(?Natural,′BigDeal,33000.,30))))

test(Megane,
succeed(
send(supplier.car.order,Need(′Jan Hiddink,′Hengelo Netherlands,′Megane)),
read(supplier.car.order,Offer(?Natural,′BigDeal,11000.,5))))

test(Astra,
succeed(
send(supplier.car.order,Need(′Hywel Thomas,′Swansea Wales,′Astra)),
read(supplier.car.order,Offer(?Natural,′WheelerDealer,18000.,30))))

test(XJ6,
succeed(
send(supplier.car.order,Need(′Iain MacKay,′Throsk Scotland,′XJ6)),
read(supplier.car.order,Offer(?Natural,′WheelerDealer,1000000.,0))))

In the validation, all but the Mondeo scenario for Supplier passes the validation. The diagnostic trace is as follows:

send(supplier.car.order,Need(mark flowers,uist scotland,mondeo))
<failure point>

The trace suggest that the **send** is fine but the **read**, which is the only action thereafter, is not successful. As the respective Mondeo scenarios for the two dealers have passed, this points to the composite service behaviour as making the error. It turns out that, in the CRESS diagram, part of the condition in the arc from node 5 to node 6 'offer.delivery > offer2.delivery' was wrong as this favours the later delivery given the same price offer from both dealers for the same car. This also affects the rest of the scenarios that have passed, just that the condition was not satisfied as the prices offered by the dealers happen to be different so the behaviour seems to be working correctly. This condition is corrected and the validation is executed again, with all validation tests achieving a pass.

4.4.2 Formal Verification

The following is the informal description of properties specified for the service.

- Freedom from deadlock
- Freedom from livelock
- All car orders must be responded to with offers
- The service will always be able to receive requests (liveness)

The CLOVE file is specified with the data enumeration, and properties which explicitly specify the latter two properties described above.

enumerate_complex(
need\(\″KEN TURNER\″,\″SCOTLAND\″,\″MONDEO\″\)
need\(\″LARRY TAN\″,\″UK\″,\″MONDEO\″\)
offer\([0−9], \″BIGDEAL\″, 20000\.0, 15\)
offer\([0−9], \″WHEELERDEALER\″, 20000\.0, 10\)
...
)

property(General_Response,
response(global,
signal(supplier.car.order,?need),
signal(supplier.car.order,?offer)))

property(Request_Liveness,
forall(**any_signal***) **exists**(**sequence**(**any_signal***, **signal**(supplier.car.order,?need))) **true**)

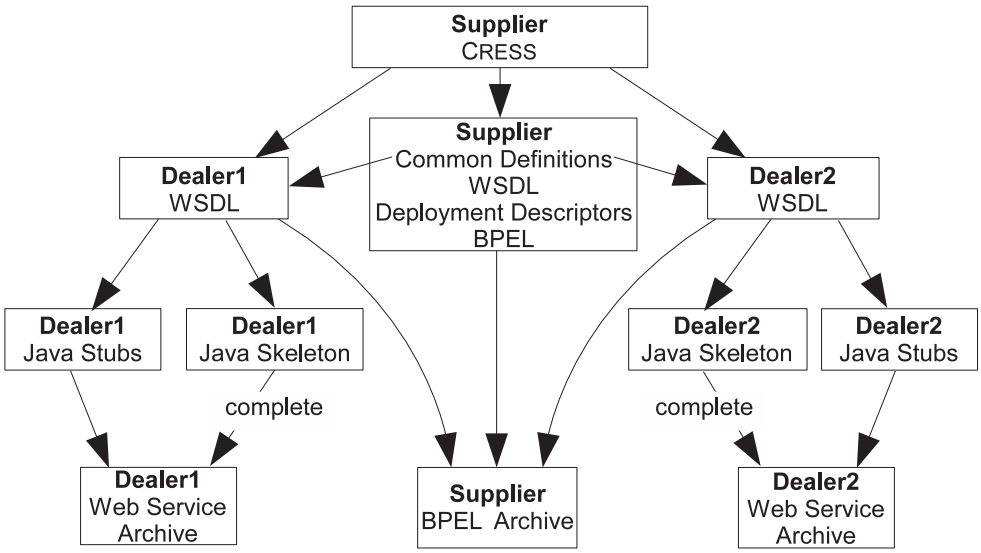


Figure 11: Implementation of Dealer1 and Dealer2

There is the use of array/list data types for vehicles, quotations and orders made. The list data types are unbounded, implying infinite state space. CLOVE's strategy for compositional verification mode supports automated use of bounded arrays and lists in a specification, making model checking possible with CADP. Bounded lists are data types that may have a large enumeration. If the enumeration range in a list element is large, a small bounded list will have a large permutation of enumerations. Therefore the specification is verified using compositional verification where the BCGs of Dealer1, and Dealer2 are constrained by only the Supplier behaviour and the actions that can be synchronised with it.

The list data types are identified to be constrained: Deals, DealOps, Vehicles, and VehiclesOps. Deals and Vehicles are the list data types where the bounds are imposed, with DealOps and VehiclesOps being the respective list data operations whose equations in an unbounded context are adapted to being bounded. The developer executes the following verification command for compositional mode, and constrains the array size for the specified data types:

cress_verify -t lotos -v ws -x deals=3,dealsops=3,vehicles=3,vehiclesops=3 supplier

The above command automates several activities in one command. The LOTOS specification is generated, imposes bounds on array list data types, annotates the specification, adapts the specification for compositional verification, translates CLOVE properties and enumeration into μ-calculus and C implementations, generates and executes the SVL which performs the compositional generation of the BCG, and then performs verification of the translated properties. The code snippet of the constraints, annotations, adaptations, translations, and script generation are listed in Appendix A.5.6 to A.5.9. The results obtained from the verification are as follows:

```
Verifying SUPPLIER GENERAL RESPONSE ...        TRUE
Verifying SUPPLIER REQUEST LIVENESS ...        TRUE
Verifying          DEADLOCK FREEDOM ...        TRUE
Verifying          LIVELOCK FREEDOM ...        TRUE
```

4.5 Implementation

This step corresponds to the implementation aspect of methodology where there is automated generation of code and deployment. As the Dealer1 and Dealer2 partner services are non-CRESS, their implementations are provided. Figure 11 illustrates how this is achieved for Dealer1 and Dealer2. Service implementation code skeletons are generated and manual code is then added. The implementation of Supplier is fully automated.

The engagement of the services is very much similar methodologically to Lender – the developer configures web service configuration diagram with the following parameters:

Deploys -c -b 2 -o 5 / SUPPLIER

This configuration **Deploys** the Supplier service, implying Dealer1 and Dealer2 partner services as they are used by the service. The BPEL implementation is in the WS-BPEL 2.0 standard (-b 2). As there is intent to per-

form implementation validation, the timeout threshold is set to five seconds (-o 5) which generates the appropriate MINT validation configuration for the three services. The code is generated with comments (-c).

The developer prepares empty directories 'dealer1' and 'dealer2' in the same location as the Supplier CRESS diagram. These directories will be used to hold the actual service implementation later. The developer executes the following command line (in the CRESS 'bpel' directory) for the automated implementation:

cress_expand -v ws main.bpel

The above command results in the generation of an implementation for Supplier and both its partners, comprising WSDL service interfaces, deployment configuration files (WSDD and PDD), code stubs translated from WSDL, and partner service code skeletons and code stubs. In the 'bpel' directory, a 'supplier' directory is created containing all the implementation code for the three services. Within the 'supplier' directory there are the 'dealer1' and 'dealer2' directories, which respectively contain their service interface, deployment files, Java code stubs, and service implementation code skeleton and stubs in the directory structure according to their namespace/package. Appendix A.6.7 lists the file structure and the code listing of all the files generated. The developer then adds a detailed implementation by extending the partner service code skeletons, and after this running the command again for the framework to pick up the developer's implementations. Dealer1 and Dealer2 implementations are packaged and deployed in Axis, whilst Supplier is deployed into ActiveBPEL.

4.5.1 Dealer1 Implementation

The code skeleton generates dummy working code for the three operations of Dealer1. The developer adds the implementation according to the informal requirements described above as <CRESS>/ws/supplier/dealer1/BigDeal/dealer1.java which is then picked up by the methodology's automated implementation franmework. As an example the full implementation of quote operation is listed in Appendix A.2.5.

```
package BigDeal;
...
   public class CarBindingImpl implements BigDeal.CarPort {
      public void cancel(DoubleQuoteDefs.Offer offer) throws java.rmi.RemoteException
      }

      public void order(DoubleQuoteDefs.Offer offer) throws java.rmi.RemoteException {
      }

      public DoubleQuoteDefs.Offer quote(DoubleQuoteDefs.Need need)
            throws java.rmi.RemoteException {
         return null;
      }
}
```

4.5.2 Dealer2 Implementation

The code skeleton generated for Dealer2 is very similar to that of Dealer1, apart from minor differences such as Java package name, hence it is not necessary to list it here. The full implementation is saved as <CRESS>/ws/supplier/dealer2/WheelerDealer/dealer2.java which is rather similar to Dealer1 – it implements the informal requirements as described above for the WheelerDealer web service partner.

4.5.3 Supplier Implementation

Supplier, being a CRESS-described service, has full automated implementation where the WSDL interface, BPEL code, and deployment descriptor files (catalogue and PDD) are generated and compiled together as a BPEL archive supplier.bpr for deployment into the ActiveBPEL container. The WSDL and BPEL code are respectively listed in Appendix A.6.9 and A.6.11.

4.5.4 Summary

Table 4 shows the number of lines of code and files that are automatically produced and manually provided in the implementation phase. The implementation for the three services is highly automated apart from the manual

Service	WSDL (incl. partners)	Java	Deployment	BPEL
Supplier	4 files (314 lines)	–	2 files (67 lines)	222 lines
Dealer1	2 files (140 lines)	8 files (997 lines)	42 lines	–
Dealer2	2 files (140 lines)	8 files (997 lines)	42 lines	–

Table 4: Code Generation Summary of Dealer1, Dealer2 and Supplier Service

implementation, which was added into the generated code skeletons. The automation of the implementation took less than a minute to produce the deployable service archives.

4.6 Implementation Validation

The MINT configuration files for each service are already generated as part of the implementation phase, and the MUSTARD scenarios for the services are already specified. Therefore the developer can readily perform implementation validation after deploying these services. The developer can execute the validation. In this case study, the command-line cress_validate is used for illustration, which is very similar to that used in the Lender (LoanStar) case study. The command line executed (by the developer) in the 'bpel/supplier' directory of CRESSis:

cress_validate -v ws -t bpel supplier

4.6.1 Implementation Validation of Dealer1

The validation of Dealer1 uses its generated MINT configuration file (Appendix A.7.7), and the MINT scenarios (Appendix A.7.8) translated from the MUSTARD descriptions. The validation stops upon the initial execution of the validation with the reported error:

Test DEALER1 Mondeo ... [Error]...
[Error] mustard: Java translation failed

This indicates that the WSDL of the deployed Dealer1 service cannot be downloaded at the URL specified by the target.url MINT property, meaning the activity of translating to Java code stubs failed (wsdl2java). It is discovered that the container has not been started. Although this is a simple situation of just starting the hosting environment, this illustrates that MINT can be used to validate the availability of services in general, specifically the service interface where potential clients obtain access. The container is started and the validation is executed again with the following results indicating passes in all the scenarios.

Test DEALER1 Mondeo ...	Pass	1 succ	0 fail	1.3 secs
Test DEALER1 A5 ...	Pass	1 succ	0 fail	1.2 secs
Test DEALER1 Megane ...	Pass	1 succ	0 fail	1.1 secs
Test DEALER1 XJ6 ...	Pass	1 succ	0 fail	1.1 secs

4.6.2 Implementation Validation of Dealer2

Dealer2's MINT configuration file (Appendix A.7.9) and translated MINT scenarios (Appendix A.7.10) from the MUSTARD descriptions are used for its validation. The results after execution are as follows:

Test DEALER2 Mondeo ...	Pass	1 succ	0 fail	1.2 secs
Test DEALER2 A5 ...	Fail	0 succ	1 fail	1.1 secs

send(dealer2.car.quote,Need(″Peter Gough″,″Congleton UK″,″A5″))
<failure point>

Test DEALER2 Astra ...	Pass	1 succ	0 fail	1.2 secs
Test DEALER2 XJ6 ...	Pass	1 succ	0 fail	1.2 secs

From the results, the A5 scenario did not pass the validation. The diagnostic trace shows that the invocation was successful, but not the return value. Narrowing this scenario is straight-forward as it involves only Dealer2's implementation, where it is found that the price for the A5 is wrongly returned. The source is corrected, automatically implemented and deployed, with validation then satisfactory and compatible with the formal validation.

4.6.3 Implementation Validation of Supplier

The validation of the Supplier composite service likewise requires its generated MINT configuration file (Appendix A.7.11) and MINT scenarios (Appendix A.7.12), automatically translated from the MUSTARD description. All the scenarios pass the validation just as for the formal validation, with the results shown below. Recall that the service description was found to be incorrect upon the execution formal validation, and it was corrected prior to implementation.

```
Test SUPPLIER Mondeo ...        Pass    1 succ    0 fail    1.7 secs
Test SUPPLIER A5 ...            Pass    1 succ    0 fail    1.2 secs
Test SUPPLIER Megane ...        Pass    1 succ    0 fail    1.2 secs
Test SUPPLIER Astra ...         Pass    1 succ    0 fail    1.2 secs
Test SUPPLIER XJ6 ...           Pass    1 succ    0 fail    1.2 secs
```

4.6.4 Performance Evaluation

Suppose it is desired that the service performance cope with a concurrent load of 100 requests with responses between 3 to maximum 7 seconds. The timeout threshold can be edited in the -o option of the service configuration to generate the MINT properties, or by editing the properties directly. The performance evaluation in sequential mode (-ps100) show consistency and good response times (a scenario run completes within 2 seconds) even with a timeout threshold of 3 sec:

```
Test DEALER1 Mondeo ...         Pass 100 succ     0 fail     0.0 secs
...
             Sequential         0 inco true cons 0.0 secs .. 1.1 secs
Test DEALER2 Mondeo ...         Pass 100 succ     0 fail     0.0 secs
             Sequential         0 inco true cons 0.0 secs .. 1.1 secs
...
Test SUPPLIER Mondeo ...        Pass 100 succ     0 fail     0.1 secs
             Sequential         0 inco true cons 0.1 secs .. 1.6 secs
```

The performance evaluation is then executed in concurrent mode by specifying the '-pc100' option using the same command as given in the implementation validation. The results below particularly show inconsistency in Supplier's concurrent performance for all its scenarios, indicating that it may not be always possible for performance (particularly Supplier) respond within 3 seconds. The diagnostics show a list of SocketTimeoutException messages which indicate that the failures were due to the invocations not being able to receive responses from the supplier.car.order synchronous operation within 3 sec. It is observed that the reaction of Dealer1 and Dealer2 are comparatively better than Supplier even though the tests are similar, involving an invocation and reading a response. This may be reasonable considering that Supplier is a composite service that involves parallel invocation of the two other web services, implying more overheads also. The concurrent performance evaluation with 4 seconds timeout threshold for all three services is successful, indicating that the service environment is able to cope with such settings (below the 7 second limit). Although the performance evaluation does not provide the means to locate and address performance issues, its execution brings about observations which can be used by developers to iteratively fine-tune the performance whereever required.

```
Test DEALER1 Mondeo ...         Pass 100 succ     0 fail 3.7 secs
             Concurrent   0 inco  true    cons 2.6 secs .. 4.0 secs
...
Test DEALER2 Mondeo ...         Pass 100 succ     0 fail 3.9 secs
             Concurrent   0 inco  true    cons 2.9 secs .. 4.3 secs
...
Test SUPPLIER Mondeo ...                          Execution error ...
    caused by java.net.SocketTimeoutException: Read timed out
                          Inconclusive            68 succ 32 fail 4.9 secs
             Concurrent   0 inco  false cons 2.9 secs .. 5.7 secs
Test SUPPLIER A5 ...                              Execution error ...
                          Inconclusive            54 succ 46 fail 5.3 secs
             Concurrent   0 inco  false cons 4.5 secs .. 5.7 secs
Test SUPPLIER Megane ...
```

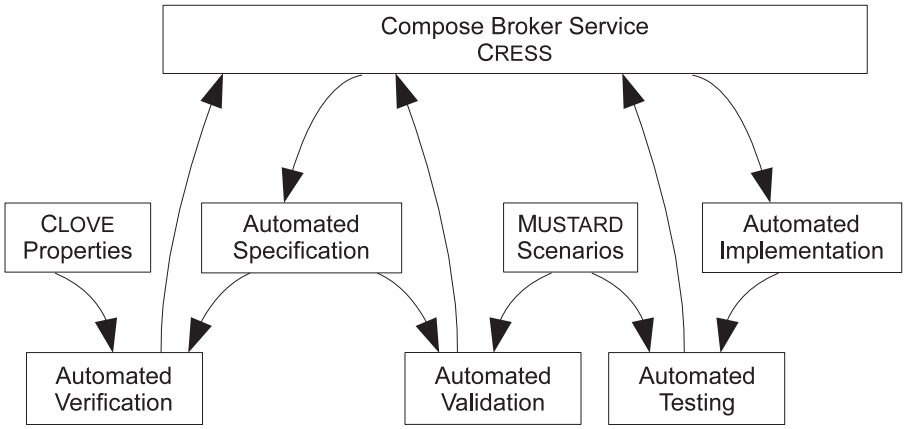


Figure 12: CarMen Composite Service Development

```
                    Inconclusive          63 succ 37 fail 5.4 secs
          Concurrent      0 inco    false cons 4.6 secs .. 5.8 secs
Test SUPPLIER Astra ...                 Execution error ...
     caused by java.net.SocketTimeoutException: Read timed out
                    Inconclusive          42 succ 58 fail 5.3 secs
          Concurrent      0 inco    false cons 4.0 secs .. 5.7 secs
Test SUPPLIER XJ6 ...                   Execution error ...
                    Inconclusive          61 succ 39 fail 5.2 secs
          Concurrent      0 inco    false cons 3.6 secs .. 5.7 secs
```

5 Development Of The Broker Composed Web Service

5.1 Introduction

This section describes the development lifecycle of CarMen, which intent involves the LoanStar and DoubleQuote services.

CarMen is the brokerage service that combines DoubleQuote and LoanStar services as a car purchase complete with a financing solution. CarMen organises the car order with DoubleQuote followed by requesting a loan at LoanStar. A loan request to the nearest rounded figure to LoanStar will only proceed should the desired car be ordered. If the loan is refused by LoanStar, then the car order made with DoubleQuote is cancelled. A successful coordinated car-finance purchase then returns schedule information to the customer. The schedule contains the order reference, car dealer, price of car, delivery period, and the loan rate. These are all obtained from the information returned by DoubleQuote and LoanStar.

The development lifecycle of the CarMen service is shown in figure 12, where the integrated methodology is applied in the order of design, formal specification, analysis (validation and verification), implementation, and implementation validation with performance evaluation. As LoanStar and DoubleQuote service have already undergone the development methodology, the specification and implementation is fully automated with no need for manual specification and implementation.

5.2 Design

5.2.1 Service Diagram

The developer designs the CarMen service with the name *broker* using the CHIVE editor, saved as broker.chx under the <CRESS>/ws/broker directory, which is illustrated in figure 13.

The rule box contains one data structure and variable definition, partner use declaration, and a definition of a constant. The complex data structure *schedule* is made up of: **Natural** reference which is the car order number; **String** dealer which identifies the dealer who organises the car; **Float** price wthat is the cost of the car; **Natural** delivery as the number of days to deliver the car; and **Float** rate as the loan rate of the car finance. The Broker

diagram uses the syntax '*/ LENDER SUPPLIER*' which means it uses the Lender and Supplier CRESS diagram descriptions, implying the definitions in their rule boxes. Therefore the Broker diagram will have access to data types such as *proposal* and *need*. The macro/constant *unpriced* is defined with the value 1000000. The Broker service has only one operation broker.car.arrange which is synchronous, having *need* as the input and *schedule* as the output.

The service behaviour of the Broker business process is the following. Broker receives the car *need* through broker.car.arrange that is defined in node 1. The car order is made with Supplier by invoking supplier.car.order at node 2 with the *need*, which returns the car offer assigned to the variable *offer*. A successful car order will have a price that is not *unpriced* described with the syntax 'offer.price != unpriced' on the outgoing arc leading to node 3. If the condition is satisfied the value for the *proposal* variable is constructed. The *proposal.name* and *proposal.address* values are assigned to the respective fields from the *need* variable. The *proposal.amount* is assigned the rounded value of *offer.price*. The loan request is made to lender.loan.quote with the *proposal* variable described in node 3. If the invocation is successful, the outgoing arc from node 3 to node 4 is followed where the *schedule* variable data values are set with the corresponding values from *offer* and *rate*. Broker returns from its operation with the value of *schedule*. If the car order is unsuccessful (meaning *offer.price* is *unpriced*), the **Else** outgoing arc to node 5 is followed which assigns the value 'car unavailable' to variable *error*. Broker then returns from broker.car.arrange operation with a fault named *refusal* with the value of *error*. Node 2 also specifies a **Compensation** handler denoted by the **Compensation** arc leading to node 7 to **Invoke** supplier.car.cancel with *offer* as the parameter. The compensation handler is only enabled if the **Invoke** at node 2 is unsuccessful for the same business process.

The entire process has a global fault handler for fault name *refusal* of String type, specified by the **Catch** *refusal.error* from the **Start** node, which sets the String value of the fault to the variable *error* (which is defined in Lender). This fault handler is specified for any potential fault that could be thrown by the Lender process in the case of rejected loans. If the fault arises, the global fault handler is enabled and the Broker process will be directed to node 7. This explicitly calls the **Compensate** action which means all **Compensation** handlers that are enabled are executed in reverse order of completion. In this case, the only compensation handler (at node 2) will be called. After compensation is completed, the process moves from node 7 to node 8, where Broker returns the fault for its operation before terminating itself at node 9.

5.2.2 Service Configuration

The developer specifies the following configuration for the Broker, illustrated in figure 14. The **Deploys** options have been explained earlier on the previous case studies, along with the service parameters (prefix, namespace, etc).

5.2.3 Checking Diagrams

The Broker diagram is checked by the CRESS tools with no reported syntax errors.

5.3 Specification

The developer or analyst can obtain the formal specification in LOTOS automatically and analyse it immediately. There is no need for any manual specification of partners in the case of Broker as the behaviour of Lender and Supplier are fully specified, along with their partner services. Therefore the explicit and implicit (direct to analysis) method can be used for specification generation. The Broker specification is listed in Appendix A.3.3. About 1298 lines of LOTOS code generated are generated.

5.4 Formal Analysis

Broker's partner services Lender and Supplier have already been previously analysed along with their partner services, which are hidden from the perspective of Broker as a client. This section therefore covers the validation and verification aspects of the methodology for Broker.

5.4.1 Formal Validation

The developer defines the following MUSTARD scenarios as broker.mstd residing in the same location as the Broker CRESS diagram. Their translation to LOTOS tests are listed in Appendix A.4.7.

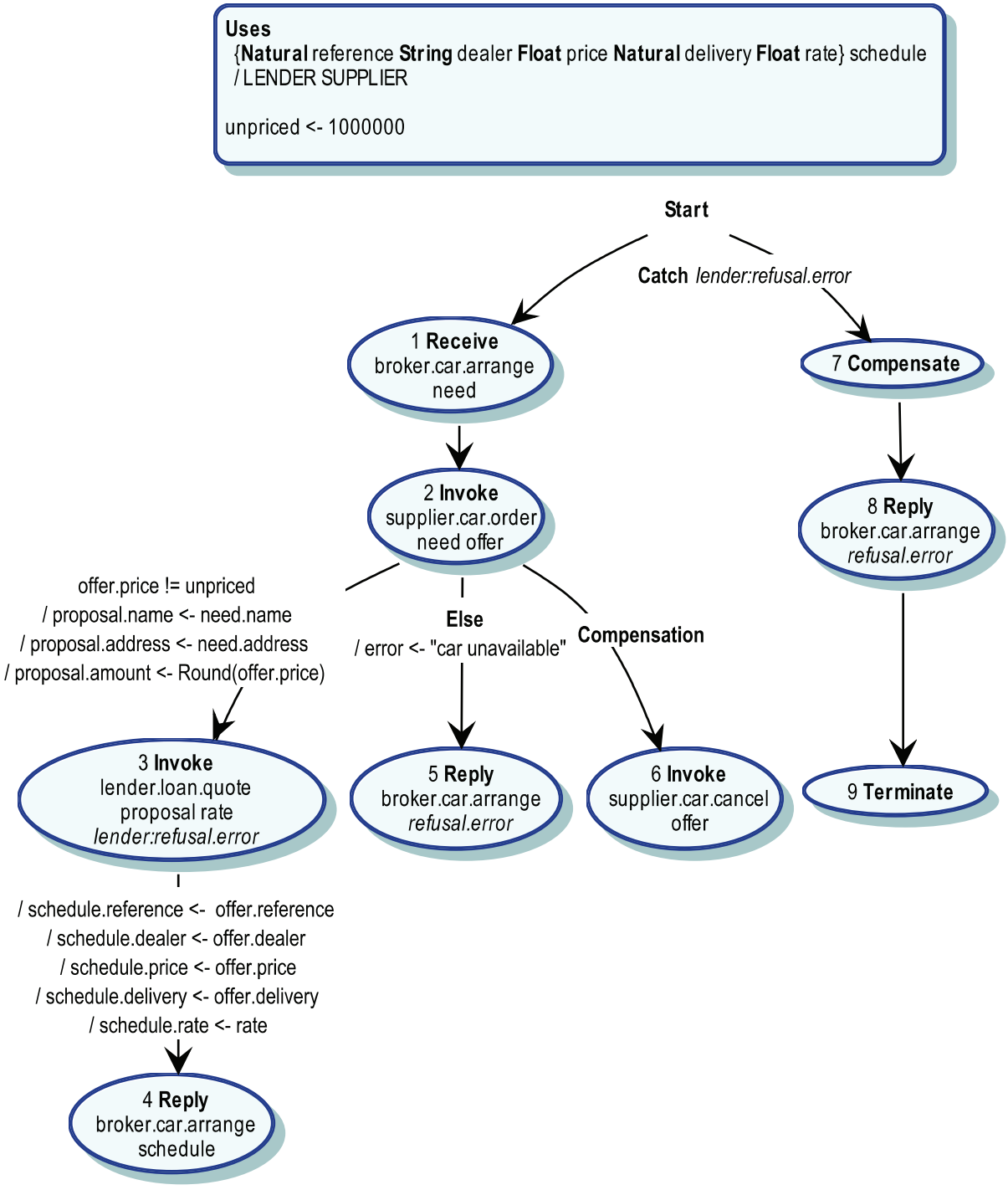


Figure 13: Broker CRESS Diagram

```
Deploys -b 2 -c -n 1 -o 5 -r / BROKER

BROKER          brok     urn:CarMen              localhost:8080/active-bpel
...
```

Figure 14: Web Service Configuration Diagram for Broker

test(Mondeo_Ken,
  succeed(
    send(broker.car.arrange,Need('Ken Boyle,'Dublin Ireland,'Mondeo)),
    read(broker.car.arrange,Schedule(?Natural,'WheelerDealer,20000.,10,3.7))))

test(A5_Scotland,
  succeed(
    send(broker.car.arrange,Need('Mary Duncan,'Wick Scotland,'A5)),
    read(broker.car.arrange,Schedule(?Natural,'BigDeal,33000.,30,4.1))))

test(Megane,
  succeed(
    send(broker.car.arrange,Need('Sally Dean,'Cardiff Wales,'Megane)),
    read(broker.car.arrange,Schedule(?Natural,'BigDeal,11000.,5,4.4))))

test(Astra,
  succeed(
    send(broker.car.arrange,Need('Ian Carey,'Croydon England,'Astra)),
    read(broker.car.arrange,refusal,'loan unacceptable)))

test(XJ6,
  succeed(
    send(broker.car.arrange,Need('Iain MacKay,'Throsk Scotland,'XJ6)),
    read(broker.car.arrange,refusal,'car unavailable)))

The developer executes the automated validation of Broker, which is illustrated here using the CHIVE editor. The editor, with Broker as the current CRESS diagram, is configured with WS (web service) application domain and LOTOS target language. The developer launches the Tools -> Validate menu item and starts the formal validation process. The validation results are listed below, and all the scenarios but XJ6 have passed the validation.

Test BROKER Mondeo Ken ...	Pass	1 succ	0 fail 1.6 secs
Test BROKER A5 Scotland ...	Pass	1 succ	0 fail 4.0 secs
Test BROKER Megane ...	Pass	1 succ	0 fail 2.6 secs
Test BROKER Astra ...	Pass	2 succ	0 fail 9.9 secs
Test BROKER XJ6 ...	Fail	0 succ	6 fail 0.6 secs

send(broker.car.arrange,Need(iain mackay,throsk scotland,xj6))
<failure point>
or
    <internal event>
<failure point>
or
    <internal event>
<failure point>
or
    <internal event>
<failure point>
or
    <internal event>
<failure point>
or
<failure point>

Of the successful scenarios, the Astra scenario has reported two successful paths in the validation which contributed to the Pass outcome. The explanation is as follows. The Astra scenario invokes the Broker service with parameters that will lead to the "loan unacceptable" refusal fault, which is actually generated from the Lender service. The Broker service handles this event with the global **Catch** arc from the **Start** node to node 7 in the CRESS diagram. This event is potentially triggered after invoking Lender at node 3, implying that the car order

has been placed (node 2). Broker explicitly performs the Compensation activity where the activities thereafter are at node 6, and that path that follows from node 7 to 9. These may be completed in a different order, such as the Reply at node 8 to Broker's client before the Invoke at node 6 or vice versa. Therefore there are two paths, and the validation process will try all both of them.

The XJ6 scenario expected the "car unavailable" refusal which the Broker service has failed to return the fault. The "car unavailable" message is returned only by Broker, which serves as a clue as to what to check. Based on the service behaviour of Broker, the Supplier is being invoked first, and then Lender if there is a valid car offer. Otherwise the Broker service replies with the "car unavailble" refusal fault. The XJ6 validation scenario specified for Supplier has passed, which indicates that the service is returning an invalid offer. This narrows the problem to the Broker service, where it was found that the labelled arc from node 2 to node 5 assigns a wrong value to the refusal fault. It was corrected to "car unavailable", and the XJ6 scenario for Broker passes the validation that is executed once more.

5.4.2 Formal Verification

The Broker service should exhibit the following properties:

- Freedom from deadlock
- Freedom from livelock
- Broker service should only have the 'arrange' operation.
- All 'need' requests should be responded to, either success schedules or faults (loan unacceptable or car unavailable)

Broker is a composite service comprising composed services that are all specified in CRESS. Compositional verification should therefore be used to achieve effective analysis. Below is the CLOVE file specified by the developer, comprising the enumerations and properties only for the last two properties, as deadlock and livelock freedom will be checked by default. These properties are general and difficult to check with validation, but verification is well-suited.

```
initials(signal(broker.car.arrange,?need))

enum_numbers(3.700000,3.700000,4.100000,4.400000)
enum_strings('loan unacceptable,'car unavailable,'low,'medium,'high)

enumerate_complex(
need\(\"KEN TURNER\",\"SCOTLAND\",\"MONDEO\"\)
need\(\"LARRY TAN\",\"UK\",\"MONDEO\"\)
proposal\(\"KEN TURNER\", \"SCOTLAND\", 20000\.0\)
proposal\(\"LARRY TAN\", \"UK\", 20000\.0\)
offer\([0−9], \"BIGDEAL\", 20000\.0, 15\)
offer\([0−9], \"WHEELERDEALER\", 20000\.0, 10\)
schedule\([0−9], \"BIGDEAL\", 20000\.0, 15\)
schedule\([0−9], \"WHEELERDEALER\", 20000\.0, 10\)
...
)

property(General_Response,
                response(global,
                        signal(broker.car.arrange,?need),
                        choice_any(signal(broker.car.arrange,?schedule),
                                signal(broker.car.arrange,refusal,'loan unacceptable),
                                signal(broker.car.arrange,refusal,'car unavailable))))
```

There are the uses of the enabling operator '$\gg$' by BROKER_EVENT, the event handling process, for describing the Compensate action. The specific part of the behaviourial expression in this process has:

```
...
  BROKER_EVENT [broker,lender,supplier]                (* do compensation *)
    (error,need,offer,proposal,rate,schedule,Tail(xstates),getScope(head(xstates)),Compensation,xvalue)
>> Accept xstates:States In                (* get rest of compensation states *)
  BROKER_EVENT [broker,lender,supplier](* continue compensation *)
    (error,need,offer,proposal,rate,schedule,xstates,xscope,compensation,xvalue)
...
```

Prior to verification, the Broker behaviourial specification has to be adapted to be free from the semantics of recursive process instantiation, which is not compatible by CADP requirements. The current limitation is that the specification be manually adapted to specify the behaviour without the semantics of recursive process instantiation. The CLOVE manual mode is used, where there is a high degree of of automated support in the LOTOS specification annotation, generation of the C implementation of data enumeration, translation of properties into μ-calculus, and generation of the SVL script which describes all the tasks in the compositional verification. The translated properties, C implementation for data enumeration, and SVL script are listed in Appendix A.5.12, A.5.13, and A.5.14. All that is required is the manual adaptation and execution of the SVL script which automates the verification. A summary of the verification outcomes is listed here:

... compositional generation of state space ...

″broker_general_response.bcg″ = verify ″broker_general_response.mcl″ **in** ″ws.bcg″

TRUE

″_deadlock_freedom.bcg″ = verify ″_deadlock_freedom.mcl″ **in** ″ws.bcg″

TRUE

″_livelock_freedom.bcg″ = verify ″_livelock_freedom.mcl″ **in** ″ws.bcg″

TRUE

″_initials_safety.bcg″ = verify ″_initials_safety.mcl″ **in** ″ws.bcg″

TRUE

The verification outcomes assert that the four specified properties are exhibited by the service behaviour. In particular the ‘general response’ property demonstrates the assurance that the service will always reply with only the specified responses.

5.5 Implementation

Confident of the service behaviour from the formal analysis, the developer therefore proceeds to implement the service.The web service configuration diagram is configured for WS-BPEL 2.0 implementation (-b 2) and also for MINT properties to be generated for a five-second service timeout (-o 5). The developer generates the implementation by executing the following command:

cress_expand -v ws main.bpel

Broker, being a CRESS composed service, has its implementation fully automated, comprising its WSDL service interfaces, partner service interfaces, BPEL process behaviour, and deployment descriptor files, which are all translated from the CRESS service and configuration diagrams. The files generated are listed in Appendix A.6.12. The overview of the service interface, common data definitions, and BPEL code are also listed respectively in Appendix A.6.13, A.6.14 and A.6.15. The Broker service implementation is packaged as broker.bpr and deployed into ActiveBPEL.

5.5.1 Summary

Table 5 shows the number of lines of code and files that are automatically generated.

WSDL	Lines	Deployment	Lines	BPEL (lines)
6 Files	420	2 Files	73	292

Table 5: Code Generation Summary for Broker Service

5.6 Implementation Validation

5.6.1 Broker Implementation Validation

The deployed Broker service can be readily validated with MINT using the MINT properties file (see Appendix A.7.13) that was generated on implementation and the MUSTARD scenarios used in the formal validation. The latter are translated into MINT scenarios (see Appendix A.7.14). For illustration, the developer executes the validation at the command line within <CRESS>/ws/broker directory:

```
cress_validate -t bpel -v ws broker
```

The results of the implementation validation are as follows:

```
Test BROKER Mondeo Ken ...      Pass    1 succ    0 fail    3.0 secs
Test BROKER A5 Scotland ...     Pass    1 succ    0 fail    1.6 secs
Test BROKER Megane ...          Pass    1 succ    0 fail    1.6 secs
Test BROKER Astra ...           Pass    1 succ    0 fail    1.6 secs
Test BROKER XJ6 ...             Pass    1 succ    0 fail    1.5 secs
```

All the scenarios have obtained a Pass in the validation. It is seen especially how the XJ6 scenario in the application of the methodology's formal validation has corrected the design, which positively impacted on the implementation via the correction made in the CRESS diagram service behaviour.

5.6.2 Performance Evaluation

suppose the Broker business process environment is expected to be able to handle an average simultaneous load of 99 transactions with responses within 10 seconds. A sequential performance evaluation is executed and successfully completed, indicating consistency of the service behaviour in a series. The concurrent performance evaluation with a load of 99 is also successful. This is due to maxThreads having been reconfigured from 150 to 500 in the Apache Tomcat server to accommodate the performance requirement for Lender (and its partners). This server is also where Supplier (with all its partners), and the Broker service are deployed. For the purposes of illustration, suppose this setting is reverted and the performance evaluation is done once more. There are the with unsuccessful outcomes shown below. The first set of evaluations resulted in a SocketTimeoutException using a threshold of 10 seconds. Subsequently, the errors reported are in the context of a connection being refused. This observation seems to indicate the service environment has degraded to inaction, not being able to receive requests. At the same time, when the series of ConnectException messages is reported by MINT, the hosting container (Tomcat/ActiveBPEL) logged a message 'SEVERE: All threads (150) are currently busy, waiting. Increase maxThreads (150) or check the servlet status'. An invocation of Broker involves all the other services directly or indirectly, which implies that there may not be adequate resources to accommodate a concurrent load of 99 requests to Broker, where each request may involve up to all 7 services. The observation can be used as a form of information to fine-tune the resources of the container, and BPEL engine, etc. until performance evaluation is satisfactory.

```
Test BROKER Mondeo Ken ...                    Execution error ...
        caused by java.net.SocketTimeoutException: Read timed out
...
                    Fail                  0 succ    99 fail          0.0 secs
            Concurrent          0 inco true cons 0.0 secs .. 0.0 secs
Test BROKER A5 Scotland ...                   Execution error ...
        caused by java.net.ConnectException: Read timed out
...
                    Fail                  0 succ    99 fail          0.0 secs
            Concurrent          0 inco true cons 0.0 secs .. 0.0 secs
Test BROKER Megane ...                        Execution error ...
```

```
        caused by java.net.ConnectException: Read timed out
...
                    Fail                0 succ    99 fail          0.0 secs
            Concurrent          0 inco true cons 0.0 secs .. 0.0 secs
Test BROKER Astra ...                   Execution error ...
 caused by java.net.ConnectException: Connection refused: connect
...
                    Fail                0 succ    99 fail          3.4 secs
            Concurrent          0 inco true cons 3.1 secs .. 3.5 secs
Test BROKER XJ6 ...                     Execution error ...
 caused by java.net.ConnectException: Connection refused: connect
...
                    Fail                0 succ    99 fail          3.5 secs
            Concurrent          0 inco true cons 3.4 secs .. 3.5 secs
```

6 Development of Allocator Composed Grid Service

6.1 Introduction

This section describes the development lifecycle flow of the Allocator service using the methodology. This is similar to the previous case study illustrations, comprising: design, specification, analysis, implementation, and implementation validation with performance evaluation.

The Allocator service brings together two resource-related grid services, Factory and Mapper, that perform occupational matching of survey datasets. The Factory grid service coordinates the allocation of resources to map occupations. The Mapper grid service performs the task of mapping occupational information using the selected resource. The Factory and Mapper services are developed according to the WS-Resource factory pattern whereby the former is the factory that creates resources and the latter is the 'instance' service that performs operations. Therefore Factory and Mapper share the same resource context.

The Allocator service combines the two services within its process behaviour to provide a complete service for mapping an occupation, which includes the coordination of resource allocation, data mapping, resource deallocation, and event handling. The Allocator has a 'translate' operation that maps an occupational title to a specific classification code. The process flow is to have the Factory service allocate a mapping resource that is accessbile via an endpoint. This addresses a particular Mapper service resource to perform the mapping. Upon successful mapping by the Mapper, the Allocator then returns the result to its client. Otherwise, if the invocation of Mapper results in a fault, then the resource is deallocated through the Factory and Allocator returns a fault to its client.

Figure 15 shows the process of the service development. The developer first describes the composed behaviour in CRESS. The composite service specification is automatically generated and the developer fills in the behaviour details of the outline behaviour for grid partner services Factory and Mapper. The developer also specifies the MUSTARD scenarios and CLOVE properties. The developer then executes the automated validation and verification, analyses the respective results, and addresses the issues identified by the analyses. Once satisfied with the analyses, the developer starts the automated implementation. The developer handcrafts the actual implementation and deployment configuration for the grid partner services. The developer starts the automated implementation validation and performance evaluation on the deployed services using the same MUSTARD scenarios used for specification, iteratively addressing implementation and resource configuration issues that are discovered.

6.2 Design

Using the CHIVE editor, the developer describes the Allocator service as a CRESS diagram, followed by the service configuration.

6.2.1 Service Diagram

The developer describes the Allocator service behaviour in CRESS, as illustrated in figure 16. This is a representation of the requirements given in the informal description. The rule box declares the user-defined data types and variables used by Allocator and partner services, similar to those in the previous case studies. In particular, there

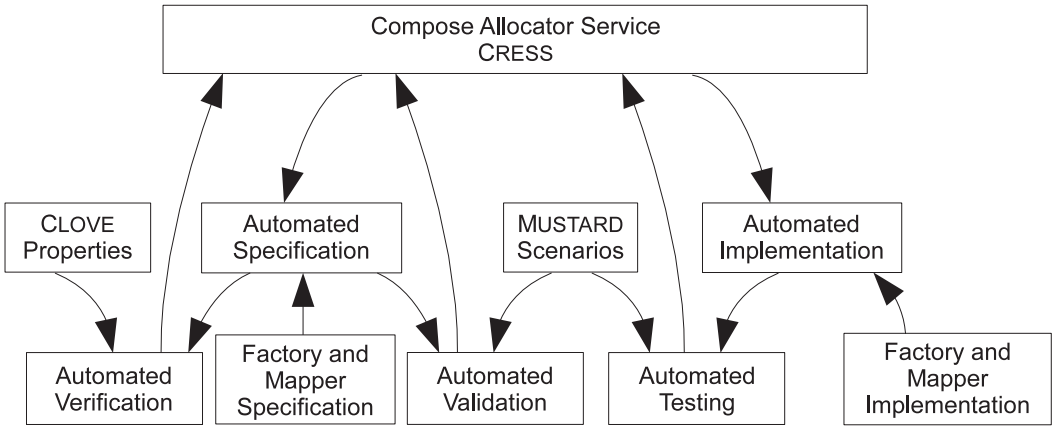


Figure 15: Allocator Composite Service Lifecycle

is a declaration of dynamic partner Mapper at its port *job*. There is a **Reference** type variable named mapperReference which is used to hold the endpoint reference to the resource created by Factory. This is associated with Mapper's *job* port after Factory invocation.

The Allocator has one operation, *translate*, that is described in node 1 **Receive** with the complex type *mapping*. The variable *scheme* is assigned the field *scheme* in *mapping*, followed by invoking factory.job.allocate (node 2) which allocates a resource according to the value in *scheme*. The reference to the created resource is returned and held in variable *mapperReference*. This reference is assigned to *mapper.job* which binds to the WS-Resource. The variable *job2* is assigned the value of the field *mapping.job* which is used in the **Invoke** activity in node 3. This ask the Mapper to perform the occupational translation of *job2* according to the classification indicated by *scheme2*. The translated result *mappedJob* is returned by Allocator in node 4. The allocation by Factory in node 2 may throw a fault, which is handled by the **Catch** *factoryError.reason* arc leading to node 5. The translation in node 3 may throw a fault, which is handled by a **Catch** .reason matching any fault name and leading to node 6. If this event occurs, then an invocation of Factory is made to deallocate the resource, followed by returning the fault value with *allocatorError* fault name at node 7 before terminating at node 8.

6.2.2 Service Configuration

The developer defines the service configuration illustrated in figure 17. Most of the **Deploys** options are familiar, with the exception of the '-m RESOURCE' parameterised option for 'merge partners', which will generate a behaviour specification for a phantom partner RESOURCE. This is used to describe the interaction of the shared resource context between Factory and Mapper.

The deployment location of Factory and Mapper is the Globus Toolkit container, distinguished by the base URL at port 8880 as the operating port used in this example.

6.2.3 Checking Diagrams

The Allocator service is syntactically parsed and reveals no syntax errors.

6.3 Formal Specification

The formal specifications of the services are automatically generated, using the following command lines in the 'lotos' directory of CRESS. Figure 18 illustrates how the specification is achieved. The following two commands generates of Factory, Mapper and Resource interfaces, and then their formal behaviour are manually completed and then automatically combined into the composed service specification of Allocator.

```
touch gs.base
tlotos gs
```

Upon the execution of the 'tlotos' command, the specification generated was found to be erroneous with the output listed below. It was detected that there was use of the variable 'scheme' which was supposed to be 'scheme2'. This leads to the second reported error as the use of 'scheme2' is invalid for node 2 (indicated by

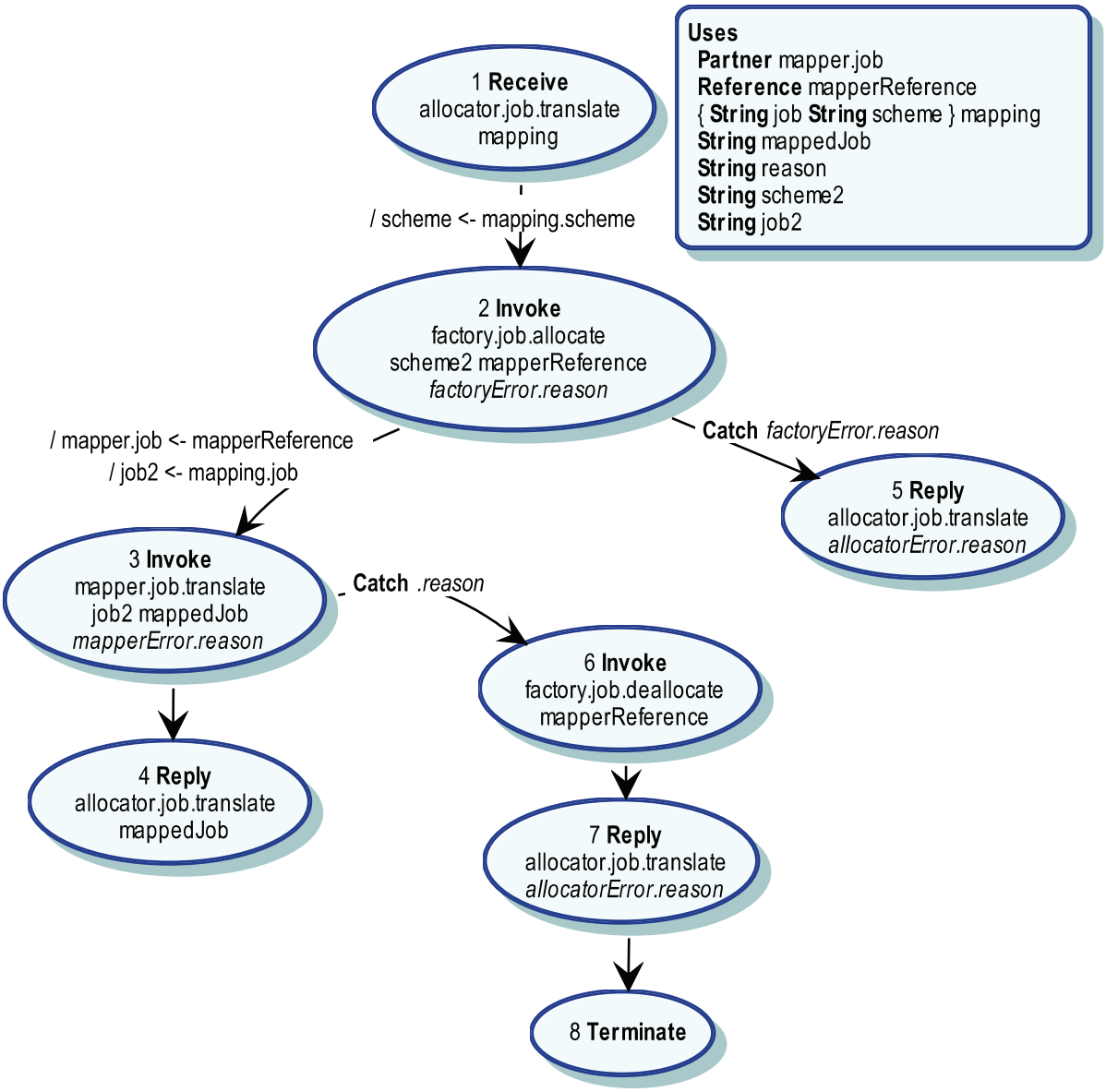


Figure 16: Allocator CRESS Diagram

Deploys -a -b 2 -c -n 1 -o 5 -r -m RESOURCE / ALLOCATOR

ALLOCATOR	allo	urn:Allocator	localhost:8080/active-bpel	
FACTORY	factory	urn:Factory	localhost:8880/wsrf	-
MAPPER	mapper	urn:Mapper	localhost:8880/wsrf	-

Figure 17: Grid Service Configuration

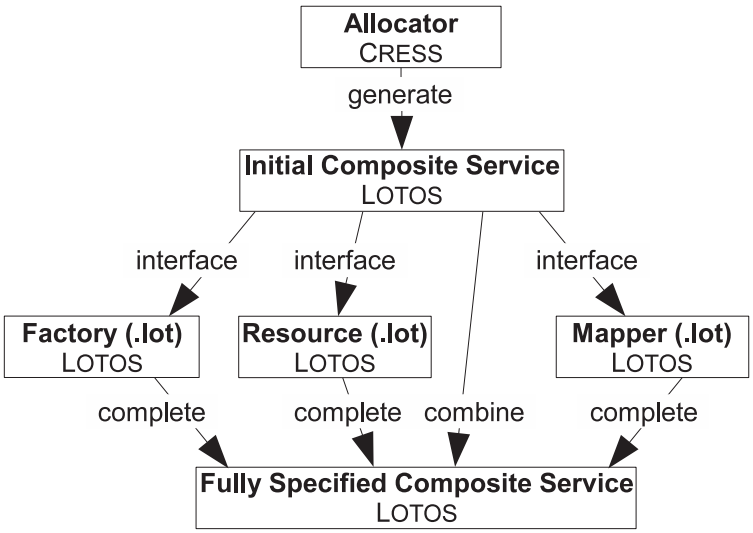


Figure 18: Specifying Factory, Mapper and Resource

ALLOCATOR.2). This semantic error is corrected in the CRESS diagram and the generation of the specification completes successfully.

[Error] cress_check: ALLOCATOR - undefined type for 'scheme'
[Error] cress_check: ALLOCATOR - unknown parameter 'scheme2' for output ALLOCATOR.2

The phantom partner Resource will have an empty LOTOS process defined for manual completion. The Factory and Mapper service have their interface behaviour generated, inferred from their operation signatures. These behaviours are completed as files resource.lot, factory.lot and mapper.lot, which are then included into the overall behaviour by running the same commands again.

6.3.1 Resource Specification

The code skeleton generated below for Resource phantom partner is simply a LOTOS Process definition with a simple exit as it has no description in the CRESS diagram, but takes on a phantom role for the purpose of being an implied shared resource in this case study.

```
Process RESOURCE [resource] : Exit(States) :=        (* RESOURCE phantom *)
  Exit(AnyStates)                                    (* dummy exit *)
EndProc                                              (* end RESOURCE *)
```

This Resource process is then manually completed, listed in Appendix B.1.1. The completed behaviour has synchronsiation events for Factory and Mapper. For successful allocation, Resource partner returns true with a reference to the allocated resource for Factory (e.g. reference 1 for SIC92 occupational scheme). Unsuccessful allocation will be returned with a false and an "Unknown scheme" message. Resource synchronises with Mapper using the resource reference in the request to handle translation requests. It defines a Translation data type with a translate operation that returns the mapped values. Successful translation, implying the job is known, returns true with the mapped value. Unsuccessful translation returns a False with an "Unknown job" message.

6.3.2 Factory Specification

The LOTOS code below is the interface behaviour generated in the overall specification for Factory. The Factory process specifies the 'resource' gate belonging to that of the phantom partner Resource.

```
Process FACTORY [factory,resource] : Exit(States) :=   (* FACTORY partner *)
  factory !job !allocate ?scheme2:Text;                (* 'allocate' input *)
  (
    factory !job !allocate !AnyNat;                    (* 'allocate' output *)
    FACTORY [factory,resource]                         (* repeat behaviour *)
  []                                                   (* or *)
    factory !job !allocate !FactoryError !AnyText;     (* 'FactoryError' fault *)
```

Allocator Automated	Factory Manual	Mapper Manual	Resource Manual
479	24	22	91

Table 6: Specification Summary of Allocator, Factory, Mapper, and Resource

```
      FACTORY [factory,resource]                    (* repeat behaviour *)
   )
[]                                                  (* or *)
   factory !job !deallocate ?mapperReference:Nat;   (* 'deallocate' input *)
   FACTORY [factory,resource]                       (* repeat behaviour *)
EndProc                                             (* end FACTORY *)
```

This specification is completed and listed in Appendix B.1.2. The Factory's 'allocate' operation synchronises with Resource using the scheme as input, and expects a response from Resource which is a successful or unsuccessful allocation as described in section 6.3.1. Factory then responds accordingly to Allocator at node 2. The deallocate operation is left specified as it is, since the behaviour is sufficient for analysis.

6.3.3 Mapper Specification

Mapper's interface behaviour is generated in a similar way to Factory, which contains a 'resource' gate for communication with the Resource partner. The completed behaviour is listed in Appendix B.1.2. The completed Mapper behaviour synchronises with the allocated resource for a translation request, and reads the response as a successful or unsuccessful translation from Resource. Mapper then responds accordingly with the mapped result, or with a mapperError fault for unsuccessful translation, to Allocator at node 3.

6.3.4 Allocator Specification

Allocator's behaviour is automatically and fully specified in LOTOS. Appendix B.2.1 lists the high-level structure of its specification, which includes the developer's specified behaviour for Resource, Factory and Mapper. In its overall specification, the ALLOCATOR process is synchronised with the RESOURCE process at its gate. The actual behaviour of the Allocator begins from the ALLOCATOR_1 process call (line 32), and does not interact with RESOURCE. FACTORY and MAPPER are exposed to the 'resource' gate in the behaviour expression of the ALLOCATOR process, which allows them to synchronise.

6.3.5 Summary

Apart from manual specification of partners Factory, Mapper, and Resource, the formal specification is fully automated with all data types and behaviour are specified. Even the manual specification itself is added to interface behaviour that is automatically generated. Using this specification, formal validation and verification can be performed. Table 6 lists the code summary in terms of lines of code generated and manually provided.

6.4 Formal Analysis

For illustration the Allocator composed service is subject to verification followed by validation, as analysis can be performed in any order.

6.4.1 Formal Verification

The informal description of the properties desired of the service is as follows:

- Allocator's translation request should result in either a successful translation, or faults of unknown job or scheme
- Allocator's translation request using the SOC2000 scheme, should result in unknown job or only numerical mappings.

- Allocator should begin its service with no other operations apart from the 'translate' operation that takes a Mapping request
- Freedom from deadlock
- Freedom from livelock

The developer specifies the enumeration and properties in the following CLOVE syntax. There is a specification of the enumeration of strings. Note that there is no explicit definition of the Mapping values. However these string values will be used in the construction of the value of Mapping, enumerating the two string fields in the structure to a total combination of 36 diferent Mapping values (6×6) where there are both valid and invalid values. These are sufficient to verify all of Allocator's key behaviour. In executing the verification, there is automated annotation of the LOTOS specification, property translation to μ-calculus, and generation of C implementation for data enumeration, which are respectively listed in Appendix B.3.1, B.3.2 and B.3.3. There is use of a regular expression in General_Response property to match the string values for successful job mapping translations returned by Allocator, instead of using '?string' which only caters for spaces and alphanumeric characters. Therefore this expression is specified verbatim in μ-calculus syntax instead of using the **signal** construct. This is also similarly applied in the SOC200_Safety property. The C data enumeration is only one of line code; it is effectively a macro that lists the values of the six strings specified.

```
enum_strings('SOC2000,'SIC92,'bookbinder,
             'cab driver,'nurse,'private detective)

initials(signal(allocator.job.translate,?mapping))

property(General_Response,
  response(global,
    signal(allocator.job.translate,?mapping),
      choice_any((''ALLOCATOR !JOB !TRANSLATE'' # '' !'' # ''[./a-zA-Z0-9]*''),
        signal(allocator.job.translate,allocatorError,'unknown scheme),
        signal(allocator.job.translate,allocatorError,'unknown job))))

property(SOC2000_Safety,
  response(global,
    (''ALLOCATOR !JOB !TRANSLATE'' # '' !'' # 'MAPPING (''.*'', ''SOC2000'')'),
      choice_any((''ALLOCATOR !JOB !TRANSLATE'' # '' !'' # ''[0-9]*''),
        signal(allocator.job.translate,allocatorError,'unknown job))))
```

The verification is successful with the following results, establishing the correctness of the specification.

```
Verifying ALLOCATOR GENERAL RESPONSE ...      TRUE
Verifying ALLOCATOR SOC2000 SAFETY ...        TRUE
Verifying          DEADLOCK FREEDOM ...       TRUE
Verifying          LIVELOCK FREEDOM ...       TRUE
Verifying          INITIALS SAFETY ...        TRUE
```

6.4.2 Formal Validation

The following are MUSTARD scenarios defined for the Allocator service, which are straightforward and specific tests. Their LOTOS translations are listed in Appendix B.4.1. Some of the validation scenarios have slight overlap with with the General_Response and SOC2000_Safety properties, but however their analysis is much quicker in a fraction of the time compared to verification. This allows immediate diagnostic pertaining to a specific case, but more importantly with a different focus. For example, SOC2000_Safety checks for valid reponses as a safety, but checking a response's actual values based on inputs would be more practical and faster with validation, particularly when using values beyond the enumeration covered by the verification (see SOC2_Unknown scenario where Sailor is used). Validation for the grid partner services are not performed as the composed service is rather straightforward and will definitely test both the partner services indirectly with a variety of Mapping values to Allocator. The Unknown_Nurse test is a specific safety validation, where the scenario of a possible mapping of a nurse is refused.

test(SOC2_Nurse,
  succeed(
    send(allocator.job.translate, Mapping(′Nurse,′SOC2000)),
    read(allocator.job.translate, ′3211)))

test(SIC_Nurse,
  succeed(
    send(allocator.job.translate, Mapping(′Nurse,′SIC92)),
    read(allocator.job.translate, ′95.14)))

test(SOC2_Unknown,
  succeed(
    send(allocator.job.translate, Mapping(′Sailor,′SOC2000)),
    read(allocator.job.translate, allocatorError, ′Unknown job)))

test(Unknown_Nurse,
  refuse(
    send(allocator.job.translate, Mapping(′Nurse,′Unknown)),
    read(allocator.job.translate, ′3211)))

test(SOC2_Unknown_Scheme,
  succeed(
    send(allocator.job.translate, Mapping(′Sailor,′SOC20000)),
    read(allocator.job.translate, allocatorError, ′Unknown scheme)))

The results listed below shows that the validation is successful for the specified scenarios. These validation results, along with the verification outcomes, establish a level of confidence in the behaviour of the services, where specified properties are satisfied and the tests demonstrate correct functionality.

Test ALLOCATOR SOC2 Nurse ...	Pass	1 succ	0 fail	1.3 secs
Test ALLOCATOR SIC Nurse ...	Pass	1 succ	0 fail	0.5 secs
Test ALLOCATOR SOC2 Unknown ...	Pass	1 succ	0 fail	0.4 secs
Test ALLOCATOR Unknown Nurse ...	Pass	1 succ	0 fail	0.4 secs
Test ALLOCATOR SOC2 Unknown Scheme ...	Pass	1 succ	0 fail	0.4 secs

6.5 Implementation

Implementation of the Allocator service is fully automated. The developer provides the implementation and deployment configuration (resource home and resource configuration) for the Factory and Mapper grid service partners, which are deployed to the GT4 framework. There is no need for the implementation of Resource as it is a 'phantom' partner used for the *specification* of shared resource. The actual implementation is an AllocatorResource class that implements the Resource object in GT4. Factory and Mapper are developed with the implementation and configuration of shared resource context. Figure 19 illustrates how the implementation is achieved for Factory and Mapper that are automatically compiled by the framework. For grid partner services, only WSDL service interfaces are generated, from which the code stubs are obtain. There are however no code skeleton and deployment descriptors generated as the GT4 service development framework has a different approach. Sections 6.5.1 and 6.5.2 describe their implementation. The developer then proceeds to implementing the services. The grid service configuration diagram is configured as follows:

Deploys -c -b 2 -o 120 -m RESOURCE / ALLOCATOR

The developer builds the implementation of the Allocator service and partner grid services by executing the following command line (in CRESS's 'bpel' directory). This generates the WSDL service interfaces and common definitions for the three services, and the deployment descriptor and BPEL file for Allocator. The grid service implementations provided by the developer for Factory and Mapper are automatically compiled and built using GT4, and packaged as grid service archives (.gar) for deployment in GT4 container. The command line is executed once to first compile and build the Factory service, which creates a Java .jar library which is imported by the Mapper service. The same command line is executed again which then builds both of the services.

cress_expand -v gs main.bpel

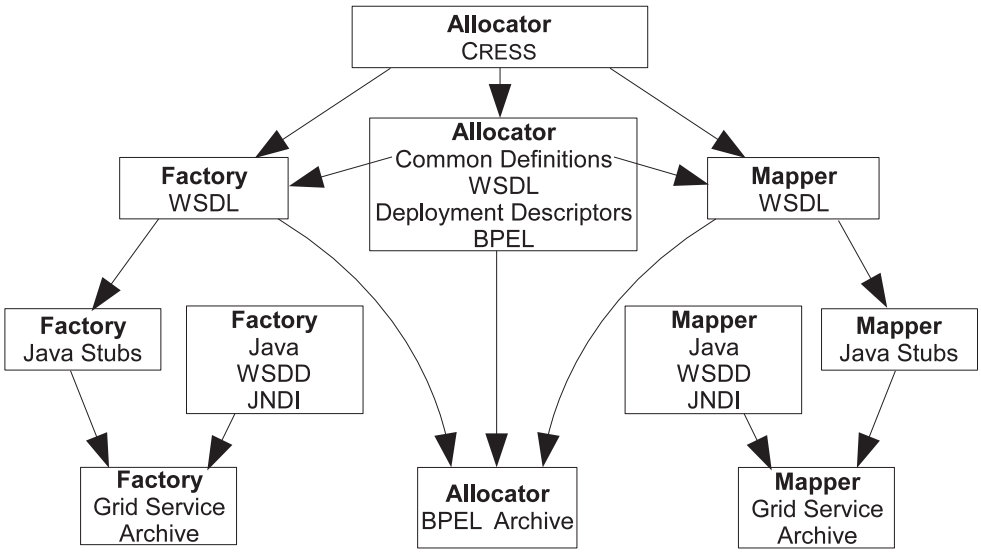


Figure 19: Implementation of Factory and Mapper

6.5.1 Implementation of Factory Service

The Factory service implements the operations 'allocate' and 'deallocate'. There is also the need to implement the 'resource home' and 'resource' that the Factory uses to manage resources. This follows the GT4 resource context framework, where the former creates the resource in the container and the latter is the resource itself which contains the allocated scheme information to return mapped values. These are respectively implemented as AllocatorResourceHome and AllocatorResource Java classes. The implementation uses the code stubs generated from its WSDL service interface and common definitions listed in Appendix B.5.5 and B.5.9. These are packaged into the grid service archive for deployment, together with the developer's specified WSDD (Web Service Deployment Descriptor) and JNDI (Java Naming and Directory Interface) resource configuration which are listed in Appendix B.5.3 and B.5.4. The WSDD is a basic one for GT4, with values configured only for the Factory service name and service WSDL location. The JNDI configures the namespace and the resource class implementation that the Factory uses.

The 'allocate' operation takes a String which contains the occupational scheme to be allocated, and invokes AllocatorResourceHome to create an instance of AllocatorResource hosting the scheme for mapping use. Upon successful resource creation, the operation returns an endpoint to the WS-Resource which the Mapper service can use. The 'deallocate' operation takes the endpoint to the WS-Resource, and uses AllocatorResourceHome to remove the identified resource. A Factory Java code snippet is listed in Appendix B.5.2. The Factory's grid service archive is deployed into the GT4 container.

6.5.2 Implementation of The Mapper Service

The Mapper service implements the 'translate' operation which uses the implicit method to associate the resource. This is done by Allocator setting mapper.job dynamically with the returned endpoint value. The 'translate' operation uses the associated resource to obtain the mapped value as a String and returns it. This is just the few lines of code shown below. Mapper uses the resource implementation already developed within the Factory service. Its generated WSDL service interface (Appendix B.5.7 and common definitions in Appendix B.5.9) are packaged into a grid service archive for deployment, together with the developer's specified WSDD and JNDI resource configuration. The WSDD is mostly similar to that of Factory apart from values pertaining to the Mapper service and its WSDL location. Its JNDI file is simply configured as a reference (see Appendix B.5.6) to the resource configured in Factory, establishing sharing of the resource context. The Mapper's grid service archive is deployed into the GT4 container.

```
AllocatorResourceHome home = (AllocatorResourceHome) context.getResourceHome();
AllocatorResource allocatorResource = (AllocatorResource) context.getResource();
return allocatorResource.mapValue(string.value);
```

6.5.3 Implementation of The Allocator Service

As mentioned, the Allocator service is fully implemented, where its WSDL service interface (Appendix B.5.8), deployment descriptors (PDD and WSDL catalogue), and BPEL code are automatically generated, compiled into a BPEL archive, and deployed in ActiveBPEL. Their code snippets are all listed in Appendix B.5.8 to B.5.11. The PDD code gives the configuration for Mapper partner as dynamic, where its endpoint (WS-Addressing) for the WS-Resource is bound during Allocator's execution in order to invoke it.

6.6 Implementation Validation

6.6.1 Allocator Implementation Validation

The deployed service can be validated as much as the formal validation. This requires the MINT properties (Appendix B.5.12) that are also generated previously in the automated implementation, and translation of the MUSTARD scenarios into MINT scenarios (Appendix B.5.13) for test interpretation by MINT. The Allocator's service is expected to respond within 15 seconds. The developer executes the following command line in <CRESS> /gs/allocator, as an illustration of using the MUSTARD tool directly instead of cress_validate.

```
mustard -v gs ALLOCATOR
```

The results of validation pass, which agrees with specification validation, confirming that the implementation behaviour is functioning the same as the specification with regard to the context of validating these scenarios.

```
Test ALLOCATOR SOC2 Nurse ...                Pass    1 succ    0 fail    1.8 secs
Test ALLOCATOR SIC Nurse ...                 Pass    1 succ    0 fail    1.3 secs
Test ALLOCATOR SOC2 Unknown ...              Pass    1 succ    0 fail    1.3 secs
Test ALLOCATOR Unknown Nurse ...             Pass    1 succ    0 fail    1.2 secs
Test ALLOCATOR SOC2 Unknown Scheme ...       Pass    1 succ    0 fail    1.2 secs
```

6.6.2 Performance Evaluation

Suppose the service is expected to handle up to 200 mappings simultaneously. The developer first evaluates the consistency of behaviour by sequential performance testing, where the results reported below show that functionality is consistent.

```
Test ALLOCATOR SOC2 Nurse ...                Pass    200 succ          0 fail 0.1 secs
                    Sequential               0 inco true cons 0.1 secs .. 1.1 secs
Test ALLOCATOR SIC Nurse ...                 Pass    200 succ          0 fail 0.1 secs
                    Sequential               0 inco true cons 0.1 secs .. 1.1 secs
Test ALLOCATOR SOC2 Unknown ...              Pass    200 succ          0 fail 0.1 secs
                    Sequential               0 inco true cons 0.1 secs .. 1.2 secs
Test ALLOCATOR Unknown Nurse ...             Pass    200 succ          0 fail 0.1 secs
                    Sequential               0 inco true cons 0.0 secs .. 1.1 secs
Test ALLOCATOR SOC2 Unknown Scheme ...Pass    200 succ          0 fail 0.1 secs
                    Sequential               0 inco true cons 0.0 secs .. 1.1 secs
```

The concurrent performance evaluation with a load of 200 requests also passes, but with a much slower completion time than the sequential evaluation (which is reasonable). The grid partner services were also indirectly tested through invocations from Allocator BPEL service, which implies that the grid partner services and their environment (GT4 container) can handle this load.

```
Test ALLOCATOR SOC2 Nurse ...                Pass    200 succ          0 fail 0.0 secs
                    Concurrent               0 inco true cons 7.5 secs .. 0.0 secs
Test ALLOCATOR SIC Nurse ...                 Pass    200 succ          0 fail 0.0 secs
                    Concurrent               0 inco true cons 5.0 secs .. 0.0 secs
Test ALLOCATOR SOC2 Unknown ...              Pass    200 succ          0 fail 0.0 secs
                    Concurrent               0 inco true cons 4.7 secs .. 0.0 secs
Test ALLOCATOR Unknown Nurse ...             Pass    200 succ          0 fail 8.8 secs
                    Concurrent               0 inco true cons 4.6 secs .. 0.0 secs
Test ALLOCATOR SOC2 Unknown Scheme ...Pass    200 succ          0 fail 9.3 secs
                    Concurrent               0 inco true cons 4.2 secs .. 0.0 secs
```

7 Evaluation

Four case studies have been used to illustrate the methodology in action from the developer's perspective. There are other service composition case studies produced earlier in [4, 5, 7, 8] to demonstrate the methodology in practice. Service compositions are described in the CRESS high-level graphical notation. Formal specifications and implementations are automatically generated, enabling rapid service prototyping and development. Partner service specifications and implementations are semi-automated, generating code when possible to reduce the developer's effort. Formal validation and verification are supported with high-level analysis descriptions (scenarios and properties). Formal analyses are automatically executed and interpreted on behalf of users, which minimises the technical interface to underlying tools, and encourages the use in a practical manner of a productive and rigorous service development. Post-implementation analysis is available as implementation validation, where functionality testing and performance can be evaluated quickly, supported by a high-level notation and automated tool support. The integration into a methodology enables rigorous development of composing services.

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] P. S. Lambert, K. L. L. Tan, K. J. Turner, V. Gayle, R. Sinnott, and K. Prandy. Data curation standards and the messy world of social science occupational information resources. In *Proc. 2nd. International Digital Curation Conference*, pages 2.1–2.8, Glasgow, Nov. 2006.

[3] K. L. L. Tan, V. Gayle, P. S. Lambert, R. O. Sinnott, and K. J. Turner. GEODE - Sharing occupational data through the grid. In S. J. Cox, editor, *Proc. 5th. UK e-Science All Hands Meeting*, pages 534–541. National e-Science Centre, Edinburgh, Sept. 2006.

[4] K. L. L. Tan and K. J. Turner. Orchestrating grid services using BPEL and Globus Toolkit 4. In M. Merabti, R. Pereira, C. Oliver, and O. Abuelma'atti, editors, *Proc. 7th PGNet Symposium*, pages 31–36. School of Computing, Liverpool John Moores University, Liverpool, UK, June 2006.

[5] K. L. L. Tan and K. J. Turner. Automated analysis and implementation of composed grid services. In D. Dranidis and I. Sakellariou, editors, *Proc. 3rd South-East European Workshop on Formal Methods*, pages 51–64. South-East European Reseach Centre, Thessaloniki, Greece, Nov. 2007.

[6] 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.

[7] 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.

[8] K. J. Turner and K. L. L. Tan. A rigorous approach to orchestrating grid services. *Computer Networks*, 51(15):4421–4441, Oct. 2007.

A Composed Web Services

A.1 Partner Specification

A.1.1 Approver Completed LOTOS Specification

```
Process APPROVER [approver] : Exit(States) :=                    (* start APPROVER *)
  approver !loan !approve ?proposal:Proposal;                    (* proposal request *)
  (
    [starts(getName(proposal),t(K)∼e∼n)] ⇒                       (* Ken? *)
      approver !loan !approve !number(+,t(3),t(8));              (* rate 3.8% *)
      APPROVER [approver]                                        (* repeat *)
  []
    [not(starts(getName(proposal),t(K)∼e∼n))] ⇒                  (* else *)
      (
    [contains(getAddress(proposal),t(S)∼c∼o∼t∼l∼a∼n∼d)] ⇒        (* Scotland? *)
      approver !loan !approve !number(+,t(4),t(1));              (* rate 4.1% *)
      APPROVER [approver]                                        (* repeat *)
      []
    [not(contains(getAddress(proposal),t(S)∼c∼o∼t∼l∼a∼n∼d))] ⇒   (* else *)
      (
        [getAmount(proposal) lt number(+,t(1)∼5∼0∼0∼0,<>)] ⇒     (* < 15000? *)
          approver !loan !approve !number(+,t(4),t(4));          (* rate 4.4% *)
          APPROVER [approver]                                    (* repeat *)
      []
        [getAmount(proposal) ge number(+,t(1)∼5∼0∼0∼0,<>)] ⇒     (* else *)
          approver !loan !approve !refusal                       (* refuse *)
        !t(L)∼o∼a∼n∼ ^ ∼u∼n∼a∼c∼c∼e∼p∼t∼a∼b∼l∼e;
          stop                                                   (* stop *)
      )
      )
  )
EndProc                                                          (* end APPROVER *)
```

A.1.2 Assessor Interface LOTOS Specification

```
Process ASSESSOR [assessor] : Exit(States) :=        (* ASSESSOR partner *)
  assessor !loan !assess ?proposal:Proposal;         (* 'assess' input *)
  assessor !loan !assess !AnyText;                   (* 'assess' output *)
  ASSESSOR [assessor]                                (* repeat behaviour *)
EndProc                                              (* end ASSESSOR *)
```

A.1.3 Assessor Completed LOTOS Specification

```
Process ASSESSOR [assessor] : Exit(States) :=                    (* start ASSESSOR *)
  assessor !loan !assess ?proposal:Proposal;                     (* assessment request *)
  (
    [ends(getName(proposal),t(T)∼u∼r∼n∼e∼r)] ⇒(* Turner? *)
      assessor !loan !assess !t(l)∼o∼w;                          (* risk low *)
      ASSESSOR [assessor]                                        (* repeat *)
  []
    [not(ends(getName(proposal),t(T)∼u∼r∼n∼e∼r))] ⇒(* else *)
      (
        [getAddress(proposal) Eq (t(U)∼K)] ⇒                     (* UK? *)
      assessor !loan !assess !t(m)∼e∼d∼ii∼u∼m;(* risk medium *)
      ASSESSOR [assessor]                                        (* repeat *)
      []
        [Not(getAddress(proposal) Eq (t(U)∼K))] ⇒(* else *)
```

```
        assessor !loan !assess !t(h)~ii~g~h;          (* risk high *)          15
        ASSESSOR [assessor]                           (* repeat *)             16
        )                                                                       17
    )                                                                           18
EndProc                                               (* end ASSESSOR *)       19
```

A.1.4 Dealer1 Interface LOTOS Specification

```
Process DEALER1 [dealer1] : Exit(States) :=           (* DEALER1 partner *)    1
  dealer1 !car !cancel ?offer:Offer;                  (* 'cancel' input *)     2
  DEALER1 [dealer1]                                   (* repeat behaviour *)   3
[]                                                    (* or *)                 4
  dealer1 !car !order ?offer:Offer;                   (* 'order' input *)      5
  DEALER1 [dealer1]                                   (* repeat behaviour *)   6
[]                                                    (* or *)                 7
  dealer1 !car !quote ?need:Need;                     (* 'quote' input *)      8
  dealer1 !car !quote !AnyOffer;                      (* 'quote' output *)     9
  DEALER1 [dealer1]                                   (* repeat behaviour *)   10
EndProc                                               (* end DEALER1 *)        11
```

A.1.5 Dealer1 Completed LOTOS Specification

```
Process DEALER1 [dealer1] : Exit(States) :=           (* start DEALER1 *)      1
                                                                                2
  DEAL1 [dealer1]                                     (* instantiate deal sub-process *) 3
   (vehicles(vehicle(...),0 Of Nat,<> Of Deals,<> Of Deals)                     4
                                                                                5
Where                                                                           6
                                                                                7
  Type Vehicle Is NumberOps,TextOps                   (* vehicle (model, price, delivery) *) 8
      AnyVehicle: -> Vehicle                          (* void vehicle *)       9
      vehicle: Text,Number,Nat -> Vehicle             (* model, price, delivery *) 10
      getModel: Vehicle -> Text                       (* get model *)          11
      getPrice: Vehicle -> Number                     (* get price *)          12
      getDelivery: Vehicle -> Nat                     (* get delivery *)       13
      _eq_,_ne_: Vehicle,Vehicle -> Bool              (* boolean comparison *) 14
  ...                                                                           15
                                                                                16
  Type Vehicles Is String ActualizedBy Vehicle,Boolean Using                    17
  (* vehicles current stock *)                                                  18
  ...                                                                           19
                                                                                20
  Type VehiclesOps Is Vehicles                        (* vehicles operations *) 21
    Opns                                                                        22
      getVehicle: Text,Vehicles -> Vehicle            (* get vehicle for model *) 23
    ...                                                                         24
                                                                                25
  Type Deal Is Offer,Need                             (* deal records a transaction (offer, need) *) 26
    Opns                                                                        27
      AnyDeal: -> Deal                                (* void deal *)          28
      Deal: Offer,Need -> Deal                        (* offer, order *)       29
      getOffer: Deal -> Offer                         (* get offer *)          30
      getNeed: Deal -> Need                           (* get need *)           31
      _eq_,_ne_: Deal,Deal -> Bool                    (* boolean comparison *) 32
    ...                                                                         33
                                                                                34
  Type Deals Is String ActualizedBy Deal,Boolean Using                          35
```

```
(* deals records a list of deal *)                                          36
...                                                                         37
                                                                            38
Type DealsOps Is Deals                          (* deals operations *)     39
  Opns                                                                      40
    getDeal: Offer,Deals → Deal                 (* get deal for offer *)   41
    remove: Deal,Deals → Deals                  (* remove deal from deals *) 42
  ...                                                                       43
                                                                            44
Process DEAL1 [dealer1]                         (* start DEAL1 *)          45
 (vehicles:Vehicles,reference:Nat,quotes,orders:Deals) : Exit(States) :=    46
  dealer1 !car !quote ?need:Need;               (* quote request *)        47
  (                                                                         48
    Let vehicle:Vehicle = getVehicle(getModel(need),vehicles) In (* match *) 49
    Let price:Number = getPrice(vehicle) In     (* get price *)            50
    Let delivery:Nat = getDelivery(vehicle) In(* get delivery *)           51
    Let offer:Offer = offer(reference,dealer1,price,delivery) In            52
    Let quotation:Deal = deal(offer,need) In    (* create offer, quotation *) 53
      dealer1 !car !quote !offer;               (* quote response *)       54
      (                                                                     55
        [price eq Million]→                     (* "infinite" price *)     56
          DEAL1 [dealer1] (vehicles,reference,quotes,orders) (* repeat *)   57
      □                                                                     58
        [price ne Million]→                     (* else *)                 59
          DEAL1 [dealer1]                       (* repeat *)               60
           (vehicles,reference + 1,quotation ∼ quotes,orders) (* update *)  61
      )                                                                     62
  )                                                                         63
□                                                                           64
  dealer1 !car !order ?offer:Offer;             (* order request *)        65
  (                                                                         66
    Let deal:Deal = getDeal(offer,quotes) In    (* get matching deal *)    67
      DEAL1 [dealer1]                           (* move deal from quotes to orders *) 68
       (vehicles,reference,remove(deal,quotes),deal ∼ orders)               69
  )                                                                         70
□                                                                           71
  dealer1 !car !cancel ?offer:Offer;            (* cancel request *)       72
  (                                                                         73
    Let deal:Deal = getDeal(offer,quotes) In    (* get matching deal *)    74
      DEAL1 [dealer1]                           (* remove deal from quotes and orders *) 75
       (vehicles,reference,remove(deal,quotes),remove(deal,orders))         76
  )                                                                         77
...                                                                         78
```

A.2 Partner Implemenetation

A.2.1 Approver Completed Implementation

```
package FirstRate;                                                          1
...                                                                         2
class LoanBindingImpl implements LoanPort {    // loan port binding        3
  ...                                                                       4
                                                                            5
  public float approve(Proposal2 proposal) throws RemoteException {         6
    String name = proposal.getName();           // get proposal name        7
    String address = proposal.getAddress();     // get proposal address     8
```

```
        int amount = proposal.getAmount().intValue(); // get proposal amount

        float rate = 0.0f;                                  // declare loan rate
        if (name.startsWith(″Ken″))                         // name starts ″Ken″?
            rate = 3.7f;
        else if (address.indexOf(″Scotland″) != −1)         // address has ″Scotland″?
            rate = 4.1f;
        else if (amount <= 15000)                           // amount less than 15000?
            rate = 4.4f;
        else {                                              // otherwise
            ... throw fault loan unacceptable
        return (rate);                                      // return loan rate
    }
}
```

A.2.2 Assessor Code Skeleton

```

package RiskTaker;

public class LoanBindingImpl implements RiskTaker.LoanPort{
    public java.lang.String assess(LoanStarDefs.Proposal proposal)
        throws java.rmi.RemoteException {
            return null;
    }

}
```

A.2.3 Assessor Completed Implementation

```
    public String assess(Proposal proposal) throws RemoteException {
        String name = proposal.getName();                   // get proposal name
        String address = proposal.getAddress();             // get proposal address
        int amount = proposal.getAmount().intValue(); // get proposal amount

        String risk;                                        // declare risk assessment
        if (name.endsWith(″Turner″))                        // name ends ″Turner″?
            risk = ″low″;
        else if (address.equals(″UK″))                      // address exactly ″UK″?
            risk = ″medium″;
        else                                                // otherwise
            risk = ″high″;
        return (risk);                                      // return risk assessment
```

A.2.4 Dealer1 Code Skeleton

```
package BigDeal;
...
    public class CarBindingImpl implements BigDeal.CarPort {
        public void cancel(DoubleQuoteDefs.Offer offer) throws java.rmi.RemoteException
        }

        public void order(DoubleQuoteDefs.Offer offer) throws java.rmi.RemoteException {
        }

        public DoubleQuoteDefs.Offer quote(DoubleQuoteDefs.Need need)
            throws java.rmi.RemoteException {
            return null;
```

```
    }
}
```

A.2.5 Dealer1 Completed Implementation

```
...
  public Offer quote(Need need) throws RemoteException {
    String name = need.getName();                    // get need name
    String address = need.getAddress();              // get need address
    String model = need.getModel();                  // get need model

    int randomReference = ((int) (Math.random() * 90)) + 10;
    NonNegativeInteger reference =                   // set random reference number
      new NonNegativeInteger(Integer.toString(randomReference));

    String dealer = ″BigDeal″;                       // set dealer name
    float price;                                     // declare price
    NonNegativeInteger delivery;                     // declare delivery period

    if (model.equals(″Mondeo″)) {                    // Mondeo?
      price = 20000.0f;
      delivery = new NonNegativeInteger(″15″);
    }
    else if (model.equals(″A5″) {                    // A5?
      price = 33000.0f;
      delivery = new NonNegativeInteger(″30″);
    }
    else if (model.equals(″Megane″)) {               // Megane?
      price = 11000.0f;
      delivery = new NonNegativeInteger(″5″);
    }
    else {                                           // unstocked car
      price = 1000000.0f;
      delivery = new NonNegativeInteger(″0″);
    }

    Offer offer =                                    // set offer
      new Offer(reference, dealer, price, delivery);
    return (offer);                                  // return offer
  }
```

A.3 Formal Specification

A.3.1 Lender LOTOS Specification

```
Specification WSSystem [lender] : Exit(States)

Library                          (* library *)
  BaseTypes,                     (* base types *)
  Boolean,                       (* boolean *)
  ...

Behaviour
  LENDER [lender]                (* call LENDER process *)

Where                            (* local definitions *)
...
```

```
Type Operation Is                                  (* operation name *)
  Sorts Operation                                  (* operation name sort *)
  Opns                                             (* operation name operations *)
    approve,assess,quote: -> Operation             (* operation name constants *)
  ...

Type Partner Is TextOps                            (* partner name *)
  Opns                                             (* partner operations *)
    approver,
    assessor,
    lender,
    ... : -> Text
  Eqns                                             (* partner equations *)
    OfSort Text                                    (* define text operations *)
      approver = t(F)~ii~r~s~t~R~a~t~e;
      assessor = t(R)~ii~s~k~T~a~k~e~r;
      lender = t(L)~o~a~n~S~t~a~r;

Type Port Is                                       (* port name *)
  Sorts Port                                       (* port name sort *)
  Opns                                             (* port name operations *)
    loan: -> Port                                  (* port name constants *)

Type Proposal Is BaseTypes                         (* proposal record *)
...

Process APPROVER [approver] : Exit(States) :=      (* APPROVER partner *)
...

Process ASSESSOR [assessor] : Exit(States) :=      (* ASSESSOR partner *)
...

Process LENDER [lender] : Exit(States) :=          (* LENDER service *)
  Hide approver,assessor In                        (* hide internal gates *)
    (
      APPROVER [approver]                          (* call APPROVER partner *)
    |||                                            (* interleaved with *)
      ASSESSOR [assessor]                          (* call ASSESSOR partner *)
    )
  |[approver,assessor]|                            (* synchronised with partners *)
    LENDER_1 [approver,assessor,lender] (* call main process *)
      ...

Where                                              (* local definitions *)
...

Process LENDER_1 [approver,assessor,lender] ...
  lender !loan !quote ?proposal:Proposal; (* LENDER receive 1 *)
...
EndSpec                                            (* end WSSystem *)
```

A.3.2 Supplier LOTOS Specification

```
Specification WSSystem [supplier] : Exit(States)   (* WS system *)
```

```
Library                                          (* library *)
...
Behaviour
  SUPPLIER [supplier]                            (* call SUPPLIER process *)

Where                                            (* local definitions *)
...
Type Need Is BaseTypes                           (* need record *)
...
Type Offer Is BaseTypes                          (* offer record *)
...
Process DEALER1 [dealer1] : Exit(States) :=      (* DEALER1 partner *)
...
Process DEALER2 [dealer2] : Exit(States) :=      (* DEALER2 partner *)
...
Process SUPPLIER [supplier] : Exit(States) :=    (* SUPPLIER service *)
  Hide dealer1,dealer2 In                        (* hide internal gates *)
    (
      DEALER1 [dealer1]                          (* call DEALER1 partner *)
    |||                                          (* interleaved with *)
      DEALER2 [dealer2]                          (* call DEALER2 partner *)
    )
  |[dealer1,dealer2]|                            (* synchronised with partners *)
    SUPPLIER_1 [dealer1,dealer2,supplier]        (* call main process *)
...
EndSpec                                          (* end WSSystem *)
```

A.3.3 Broker LOTOS Specification

```
Specification WSSystem [broker] : Exit(States)   (* WS system *)

Library                                          (* library *)
...

Behaviour
  BROKER [broker]                                (* call BROKER process *)

Where                                            (* local definitions *)

...
Type Schedule Is BaseTypes                       (* schedule record *)
...

Process APPROVER [approver] : Exit(States) :=    (* APPROVER partner *)

Process ASSESSOR [assessor] : Exit(States) :=    (* ASSESSOR partner *)

Process BROKER [broker] : Exit(States) :=        (* BROKER service *)
  Hide lender,supplier In                        (* hide internal gates *)
    (
      LENDER [lender]                            (* call LENDER partner *)
    |||                                          (* interleaved with *)
      SUPPLIER [supplier]                        (* call SUPPLIER partner *)
    )
  |[lender,supplier]|                            (* synchronised with partners *)
    BROKER_1 [broker,lender,supplier]            (* call main process *)
```

```
...

Process DEALER1 [dealer1] : Exit(States) :=          (* DEALER1 partner *)
...
Process DEALER2 [dealer2] : Exit(States) :=          (* DEALER2 partner *)
...
Process LENDER [lender] : Exit(States) :=            (* LENDER service *)
...
Process SUPPLIER [supplier] : Exit(States) :=        (* SUPPLIER service *)
...
EndSpec                                              (* end WSSystem *)
```

A.4 Formal Validation

A.4.1 Approver Translated LOTOS Scenarios

```
Process APPROVER_Low_Rate [approver,OK] : NoExit :=
  approver !loan !approve !proposal(t(K)~e~n~ ^ ~S~m~ii~t~h,
    t(L)~ii~v~e~r~p~o~o~l~ ^ ~U~K,
    Number(+,t(6)~0~0~0,<>));
  approver !loan !approve !Number(+,t(3),t(7));
  OK;
  Stop
EndProc (* APPROVER_Low_Rate *)


Process APPROVER_Medium_Rate [approver,OK] : NoExit :=
  approver !loan !approve !proposal(t(A)~n~g~u~s~ ^ ~O~g,
    t(A)~ii~r~t~h~ ^ ~S~c~o~t~l~a~n~d,
    Number(+,t(2)~0~0~0~0,<>));
  approver !loan !approve !Number(+,t(4),t(1));
  OK;
  Stop
EndProc (* APPROVER_Medium_Rate *)

Process APPROVER_High_Rate [approver,OK] : NoExit :=
  approver !loan !approve !proposal(t(N)~a~n~c~y~ ^ ~T~u~r~n~e~r,
    t(M)~a~n~c~h~e~s~t~e~r~ ^ ~E~n~g~l~a~n~d,
    Number(+,t(1)~4~9~9~9,<>));
  approver !loan !approve !Number(+,t(4),t(4));
  OK;
  Stop
EndProc (* APPROVER_High_Rate *)

Process APPROVER_Loan_Unacceptable [approver,OK] : NoExit :=
  approver !loan !approve !proposal(t(II)~a~n~ ^ ~C~a~r~e~y,
    t(C)~r~o~y~d~o~n~ ^ ~E~n~g~l~a~n~d,
    Number(+,t(1)~5~0~0~0,<>));
  approver !loan !approve !refusal !t(l)~o~a~n~ ^ ~u~n~a~c~c~e~p~t~a~b~l~e;
  OK;
  Stop
EndProc (* APPROVER_Loan_Unacceptable *)
```

A.4.2 Assessor Translated LOTOS Scenarios

```
Process ASSESSOR_Low_Risk [assessor,OK] : NoExit :=
  assessor !loan !assess !proposal(t(M)~ii~k~e~ ^ ~T~u~r~n~e~r,
```

```
      t(C)∼a∼r∼l∼ii∼s∼l∼e∼ ^ ∼U∼K,Number(+,t(2)∼0∼0∼0∼0,<>));
   assessor !loan !assess !t(l)∼o∼w;
   OK;
   Stop
EndProc (* ASSESSOR_Low_Risk *)

Process ASSESSOR_Medium_Risk [assessor,OK] : NoExit :=
   assessor !loan !assess !proposal(t(F)∼r∼e∼d∼ ^ ∼H∼o∼y∼l∼e,
      t(U)∼K,Number(+,t(5)∼0∼0∼0,<>));
   assessor !loan !assess !t(m)∼e∼d∼ii∼u∼m;
   OK;
   Stop
EndProc (* ASSESSOR_Medium_Risk *)

Process ASSESSOR_High_Risk [assessor,OK] : NoExit :=
   assessor !loan !assess !proposal(t(P)∼a∼t∼r∼ii∼c∼e∼ ^ ∼T∼o∼u∼v∼e∼t,
      t(P)∼a∼r∼ii∼s∼ ^ ∼F∼r∼a∼n∼c∼e,Number(+,t(1)∼0∼0∼0,<>));
   assessor !loan !assess !t(h)∼ii∼g∼h;
   OK;
   Stop
EndProc (* ASSESSOR_High_Risk *)
```

A.4.3 Lender Translated LOTOS Scenarios

```
Process LENDER_Little_Low_Risk [lender,OK] : NoExit :=
   lender !loan !quote !proposal(t(N)∼a∼n∼c∼y∼ ^ ∼T∼u∼r∼n∼e∼r,
   t(M)∼a∼n∼c∼h∼e∼s∼t∼e∼r∼ ^ ∼E∼n∼g∼l∼a∼n∼d,Number(+,t(9)∼9∼9∼9,<>));
   lender !loan !quote !Number(+,t(3),t(5));
   OK;
   Stop
EndProc (* LENDER_Little_Low_Risk *)

Process LENDER_No_Risk_Assess_Low [lender,OK] : NoExit :=
   lender !loan !quote !proposal(t(N)∼a∼n∼c∼y∼ ^ ∼T∼u∼r∼n∼e∼r,
   t(M)∼a∼n∼c∼h∼e∼s∼t∼e∼r∼ ^ ∼E∼n∼g∼l∼a∼n∼d,
   Number(+,t(1)∼0∼0∼0∼0,<>));
   (
      lender !loan !quote !Number(+,t(3),t(5));
      Stop
   []
      I;
      OK;
      Stop
   )
EndProc (* LENDER_No_Risk_Assess_Low *)

Process LENDER_Lots_Ken [lender,OK] : NoExit :=
   lender !loan !quote !proposal(t(K)∼e∼n∼ ^ ∼B∼o∼y∼l∼e,
      t(D)∼u∼b∼l∼ii∼n∼ ^ ∼II∼r∼e∼l∼a∼n∼d,Number(+,t(1)∼0∼0∼0∼0,<>));
   lender !loan !quote !Number(+,t(3),t(7));
   OK;
   Stop
EndProc (* LENDER_Lots_Ken *)

Process LENDER_Lots_Scotland [lender,OK] : NoExit :=
   lender !loan !quote !proposal(t(M)∼a∼r∼y∼ ^ ∼D∼u∼n∼c∼a∼n,
```

```
      t(W)~ii~c~k~ ^ ~S~c~o~t~l~a~n~d,Number(+,t(2)~0~0~0~0,<>));
   lender !loan !quote !Number(+,t(4),t(1));
   OK;
   Stop
EndProc (* LENDER_Lots_Scotland *)

Process LENDER_Lots_Under_15000 [lender,OK] : NoExit :=
   lender !loan !quote !proposal(t(S)~a~l~l~y~ ^ ~D~e~a~n,
      t(C)~a~r~d~ii~f~f~ ^ ~W~a~l~e~s,Number(+,t(1)~4~9~9~9,<>));
   lender !loan !quote !Number(+,t(4),t(4));
   OK;
   Stop
EndProc (* LENDER_Lots_Under_15000 *)

Process LENDER_Lots_Exceeds_15000 [lender,OK] : NoExit :=
   lender !loan !quote !proposal(t(II)~a~n~ ^ ~C~a~r~e~y,
      t(C)~r~o~y~d~o~n~ ^ ~E~n~g~l~a~n~d,Number(+,t(1)~5~0~0~0,<>));
   lender !loan !quote !refusal !t(l)~o~a~n~ ^ ~u~n~a~c~c~e~p~t~a~b~l~e;
   OK;
   Stop
EndProc (* LENDER_Lots_Exceeds_15000 *)
```

A.4.4 Dealer1 Translated LOTOS Scenarios

```
Process DEALER1_Mondeo [dealer1,OK] : NoExit :=
   dealer1 !car !quote
      !need(t(M)~a~r~k~ ^ ~F~l~o~w~e~r~s,
            t(U)~ii~s~t~ ^ ~S~c~o~t~l~a~n~d,
            t(M)~o~n~d~e~o);
   dealer1 !car !quote ?offer:offer
      [offer eq offer(SomeNat,
                      t(B)~ii~g~D~e~a~l,
                      Number(+,t(2)~0~0~0~0,<>),
                      (1&5) Of Nat)];
   OK;
   Stop
EndProc (* DEALER1_Mondeo *)

Process DEALER1_A5 [dealer1,OK] : NoExit :=
   dealer1 !car !quote
      !need(t(P)~e~t~e~r~ ^ ~G~o~u~g~h,
            t(C)~o~n~g~l~e~t~o~n~ ^ ~U~K,
            t(A)~5);
   dealer1 !car !quote ?offer:offer
      [offer eq offer(SomeNat,
                      t(B)~ii~g~D~e~a~l,
                      Number(+,t(3)~3~0~0~0,<>),
                      (3&0) Of Nat)];
   OK;
   Stop
EndProc (* DEALER1_A5 *)

Process DEALER1_Megane [dealer1,OK] : NoExit :=
   dealer1 !car !quote
      !need(t(J)~a~n~ ^ ~H~ii~d~d~ii~n~k,
            t(H)~e~n~g~e~l~o~ ^ ~N~e~t~h~e~r~l~a~n~d~s,
```

```
            t(M)∼e∼g∼a∼n∼e);
   dealer1 !car !quote ?offer:offer
      [offer eq offer(SomeNat,
                      t(B)∼ii∼g∼D∼e∼a∼l,
                      Number(+,t(1)∼1∼0∼0∼0,<>),
                      5 Of Nat)];
   OK;
   Stop
EndProc (* DEALER1_Megane *)

Process DEALER1_XJ6 [dealer1,OK] : NoExit :=
   dealer1 !car !quote
      !need(t(II)∼a∼ii∼n∼ ^ ∼M∼a∼c∼K∼a∼y,
            t(T)∼h∼r∼o∼s∼k∼ ^ ∼S∼c∼o∼t∼l∼a∼n∼d,
            t(X)∼J∼6);
   dealer1 !car !quote ?offer:offer
      [offer eq offer(SomeNat,
                      t(B)∼ii∼g∼D∼e∼a∼l,
                      Number(+,t(1)∼0∼0∼0∼0∼0∼0,<>),
                      0 Of Nat)];
   OK;
   Stop
EndProc (* DEALER1_XJ6 *)
```

A.4.5 Dealer2 Translated LOTOS Scenarios

```
Process DEALER2_Mondeo [dealer2,OK] : NoExit :=
   dealer2 !car !quote
      !need(t(M)∼a∼r∼k∼ ^ ∼F∼l∼o∼w∼e∼r∼s,
            t(U)∼ii∼s∼t∼ ^ ∼S∼c∼o∼t∼l∼a∼n∼d,
            t(M)∼o∼n∼d∼e∼o);
   dealer2 !car !quote ?offer:offer
      [offer eq offer(SomeNat,
                      t(W)∼h∼e∼e∼l∼e∼r∼D∼e∼a∼l∼e∼r,
                      Number(+,t(2)∼0∼0∼0∼0,<>),
                      (1&0) Of Nat)];
   OK;
   Stop
EndProc (* DEALER2_Mondeo *)

Process DEALER2_A5 [dealer2,OK] : NoExit :=
   dealer2 !car !quote
      !need(t(P)∼e∼t∼e∼r∼ ^ ∼G∼o∼u∼g∼h,
            t(C)∼o∼n∼g∼l∼e∼t∼o∼n∼ ^ ∼U∼K,
            t(A)∼5);
   dealer2 !car !quote ?offer:offer
      [offer eq offer(SomeNat,
                      t(W)∼h∼e∼e∼l∼e∼r∼D∼e∼a∼l∼e∼r,
                      Number(+,t(3)∼5∼0∼0∼0,<>),
                      (2&0) Of Nat)];
   OK;
   Stop
EndProc (* DEALER2_A5 *)

Process DEALER2_Astra [dealer2,OK] : NoExit :=
   dealer2 !car !quote
```

```
      !need(t(H)~y~w~e~l~ ^ ~T~h~o~m~a~s,
            t(S)~w~a~n~s~e~a~ ^ ~W~a~l~e~s,
            t(A)~s~t~r~a);
   dealer2 !car !quote ?offer:offer
      [offer eq offer(SomeNat,
                      t(W)~h~e~e~l~e~r~D~e~a~l~e~r,
                      Number(+,t(1)~8~0~0~0,<>),
                      (3&0) Of Nat)];
   OK;
   Stop
EndProc (* DEALER2_Astra *)

Process DEALER2_XJ6 [dealer2,OK] : NoExit :=
   dealer2 !car !quote
      !need(t(II)~a~ii~n~ ^ ~M~a~c~K~a~y,
            t(T)~h~r~o~s~k~ ^ ~S~c~o~t~l~a~n~d,
            t(X)~J~6);
   dealer2 !car !quote ?offer:offer
      [offer eq offer(SomeNat,
                      t(W)~h~e~e~l~e~r~D~e~a~l~e~r,
                      Number(+,t(1)~0~0~0~0~0~0,<>),
                      0 Of Nat)];
   OK;
   Stop
EndProc (* DEALER2_XJ6 *)
```

A.4.6 Supplier Translated LOTOS Scenarios

```
Process SUPPLIER_Mondeo [supplier,OK] : NoExit :=
   supplier !car !order
      !need(t(M)~a~r~k~ ^ ~F~l~o~w~e~r~s,
            t(U)~ii~s~t~ ^ ~S~c~o~t~l~a~n~d,
            t(M)~o~n~d~e~o);
   supplier !car !order ?offer:offer
      [offer eq offer(SomeNat,
                      t(W)~h~e~e~l~e~r~D~e~a~l~e~r,
                      Number(+,t(2)~0~0~0~0,<>),
                      (1&0) Of Nat)];
   OK;
   Stop
EndProc (* SUPPLIER_Mondeo *)

Process SUPPLIER_A5 [supplier,OK] : NoExit :=
   supplier !car !order
      !need(t(P)~e~t~e~r~ ^ ~G~o~u~g~h,
            t(C)~o~n~g~l~e~t~o~n~ ^ ~U~K,
            t(A)~5);
   supplier !car !order ?offer:offer
      [offer eq offer(SomeNat,
                      t(B)~ii~g~D~e~a~l,
                      Number(+,t(3)~3~0~0~0,<>),
                      (3&0) Of Nat)];
   OK;
   Stop
EndProc (* SUPPLIER_A5 *)

Process SUPPLIER_Megane [supplier,OK] : NoExit :=
```

```
    supplier !car !order
      !need(t(J)~a~n~ ^ ~H~ii~d~d~ii~n~k,
            t(H)~e~n~g~e~l~o~ ^ ~N~e~t~h~e~r~l~a~n~d~s,t(M)~e~g~a~n~e);
    supplier !car !order ?offer:offer
      [offer eq offer(SomeNat,
                      t(B)~ii~g~D~e~a~l,
                      Number(+,t(1)~1~0~0~0,<>),
                      5 Of Nat)];
    OK;
    Stop
  EndProc (* SUPPLIER_Megane *)

  Process SUPPLIER_Astra [supplier,OK] : NoExit :=
    supplier !car !order
      !need(t(H)~y~w~e~l~ ^ ~T~h~o~m~a~s,
            t(S)~w~a~n~s~e~a~ ^ ~W~a~l~e~s,
            t(A)~s~t~r~a);
    supplier !car !order ?offer:offer
      [offer eq offer(SomeNat,
                      t(W)~h~e~e~l~e~r~D~e~a~l~e~r,
                      Number(+,t(1)~8~0~0~0,<>),
                      (3&0) Of Nat)];
    OK;
    Stop
  EndProc (* SUPPLIER_Astra *)

  Process SUPPLIER_XJ6 [supplier,OK] : NoExit :=
    supplier !car !order
      !need(t(II)~a~ii~n~ ^ ~M~a~c~K~a~y,
            t(T)~h~r~o~s~k~ ^ ~S~c~o~t~l~a~n~d,
            t(X)~J~6);
    supplier !car !order ?offer:offer
      [offer eq offer(SomeNat,
                      t(W)~h~e~e~l~e~r~D~e~a~l~e~r,
                      Number(+,t(1)~0~0~0~0~0~0,<>),
                      0 Of Nat)];
    OK;
    Stop
  EndProc (* SUPPLIER_XJ6 *)
```

A.4.7 Broker Translated LOTOS Scenarios

```
  Process BROKER_Mondeo_Ken [broker,OK] : NoExit :=
    broker !car !arrange
      !need(t(K)~e~n~ ^ ~B~o~y~l~e,
            t(D)~u~b~l~ii~n~ ^ ~II~r~e~l~a~n~d,
            t(M)~o~n~d~e~o);
    broker !car !arrange ?schedule:schedule
      [schedule eq schedule(SomeNat,
                            t(W)~h~e~e~l~e~r~D~e~a~l~e~r,
                            Number(+,t(2)~0~0~0~0,<>),
                            (1&0) Of Nat,
                            Number(+,t(3),t(7)))];
    OK;
    Stop
  EndProc (* BROKER_Mondeo_Ken *)
```

```
Process BROKER_A5_Scotland [broker,OK] : NoExit :=
   broker !car !arrange
      !need(t(M)~a~r~y~ ^ ~D~u~n~c~a~n,
            t(W)~ii~c~k~ ^ ~S~c~o~t~l~a~n~d,
            t(A)~5);
   broker !car !arrange ?schedule:schedule
      [schedule eq schedule(SomeNat,
                            t(B)~ii~g~D~e~a~l,
                            Number(+,t(3)~3~0~0~0,<>),
                            (3&0) Of Nat,
                            Number(+,t(4),t(1)))];
   OK;
   Stop
EndProc (* BROKER_A5_Scotland *)

Process BROKER_Megane [broker,OK] : NoExit :=
   broker !car !arrange
      !need(t(S)~a~l~l~y~ ^ ~D~e~a~n,
            t(C)~a~r~d~ii~f~f~ ^ ~W~a~l~e~s,
            t(M)~e~g~a~n~e);
   broker !car !arrange ?schedule:schedule
      [schedule eq schedule(SomeNat,
                            t(B)~ii~g~D~e~a~l,
                            Number(+,t(1)~1~0~0~0,<>),
                            5 Of Nat,
                            Number(+,t(4),t(4)))];
   OK;
   Stop
EndProc (* BROKER_Megane *)

Process BROKER_Astra [broker,OK] : NoExit :=
   broker !car !arrange
      !need(t(II)~a~n~ ^ ~C~a~r~e~y,
            t(C)~r~o~y~d~o~n~ ^ ~E~n~g~l~a~n~d,
            t(A)~s~t~r~a);
   broker !car !arrange !refusal !t(l)~o~a~n~ ^ ~u~n~a~c~c~e~p~t~a~b~l~e;
   OK;
   Stop
EndProc (* BROKER_Astra *)

Process BROKER_XJ6 [broker,OK] : NoExit :=
   broker !car !arrange
      !need(t(II)~a~ii~n~ ^ ~M~a~c~K~a~y,
            t(T)~h~r~o~s~k~ ^ ~S~c~o~t~l~a~n~d,
            t(X)~J~6);
   broker !car !arrange !refusal !t(c)~a~r~ ^ ~u~n~a~v~a~ii~l~a~b~l~e;
   OK;
   Stop
EndProc (* BROKER_XJ6 *)
```

A.5 Formal Verification

A.5.1 Annotated Lender LOTOS Specification

```
Specification WSSystem [lender] : Exit(States)          (* WS system *)          1
Type wssystem Is                                                                  2
   Sorts                                                                          3
```

```
    ...                                                                          4
    proposal (*! implementedby PROPOSAL_SORT                                    5
                comparedby PROPOSAL_COMP                                         6
                iteratedby PROPOSAL_FIRST and PROPOSAL_NEXT                      7
                printedby PROPOSAL_PRINT *),                                     8
    ...                                                                          9
  Opns                                                                           10
    ...                                                                          11
    proposal (*! implementedby Proposal constructor *) : text,text,number -> proposal   12
  Eqns                                                                           13
    ...                                                                          14
Behaviour                                                                        15
  LENDER [lender]                                  (* call LENDER process *)    16
                                                                                 17
Where                                              (* local definitions *)      18
... behaviorial specification remains the same                                   19
```

A.5.2 Translated μ-calculus Properties for Lender

```
(* Property name : GENERAL RESPONSE *)

[(((((true)*) . ((″LENDER !LOAN !QUOTE″ # ″ !″ # 'PROPOSAL (.*)')))))]
    mu X. (<((true))> (true) and
      [(not((((″LENDER !LOAN !QUOTE″ # ″ !″ # '[0−9][0−9]*[.][0−9]*'))
        or ((″LENDER !LOAN !QUOTE″ # ″ !″ # ″REFUSAL″ # ″ !″ #
            ″\″LOAN UNACCEPTABLE\″″)))))] X)


(* Property name : SPECIFIC RESPONSE *)

[(((((true)*) . ((″LENDER !LOAN !QUOTE″ # ″ !″ # 'PROPOSAL (.*)')))))]
  mu X. (<((true))> (true) and
    [(not((((″LENDER !LOAN !QUOTE″ # ″ !″ # ″3.500000″)) or
            ((″LENDER !LOAN !QUOTE″ # ″ !″ # ″3.700000″)) or
            ((″LENDER !LOAN !QUOTE″ # ″ !″ # ″4.100000″)) or
            ((″LENDER !LOAN !QUOTE″ # ″ !″ # ″4.400000″)) or
            ((″LENDER !LOAN !QUOTE″ # ″ !″ # ″REFUSAL″ # ″ !″
              # ″\″LOAN UNACCEPTABLE\″″)))))] X)

(* Property name : DEADLOCK FREEDOM *)

[((true)*)] <((true))> true


(* Property name : LIVELOCK FREEDOM *)

<((true)*)> @ (″i″) equ false
(* Property name : INITIALS SAFETY *)

[(not((((″LENDER !LOAN !QUOTE″ # ″ !″ # 'PROPOSAL (.*)')))))] false
```

A.5.3 Generated C Code of User Types for Lender

```
#define PROPOSAL_ENUM Proposal(″KEN TURNER″,″UK″,5000.0),...                     1
                                                                                 2
/* Iteration macros and functions for PROPOSAL_SORT */                           3
#undef PROPOSAL_FIRST                                                            4
```

```
#define PROPOSAL_FIRST() (PROPOSAL_FIRST_FUNC())

// Definition of the first in iteration of PROPOSAL_SORT
PROPOSAL_SORT PROPOSAL_FIRST_FUNC() {
    PROPOSAL_SORT proposal_permutations[] = {PROPOSAL_ENUM};
    return proposal_permutations[0];
}

#undef PROPOSAL_NEXT
#define PROPOSAL_NEXT(PROPOSAL_PARAM)
  (PROPOSAL_NEXT_FUNC(&(PROPOSAL_PARAM)))

// Definition of the next in iteration of PROPOSAL_SORT
int PROPOSAL_NEXT_FUNC(PROPOSAL_PARAM)
PROPOSAL_SORT* PROPOSAL_PARAM;
{
    // if has next, return 1 and set the value of next to pointer
    // otherwise return 0
/* End iteration macros and functions for PROPOSAL_SORT */
```

A.5.4 Annotated Supplier LOTOS Specification

```
Specification WSSystem [supplier] : Exit(States)          (* WS system *)
Type wssystem Is
  Sorts
    offer (*! implementedby OFFER_SORT comparedby OFFER_COMP
              iteratedby OFFER_FIRST and OFFER_NEXT
              printedby OFFER_PRINT *),
    need (*! implementedby NEED_SORT comparedby NEED_COMP
             iteratedby NEED_FIRST and NEED_NEXT
             printedby NEED_PRINT *),
  Opns
    ...
    deal (*! constructor *) : offer,need -> deal
    ...
    need (*! implementedby Need constructor *) : text,text,
                                                  text -> need
    ...
    offer (*! implementedby Offer constructor *) : nat,text,number,
                                                    nat -> offer
    offer : value -> offer
    ...
  Eqns
    ...
EndType

Behaviour
  SUPPLIER [supplier]                             (* call SUPPLIER process *)
Where                                             (* local definitions *)
... behaviorial specification remains the same ...
```

A.5.5 Automated Adaptation of Supplier LOTOS Specification for Compositional Verification

```
Specification WSSystem [supplier] : Exit(States) (* WS system *)
Type wssystem Is
  ... flattened and annotated data types ...
Behaviour
```

```
   SUPPLIER [supplier]                                   (* call SUPPLIER process *)
... specification behaviour identical with the insertion of
Process SUPPLIER_ONLY [dealer1,dealer2,supplier] : Exit(States) :=
SUPPLIER_1 [dealer1,dealer2,supplier]
       (AnyNeed,AnyOffer,AnyOffer,AnyStates)

EndProc
...
```

A.5.6 Constrained Data Types and Operations for Supplier

```
Type Deals Is LimitedString ActualizedBy Deal,Boolean Using   (* deals *)
  SortNames                                                   (* rename sorts *)
    Nat For FNat                                              (* use real natural *)
    Deals For LimitedString                                   (* deal list *)
    Deal For Element                                          (* deal element *)
    Bool For FBool                                            (* use real booleans *)
  OpnNames
    3 For StringMax                                           (* maximum length 3 for Deals *)
    deals For string                                          (* deal to deals *)
EndType                                                       (* end Deals *)
```

A.5.7 Translated μ-calculus Properties for Supplier

```
(* Property name : GENERAL RESPONSE *)

[(((((true)*) . ((″SUPPLIER !CAR !ORDER″ # ″ !″ # 'NEED (.*)')))))]
    mu X. (<((true))> (true) and
        [(not((″SUPPLIER !CAR !ORDER″ # ″ !″ # 'OFFER (.*)')))] X)

(* Property name : DEADLOCK FREEDOM *)

[((true)*)] <((true))> true

(* Property name : LIVELOCK FREEDOM *)

<((true)*)> @ (″i″) equ false
```

A.5.8 Generated C Code of User Types for Supplier

```
#define OFFER_ENUM Offer(0, ″BIGDEAL″, 20000.0, 15),...

/* Iteration macros and functions for OFFER_SORT */
#undef OFFER_FIRST
#define OFFER_FIRST() (OFFER_FIRST_FUNC())

// Definition of the first in iteration of OFFER_SORT
OFFER_SORT OFFER_FIRST_FUNC() {
          OFFER_SORT offer_permutations[] = {OFFER_ENUM};
          return offer_permutations[0];
}

#undef OFFER_NEXT
#define OFFER_NEXT(OFFER_PARAM) (OFFER_NEXT_FUNC(&(OFFER_PARAM)))

// Definition of the next in iteration of OFFER_SORT
int OFFER_NEXT_FUNC(OFFER_PARAM)
```

```
OFFER_SORT* OFFER_PARAM;
{
        OFFER_SORT offer_permutations[] = {OFFER_ENUM};
        int size = sizeof(offer_permutations)/sizeof(OFFER_SORT);
        int index = 0;
        int found = 0;
        int found_index = -1;
        for(index = 0; index < size && !found; index++) {
                if (OFFER_COMP(*OFFER_PARAM, offer_permutations[index])) {
                        found = 1;
                        found_index = index;
                }
        }

        if (found && found_index < size - 1) {
                *OFFER_PARAM = offer_permutations[found_index+1];
                return 1;
        } else {
                return 0;
        }
}
/* End iteration macros and functions for OFFER_SORT */

#define NEED_ENUM Need("KEN TURNER","SCOTLAND","MONDEO"),...
... iteration code functions similar to OFFER
```

A.5.9 Generated SVL Script for Supplier

```
% DEFAULT_LOTOS_FILE=ws.lotos
"supplier_only.bcg" = strong reduction of "SUPPLIER_ONLY";
"supplier_only_dealer1.bcg" = strong reduction of
    DEALER1 -|[dealer1]| "supplier_only.bcg";
"supplier_only_dealer2.bcg" = strong reduction of
    DEALER2 -|[dealer2]| "supplier_only.bcg";
"ws.bcg" = strong reduction of          hide dealer1,dealer2 in
      (
        "supplier_only_dealer1.bcg"
      |||
        "supplier_only_dealer2.bcg"
      )
   |[dealer1,dealer2]|
     "supplier_only.bcg";
```

A.5.10 Annotated Broker LOTOS Specification

```
Specification WSSystem [broker] : Exit(States)          (* WS system *)

Type wssystem Is
  Sorts
    schedule (*! implementedby SCHEDULE_SORT
                comparedby SCHEDULE_COMP
                iteratedby SCHEDULE_FIRST and SCHEDULE_NEXT
                printedby SCHEDULE_PRINT *),
    proposal (*! implementedby PROPOSAL_SORT
                comparedby PROPOSAL_COMP
                iteratedby PROPOSAL_FIRST and PROPOSAL_NEXT
                printedby PROPOSAL_PRINT *),
    offer (*! implementedby OFFER_SORT
```

```
                comparedby OFFER_COMP
                iteratedby OFFER_FIRST and OFFER_NEXT
                printedby OFFER_PRINT *),
    need (*! implementedby NEED_SORT
              comparedby NEED_COMP
              iteratedby NEED_FIRST and NEED_NEXT
              printedby NEED_PRINT *),
  Opns
   ...

Behaviour
  BROKER [broker]                                  (* call BROKER process *)

Where                                              (* local definitions *)
...
Process BROKER_6 [broker,lender,supplier]          (* BROKER from 6 *)
 (error:Text,need:Need,offer:Offer,proposal:Proposal,
  rate:Number,schedule:Schedule,xstates:States) : Exit(States) :=
  supplier !car !cancel !offer;                    (* BROKER invoke 6 *)
  Exit(xstates)
EndProc                                            (* end BROKER_6 *)

Process BROKER_7 [broker,lender,supplier]          (* BROKER from 7 *)
 (error:Text,need:Need,offer:Offer,proposal:Proposal,
  rate:Number,schedule:Schedule,xstates:States) : Exit(States) :=
  BROKER_EVENT [broker,lender,supplier]            (* BROKER compensate 7 *)
   (error,need,offer,proposal,rate,schedule,xstates,0 Of Nat,Compensation,AnyValue)
>> Accept xstates:States In                        (* accept compensation states *)
  broker !car !arrange !Refusal !error;            (* BROKER reply 8 *)
  BROKER_1 [broker,lender,supplier]                (* repeat behaviour *)
   (AnyText,AnyNeed,AnyOffer,AnyProposal,AnyNumber,AnySchedule,AnyStates)
EndProc                                            (* end BROKER_7 *)

Process BROKER_EVENT [broker,lender,supplier] (* event dispatcher *)
 (error:Text,need:Need,offer:Offer,proposal:Proposal,rate:Number,schedule:Schedule,
 xstates:States,xscope:Nat,xevent:Event,xvalue:Value) : Exit(States) :=
  Let xkind:Kind = kind(xvalue) In                 (* get value kind *)
    [xscope eq 0]->                                (* platform scope? *)
      (
        [match(xevent,CatchAll)]->                 (* match for 'CatchAll'? *)
          BROKER_1 [broker,lender,supplier] (* repeat behaviour *)
           (AnyText,AnyNeed,AnyOffer,AnyProposal,AnyNumber,AnySchedule,AnyStates)
      []                                           (* or *)
        [match(xevent,Compensation)]->             (* match for 'Compensation'? *)
          (
            [xstates ne <>]->                      (* compensation states? *)
              (
                BROKER_EVENT [broker,lender,supplier] (* do compensation *)
                 (error,need,offer,proposal,rate,schedule,Tail(xstates),
                  getScope(head(xstates)),Compensation,xvalue)
              >> Accept xstates:States In          (* get rest of compensation states *)
                BROKER_EVENT [broker,lender,supplier](* continue compensation *)
                 (error,need,offer,proposal,rate,schedule,
                  xstates,xscope,compensation,xvalue)
              )
```

```
            []                                          (* or *)
              [xstates eq <>]->                         (* no compensation states? *)
                Exit(xstates)                           (* end compensation *)
            )
        )
    []                                                  (* or *)
      [xscope eq 1]->                                   (* start scope? *)
        (
          [match(xevent,xkind,Refusal,textKind)]->      (* match for 'refusal.error'? *)
            BROKER_7 [broker,lender,supplier]           (* call event handler *)
             (text(xvalue),need,offer,proposal,rate,schedule,xstates)
          []                                            (* or *)
            [not(match(xevent,xkind,Refusal,textKind))]-> (* else *)
              (
                [match(xevent,CatchAll)]->              (* match for 'CatchAll'? *)
                  BROKER_1 [broker,lender,supplier]     (* repeat behaviour *)
                   (AnyText,AnyNeed,AnyOffer,AnyProposal,AnyNumber,AnySchedule,AnyStates)
              )
        )
    []                                                  (* or *)
      [xscope eq 2]->                                   (* BROKER 2 scope? *)
        (
          [match(xevent,xkind,Refusal,textKind)]->      (* match for 'refusal.error'? *)
            BROKER_7 [broker,lender,supplier] (* call event handler *)
             (text(xvalue),need,offer,proposal,rate,schedule,xstates)
          []                                            (* or *)
            [not(match(xevent,xkind,Refusal,textKind))]->(* else *)
              (
                [match(xevent,CatchAll)]->              (* match for 'CatchAll'? *)
                  BROKER_1 [broker,lender,supplier]     (* repeat behaviour *)
                   (AnyText,AnyNeed,AnyOffer,AnyProposal,
                    AnyNumber,AnySchedule,AnyStates)
              []                                        (* or *)
                [match(xevent,Compensation)]->          (* match for 'Compensation'? *)
                  BROKER_6 [broker,lender,supplier]     (* call event handler *)
                   (error,need,offer,proposal,rate,schedule,xstates)
              )
        )
EndProc                                                 (* end BROKER_EVENT *)

EndProc                                                 (* end BROKER *)

EndSpec                                                 (* end WSSystem *)
```

A.5.11 Manual Adaptation of Annotated Broker LOTOS Specification

```
Specification WSSystem [broker] : Exit(States)          (* WS system *)

Type wssystem Is
  Sorts
    ...
    schedule (*! implementedby SCHEDULE_SORT
                  comparedby SCHEDULE_COMP
                  iteratedby SCHEDULE_FIRST and SCHEDULE_NEXT
                  printedby SCHEDULE_PRINT *),
    proposal (*! implementedby PROPOSAL_SORT
```

```
                    comparedby PROPOSAL_COMP                                    11
                    iteratedby PROPOSAL_FIRST and PROPOSAL_NEXT                 12
                    printedby PROPOSAL_PRINT *),                                13
    offer (*! implementedby OFFER_SORT                                          14
                    comparedby OFFER_COMP                                       15
                    iteratedby OFFER_FIRST and OFFER_NEXT                       16
                    printedby OFFER_PRINT *),                                   17
    need (*! implementedby NEED_SORT                                           18
              comparedby NEED_COMP                                              19
              iteratedby NEED_FIRST and NEED_NEXT                               20
              printedby NEED_PRINT *),                                          21
    ...                                                                         22
  Opns                                                                          23
  ...                                                                           24
Behaviour                                                                       25
  BROKER [broker]                                (* call BROKER process *)     26
                                                                                27
Where                                            (* local definitions *)       28
... behaviour99 specification99 remains the same                                29
    with insertion of \_ONLY processes for compositions                         30
    and the following adaptation                                                31
...                                                                             32
Process BROKER_6 [broker,lender,supplier]        (* BROKER from 6 *)           33
 (error:Text,need:Need,offer:Offer,proposal:Proposal,                           34
  rate:Number,schedule:Schedule,xstates:States) : Exit(States) :=               35
  supplier !car !cancel !offer;                  (* BROKER invoke 6 *)         36
  broker !car !arrange !Refusal !error;          (* BROKER reply 8 *)          37
  BROKER_1 [broker,lender,supplier]              (* repeat behaviour *)        38
   (AnyText,AnyNeed,AnyOffer,AnyProposal,                                       39
    AnyNumber,AnySchedule,AnyStates)                                            40
EndProc                                          (* end BROKER_6 *)            41
                                                                                42
Process BROKER_7 [broker,lender,supplier]        (* BROKER from 7 *)           43
 (error:Text,need:Need,offer:Offer,proposal:Proposal,                           44
  rate:Number,schedule:Schedule,xstates:States) : Exit(States) :=               45
  BROKER_EVENT [broker,lender,supplier]          (* BROKER compensate 7 *)     46
   (error,need,offer,proposal,rate,schedule,                                    47
    xstates,0 Of Nat,Compensation,AnyValue)                                     48
   (* calls compensate *)                                                      49
EndProc                                          (* end BROKER_7 *)            50
                                                                                51
Process BROKER_EVENT [broker,lender,supplier] (* event dispatcher *)            52
 (error:Text,need:Need,offer:Offer,proposal:Proposal,                           53
  rate:Number,schedule:Schedule,xstates:States,                                 54
  xscope:Nat,xevent:Event,xvalue:Value) : Exit(States) :=                       55
  Let xkind:Kind = kind(xvalue) In               (* get value kind *)          56
    [xscope eq 0]->                              (* platform scope? *)         57
      (                                                                         58
        [match(xevent,CatchAll)]->               (* match for 'CatchAll'? *)   59
          BROKER_1 [broker,lender,supplier]      (* repeat behaviour *)        60
           (AnyText,AnyNeed,AnyOffer,AnyProposal,                               61
            AnyNumber,AnySchedule,AnyStates)                                    62
      []                                         (* or *)                      63
        [match(xevent,Compensation)]->           (* match for 'Compensation'? *) 64
          (                                                                     65
```

```
                [xstates ne <>]->                    (* compensation states? *)
                  (
                    BROKER_EVENT [broker,lender,supplier] (* do compensation *)
                     (error,need,offer,proposal,rate,schedule,Tail(xstates),
                     getScope(head(xstates)),Compensation,xvalue)
                  )
                []                                   (* or *)
                  [xstates eq <>]->                  (* no compensation states? *)
                    Exit(xstates)                    (* end compensation *)
              )
          )
      []                                             (* or *)
        [xscope eq 1]->                              (* start scope? *)
          (
            [match(xevent,xkind,Refusal,textKind)]->(* match for 'refusal.error'? *)
              BROKER_7 [broker,lender,supplier]      (* call event handler *)
               (text(xvalue),need,offer,proposal,rate,schedule,xstates)
            []                                       (* or *)
              [not(match(xevent,xkind,Refusal,textKind))]-> (* else *)
                (
                  [match(xevent,CatchAll)]->         (* match for 'CatchAll'? *)
                    BROKER_1 [broker,lender,supplier] (* repeat behaviour *)
                     (AnyText,AnyNeed,AnyOffer,AnyProposal,AnyNumber,AnySchedule,AnyStates)
                )
          )
      []                                             (* or *)
        [xscope eq 2]->                              (* BROKER 2 scope? *)
          (
            [match(xevent,xkind,Refusal,textKind)]->(* match for 'refusal.error'? *)
              BROKER_7 [broker,lender,supplier]      (* call event handler *)
               (text(xvalue),need,offer,proposal,rate,schedule,xstates)
            []                                       (* or *)
              [not(match(xevent,xkind,Refusal,textKind))]->(* else *)
                (
                  [match(xevent,CatchAll)]->         (* match for 'CatchAll'? *)
                    BROKER_1 [broker,lender,supplier]                (* repeat behaviour *)
                     (AnyText,AnyNeed,AnyOffer,AnyProposal,AnyNumber,AnySchedule,AnyStates)
                  []                                 (* or *)
                    [match(xevent,Compensation)]->   (* match for 'Compensation'? *)
                      BROKER_6 [broker,lender,supplier] (* call event handler *)
                       (error,need,offer,proposal,rate,schedule,xstates)
                )
          )
    EndProc                                          (* end BROKER_EVENT *)

    EndProc                                          (* end BROKER *)
```

A.5.12 Translated μ-calculus Properties for Broker

```
(* Property name : GENERAL RESPONSE *)

[(((((true)*) . ((''BROKER !CAR !ARRANGE'' # '' !'' # 'NEED (.*)')))))]
   mu X. (<((true))> (true) and
     [(not((((''BROKER !CAR !ARRANGE'' # '' !'' # 'SCHEDULE (.*)')) or
       (((''BROKER !CAR !ARRANGE'' # '' !'' # ''REFUSAL'' # '' !''
         # ''\''LOAN UNACCEPTABLE\''''')) or
```

```
((″BROKER !CAR !ARRANGE″ # ″ !″ # ″REFUSAL″ # ″ !″
  # ″\″CAR UNAVAILABLE\″″)))))] X)

(* Property name : DEADLOCK FREEDOM *)

[((true)*)] <((true))> true

(* Property name : LIVELOCK FREEDOM *)

<((true)*)> @ (″i″) equ false

(* Property name : INITIALS SAFETY *)

[(not((((″BROKER !CAR !ARRANGE″ # ″ !″ # 'NEED (.*)')))))] false
```

A.5.13 Generated C Code of User Types for Broker

```
#define OFFER_ENUM Offer(0, ″BIGDEAL″, 20000.0, 15),...

/* Iteration macros and functions for OFFER_SORT */
#undef OFFER_FIRST
#define OFFER_FIRST() (OFFER_FIRST_FUNC())

// Definition of the first in iteration of OFFER_SORT
OFFER_SORT OFFER_FIRST_FUNC() {
        OFFER_SORT offer_permutations[] = {OFFER_ENUM};
        return offer_permutations[0];
}

#undef OFFER_NEXT
#define OFFER_NEXT(OFFER_PARAM) (OFFER_NEXT_FUNC(&(OFFER_PARAM)))

// Definition of the next in iteration of OFFER_SORT
int OFFER_NEXT_FUNC(OFFER_PARAM)
OFFER_SORT* OFFER_PARAM;
{
        OFFER_SORT offer_permutations[] = {OFFER_ENUM};
        int size = sizeof(offer_permutations)/sizeof(OFFER_SORT);
        int index = 0;
        int found = 0;
        int found_index = −1;
        for(index = 0; index < size && !found; index++) {
                if (OFFER_COMP(*OFFER_PARAM, offer_permutations[index])) {
                        found = 1;
                        found_index = index;
                }
        }

        if (found && found_index < size - 1) {
                *OFFER_PARAM = offer_permutations[found_index+1];
                return 1;
        } else {
                return 0;
        }
}
/* End iteration macros and functions for OFFER_SORT */
```

```
#define NEED_ENUM Need("KEN TURNER","SCOTLAND","MONDEO"),...

/* Iteration macros and functions for NEED_SORT */
...

#define SCHEDULE_ENUM Schedule(0, "BIGDEAL", 20000.0, 15),...

/* Iteration macros and functions for SCHEDULE_SORT */
...

#define PROPOSAL_ENUM Proposal("KEN TURNER", "SCOTLAND", 20000.0),...

/* Iteration macros and functions for PROPOSAL_SORT */
...
```

A.5.14 SVL Script for Broker

```
% DEFAULT_LOTOS_FILE=ws.lotos
"broker_only.bcg" = strong reduction of "BROKER_ONLY";
"broker_only_lender_only.bcg" = strong reduction of LENDER_ONLY
                                    -|[lender]| "broker_only.bcg";
"broker_only_lender_only_approver.bcg" = strong reduction of APPROVER
                                -|[approver]| "broker_only_lender_only.bcg";
"broker_only_lender_only_assessor.bcg" = strong reduction of ASSESSOR
                                -|[assessor]| "broker_only_lender_only.bcg";
"broker_only_lender.bcg" = strong reduction of hide approver,assessor in
      (
        "broker_only_lender_only_approver.bcg"
      |||
        "broker_only_lender_only_assessor.bcg"
      )
    |[approver,assessor]|
    "broker_only_lender_only.bcg";
"broker_only_supplier_only.bcg" = strong reduction of SUPPLIER_ONLY
                                    -|[supplier]| "broker_only.bcg";
"broker_only_supplier_only_dealer1.bcg" = strong reduction of DEALER1
                                -|[dealer1]| "broker_only_supplier_only.bcg";
"broker_only_supplier_only_dealer2.bcg" = strong reduction of DEALER2
                                -|[dealer2]| "broker_only_supplier_only.bcg";
"broker_only_supplier.bcg" = strong reduction of hide dealer1,dealer2 in
      (
        "broker_only_supplier_only_dealer1.bcg"
      |||
        "broker_only_supplier_only_dealer2.bcg"
      )
    |[dealer1,dealer2]|
    "broker_only_supplier_only.bcg";
"ws.bcg" = strong reduction of hide lender,supplier in
      (
        "broker_only_lender.bcg"
      |||
        "broker_only_supplier.bcg"
      )
    |[lender,supplier]|
     "broker_only.bcg";
(* Verifying property: broker_general_response *)
```

```
″broker_general_response.bcg″ = verify ″broker_general_response.mcl″ in ″ws.bcg″;

(* Verifying property: _deadlock_freedom *)

″_deadlock_freedom.bcg″ = verify ″_deadlock_freedom.mcl″ in ″ws.bcg″;

(* Verifying property: _livelock_freedom *)

″_livelock_freedom.bcg″ = verify ″_livelock_freedom.mcl″ in ″ws.bcg″;

(* Verifying property: _initials_safety *)

″_initials_safety.bcg″ = verify ″_initials_safety.mcl″ in ″ws.bcg″;
```

A.6 Implementation

A.6.1 File Structure of Generated Code in Lender

Tree structure in directory <CRESS>/bpel/lender

```
|   approver.properties
|   approver.wsdl
|   approver.wsr
|   assessor.properties
|   assessor.wsdl
|   assessor.wsr
|   lender.bpel
|   lender.bpr
|   lender.pdd
|   lender.properties
|   lender.wsdl
|   lender_defs.wsdl
|
+---approver
|   |   approver.wsdl
|   |   lender_defs.wsdl
|   |
|   +---FirstRate
|   |       approver.java
|   |       ApproverService.class
|   |       ApproverService.java
|   |       ApproverServiceLocator.class
|   |       ApproverServiceLocator.java
|   |       deploy.wsdd
|   |       LoanBindingImpl.class
|   |       LoanBindingSkeleton.class
|   |       LoanBindingSkeleton.java
|   |       LoanBindingStub.class
|   |       LoanBindingStub.java
|   |       LoanPort.class
|   |       LoanPort.java
|   |       undeploy.wsdd
|   |
|   +---FirstRateDefs
|   |       Proposal2.class
```

```
|   |       Proposal2.java
|   |
|   +---LoanStarDefs
|   |       StringMessage.class
|   |       StringMessage.java
|   |
|   +---META-INF
|           deploy.wsdd
|
+---assessor
|   |   assessor.wsdl
|   |   lender_defs.wsdl
|   |
|   +---LoanStarDefs
|   |       Proposal.class
|   |       Proposal.java
|   |
|   +---META-INF
|   |       deploy.wsdd
|   |
|   +---RiskTaker
|           assessor.java
|           AssessorService.class
|           AssessorService.java
|           AssessorServiceLocator.class
|           AssessorServiceLocator.java
|           deploy.wsdd
|           LoanBindingImpl.class
|           LoanBindingSkeleton.class
|           LoanBindingSkeleton.java
|           LoanBindingStub.class
|           LoanBindingStub.java
|           LoanPort.class
|           LoanPort.java
|           undeploy.wsdd
|
+---lender
    |   approver.wsdl
    |   assessor.wsdl
    |   lender.bpel
    |   lender.pdd
    |   lender.wsdl
    |   lender_defs.wsdl
    |
    +---FirstRateDefs
    |       Proposal2.java
    |
    +---LoanStarDefs
    |       Proposal.java
    |
    +---META-INF
            wsdlCatalog.xml
```

A.6.2 Approver WSDL

The approver.wsdl and lender_defs.wsdl files are the service interface and data type definition respectively. The high-level structure of the approver.wsdl including generated comments shown below.

```
<definitions name="ApproverDefinitions"                                    0
 ...>                                                                      1
                                                                           2
  <import namespace="urn:LoanStarDefs"                                     3
   location="lender_defs.wsdl"/>                                           4
                                                                           5
  <portType name="loanPort">                                               6
    <operation name="approve">                                             7
      <input message="defs:proposal2Message"/>                             8
      <output message="defs:floatMessage"/>                                9
      <fault name="refusal" message="defs:stringMessage"/>                 10
    </operation>                                                           11
  </portType>                                                              12
                                                                           13
  <binding name="loanBinding" type="app:loanPort">                         14
    <soap:binding style="rpc"                                              15
     transport="http://schemas.xmlsoap.org/soap/http"/>                    16
    <operation name="approve">                                             17
      <soap:operation                                                      18
       soapAction="http://www.cs.stir.ac.uk/schemas/ApproverService/approve"/>  19
      <input>                                                              20
        <soap:body use="encoded" namespace="urn:FirstRate"                 21
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>       22
      </input>                                                             23
      <output>                                                             24
        ...                                                                25
      </output>                                                            26
      <fault name="refusal">                                               27
        ...                                                                28
      </fault>                                                             29
    </operation>                                                           30
  </binding>                                                               31
                                                                           32
  <service name="ApproverService">                                         33
    <port name="ApproverLoan" binding="app:loanBinding">                   34
      <soap:address                                                        35
       location="http://localhost:8080/active-bpel/services/ApproverLoan"/>  36
    </port>                                                                37
  </service>                                                               38
</definitions>                                                             39
```

A.6.3 Assessor WSDL

```
<definitions name="AssessorDefinitions"                                    0
 ...>                                                                      1
                                                                           2
  <import namespace="urn:LoanStarDefs"                                     3
   location="lender_defs.wsdl"/>                                           4
                                                                           5
  <portType name="loanPort">                                               6
    <operation name="assess">                                              7
      <input message="defs:proposalMessage"/>                              8
```

```
      <output message="defs:stringMessage"/>
    </operation>
  </portType>

  <binding name="loanBinding" type="ass:loanPort">
  ...
  <service name="AssessorService">
    <port name="AssessorLoan" binding="ass:loanBinding">
      <soap:address
       location="http://localhost:8080/active-bpel/services/AssessorLoan"/>
    </port>
  </service>
</definitions>
```

A.6.4 Lender WSDL

```
<definitions name="LenderDefinitions"
 ...>
  <import namespace="urn:FirstRate" location="approver.wsdl"/>
  <import namespace="urn:RiskTaker" location="assessor.wsdl"/>
  <import namespace="urn:LoanStarDefs" location="lender_defs.wsdl"/>

  ... port type ...
  ... binding ...

  <plnk:partnerLinkType name="approverLoanLink">
    <plnk:role name="approver">
      <plnk:portType name="app:loanPort"/>
    </plnk:role>
  </plnk:partnerLinkType>

  <plnk:partnerLinkType name="assessorLoanLink">
    <plnk:role name="assessor">
      <plnk:portType name="ass:loanPort"/>
    </plnk:role>
  </plnk:partnerLinkType>

  <plnk:partnerLinkType name="lenderLoanLink">
    <plnk:role name="lender">
      <plnk:portType name="lend:loanPort"/>
    </plnk:role>
  </plnk:partnerLinkType>

  ... service ...
</definitions>
```

A.6.5 Common Definitions - lender_defs.wsdl

The lender_defs.wsdl contains the common definitions of the data types and messages used by all the services. Its definition is the following:

```
<definitions name="LenderCommonDefinitions"
 ...>
  <types>
    <xsd:schema ...>
      <complexType name="proposal">
        <sequence>
          <element name="name" type="xsd:string"/>
          <element name="address" type="xsd:string"/>
```

```
            <element name="amount" type="xsd:integer"/>
          </sequence>
        </complexType>
      </xsd:schema>

      <xsd:schema targetNamespace="urn:FirstRateDefs" ...>
        <complexType name="proposal2">
          ...
      ...
    </types>

    <message name="stringMessage">
      <part name="string" type="xsd:string"/>
    </message>

    <message name="proposalMessage">
      <part name="proposal" type="defs:proposal"/>
    </message>

    <message name="proposal2Message">
      <part name="proposal2" type="app_defs:proposal2"/>
    </message>

    <message name="floatMessage">
    ...

</definitions>
```

A.6.6 Lender BPEL

```
<process name="LenderProcess"
 ...>

  <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
   namespace="urn:LoanStar" location="lender.wsdl"/>
  <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
   namespace="urn:LoanStarDefs" location="lender_defs.wsdl"/>

  <partnerLinks>
  ...
  <variables>
  ...
  <flow>
    <links>
      <link name="LENDER.1-LENDER.2"/>
      ...

    <sequence>
      <sources>
        <source linkName="LENDER.1-LENDER.2">
          <transitionCondition>
            $proposal.proposal/amount &gt;= 10000
          </transitionCondition>
        </source>
        <source linkName="LENDER.1-LENDER.6">
          <transitionCondition>
```

```
                not($proposal.proposal/amount &gt;= 10000)                    26
              </transitionCondition>                                          27
            </source>                                                         28
          </sources>                                                          29
          <receive name="LENDER.1"                                            30
           partnerLink="lenderLoan" portType="lend:loanPort"                  31
           operation="quote" variable="proposal" createInstance="yes">        32
                                                                              33
          </receive>                                                          34
          <if>                                                                35
            <condition>$proposal.proposal/amount &gt;= 10000</condition>      36
                                                                              37
              <assign>                                                        38
                <copy>                                                        39
                  <from variable="proposal" part="proposal"/>                 40
                                                                              41
                  <to variable="proposal2" part="proposal2"/>                 42
                                                                              43
                </copy>                                                       44
              </assign>                                                       45
          </if>                                                               46
        </sequence>                                                           47
      ...                                                                     48
    </process>                                                                49
```

A.6.7 File Structure of Generated Code in Supplier

Tree structure of <CRESS>/bpel/supplier

```
|   dealer1.properties
|   dealer1.wsdl
|   dealer1.wsr
|   dealer2.properties
|   dealer2.wsdl
|   dealer2.wsr
|   supplier.bpel
|   supplier.bpr
|   supplier.pdd
|   supplier.properties
|   supplier.wsdl
|   supplier_defs.wsdl
|
+---dealer1
|   |   dealer1.wsdl
|   |   supplier_defs.wsdl
|   |
|   +---BigDeal
|   |       CarBindingImpl.class
|   |       CarBindingSkeleton.class
|   |       CarBindingSkeleton.java
|   |       CarBindingStub.class
|   |       CarBindingStub.java
|   |       CarPort.class
|   |       CarPort.java
|   |       dealer1.java
|   |       Dealer1Service.class
```

```
|   |       Dealer1Service.java
|   |       Dealer1ServiceLocator.class
|   |       Dealer1ServiceLocator.java
|   |       deploy.wsdd
|   |       undeploy.wsdd
|   |
|   +---DoubleQuoteDefs
|   |       Need.class
|   |       Need.java
|   |       Offer.class
|   |       Offer.java
|   |
|   +---META-INF
|           deploy.wsdd
|
+---dealer2
|   |   dealer2.wsdl
|   |   supplier_defs.wsdl
|   |
|   +---DoubleQuoteDefs
|   |       Need.class
|   |       Need.java
|   |       Offer.class
|   |       Offer.java
|   |
|   +---META-INF
|   |       deploy.wsdd
|   |
|   +---WheelerDealer
|           CarBindingImpl.class
|           CarBindingSkeleton.class
|           CarBindingSkeleton.java
|           CarBindingStub.class
|           CarBindingStub.java
|           CarPort.class
|           CarPort.java
|           dealer2.java
|           Dealer2Service.class
|           Dealer2Service.java
|           Dealer2ServiceLocator.class
|           Dealer2ServiceLocator.java
|           deploy.wsdd
|           undeploy.wsdd
|
+---supplier
    |   dealer1.wsdl
    |   dealer2.wsdl
    |   supplier.bpel
    |   supplier.pdd
    |   supplier.wsdl
    |   supplier_defs.wsdl
    |
    +---DoubleQuoteDefs
    |       Need.java
    |       Offer.java
```

```
|
+---META-INF
        wsdlCatalog.xml
```

A.6.8 Dealer1 WSDL

Dealer2's WSDL is similar to this, and therefore its WSDL is not listed.

```
<definitions name="Dealer1Definitions"
 targetNamespace="urn:BigDeal"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:deal1="urn:BigDeal"
 xmlns:defs="urn:DoubleQuoteDefs"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <import namespace="urn:DoubleQuoteDefs"
   location="supplier_defs.wsdl"/>

  <portType name="carPort">
    <operation name="cancel">
      <input message="defs:offerMessage"/>
    </operation>
    <operation name="order">
      <input message="defs:offerMessage"/>
    </operation>
    <operation name="quote">
      <input message="defs:needMessage"/>
      <output message="defs:offerMessage"/>
    </operation>
  </portType>

  <binding name="carBinding" type="deal1:carPort">
  ...

  <service name="Dealer1Service">
    <port name="Dealer1Car" binding="deal1:carBinding">
      <soap:address
       location="http://localhost:8080/active-bpel/services/Dealer1Car"/>

    </port>
  </service>
</definitions>
```

A.6.9 Supplier WSDL

```
<definitions name="SupplierDefinitions"
 targetNamespace="urn:DoubleQuote"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:deal1="urn:BigDeal"
 xmlns:deal2="urn:WheelerDealer"
 xmlns:supp="urn:DoubleQuote"
 xmlns:defs="urn:DoubleQuoteDefs"
 xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <import namespace="urn:BigDeal"
```

```
 location="dealer1.wsdl"/>
<import namespace="urn:WheelerDealer"
 location="dealer2.wsdl"/>
<import namespace="urn:DoubleQuoteDefs"
 location="supplier_defs.wsdl"/>

<portType name="carPort">
  <operation name="cancel">
    <input message="defs:offerMessage"/>
  </operation>
  <operation name="order">
    <input message="defs:needMessage"/>
    <output message="defs:offerMessage"/>
  </operation>
</portType>

<binding name="carBinding" type="supp:carPort">
...

<plnk:partnerLinkType name="dealer1CarLink">
  <plnk:role name="dealer1">
    <plnk:portType name="deal1:carPort"/>
  </plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="dealer2CarLink">
...

<plnk:partnerLinkType name="supplierCarLink">

  <plnk:role name="supplier">
    <plnk:portType name="supp:carPort"/>
  </plnk:role>
</plnk:partnerLinkType>

<service name="SupplierService">
  <port name="SupplierCar" binding="supp:carBinding">
    <soap:address
     location="http://localhost:8080/active-bpel/services/SupplierService"/>
  </port>
 </service>
</definitions>
```

A.6.10 Common Definitions - supplier_defs.wsdl

```
<definitions name="SupplierCommonDefinitions"
 targetNamespace="urn:DoubleQuoteDefs"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:deal1_defs="urn:BigDealDefs"
 xmlns:deal2_defs="urn:WheelerDealerDefs"
 xmlns:defs="urn:DoubleQuoteDefs"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <types>

  <xsd:schema
   targetNamespace="urn:DoubleQuoteDefs"
```

```
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

       <complexType name="need">
         <sequence>
           <element name="name" type="xsd:string"/>
           <element name="address" type="xsd:string"/>
           <element name="model" type="xsd:string"/>
         </sequence>
       </complexType>

       <complexType name="offer">
         <sequence>
           <element name="reference" type="xsd:nonNegativeInteger"/>
           <element name="dealer" type="xsd:string"/>
           <element name="price" type="xsd:float"/>
           <element name="delivery" type="xsd:nonNegativeInteger"/>

         </sequence>
       </complexType>
   ...

   <message name="needMessage">
     <part name="need" type="defs:need"/>
   </message>

   <message name="offerMessage">
     <part name="offer" type="defs:offer"/>
   </message>

 </definitions>
```

A.6.11 Supplier BPEL

```
 <process name="SupplierProcess"
  ...>

   <bpel:extensions>
     <bpel:extension mustUnderstand="yes"
      namespace="http://www.activebpel.org/2006/09/bpel/extension/query_handling"/>
   </bpel:extensions>

   <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
    namespace="urn:DoubleQuote" location="supplier.wsdl"/>
   <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
    namespace="urn:DoubleQuoteDefs" location="supplier_defs.wsdl"/>

   <partnerLinks>
   ...
   <variables>
   ...
   <flow>

     <links>
       <link name="SUPPLIER.1-SUPPLIER.2"/>
       ...
     ...
```

```
<pick name="SUPPLIER.1" createInstance="yes">
  <onMessage
   partnerLink="supplierCar" portType="supp:carPort"
   operation="order" variable="need">
    <empty>
      <sources>
        <source linkName="SUPPLIER.1-SUPPLIER.2"/>
      </sources>
    </empty>
  </onMessage>

  <onMessage
   partnerLink="supplierCar" portType="supp:carPort"
   operation="cancel" variable="offer">
    <empty>
      <sources>
        <source linkName="SUPPLIER.10-SUPPLIER.11">
          <transitionCondition>
           $offer.offer/dealer = 'BigDeal'
          </transitionCondition>
        </source>
        <source linkName="SUPPLIER.10-SUPPLIER.12">
          <transitionCondition>
           not($offer.offer/dealer = 'BigDeal')
          </transitionCondition>
...
<empty name="SUPPLIER.2">
  <targets>
    <target linkName="SUPPLIER.1-SUPPLIER.2"/>
  </targets>
  <sources>
    <source linkName="SUPPLIER.2-SUPPLIER.3"/>
    <source linkName="SUPPLIER.2-SUPPLIER.4"/>
...
<invoke name="SUPPLIER.3"
...
<invoke name="SUPPLIER.4"
  ...
</process>
```

A.6.12 File Structure of Generated Code in Broker

Tree structure in <CRESS>/bpel/broker

```
|   broker.bpel
|   broker.bpr
|   broker.pdd
|   broker.properties
|   broker.wsdl
|   broker_defs.wsdl
|   lender_broker.wsdl
|   lender_defs.wsdl
|   supplier_broker.wsdl
|   supplier_defs.wsdl
|
+---broker
|   |   broker.bpel
```

```
|   |   broker.pdd
|   |   broker.wsdl
|   |   broker_defs.wsdl
|   |   lender_broker.wsdl
|   |   lender_defs.wsdl
|   |   supplier_broker.wsd
|   |   supplier_defs.wsdl
|   |
|   +---CarMenDefs
|   |       Schedule.java
|   |
|   +---META-INF
|           wsdlCatalog.xml
```

A.6.13 Broker WSDL

```
<definitions name="BrokerDefinitions"
 ...>

  <import namespace="urn:CarMenDefs"
   location="broker_defs.wsdl"/>
  <import namespace="urn:LoanStarDefs"
   location="lender_defs.wsdl"/>
  <import namespace="urn:DoubleQuoteDefs"
   location="supplier_defs.wsdl"/>

  <portType name="carPort">
    <operation name="arrange">
      <input message="supp_defs:needMessage"/>
      <output message="defs:scheduleMessage"/>
      <fault name="refusal" message="lend_defs:stringMessage"/>
    </operation>
  </portType>

  <binding name="carBinding" type="brok:carPort">
  ...

  <plnk:partnerLinkType name="brokerCarLink">
    <plnk:role name="broker">
      <plnk:portType name="brok:carPort"/>
    </plnk:role>
  </plnk:partnerLinkType>

  <plnk:partnerLinkType name="lenderLoanLink">
    <plnk:role name="lender">
      <plnk:portType name="lend:loanPort"/>
    </plnk:role>
  </plnk:partnerLinkType>

  <plnk:partnerLinkType name="supplierCarLink">

    <plnk:role name="supplier">
      <plnk:portType name="supp:carPort"/>
    </plnk:role>
  </plnk:partnerLinkType>
```

```
  <service name="BrokerService">                                                  40
    <port name="BrokerCar" binding="brok:carBinding">                             41
      <soap:address                                                               42
       location="http://localhost:8080/active-bpel/services/BrokerService"/>     43
    </port>                                                                       44
  </service>                                                                      45
</definitions>                                                                    46
```

A.6.14 WSDL Common Definitions

```
<definitions name="BrokerCommonDefinitions"                                       0
 targetNamespace="urn:CarMenDefs"                                                 1
 xmlns="http://schemas.xmlsoap.org/wsdl/"                                         2
 xmlns:defs="urn:CarMenDefs"                                                      3
 xmlns:lend_defs="urn:LoanStarDefs"                                               4
 xmlns:supp_defs="urn:DoubleQuoteDefs"                                            5
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                    6
                                                                                  7
  <import namespace="urn:LoanStarDefs"                                            8
   location="lender_defs.wsdl"/>                                                  9
  <import namespace="urn:DoubleQuoteDefs"                                         10
   location="supplier_defs.wsdl"/>                                                11
                                                                                  12
  <types>                                                                         13
    <xsd:schema targetNamespace="urn:CarMenDefs" ...>                             14
      <complexType name="schedule">                                               15
        <sequence>                                                                16
          <element name="reference" type="xsd:nonNegativeInteger"/>               17
          <element name="dealer" type="xsd:string"/>                              18
          <element name="price" type="xsd:float"/>                                19
          <element name="delivery" type="xsd:nonNegativeInteger"/>                20
          <element name="rate" type="xsd:float"/>                                 21
        </sequence>                                                               22
      </complexType>                                                              23
    </xsd:schema>                                                                 24
  </types>                                                                        25
  <message name="scheduleMessage">                                                26
    <part name="schedule" type="defs:schedule"/>                                  27
  </message>                                                                      28
</definitions>                                                                    29
```

A.6.15 Broker BPEL

```
<process name="BrokerProcess" ...>                                                0
  ...                                                                             1
  <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"                      2
   namespace="urn:CarMen" location="broker.wsdl"/>                                3
  <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"                      4
   namespace="urn:CarMenDefs" location="broker_defs.wsdl"/>                       5
                                                                                  6
  <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"                      7
   namespace="urn:LoanStar" location="lender.wsdl"/>                              8
   ...                                                                            9
                                                                                  10
  <partnerLinks>                                                                  11
    <partnerLink name="brokerCar" partnerLinkType="brok:brokerCarLink"            12
     myRole="broker"/>                                                            13
```

```
    ...                                                                    14
  </partnerLinks>                                                          15
                                                                           16
  <variables>                                                              17
    <variable name="error" messageType="lend_defs:stringMessage"/>         18
    <variable name="need" messageType="supp_defs:needMessage"/>            19
    ...                                                                    20
    <variable name="schedule" messageType="defs:scheduleMessage"/>         21
  </variables>                                                             22
                                                                           23
  <flow>                                                                   24
    <links>                                                                25
      <link name="BROKER.1-BROKER.2"/>                                     26
      ...                                                                  27
    <scope>                                                                28
      <faultHandlers>                                                      29
        <catch faultName="lend:refusal" faultVariable="error"              30
         faultMessageType="lend_defs:stringMessage">                       31
          <flow>                                                           32
            <compensate>                                                   33
              <sources>                                                    34
                <source linkName="BROKER.7-BROKER.8"/>                     35
              </sources>                                                   36
            </compensate>                                                  37
            <reply name="BROKER.8"                                         38
             partnerLink="brokerCar" portType="brok:carPort"               39
             operation="arrange" variable="error" faultName="brok:refusal">40
              <targets>                                                    41
                <target linkName="BROKER.7-BROKER.8"/>                     42
              </targets>                                                   43
              <sources>                                                    44
                <source linkName="BROKER.8-BROKER.9"/>                     45
              </sources>                                                   46
            </reply>                                                       47
            <exit name="BROKER.9">                                         48
              <targets>                                                    49
                <target linkName="BROKER.8-BROKER.9"/>                     50
              </targets>                                                   51
            ...                                                            52
      ...                                                                  53
      <flow>                                                               54
        <receive name="BROKER.1"                                           55
         partnerLink="brokerCar" portType="brok:carPort"                   56
         operation="arrange" variable="need" createInstance="yes">         57
          <sources>                                                        58
            <source linkName="BROKER.1-BROKER.2"/>                         59
          </sources>                                                       60
        </receive>                                                         61
...                                                                        62
```

A.7 Implementation Validation

A.7.1 Approver MINT Properties

```
# Service URL
target.url=http://localhost:8080/active-bpel/services/ApproverLoan
```

```
# Service Timeout (msec, default 0 = no timeout)
service.timeout=5000

# Service Package
service.package.approver=FirstRate

# Fault Classes
class.approver.loan.approve.fault.refusal=LoanStarDefs.StringMessage
```

A.7.2 Approver Translated MINT Scenarios

```
test(APPROVER_Low_Rate,
   sequence(
      send(approver.loan.approve,Proposal2(″Ken Smith″,″Liverpool UK″,6000.)),
      read(approver.loan.approve,3.7),OK))

test(APPROVER_Medium_Rate,
   sequence(
      send(approver.loan.approve,Proposal2(″Angus Og″,″Airth Scotland″,20000.)),
      read(approver.loan.approve,4.1),OK))

test(APPROVER_High_Rate,
   sequence(
      send(approver.loan.approve,Proposal2(″Nancy Turner″,″Manchester England″,14999.)),
      read(approver.loan.approve,4.4),OK))

test(APPROVER_Loan_Unacceptable,
   sequence(
      send(approver.loan.approve,Proposal2(″Ian Carey″,″Croydon England″,15000.)),
      read(approver.loan.approve,refusal,″loan unacceptable″),OK))
```

A.7.3 Assessor MINT Properties

```
# Service URL
target.url=http://localhost:8080/active−bpel/services/AssessorLoan

# Service Timeout (msec, default 0 = no timeout)
service.timeout=5000

# Service Package
service.package.assessor=RiskTaker
```

A.7.4 Assessor Translated MINT Scenarios

```
test(ASSESSOR_Low_Risk,
   sequence(
      send(assessor.loan.assess,Proposal(″Mike Turner″,″Carlisle UK″,20000.)),
      read(assessor.loan.assess,″low″),OK))

test(ASSESSOR_Medium_Risk,
   sequence(
      send(assessor.loan.assess,Proposal(″Fred Hoyle″,″UK″,5000.)),
      read(assessor.loan.assess,″medium″),OK))

test(ASSESSOR_High_Risk,
   sequence(
      send(assessor.loan.assess,Proposal(″Patrice Touvet″,″Paris France″,1000.)),
```

read(assessor.loan.assess,″high″),**OK**))

A.7.5 Lender MINT Properties

```
# Service URL
target.url=http://localhost:8080/active−bpel/services/LenderService

# Service Timeout (msec, default 0 = no timeout)
service.timeout=5000

# Service Package
service.package.lender=LoanStar

# Fault Classes
class.lender.loan.quote.fault.refusal=LoanStarDefs.StringMessage
```

A.7.6 Lender Translated MINT Scenarios

test(LENDER_Little_Low_Risk,
  sequence(**send**(lender.loan.quote,Proposal(″Nancy Turner″,″Manchester England″,9999.)),
    read(lender.loan.quote,3.5),**OK**))

test(LENDER_No_Risk_Assess_Low,
  sequence(**send**(lender.loan.quote,Proposal(″Nancy Turner″,″Manchester England″,10000.)),
    offer(**read**(lender.loan.quote,3.5),**OK**)))

test(LENDER_Lots_Ken,
  sequence(**send**(lender.loan.quote,Proposal(″Ken Boyle″,″Dublin Ireland″,10000.)),
    read(lender.loan.quote,3.7),**OK**))

test(LENDER_Lots_Scotland,
  sequence(**send**(lender.loan.quote,Proposal(″Mary Duncan″,″Wick Scotland″,20000.)),
    read(lender.loan.quote,4.1),**OK**))

test(LENDER_Lots_Under_15000,
  sequence(**send**(lender.loan.quote,Proposal(″Sally Dean″,″Cardiff Wales″,14999.)),
    read(lender.loan.quote,4.4),**OK**))

test(LENDER_Lots_Exceeds_15000,
  sequence(**send**(lender.loan.quote,Proposal(″Ian Carey″,″Croydon England″,15000.)),
    read(lender.loan.quote,refusal,″loan unacceptable″),**OK**))

A.7.7 Dealer1 MINT Properties

```
# Service URL
target.url=http://localhost:8080/active-bpel/services/Dealer1Car

# Service Timeout (msec, default 0 = no timeout)
service.timeout=5000

# Service Package
service.package.dealer1=BigDeal
```

A.7.8 Dealer1 Translated MINT Scenarios

```
test(DEALER1_Mondeo,
  sequence(
    send(dealer1.car.quote,Need(″Mark Flowers″,″Uist Scotland″,″Mondeo″)),
    read(dealer1.car.quote,Offer(?NonNegativeInteger,″BigDeal″,20000.,15)),OK))

test(DEALER1_A5,
  sequence(
    send(dealer1.car.quote,Need(″Peter Gough″,″Congleton UK″,″A5″)),
    read(dealer1.car.quote,Offer(?NonNegativeInteger,″BigDeal″,35000.,30)),OK))

test(DEALER1_Megane,
  sequence(
    send(dealer1.car.quote,Need(″Jan Hiddink″,″Hengelo Netherlands″,″Megane″)),
    read(dealer1.car.quote,Offer(?NonNegativeInteger,″BigDeal″,11000.,5)),OK))

test(DEALER1_XJ6,
  sequence(
    send(dealer1.car.quote,Need(″Iain MacKay″,″Throsk Scotland″,″XJ6″)),
    read(dealer1.car.quote,Offer(?NonNegativeInteger,″BigDeal″,1000000.,0)),OK))
```

A.7.9 Dealer2 MINT Properties

```
# Service URL
target.url=http://localhost:8080/active-bpel/services/Dealer2Car

# Service Timeout (msec, default 0 = no timeout)
service.timeout=5000

# Service Package
service.package.dealer2=WheelerDealer
```

A.7.10 Dealer2 Translated MINT Scenarios

```
test(DEALER2_Mondeo,
  sequence(
    send(dealer2.car.quote,Need(″Mark Flowers″,″Uist Scotland″,″Mondeo″)),
    read(dealer2.car.quote,Offer(?NonNegativeInteger,″WheelerDealer″,20000.,10)),OK))

test(DEALER2_A5,
  sequence(
    send(dealer2.car.quote,Need(″Peter Gough″,″Congleton UK″,″A5″)),
    read(dealer2.car.quote,Offer(?NonNegativeInteger,″WheelerDealer″,35000.,20)),OK))

test(DEALER2_Astra,
  sequence(
    send(dealer2.car.quote,Need(″Hywel Thomas″,″Swansea Wales″,″Astra″)),
    read(dealer2.car.quote,Offer(?NonNegativeInteger,″WheelerDealer″,18000.,30)),OK))

test(DEALER2_XJ6,
  sequence(
    send(dealer2.car.quote,Need(″Iain MacKay″,″Throsk Scotland″,″XJ6″)),
    read(dealer2.car.quote,Offer(?NonNegativeInteger,″WheelerDealer″,1000000.,0)),OK))
```

A.7.11 Supplier MINT Properties

```
# Service URL
target.url=http://localhost:8080/active-bpel/services/SupplierService
```

```
# Service Timeout (msec, default 0 = no timeout)
service.timeout=5000

# Service Package service.package.supplier=DoubleQuote
```

A.7.12 Supplier Translated MINT Scenarios

```
test(SUPPLIER_Mondeo,
  sequence(
    send(supplier.car.order,Need("Mark Flowers","Uist Scotland","Mondeo")),
    read(supplier.car.order,Offer(?NonNegativeInteger,"WheelerDealer",20000.,10)),OK))

test(SUPPLIER_A5,
  sequence(
    send(supplier.car.order,Need("Peter Gough","Congleton UK","A5")),
    read(supplier.car.order,Offer(?NonNegativeInteger,"BigDeal",33000.,30)),OK))

test(SUPPLIER_Megane,
  sequence(
    send(supplier.car.order,Need("Jan Hiddink","Hengelo Netherlands","Megane")),
    read(supplier.car.order,Offer(?NonNegativeInteger,"BigDeal",11000.,5)),OK))

test(SUPPLIER_Astra,
  sequence(
    send(supplier.car.order,Need("Hywel Thomas","Swansea Wales","Astra")),
    read(supplier.car.order,Offer(?NonNegativeInteger,"WheelerDealer",18000.,30)),OK))

test(SUPPLIER_XJ6,
  sequence(
    send(supplier.car.order,Need("Iain MacKay","Throsk Scotland","XJ6")),
    read(supplier.car.order,Offer(?NonNegativeInteger,"WheelerDealer",1000000.,0)),OK))
```

A.7.13 Broker MINT Properties

```
# Service URL
target.url=http://localhost:8080/active−bpel/services/BrokerService

# Service Timeout (msec, default 0 = no timeout)
service.timeout=10000

# Service Package
service.package.broker=CarMen

# Fault Classes
class.broker.car.arrange.fault.refusal=LoanStarDefs.StringMessage
```

A.7.14 Broker Translated MINT Scenarios

```
test(BROKER_Mondeo_Ken,
  sequence(
    send(broker.car.arrange,Need("Ken Boyle","Dublin Ireland","Mondeo")),
    read(broker.car.arrange,
      Schedule(?NonNegativeInteger,"WheelerDealer",20000.,10,3.7)),OK))

test(BROKER_A5_Scotland,
  sequence(
    send(broker.car.arrange,Need("Mary Duncan","Wick Scotland","A5")),
```

```
      read(broker.car.arrange,
        Schedule(?NonNegativeInteger,″BigDeal″,33000.,30,4.1)),OK))

  test(BROKER_Megane,
    sequence(
      send(broker.car.arrange,Need(″Sally Dean″,″Cardiff Wales″,″Megane″)),
      read(broker.car.arrange,
        Schedule(?NonNegativeInteger,″BigDeal″,11000.,5,4.4)),OK))

  test(BROKER_Astra,
    sequence(
      send(broker.car.arrange,Need(″Ian Carey″,″Croydon England″,″Astra″)),
      read(broker.car.arrange,refusal,″loan unacceptable″),OK))

  test(BROKER_XJ6,
    sequence(
      send(broker.car.arrange,Need(″Iain MacKay″,″Throsk Scotland″,″XJ6″)),
      read(broker.car.arrange,refusal,″car unavailable″),OK))
```

B Allocator Composed Service

B.1 Partner Specification

B.1.1 Resource Completed LOTOS Specification

```
Process RESOURCE [resource] : Exit(States) :=          (* RESOURCE phantom *)
  resource ?scheme:Text;                               (* get scheme name *)
  (
    [scheme eq (t(s)∼iI∼c∼9∼2)] ->                     (* SIC92 request? *)
      resource !True !1 Of Nat;                        (* return EPR 1 *)
      RESOURCE [resource]                              (* repeat behaviour *)
  []                                                   (* or *)
    [scheme eq (t(s)∼o∼c∼2∼0∼0∼0)] ->                  (* SOC2000 request? *)
      resource !True !2 Of Nat;                        (* return EPR 2 *)
      RESOURCE [resource]                              (* repeat behaviour *)
  []                                                   (* or *)
    [(scheme ne (t(s)∼iI∼c∼9∼2)) and                   (* not SIC92/SOC2000? *)
     (scheme ne (t(s)∼o∼c∼2∼0∼0∼0))] ->
      resource !False !t(U)∼n∼k∼n∼o∼w∼n∼ ^ ∼s∼c∼h∼e∼m∼e; (* return fault *)
    RESOURCE [resource]                                (* repeat behaviour *)
  )
[]                                                     (* or *)
  resource ?epr:Nat ?occupation:Text;                  (* translation request? *)
  (
    Let code:Text = translate(epr,occupation) In       (* get translation *)
      (
        [code ne <>] ->                                (* job translated? *)
          resource !True !code;                        (* return translation *)
          RESOURCE [resource]                          (* repeat behaviour *)
      []                                               (* or *)
        [code eq <>] ->                                (* job not translated? *)
          resource !False !t(U)∼n∼k∼n∼o∼w∼n∼ ^ ∼j∼o∼b; (* return fault *)
          RESOURCE [resource]                          (* repeat behaviour *)
      )
  )
```

```
Where                                                  (* local definitions *)

  Type Translation Is TextOps                          (* job translation *)
    Opns
      translate: Nat,Text -> Text                      (* scheme EPR and job to code *)
    Eqns
    (* rewriting equations for translated occupation values *)
    ...
```

B.1.2 Factory Completed LOTOS Specification

```
Process FACTORY [factory,resource] : Exit(States) :=          (* FACTORY partner *)
  factory !job !allocate ?scheme:Text;                        (* get scheme name *)
  resource !scheme;                                           (* send scheme name *)
  (
    resource !True ?epr:Nat;                                  (* get EPR *)
    factory !job !allocate !epr;                              (* return mapping EPR *)
    FACTORY [factory,resource]                                (* repeat behaviour *)
  []                                                          (* or *)
    resource !False ?reason:Text;                             (* get fault *)
    factory !job !allocate !factoryError !reason; (* return factory fault *)
    FACTORY [factory,resource]                                (* repeat behaviour *)
  )
[]
  factory !job !deallocate ?epr:Nat;
  FACTORY [factory,resource]                                  (* repeat behaviour *)
EndProc                                                       (* end FACTORY *)
```

B.1.3 Mapper Completed LOTOS Specification

```
Process MAPPER [mapper,resource] : Exit(States) :=            (* MAPPER partner *)
  mapper !job ?epr:Nat;                                       (* get resource reference *)
  mapper !job !translate ?occupation:Text;                    (* get job name *)
  resource !epr !occupation;                                  (* send EPR and job name *)
  (
    resource !True ?code:Text;                                (* get job code *)
    mapper !job !translate !code;                             (* return job code *)
    MAPPER [mapper,resource]                                  (* repeat behaviour *)
  []                                                          (* or *)
    resource !False ?reason:Text;                             (* get failure reason *)
    mapper !job !translate !mapperError !reason; (* return failure reason *)
    MAPPER [mapper,resource]                                  (* repeat behaviour *)
  )
EndProc                                                       (* end MAPPER *)
```

B.2 Formal Specification

B.2.1 Allocator LOTOS Specification

```
Specification GSSystem [allocator] : Exit(States)             (* GS system *)

Library                                                       (* library *)
...
Behaviour
Hide resource In                                              (* hide internal gates *)
  RESOURCE [resource]                                         (* call RESOURCE process *)
|[resource]|                                                  (* synchronised with services *)
  ALLOCATOR [allocator,resource]                              (* call ALLOCATOR process *)
```

```
Where                                                    (* local definitions *)
Type Mapping Is BaseTypes                                (* mapping record *)
  Sorts Mapping                                          (* mapping sort *)
  Opns                                                   (* mapping operations *)
    AnyMapping: -> Mapping
    _eq_,_ne_: Mapping,Mapping -> Bool
    mapping: Text,Text -> Mapping
    getJob: Mapping -> Text
    setJob: Mapping,Text -> Mapping
    getScheme: Mapping -> Text
    setScheme: Mapping,Text -> Mapping
...
Process ALLOCATOR [allocator,resource] : Exit(States) := (* ALLOCATOR service *)
  Hide factory,mapper In                                 (* hide internal gates *)
    (
      FACTORY [factory,resource]                         (* call FACTORY partner *)
    |||                                                  (* interleaved with *)
      MAPPER [mapper,resource]                           (* call MAPPER partner *)
    )
  ||[factory,mapper]||                                   (* synchronised with partners *)
    ALLOCATOR_1 [allocator,factory,mapper]               (* call main process *)
     (AnyText,AnyText,AnyNat,AnyNat,AnyMapping,AnyText,AnyText)

Where                                                    (* local definitions *)
Process FACTORY [factory,resource] : Exit(States) :=     (* FACTORY partner *)
...
Process MAPPER [mapper,resource] : Exit(States) :=       (* MAPPER partner *)
...
Process RESOURCE [resource] : Exit(States) :=            (* RESOURCE phantom *)
...
```

B.3 Formal Verification

B.3.1 Annotated LOTOS Specification

```
Specification GSSystem [allocator] : Exit(States)        (* GS system *)

Type gssystem Is
  Sorts
    ...
    mapping (*! implementedby MAPPING_SORT
                comparedby MAPPING_COMP
                iteratedby MAPPING_FIRST and MAPPING_NEXT
                printedby MAPPING_PRINT *),
    ...
...
Behaviour
... specification remain unchanged as per generated ...
```

B.3.2 Translated μ-calculus

```
(* Property name : GENERAL RESPONSE *)

[(((((true)*) . ((″ALLOCATOR !JOB !TRANSLATE″ # ″ !″ # 'MAPPING (.*)')))))]
  mu X. (<((true))> (true) and
    [(not((((″ALLOCATOR !JOB !TRANSLATE″ # ″ !″ # '″[./a−zA−Z0−9]*″')) or
      ((″ALLOCATOR !JOB !TRANSLATE″ # ″ !″ # ″ALLOCATORERROR″ # ″ !″
          # ″\″UNKNOWN SCHEME\″″)) or
      ((″ALLOCATOR !JOB !TRANSLATE″ # ″ !″ # ″ALLOCATORERROR″ # ″ !″
```

```
            # ″\″UNKNOWN JOB\″″)))))] X)

(* Property name : SOC2000 SAFETY *)

[(((((true)*) . ((″ALLOCATOR !JOB !TRANSLATE″ # ″ !″ # ’MAPPING (″.*″, ″SOC2000″)’)))))]
   mu X. (<((true))> (true) and
      [(not((((″ALLOCATOR !JOB !TRANSLATE″ # ″ !″ # ’″[0-9]*″’)) or
         ((″ALLOCATOR !JOB !TRANSLATE″ # ″ !″ # ″ALLOCATORERROR″
            # ″ !″ # ″\″UNKNOWN JOB\″″)))))] X)

(* Property name : DEADLOCK FREEDOM *)

[((true)*)] <((true))> true

(* Property name : LIVELOCK FREEDOM *)

<((true)*)> @ (″i″) equ false

(* Property name : INITIALS SAFETY *)

[(not((((″ALLOCATOR !JOB !TRANSLATE″ # ″ !″ # ’MAPPING (.*)’)        ))))] false
```

B.3.3 Generated C Code of User Types

```
#define CRESS_TEXT_ENUM ″BOOKBINDER″,″CAB DRIVER″,
   ″NURSE″,″PRIVATE DETECTIVE″,″SIC92″,″SOC2000″
```

B.4 Formal Validation

B.4.1 Allocator Translated LOTOS Scenarios

```
Process ALLOCATOR_SOC2_Nurse [allocator,OK] : NoExit :=
   allocator !job !translate !mapping(t(N)∼u∼r∼s∼e,t(S)∼O∼C∼2∼0∼0∼0);
   allocator !job !translate !t(3)∼2∼1∼1;
   OK;
   Stop
EndProc (* ALLOCATOR_SOC2_Nurse *)

Process ALLOCATOR_SIC_Nurse [allocator,OK] : NoExit :=
   allocator !job !translate !mapping(t(N)∼u∼r∼s∼e,t(S)∼II∼C∼9∼2);
   allocator !job !translate !t(9)∼5∼ .
   ∼1∼4;
   OK;
   Stop
EndProc (* ALLOCATOR_SIC_Nurse *)

Process ALLOCATOR_SOC2_Unknown [allocator,OK] : NoExit :=
   allocator !job !translate !mapping(t(S)∼a∼ii∼l∼o∼r,t(S)∼O∼C∼2∼0∼0∼0);
   allocator !job !translate !allocatorError !t(U)∼n∼k∼n∼o∼w∼n∼ ^ ∼j∼o∼b;
   OK;
   Stop
EndProc (* ALLOCATOR_SOC2_Unknown *)

Process ALLOCATOR_Unknown_Nurse [allocator,OK] : NoExit :=
   allocator !job !translate !mapping(t(N)∼u∼r∼s∼e,t(U)∼n∼k∼n∼o∼w∼n);
   (
      allocator !job !translate !t(3)∼2∼1∼1;
```

```
      Stop
    []
      I;
      OK;
      Stop
    )
  EndProc (* ALLOCATOR_Unknown_Nurse *)

  Process ALLOCATOR_SOC2_Unknown_Scheme [allocator,OK] : NoExit :=
    allocator !job !translate !mapping(t(S)~a~ii~l~o~r,t(S)~O~C~2~0~0~0~0);
    allocator !job !translate !allocatorError !t(U)~n~k~n~o~w~n~ ^ ~s~c~h~e~m~e;
    OK;
    Stop
  EndProc (* ALLOCATOR_SOC2_Unknown_Scheme *)
```

B.5 Implementation

B.5.1 File Structure of Generated Code

```
|   allocator.bpel
|   allocator.bpr
|   allocator.pdd
|   allocator.properties
|   allocator.wsdl
|   allocator_defs.wsdl
|   factory.gar
|   factory.properties
|   factory.wsdl
|   filestruct.txt
|   mapper.gar
|   mapper.properties
|   mapper.wsdl
|
+---allocator
|   |   allocator.bpel
|   |   allocator.pdd
|   |   allocator.wsdl
|   |   allocator_defs.wsdl
|   |   factory.wsdl
|   |   mapper.wsdl
|   |   rpw-2.wsdl
|   |
|   +---AllocatorDefs
|   |       Mapping.java
|   |
|   +---META-INF
|           wsdlCatalog.xml
|
|
+---factory
|   |   jndi-config-deploy.xml
|   |   server-deploy.wsdd
|   |
|   +---AllocatorDefs
|   |       Mapping.class
|   |       Mapping.java
```

```
|   |       StringMessage.class
|   |       StringMessage.java
|   |
|   +---Factory
|   |       FactoryService.class
|   |       FactoryService.java
|   |       FactoryServiceAddressing.class
|   |       FactoryServiceAddressing.java
|   |       FactoryServiceAddressingLocator.class
|   |       FactoryServiceAddressingLocator.java
|   |       FactoryServiceLocator.class
|   |       FactoryServiceLocator.java
|   |       JobBindingStub.class
|   |       JobBindingStub.java
|   |       JobPort.class
|   |       JobPort.java
|   |
|   +---lib
|   |       factory.jar
|   |
|   +---META-INF
|   +---schema
|   |       allocator_defs.wsdl
|   |       factory.wsdl
|   |
|   \---uk
|       \---ac
|           \---stir
|               \---cs
|                   \---factory
|                           AllocatorQNames.class
|                           AllocatorQNames.java
|                           AllocatorResource.class
|                           AllocatorResource.java
|                           AllocatorResourceHome.class
|                           AllocatorResourceHome.java
|                           FactoryService.class
|                           FactoryService.java
|
\---mapper
    |   jndi-config-deploy.xml
    |   server-deploy.wsdd
    |
    +---AllocatorDefs
    |       StringMessage.class
    |       StringMessage.java
    |
    +---lib
    |       factory.jar
    |       mapper.jar
    |
    +---Mapper
    |       JobBindingStub.class
    |       JobBindingStub.java
    |       JobPort.class
```

```
|       JobPort.java
|       MapperService.class
|       MapperService.java
|       MapperServiceAddressing.class
|       MapperServiceAddressing.java
|       MapperServiceAddressingLocator.class
|       MapperServiceAddressingLocator.java
|       MapperServiceLocator.class
|       MapperServiceLocator.java
|
+---META-INF
+---schema
|       allocator_defs.wsdl
|       mapper.wsdl
|
\---uk
    \---ac
        \---stir
            \---cs
                \---mapper
                        MapperService.class
                        MapperService.java
```

B.5.2 Factory Implementation

```
package uk.ac.stir.cs.factory;

... imports ...

public class FactoryService {

   public EndpointReferenceType allocate(String scheme)
            throws RemoteException, StringMessage {
      ...

      try {
         ...
         home = (AllocatorResourceHome) ctx.getResourceHome();
         key = home.create(scheme);
         ...

      EndpointReferenceType epr = null;

      ... create the value of epr ...

      return(epr);
   }

      public void deallocate(EndpointReferenceType reference)
               throws java.rmi.RemoteException {
         try {
            ResourceContext context = ResourceContext.getResourceContext();
            AllocatorResourceHome home =
                        (AllocatorResourceHome) context.getResourceHome();
            ResourceKey resourceKey = this.getAsResourceKey(reference);
            if (resourceKey != null) {
```

```
            AllocatorResource resource = (AllocatorResource) home.find(resourceKey);
            home.remove(resourceKey);
            ...
        }

    private ResourceKey getAsResourceKey(EndpointReferenceType epr)
            throws Exception {
        ...
    }

}
```

B.5.3 Factory WSDD

```
<deployment name="FactoryConfiguration"
 ...>

  <service name="FactoryJob" provider="Handler" style="document"
   use="literal">
    <parameter name="className" value="uk.ac.stir.cs.factory.FactoryService"/>
    <wsdlFile>share/schema/factory.wsdl</wsdlFile>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="scope" value="Application"/>
    <parameter name="providers" value="GetRPProvider"/>
    <parameter name="loadOnStartup" value="true"/>
    <parameter name="instance" value="MapperJob"/>
  </service>
</deployment>
```

B.5.4 Factory JNDI Configuration

```
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

  <service name="FactoryJob">
    <resource name="home"
      type="uk.ac.stir.cs.factory.AllocatorResourceHome">
    <resourceParams>

    <parameter>
        <name>resourceClass</name>
        <value>uk.ac.stir.cs.factory.AllocatorResource</value>
    </parameter>

    <parameter>
        <name>factory</name>
        <value>org.globus.wsrf.jndi.BeanFactory</value>
    </parameter>

    <parameter>
        <name>resourceKeyType</name>
        <value>java.lang.Integer</value>
    </parameter>

    <parameter>
        <name>resourceKeyName</name>
        <value>
        {http://www.cs.stir.ac.uk/allocator/resource}AllocatorResourceKey
```

```
            </value>
        </parameter>
...
```

B.5.5 Factory WSDL

```
<definitions name="FactoryDefinitions"
 ...>

  <import namespace="urn:AllocatorDefs"
   location="allocator_defs.wsdl"/>

  <portType name="jobPort">
    <operation name="allocate">
      <input message="defs:stringMessage"/>
      <output message="defs:referenceMessage"/>
      <fault name="factoryError" message="defs:stringMessage"/>
    </operation>
    <operation name="deallocate">
      <input message="defs:referenceMessage"/>
    </operation>
  </portType>

  <binding name="jobBinding" type="factory:jobPort">
   ...
  </binding>

  <service name="FactoryService">
    <port name="FactoryJob" binding="factory:jobBinding">
      <soap:address
       location="http://localhost:8880/wsrf/services/FactoryJob"/>
    </port>
  </service>
</definitions>
```

B.5.6 Mapper JNDI Configuration

```
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

  <service name="MapperJob">
    <resourceLink name="home"
      target="java:comp/env/services/FactoryJob/home"/>
  </service>

</jndiConfig>
```

B.5.7 Mapper WSDL

```
<definitions name="MapperDefinitions"
 ...>

  <import namespace="urn:AllocatorDefs"
   location="allocator_defs.wsdl"/>

  <portType name="jobPort">
    <operation name="translate">
      <input message="defs:stringMessage"/>
      <output message="defs:stringMessage"/>
```

```
      <fault name="mapperError" message="defs:stringMessage"/>
    </operation>
  </portType>

  <binding name="jobBinding" type="mapper:jobPort">
    ...
  </binding>

  <service name="MapperService">
    <port name="MapperJob" binding="mapper:jobBinding">
      <soap:address
       location="http://localhost:8880/wsrf/services/MapperJob"/>
    </port>
  </service>
</definitions>
```

B.5.8 Allocator WSDL

```
<definitions name="AllocatorDefinitions"
 ...>

  <import namespace="urn:AllocatorDefs"
   location="allocator_defs.wsdl"/>
  <import namespace="urn:Factory"
   location="factory.wsdl"/>
  <import namespace="urn:Mapper"
   location="mapper.wsdl"/>

  <portType name="jobPort">
    <operation name="translate">
      <input message="defs:mappingMessage"/>
      <output message="defs:stringMessage"/>
      <fault name="allocatorError" message="defs:stringMessage"/>
    </operation>
  </portType>

  <binding name="jobBinding" type="allo:jobPort">
    ...
  </binding>

  <plnk:partnerLinkType name="allocatorJobLink">
    <plnk:role name="allocator">
      <plnk:portType name="allo:jobPort"/>
    </plnk:role>
  </plnk:partnerLinkType>

  <plnk:partnerLinkType name="factoryJobLink">
    <plnk:role name="factory">
      <plnk:portType name="factory:jobPort"/>
    </plnk:role>
  </plnk:partnerLinkType>

  <plnk:partnerLinkType name="mapperJobLink">
    <plnk:role name="mapper">
      <plnk:portType name="mapper:jobPort"/>
    </plnk:role>
```

```
  </plnk:partnerLinkType>

  <service name="AllocatorService">
    <port name="AllocatorJob" binding="allo:jobBinding">
      <soap:address
       location="http://localhost:8080/active-bpel/services/AllocatorService"/>
    </port>
  </service>
</definitions>
```

B.5.9 Common Definitions - allocator_defs.wsdl

```
<definitions name="AllocatorCommonDefinitions"
 ...>

  <types>
    <xsd:schema
     targetNamespace="urn:AllocatorDefs"
     xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://www.w3.org/2005/08/addressing"
       schemaLocation="..."/>
    </xsd:schema>

    <xsd:schema
     targetNamespace="urn:AllocatorDefs"
     xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <complexType name="mapping">
        <sequence>
          <element name="job" type="xsd:string"/>
          <element name="scheme" type="xsd:string"/>
        </sequence>
      </complexType>
    </xsd:schema>
  </types>

  <message name="stringMessage">
    <part name="string" type="xsd:string"/>
  </message>

  <message name="referenceMessage">
    <part name="reference" type="wsa:EndpointReferenceType"/>
  </message>
  <message name="mappingMessage">
    <part name="mapping" type="defs:mapping"/>
  </message>

</definitions>
```

B.5.10 Allocator BPEL

```
<process name="AllocatorProcess"
 ...>
  <bpel:extensions>
    <bpel:extension mustUnderstand="yes"
     namespace="..."/>
```

```
...
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
 namespace="urn:Allocator" location="allocator.wsdl"/>
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
 namespace="urn:AllocatorDefs" location="allocator_defs.wsdl"/>

<partnerLinks>
 ...
<variables>
  ...
  <variable name="dynamicMapperJobEPR"
            element="wsa:EndpointReference"/>
  <variable name="mapperReference"
            messageType="defs:referenceMessage"/>
  ...
<flow>
  <links>
  ...

  <sequence>
    <targets>
      <target linkName="ALLOCATOR.1-ALLOCATOR.2"/>
    </targets>
    <sources>
      <source linkName="ALLOCATOR.2-ALLOCATOR.3"/>
    </sources>
    <invoke name="ALLOCATOR.2"...>
      <catch faultName="factory:factoryError" faultVariable="reason"
       faultMessageType="defs:stringMessage">
        <flow>
          <reply name="ALLOCATOR.5"...>
    ...
    <assign>
      <copy>
        <from variable="mapperReference" part="reference"/>
        <to variable="dynamicMapperJobEPR"/>
      </copy>
      <copy>
        <from variable="dynamicMapperJobEPR"/>
        <to partnerLink="mapperJob"/>
      </copy>
    ...
...
```

B.5.11 Allocator PDD

```
<process name="allo:AllocatorProcess" location="allocator.bpel"
 xmlns="http://schemas.active-endpoints.com/pdd/2004/09/pdd.xsd"
 ...>

  <partnerLinks>
    <partnerLink name="allocatorJob">
      <myRole service="AllocatorService" binding="RPC"/>
    </partnerLink>

    <partnerLink name="factoryJob">
      <partnerRole endpointReference="static">
```

```
            <wsa:EndpointReference xmlns:factory="urn:Factory">
              <wsa:Address>factory:anyURI</wsa:Address>
              <wsa:ServiceName PortName="FactoryJob">
                factory:FactoryService
              </wsa:ServiceName>
            </wsa:EndpointReference>
          </partnerRole>
        </partnerLink>

        <partnerLink name="mapperJob">
          <partnerRole endpointReference="dynamic"
            invokeHandler="default:Address"/>
    ...

    <wsdlReferences>
      <wsdl namespace="urn:Allocator"
       location="allocator.wsdl"/>
      <wsdl namespace="urn:AllocatorDefs"
       location="allocator_defs.wsdl"/>
       ...
    </wsdlReferences>
  </process>
```

B.5.12 Allocator MINT Properties

```
# Service URL
target.url=http://localhost:8080/active−bpel/services/AllocatorService

# Service Timeout (msec, default 0 = no timeout)
service.timeout=15000

# Service Package
service.package.allocator=Allocator

# Fault Classes
class.allocator.job.translate.fault.allocatorError=AllocatorDefs.StringMessage
```

B.5.13 Allocator Translated MINT Scenarios

```
test(ALLOCATOR_SOC2_Nurse,
  sequence(
    send(allocator.job.translate,Mapping("Nurse","SOC2000")),
    read(allocator.job.translate,"3211"),OK))

test(ALLOCATOR_SIC_Nurse,
  sequence(
    send(allocator.job.translate,Mapping("Nurse","SIC92")),
    read(allocator.job.translate,"95.14"),OK))

test(ALLOCATOR_SOC2_Unknown,
  sequence(
    send(allocator.job.translate,Mapping("Sailor","SOC2000")),
    read(allocator.job.translate,allocatorError,"Unknown job"),OK))

test(ALLOCATOR_Unknown_Nurse,
  sequence(
    send(allocator.job.translate,Mapping("Nurse","Unknown")),
```

```
      offer(read(allocator.job.translate,″3211″),OK)))

test(ALLOCATOR_SOC2_Unknown_Scheme,
   sequence(
      send(allocator.job.translate,Mapping(″Sailor″,″SOC20000″)),
      read(allocator.job.translate,allocatorError,″Unknown scheme″),OK))
```