Imperial College London
University Of London
Department of Computing

# A Rigorous Approach To Engineering Web Service Compositions

Howard Foster

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering of the University of London, and for the Diploma of the Imperial College of London.

January 2006

# Abstract

Despite the emergence of standards to define and compose Web Services to form more complex systems, as yet, there is little support for engineering systems composed from multiple services. As web technology has evolved, tools have been developed that support the design of both visual content and functional services for users. Web Services however, concentrate on the view of systems inter-operating with other systems rather than that of actual human actors, yet the concepts related to ease of design are still highly relevant. However, as yet, tools which assist service design and composition provide only basic capabilities.

The main contribution of this work is to provide a rigorous approach to specifying, modelling, verifying and validating the behaviour of web service compositions with the goal of simplifying the task of designing coordinated distributed services and their interaction requirements. We address these issues through the use of rigorous software process analysis techniques. The thesis specifies semantics for web service composition standards and develops an accessible, mechanical tool, which automates the tasks involved.

This thesis presents a model-based approach built upon formal verification, validation and simulation techniques, utilising scenario-based design and implementations built in service composition standards. The work assigns the semantics of compositions through the use of Labelled Transition Systems (LTS) in the form of Finite State Processes (FSP). A tool suite is also presented, forming an environment to assist in undertaking the approach, and featuring an extendable and flexible architecture for the variety of compositional standards that exist. The approach is validated using a case study as a result of collaborative work with the UK Police Information Technology Organisation (PITO) and through example compositions published by the International Business Machines Corporation (IBM).

# Acknowledgements

This thesis is dedicated to my family, friends and colleagues whose support has proved invaluable to my research. Through the progress of undertaking this research, I have felt extremely fortunate to have contributed in part to the emerging use of standards and the application of formal modelling techniques to analyse software processes constructed in these standards. In particular, the many varied discussions I have had surrounding the issues associated with the validation and verification of processes for web service composition has continually boosted my endurance to pursue new ideas and here I mention those who have been part of this work.

In particular, I would like to thank my supervisor Jeff Magee for discussions and guidance, who has not only been a source for academic progression, but has also provided a friendship and understanding throughout my research. Additionally the support from Jeff Kramer and Sebastian Uchitel also proved highly complementary to working through ideas and considering possible solutions. Thanks are also due for Robert Chatley, who assisted in building early prototypes of the tool described in this thesis and continuing to update the core analyser and other plug-ins. I would also like to thank those beta testers of the tool (potentially unknown to them that they were!) for returning comments and highlighting where the use of the tool for the approach could be improved.

I will also fondly remember all those I met at the annual conferences on software engineering and web service research, and particularly, the growing emphasis of service composition and choreography reasoning playing an increasing part of their publications.

*"...the best test of truth is the power of the thought to get itself*
*accepted in the competition of the market."*
*(Oliver Wendel Homes Jnr, dissent. Abrams v. United States, 1919)*

---

Financial Support for this thesis has been provided in part by a series of IBM Eclipse Innovation Awards (2004, 2005) and with the assistance of the Department of Computing, Imperial College London.

# Table of Contents

**ABSTRACT** ..... **2**

**ACKNOWLEDGEMENTS** ..... **3**

**LIST OF FIGURES** ..... **8**

**GLOSSARY** ..... **12**

**CHAPTER 1 INTRODUCTION** ..... **13**

1.1 MOTIVATION ..... 13
1.2 THE APPROACH ..... 17
1.3 MOTIVATING EXAMPLE ..... 19
1.4 CONTRIBUTIONS ..... 21
1.5 THESIS OUTLINE ..... 22

**CHAPTER 2 BACKGROUND** ..... **24**

2.1 EVOLUTION OF THE COMPUTING NETWORK ..... 24
2.2 EVOLUTION OF DISTRIBUTED COMPUTING ..... 25
2.3 WEB SERVICES ARCHITECTURE ..... 27
2.4 WEB SERVICE BEHAVIOUR ..... 28
*2.4.1 The Problem Domain* ..... *30*
*2.4.2 Web Service Interfaces* ..... *30*
*2.4.3 Web Service Compositions* ..... *31*
*2.4.4 Web Service Choreography* ..... *33*
*2.4.5 The Service-Oriented Model (SOM)* ..... *34*
*2.4.6 Service Goals, Policies and Obligations* ..... *35*
*2.4.7 Goal-Oriented Requirements Engineering* ..... *37*
2.5 SOFTWARE PROCESS ANALYSIS ..... 38
*2.5.1 Software Process Models* ..... *39*
*2.5.2 π-calculus* ..... *40*
*2.5.3 Petri-Nets* ..... *41*
*2.5.4 Finite State Process* ..... *42*
*2.5.5 Comparison* ..... *42*
2.6 REVIEW OF RELATED WORK ..... 43
*2.6.1 Web Service Specifications* ..... *43*
*2.6.2 Modelling Web Service Compositions and Choreography* ..... *44*
*2.6.3 Verification and Behaviour Analysis* ..... *47*
*2.6.4 Tool Support and Case Studies* ..... *48*
*2.6.5 Summary of related work and our approach* ..... *49*
2.7 SUMMARY AND DISCUSSION ..... 50

**CHAPTER 3 WEB SERVICE SPECIFICATIONS ..... 52**

3.1 SPECIFYING WEB SERVICE COMPOSITIONS ..... 52
3.2 THE SCENARIO APPROACH ..... 54
*3.2.1 Basic Message Sequence Charts (bMSC)* ..... *55*
*3.2.2 High Level Message Sequence Charts (hMSC)* ..... *56*
3.3 MSCS, COMPOSITIONS AND CHOREOGRAPHY ..... 57
*3.3.1 Mapping MSCs elements to Web Service Composition Behaviour* ..... *57*
*3.3.2 Web Service Compositions as MSCs* ..... *57*
*3.3.3 Web Service Choreography as MSCs* ..... *60*
3.4 SYNTHESISING MSCS TO LABELED TRANSITION SYSTEMS ..... 61
3.5 SUMMARY AND DISCUSSION ..... 62

**CHAPTER 4 MODELLING WEB SERVICE COMPOSITIONS ..... 64**

4.1 MODELLING BPEL4WS PROCESSES ..... 64
*4.1.1 Overview of BPEL4WS* ..... *64*
*4.1.2 BPEL4WS Processes and Business Protocols* ..... *65*
*4.1.3 Private Process Structure* ..... *66*
*4.1.4 Mapping BPEL4WS Processes to FSP* ..... *68*
4.2 MAPPING PRIMITIVE ACTIVITIES ..... 69
*4.2.1 Label Abstraction of Web Service Interactions* ..... *69*
*4.2.2 Invoke, Receive, Reply* ..... *70*
*4.2.3 Wait and Empty* ..... *71*
*4.2.4 Terminate* ..... *72*
4.3 STRUCTURED ACTIVITIES ..... 72
*4.3.1 Sequences of Activities* ..... *73*
*4.3.2 Concurrent Activities* ..... *73*
*4.3.3 Linked Transitions* ..... *75*
4.4 GUARDED PROCESS ACTIVITIES ..... 76
*4.4.1 Variable Abstraction and Guards* ..... *76*
*4.4.2 Assign* ..... *78*
*4.4.3 While* ..... *78*
*4.4.4 Switch/Case* ..... *79*
*4.4.5 Pick/onMessage* ..... *82*
4.5 FAULT AND COMPENSATION HANDLERS ..... 84
*4.5.1 Modelling Fault Handling* ..... *84*
*4.5.2 Throw* ..... *84*
*4.5.3 Modelling Compensation Handling* ..... *85*
4.6 A COMPLETE EXAMPLE ..... 90
4.7 ASSUMPTIONS AND LIMITATIONS ..... 95
4.8 SUMMARY AND DISCUSSION ..... 97

**CHAPTER 5 MODELLING WEB SERVICE CHOREOGRAPHY ..... 98**

5.1 WEB SERVICE INTERACTIONS AND CHOREOGRAPHY ..... 98
5.2 MODELLING WEB SERVICE INTERACTIONS ..... 100
*5.2.1 Service Conversations ..... 100*
*5.2.2 Service Partners and Roles ..... 101*
*5.2.3 Linking Composition Interactions ..... 102*
*5.2.4 An Interaction Modelling Algorithm ..... 105*
5.3 BUILDING INTERACTION MODELS ..... 107
*5.3.1 Composition Process Interactions ..... 107*
*5.3.2 Building a Set of Processes ..... 108*
*5.3.3 Messaging Port Connector Models ..... 109*
5.4 SUMMARY AND DISCUSSION ..... 116

**CHAPTER 6 ANALYSIS FOR THE SERVICE-ORIENTED MODEL ..... 117**

6.1 ANALYSIS OF WEB SERVICE COMPOSITIONS AND CHOREOGRAPHY ..... 117
*6.1.1 Approach to Analysis of the SOM ..... 117*
*6.1.2 Techniques used in the Analysis ..... 119*
6.2 PREPARATION FOR ANALYSIS ..... 121
*6.2.1 Types of Preparation Activities ..... 121*
*6.2.2 Preparation for Composition Abstraction and Mappings ..... 122*
*6.2.3 Sample Scenario for Verification and Validation ..... 123*
6.3 REFINING COMPOSITION BEHAVIOUR MODELS ..... 124
*6.3.1 Reduction of Implementation Specific Activities ..... 124*
*6.3.2 Grouping Design and Implementation Activities between Models ..... 125*
*6.3.3 Building an Architecture Model for Analysis ..... 126*
6.4 ANALYSIS OF COMPOSITION BEHAVIOUR MODELS ..... 127
*6.4.1 Composition Design and Implementation Equivalence ..... 128*
*6.4.2 Compatibility of Service Composition Interactions ..... 135*
*6.4.3 Other Properties ..... 139*
6.5 VALIDATION ANALYSIS OF BEHAVIOUR MODELS ..... 143
*6.5.1 Composition Validation through Animation ..... 144*
6.6 SUMMARY AND DISCUSSION ..... 147

**CHAPTER 7 TOOL SUPPORT AND CASE STUDY ..... 148**

7.1 TOOL SUPPORT ..... 148
*7.1.1 Tool Architecture ..... 149*
*7.1.2 Initial Prototype as Plug-in for LTSA ..... 150*
*7.1.3 Migrating the tool to the Eclipse Environment ..... 153*
7.2 CASE STUDY: UK NATIONAL POLICE IT WEB SERVICE COMPOSITIONS ..... 155
*7.2.1 Introduction ..... 155*
*7.2.2 Scope ..... 155*

*7.2.3 Issues and Our Contribution* ... *156*
*7.2.4 Requirements* ... *157*
*7.2.5 Specification* ... *158*
*7.2.6 Implementation and Analysis* ... *162*
*7.2.7 Choreography* ... *170*
*7.2.8 Summary and Discussion* ... *175*

**CHAPTER 8 EVALUATION AND CONCLUSIONS ... 176**

8.1 EVALUATION OF APPROACH ... 176
*8.1.1 On Design Specifications* ... *177*
*8.1.2 On Modelling Implementations* ... *179*
*8.1.3 On Verification and Validation* ... *181*
*8.1.4 On Iteration* ... *182*
8.2 EVALUATION OF TOOL SUPPORT ... 183
*8.2.1 Ease of learning* ... *183*
*8.2.2 Early Payback* ... *184*
*8.2.3 Efficiency* ... *185*
*8.2.4 Incremental gain for incremental effort* ... *185*
*8.2.5 Orientation toward error detection* ... *185*
*8.2.6 Integrated use* ... *186*
*8.2.7 Focused Analysis* ... *186*
*8.2.8 Evolutionary Development* ... *187*
8.3 SUMMARY OF CONTRIBUTIONS ... 187
8.4 FUTURE WORK ... 188
8.5 CLOSING REMARKS ... 190

**BIBLIOGRAPHY ... 191**

**APPENDIX A – WS-* ... 202**

A.1 WEB SERVICE STANDARDS ... 202

**APPENDIX B – FSP SEMANTICS ... 203**

B.1 PROCESSES ... 203
B.2 COMPOSITE PROCESSES ... 203
B.3 COMMON OPERATORS ... 204
B.4 PROPERTIES ... 204

**APPENDIX C – BPEL4WS TO FSP ... 205**

C.1 PRIMITIVE ACTIVITIES ... 205
C.2 STRUCTURED ACTIVITIES ... 205
C.3 GUARDED ACTIVITIES ... 206
C.4 FAULT HANDLING ACTIVITIES ... 207

# List of Figures

Figure 1-1 Web Services, Compositions and Choreography in a Police Enquiry Collaboration ..... 15
Figure 1-2 An Approach to Rigorous Engineering of Web Service Compositions ..... 18
Figure 1-3 UK PITO Case Study – Police Enquiry Service Domain and Hierarchy ..... 20
Figure 1-4 Chapters and subtopics of thesis ..... 23
Figure 2-1 Evolution of Distributed Computing Architecture Styles ..... 27
Figure 2-2 Web Services Standards Stack ..... 28
Figure 2-3 Web Services and Software Process Elements ..... 29
Figure 2-4 WSDL Structure ..... 31
Figure 2-5. Elements and relationships of a Service Oriented Model ..... 35
Figure 2-6 Elements of SOM for Verification and Validation of Services ..... 36
Figure 2-7 Scenario of shared (printer) resource between server and client ..... 40
Figure 2-8 A Petri-Net example for a Simplified Alternating Bit process ..... 41
Figure 2-9 FSP and LTS of a sequential process composition ..... 42
Figure 3-1 Example service composition for a new loan request ..... 53
Figure 3-2 A Rich-Picture of viewpoints in a loan selection service composition ..... 54
Figure 3-3 Example scenario of a loan offer service composition ..... 55
Figure 3-4 Example bMSCs for scenarios in the loan selection service composition ..... 56
Figure 3-5. High Level MSC (hMSC) for Loan Selection Service composition ..... 57
Figure 3-6. Basic MSCs and Web Service Composition elements ..... 59
Figure 3-7 Basic MSC and Web Service Choreography elements forming a Collaboration Group ..... 61
Figure 3-8 Architectural Model LTS of Loan Selection Composition ..... 62
Figure 3-9 LTS for Loan Selection Service Process ..... 62
Figure 3-10 Elements of the approach discussed in chapter 3. ..... 63
Figure 4-1 Basic BPEL4WS Process Structure and Activity Groups ..... 66
Figure 4-2 Standard Transitional Attributes and Elements Tags of BPEL4WS Activities ..... 68
Figure 4-3. Basic Service Activity Labelling ..... 70
Figure 4-4 Invoke, Receive and Reply constructs and mapping to FSP ..... 71
Figure 4-5 Example mapping of terminate activity as LTS process ..... 72
Figure 4-6 Sequence construct and mapping to FSP ..... 73
Figure 4-7 Flow construct and mapping to FSP for concurrent activities ..... 74
Figure 4-8 Mapping link Semantics for part of a loan approval process ..... 75
Figure 4-9 The Variable form in BPEL4WS ..... 76
Figure 4-10 Read-write Models for BPEL4WS Variables. FSP (top), LTS (bottom) ..... 77
Figure 4-11 Assign construct and FSP mapping ..... 79
Figure 4-12 While construct and mapping to FSP ..... 80
Figure 4-13 Switch/Case construct and mapping to FSP ..... 81
Figure 4-14 FSP Code for Pick..OnMessage event model for an ATM Logon ..... 83
Figure 4-15 LTS of Fault Handler and normal execution activity scope ..... 86

Figure 4-16 LTS of Throw activity model and process synchronisation ..... 87
Figure 4-17 Compensation Handlers as inline (top-left) or scoped (top-right) and activity (bottom) ..... 88
Figure 4-18 Compensate (inline) choice of execution paths ..... 89
Figure 4-19 LTS of scoped compensation handler activities ..... 90
Figure 4-20 LoanApproval models (bottom) produced from Linked Receive Activity (top) ..... 92
Figure 4-21 LTS for models produced from InvokeAssessor Activity ..... 92
Figure 4-22 LTS for models produced from InvokeApprover Activity ..... 93
Figure 4-23 Assign activity to set reply message content ..... 94
Figure 4-24 LTS for models produced from Reply Message Assign Activity ..... 94
Figure 4-25 LTS for model produced by mapping of reply activity ..... 95
Figure 4-26 Architecture Model of Loan Approval Web Service Composition ..... 96
Figure 4-27. Elements of approach discussed in chapter 4 ..... 97
Figure 5-1 View of multiple service compositions interacting and choreography layer ..... 99
Figure 5-2. Composition Interaction Analysis Sub-Action Diagram ..... 101
Figure 5-3. Service Partners, PartnerLinks and Roles in Composition Linking ..... 103
Figure 5-4. PartnerLinkType, PartnerLink and Partner construct forms ..... 104
Figure 5-5. Flow-chart of algorithm for Modelling Composition Interactions ..... 106
Figure 5-6 Web Service Composition and Port Channels ..... 107
Figure 5-7. View of Multiple Web Service Compositions Interacting in a Police Enquiry Scenario ..... 108
Figure 5-8 Scenario and Diagram for a MarketPlace Service Composition ..... 109
Figure 5-9. FSP Code for Buyer and Seller Interactions with a MarketPlace Process ..... 110
Figure 5-10 Channels and Interaction Activities of Web Service Compositions ..... 110
Figure 5-11 LTS of Model for Request Only Port Connector ..... 111
Figure 5-12. LTS of Model for Synchronous Rendezvous Port Connector ..... 112
Figure 5-13. Mapping Activities Between Port Connector and BPEL4WS for ..... 113
Figure 5-14 FSP Code segments for mapping activities ..... 114
Figure 5-15 LTS for Partial Set of Interactions between Seller, Buyer and Marketplace Compositions ..... 115
Figure 5-16. Elements of the approach discussed in chapter 5. ..... 116
Figure 6-1 Approach to analysis of Service Specifications and Implementation Models ..... 118
Figure 6-2. Behaviour Refinement through Analysis and Abstraction ..... 122
Figure 6-3 Example scenario of activities in a Message Auditing Service Composition ..... 123
Figure 6-4. Composition Implementation Alphabet before Reduction ..... 124
Figure 6-5 FSP code for Refined Composition Architecture Model ..... 127
Figure 6-6 Approach for Verification Analysis of Composition Models ..... 128
Figure 6-7 bMSCs for scenarios in the echo-audit service composition ..... 130
Figure 6-8 hMSC of echo-audit service composition ..... 130
Figure 6-9 LTS model for MSC scenario Echo-Audit Composition ..... 131
Figure 6-10 BPEL4WS Process Structure for ..... 131
Figure 6-11 LTS model for BPEL4WS Provider Service Activities in Echo-Audit Composition ..... 132
Figure 6-12 LTS model for BPEL4WS Provider Service mapped to MSC activities ..... 133
Figure 6-13 FSP code for equivalence verification of BPEL4WS against MSC models ..... 134

Figure 6-14 Trace run example of trace equivalence of MSC and BPEL4WS models ..... 134
Figure 6-15 FSP code for equivalence verification of BPEL4WS against MSC models ..... 134
Figure 6-16 BPEL4WS Process Structures for Services in Echo-Audit Composition Example ..... 137
Figure 6-17 FSP code for Client-Provider port connector model ..... 137
Figure 6-18 Port Connectors for Services in Echo-Audit Composition Example ..... 138
Figure 6-19 FSP code for parallel composition of BPEL4WS services and port connectors ..... 138
Figure 6-20 Deadlock example of compatibility verification BPEL4WS and partnered services ..... 138
Figure 6-21 FSP code for safety property that a request to log a client is made ..... 140
Figure 6-22 LTS model of a violation of a safety property in the Provider Service Composition ..... 141
Figure 6-23 FSP code for progress property that a reply to a client is always made ..... 142
Figure 6-24 FSP code for equivalence verification of BPEL4WS against MSC models ..... 142
Figure 6-25 Approach for Validation Analysis of Composition Models ..... 144
Figure 6-26 A sample validation of a sequence using LTSA Animator function ..... 145
Figure 6-27 A sample validation of alternative paths using LTSA Animator function ..... 146
Figure 6-28 The alternative paths available using LTSA Animator function ..... 146
Figure 6-29. Elements of the approach discussed in chapter 6. ..... 147
Figure 7-1 LTSA-WS Tool Component Architecture ..... 149
Figure 7-2 LTSA-MSC: hMSC ..... 151
Figure 7-3 LTSA-MSC: bMSC ..... 151
Figure 7-4 LTSA-WS Interface and LTSA plug-in framework ..... 152
Figure 7-5 LTSA-WS: FSP ..... 152
Figure 7-6 LTSA-WS: Verification ..... 152
Figure 7-7 LTSA-WS: Validation and Animation ..... 153
Figure 7-8 Web Service Composition Development with LTSA-Eclipse ..... 154
Figure 7-9 PITO Web Services Architecture Scope ..... 156
Figure 7-10 A Pilot Project Scenario for Web Service Composition in PITO ..... 157
Figure 7-11 Initial specification for a PITO police enquiry web service composition ..... 158
Figure 7-12 Concurrent interactions introduced in to the PITO composition specification ..... 159
Figure 7-13 Partial scenario for Vehicle Enquiry reply and ANPR request constraint ..... 160
Figure 7-14 Partial specification scenario to constrain nominal enquiry with result of insurance enquiry ..... 160
Figure 7-15 Partial specification scenario to constrain nominal enquiry with result of vehicle enquiry ..... 161
Figure 7-16 hMSC for PITO Police Enquiry composition ..... 161
Figure 7-17 PITO Police Enquiry Basic BPEL4WS Process structure (interactions only) ..... 162
Figure 7-18 Partial PITO Police Enquiry Basic BPEL4WS Process with assignments ..... 163
Figure 7-19 Partial BPEL4WS Process sequence with assignments ..... 163
Figure 7-20 Graphical LTS view of Police Enquiry Composition with abstraction ..... 164
Figure 7-21 Results of trace equivalence test to check BPEL4WS partially fulfils MSC specification ..... 165
Figure 7-22 Results of trace equivalence test to check scenarios not covered by BPEL4WS composition ..... 166
Figure 7-23 Modified BPEL4WS Process to support FLOW of Vehicle Enquiry and Insurance Invocations ..... 167
Figure 7-24 Trace equivalence verification to check current vehicle enquiries in BPEL4WS composition ..... 167
Figure 7-25 BPEL4WS Process to support LINKED transitions of Vehicle Enquiry and Nominal Enquiry ..... 168

Figure 7-26 Final BPEL4WS process for verification ..... 169
Figure 7-27 Final BPEL4WS process verification against MSC specification ..... 169
Figure 7-28 A Pilot Project Scenario for Web Service Composition in PITO ..... 170
Figure 7-29 Overview of choreography of elaborated composition scenario ..... 171
Figure 7-30 Specification for scenario of Vehicle, ANPR and Authorisation Enquiries ..... 172
Figure 7-31 Police Enquiry composition in Choreography example ..... 172
Figure 7-32 Vehicle Enquiry composition in Choreography example ..... 173
Figure 7-33 ANPR Enquiry (Traffic Services) in Choreography example ..... 173
Figure 7-34 Port Connector model between Police Enquiry and Vehicle Enquiry compositions ..... 174
Figure 7-35 Deadlock example of compatibility verification BPEL4WS and partnered compositions ..... 174
Figure 7-36 Results of trace equivalence test to check BPEL4WS partially fulfils MSC specification ..... 175

# Glossary

The following glossary terms are taken from the Open Distributed Processing Reference Model (ISO 1995), and various Web Service specifications including; (Christensen, Curbera et al. 2001; Leymann 2001; Haas 2002; Schlimmer 2002; Christensen, Curbera et al. 2003; Gudgin and Hadley 2003; Booth, Haas et al. 2004; Gudgin and Hadley 2004). Traditional and more general descriptions are cross-referenced with relation to web services.

| Term Applied To Web Services | Definition |
|---|---|
| **Behaviour** | A web service's behaviour is defined by the set of activities behind that service and mapping those activities to message exchanges. |
| **Choreography** | Choreography describes the collective message exchange among interacting Web Services, providing a global, message-oriented view of the interactions (observing and controlling a many to many relationship). |
| **Composition** | A web service composition consists of an orchestration of web service interactions defined in a local process (itself potentially a service). Static web service compositions are those which use services known at design time and are bound to a composition at design time. Dynamic web service compositions those which define web service interactions where the services are not known at design time, and which are discovered or their properties resolved based upon a criteria process set at design time. |
| **Interface** | A service interface is the abstract boundary that a service exposes. It defines the types of messages and the message exchange patterns that are involved in interacting with the service, together with any conditions implied by those messages. A web service's interface describes the operations provided by that service and biographical information about where the service may be referenced (e.g. network address). |
| **Orchestration** | Describes the definition and the implementation of processes that drives the message exchanges between one or more web services. The BPEL4WS standard refers to participating services as composition Partners. Interaction is seen between one process and many services (i.e. one to many). |
| **Problem Domain** | The functional area of interest, or under control, by individual or groups of (web) services hosted on the internet and accessible either locally or globally (to other service groups) to fulfil a task or a series of tasks within the function area. |
| **Service** | A web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols. |

# Chapter 1

# Introduction

*"…the inherent complexity of a software system is related to the problem it is trying to solve. The actual complexity is related to the size and structure of the software system as built."* (Kevin Henney, 1999)

## 1.1 Motivation

Distributed software systems, and the interactions between components within these systems, can exhibit a high level of complexity and lead to difficulty in the assessment of what system behaviour is possible in multiple scenarios (Hogg and Huberman 1991). Constraining such a system requires us to fully understand the behaviour of the system and place controls on which sets of activities a system can perform. A distributed software system also encourages system evolution, by offering reusable services so that other systems may also include components from each other without reengineering solutions. Web Services (components interfaced using XML and standard internet protocols) are one such software architecture to exhibit this need for control, combining the flexibility and reach of the internet, the principles of reusability, with that of conventional distributed systems engineering practices.

The effect of using earlier distributed architecture styles has been prone to issues of semantic failure (where processes fail to achieve a goal due to transaction failures) and difficulties in providing the necessary distributed compensation handling sequences (Bukhres and Crawley 1996). There have also been difficulties attributed to the strict binding of compositions with specific technologies. Where previously designers of compositions had to work very closely with the developers of a technical solution, there is now a mechanism to support technology

independent software component invocation through the standards used in Web Service architectures. This provides opportunity for software designers to concentrate on the exact processes required from the services without hindrance from limitations of technical possibilities or great effort required to implement them. As web technology has evolved, the emphasis has been placed on providing ease of design and implementation, with the desirable "what-you-see-is-what-you-get - WYSIWYG" (Johnson, Roberts et al. 1989) now the normal rather than the exception for rapidly building web served applications. This is equally applicable to the domain of web services. Even though the web services concept focuses on a view of systems to systems rather than actual human actors, the concepts of ease of design and implementation for system interactions are highly related.

Web Services exhibit many similarities to traditional software components that amongst which resemble hosted objects which have a simple, well-defined interface, and that are designed with the expectation of reuse. In fact, the notions and ideas behind constructing reusable software components are highly applicable to web services, as they could simply be viewed as a type of software component architecture but with the addition of yielding a standard communication model. Some problems of component composition have been reported in (Fowler 2003) as:

- Identifying the appropriate components to implement the desired functionality
- Determining and resolving gaps between desired functionality and the component's functionality
- Specifying the component interactions

These issues are equally applicable to web service components. As web service deployment and use becomes more widespread, the notion of managing the composition of web services to integrate software processes together is being highlighted with concerns within research and the adopted standards (Yang and Papazoglou 2003). Web Service compositions focus on a group of services offering functional roles and activities to achieve a goal. An "orchestration" of services is assembled to achieve a collaborative effort between this group of services. The difference with web service compositions over that of traditional software component compositions is that web services compositions focus on the "autonomic open system"; in that they are designed to exhibit a service for varied client use and these can be reused without significant changes incurred to the design, potentially by any interested party.

They also exhibit some operational differences where run-time binding is loosely coupled assisting dynamic service invocation. If a web service or web service composition (a process interacting with one or more web services) is offered, the interests lie in who will use the service, how they will use the service and what they will expect to be invoked when requesting the service. These issues encourage us to consider how the analysis and verification of service offerings are incorporated into a service-oriented development approach, and specifically how prior to deployment, modelling and behaviour analysis can assist in determining the impact of solution given, and thus providing reassurance that the services are constrained appropriately. As an introduction to the domains covered in this thesis, the architecture of a web services solution with multiple compositions collaborating in a theoretical Police Enquiry services environment, illustrates how complex compositions could become (Figure 1-1).

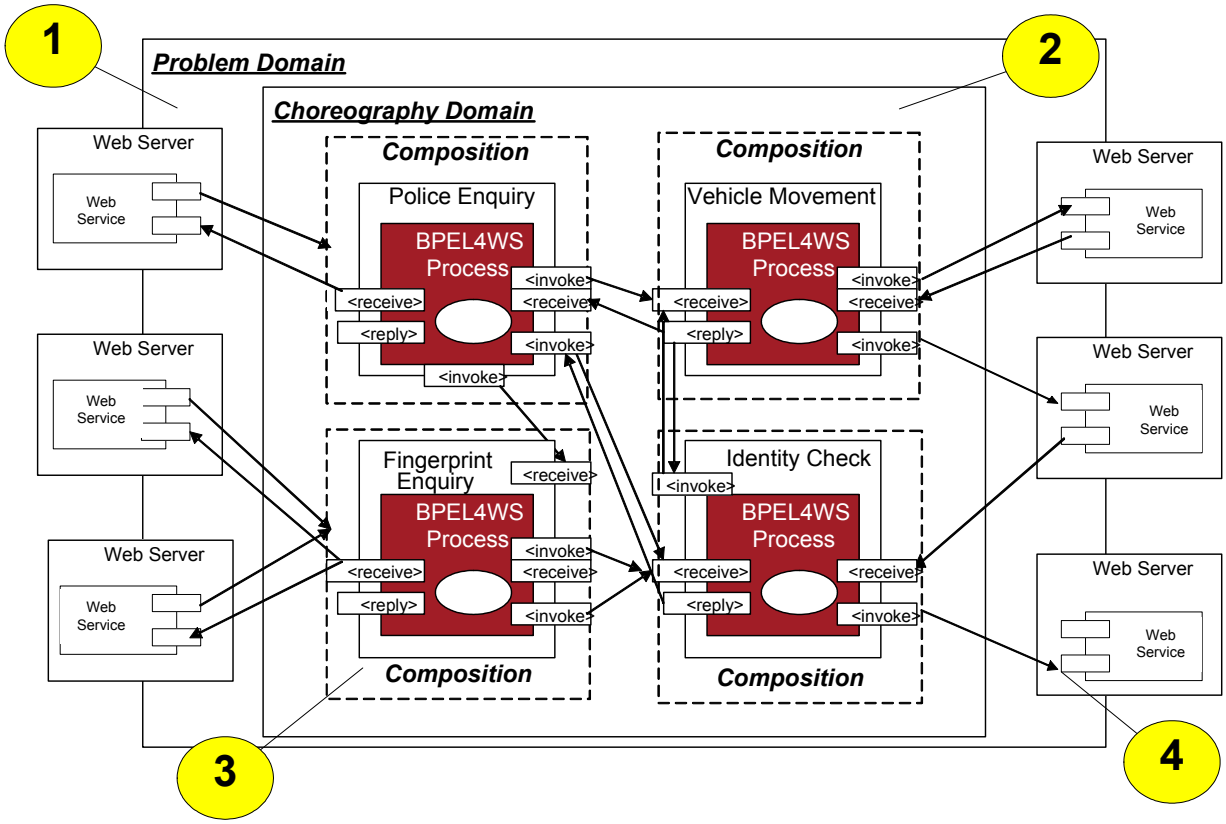



**Figure 1-1 Web Services, Compositions and Choreography in a Police Enquiry Collaboration**

This example, of four compositions collaborating to fulfil a set of police officer enquiry scenarios, illustrates how the layers of a services architecture firstly, *communicate* from basic invoke, request and reply actions (4), are *orchestrated* in a common process through standard

web service composition languages such as the Business Process Execution Language for Web Services (BPEL4WS) (3), and are kept consistent for transactional and process state purposes through Choreography (2). Brought together, these elements form a view of the problem domain (1) that we consider in the web services architecture.

An approach to service process modelling and behaviour analysis of these software architectures can be constructed to assist in providing a solution to some of the difficulties inherent in these types of architectures (Magee, Kramer et al. 1999). Whilst process behaviour analysis has been used for various themes of verification and validation, including analysing systems for deadlock, safety and progress properties (Holzmann 1997; Magee, Kramer et al. 1997), the core of these approaches can be seen in forming process models for analysis from a given specification or implementation. Aligned with these models, we can explore an approach of tasks for web service composition analysis that can be summarised using the analysis terms of:

- Verification (checking specified properties of the model, individually or against design specifications)
- Validation (simulation to validate requirements against design or implementation)
- Compatibility (that service compositions can cooperatively carry out shared tasks)

The motivation in collaborative web service engineering is similar to those efforts applied in UML for software process models (Jiang, Mair et al. 2003). The model may also be used in the design and implementation steps to derive service compositions from models of specifications. This promotes a collaborative distributed system design as specifications may be built locally and then compiled as composite processes. However, there is a need to explore how process modelling, and its formal background, can assist in these areas. The theme of this thesis therefore also considers the related areas of;

- Composition Architecture (building a model representation of service components)
- Role-based Decomposition (decomposing a model into one or more web service compositions based upon the roles and domains of service hosts and processes).

The motivation for this work also aligns closely with similar goals formed by the web service standards community on web service compositions and choreography, such as in the criteria of OASIS (OASIS 1993) and the W3C (W3C-Org 1994) focus groups, which provides greater confidence in the reasoning behind this area of work. At the time of writing this thesis, their work is under review but clearly indicates the need for early design verification as part of a standards based process for developing web service composition behaviour.

## 1.2 The Approach

Web service composition architectures aim to provide a technology independent means of integration, the ability to verify workflows is inherently not a technology issue but of state, behaviour and identity (Hruby 1998). A process has behaviour in the way in which it responds to a certain set of conditions, persistent state which is not visible to the client and persistent identity which is visible through explicit user defined operations (OMG-CORBA 2002). Web service compositions (as a set of service processes) equally hold these characteristics. An approach therefore is required to provide steps to analyse each of these characteristics, but without limitations imposed from the underlying technologies involved. The approach therefore considers analysis of a web service composition process from two viewpoints. Firstly, process model verification can be used to identify parts of the process behaviour that have been implemented incorrectly, or perhaps have unforeseen property results. Whilst there have been other attempts to use model-checking techniques for reliable web service verification (Nakajima 2002; Narayanan and Mcllraith 2002), they have concentrated on property specifications in domain specific language notations (e.g. Promela, the implementation language of SPIN). Our verification approach is from an abstract behavioural specification using the Message Sequence Chart (MSC) (ITU 1996) notation. The approach uses the UML (OMG 2002) style design of these sequences away from a technical implementation, and evaluates their transitional state and behaviour locally before deploying any parts of the workflow, and realizing the true effect of the process flow. The verification side of the approach aims to provide a mechanism to support such questions as; can the implementation fulfil the interaction requirements and did we build the *process interactions correctly*?

The second viewpoint is from that of validation. The focus of validation is clarifying the understanding of requirements against that of the web service composition implementation.

Some questions help us identify the validation areas that the approach can assist in with this. For example, has the implementer understood the needs of all expected clients, their intended use of the process and in all possible contexts? Ultimately, the result of validation is to ensure that the *right process was built*. Validation allows the designers and also prospective users of the process to step through the model and determine whether the design is fit for their requirements. Validation of web service composition specification models are a useful step prior to verification of implemented web service compositions, such that designers and users can evaluate a modal aimed at representing an equal view of their requirements. The approach for this is built from a series of steps undertaken by service designers, implementers (BPEL4WS engineers), those who deploy the services and clients (partners) of the service. Figure 1-2 illustrates the approach taken to verification and validation of web service compositions.

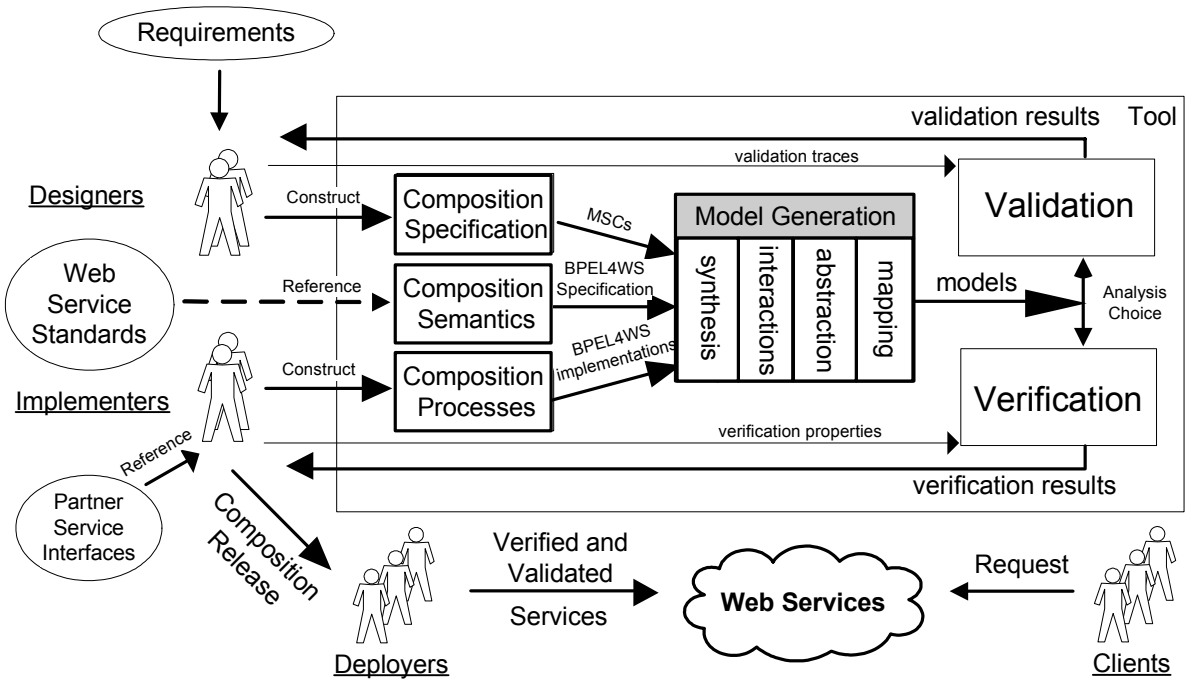



**Figure 1-2 An Approach to Rigorous Engineering of Web Service Compositions**

The approach is undertaken as follows; a designer, given a set of web service requirements, specifies a series of MSCs to describe how the services will be used and to model how each service requests or receives a reply in a series of service scenarios. The resulting set of scenarios is synthesized to generate a behavioural model, in the form of a state transition

system. The service implementation is undertaken by a BPEL4WS engineer, who builds the BPEL4WS process from either specification or requirements. The BPEL4WS specification is used to generate a second behavioural model (transition system) by a process of abstracting the BPEL4WS, with respect to data, to yield a model of interaction. Validation and verification consists of comparing and observing states of these two transition systems. The approach can assist in determining whether the implementation contains all the specified scenarios and whether any additional scenarios implied by the implementation are acceptable to the end-user. In addition, checks can be made on the models with respect to desirable general global properties such as absence of deadlock and liveness (using model-checking). Feedback to the user is in the form of UML style MSCs. The aim is to hide the underlying Labelled Transition System (LTS) representations and let the user view only the BPLE4WS implementations and the MSCs as a simple intuitive and visual formalism accessible to most engineers (Uchitel and Kramer 2001).

## 1.3 Motivating Example

Web Service compositions aim to fulfil the requirement of a standards based coordinated and collaborative service invocation specification to support multi-stake holder, multi-service application transactions (Hall 2003). This is seen as an important element of making the web services architecture viable for wide spread use, and to provide a closer representation of business transactions in cross-enterprise domains. One language which aims to consolidate previous efforts of specifying a composition language is the Business Process Language for Web Services (BPEL4WS) (Curbera, Goland et al. 2002). BPEL4WS is mostly the result of work undertaken previously by industry to build such specifications, such as that from the XLANG (Microsoft Corporation) specification and Web Service Flow Language (WSFL) by International Business Machines Corporation (IBM). Whilst BPEL4WS provides an *orchestration* of web service interactions at a local (single party) viewpoint, it does not provide a level of coordination between processes or manage long-running transactions (LRTs). This additional level of specification is encompassed in the term *Choreography*. Related specifications for this are still in an early stage of design, however, the Web Service Choreography Description Language (WS-CDL) (Kavantzas, Burdett et al. 2004) and Web Service Choreography Interface (WS-CI) (Cabrera, Copeland et al. 2002) are being marketed as a response to this requirement.

At the time of writing this thesis, BPEL4WS has been voted in as a standard service composition language by OASIS and has been positioned in a standards stack for clarification and consensus of where it is related with other emerging standards. The momentum of BPEL4WS is seen through case studies and industry BPEL engine implementations suggesting that BPEL4WS will be the standard of choice for coordinated web service composition implementations. Of the BPEL4WS examples used in this thesis, one is of the UK Police IT Organisation project (detailed in Chapter 7). Their UK national web services project is spread across heterogeneous systems including varied enquiry services (Finger Print, Vehicle etc) and centrally to the UK Police National Criminal Database (PNCD).

A series of web services, such as those scoped in the PITO project (Figure 1-3), clearly requires management and coordination. If this coordination is implemented as a process in BPEL4WS or other web service orchestration language, then the implementation needs to be constructed for a series of differing scenarios and verified and validated thoroughly for desirable execution paths in each. The use of web technology for services provides an example of how flexible distributed system computing has become. From a specification perspective, the focus is on appropriate service interaction, yet it is important to compose the web service workflow correctly for all service actors and more importantly, verify this flow before deployment is undertaken. This issue is particularly important in sensitive and critical system domains such as civil, emergency and other national infrastructure services.

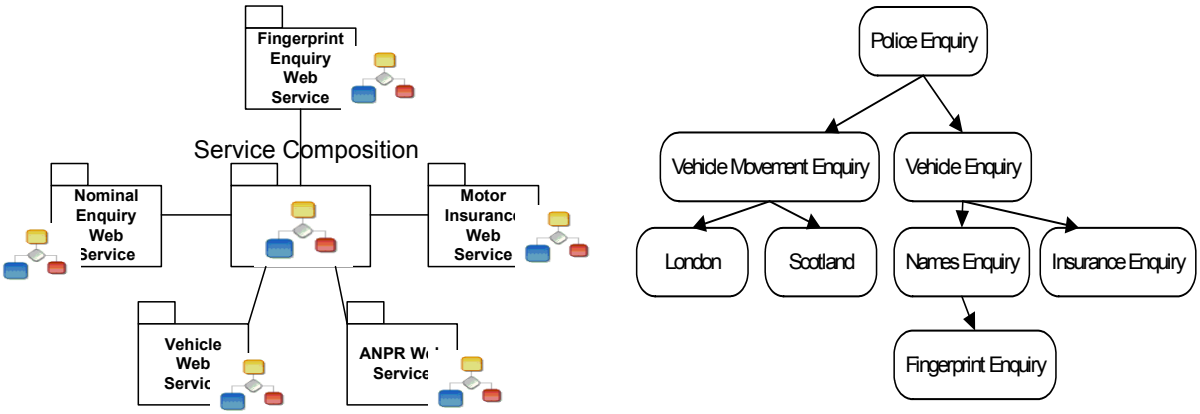



**Figure 1-3 UK PITO Case Study – Police Enquiry Service Domain and Hierarchy**

## 1.4 Contributions

To focus on the goals of undertaking this work we constructed the following statement. This statement was established at an early stage to guide the research to a common goal and to form the basis to address the areas discussed previously.

> **"*The main objective of this work is to provide a rigorous approach to specifying, modelling, verifying and validating the behaviour of web service compositions with the goal of simplifying the task of designing coordinated distributed services and their interaction requirements*. "**

This thesis aims to satisfy the objective. We address the issues in designing web service compositions by modelling the required processes in an accessible and concise notation which can then be used to verify, not only web service workflows but behaviour over cross-domain services. In this thesis an approach is presented which specifically addresses adding semantic representation to the BPEL4WS compositions and extends a tool to support a mechanical aid for verification and validation of these processes. The approach has been illustrated both from constructing workflows from a specification, modelling and transformation perspective, and by translating existing BPEL4WS processes to produce models for verification. Furthermore, by automating the process specified in this thesis, a framework can be added to the web service engineering life cycle to support modelling, verification, validation and implementation in the notation of choice. We have chosen BPEL4WS to use as an example, yet verification and validation of other web service workflow specifications may be undertaken using the same approach. The approach also provides scope to enhance the specification of BPEL4WS with availability, reliability and service performance consideration. With a medium such as the Internet, these are becoming increasingly important to verify before deployment of services is undertaken. It is our goal that the approach, combined with an extensive analysis tool will provide both academic and industrial areas with a rigorous tool for design and implementation analysis. Indeed, there are already references to this approach and the tool set being used for the education of BPEL4WS (Austin 2004) and as part of distributed system engineering lectures. Additionally, references have been made for industry community web sites focusing on BPEL (Foster 2004b), and in other service design and verification projects (Maghrabi 2004).

The work presented in this thesis is based upon, and extends, several papers and reports that have been published in the last three years (Foster, Uchitel et al. 2003a; Foster 2003b; Foster, Uchitel et al. 2004a; Foster, Uchitel et al. 2005; Foster, Uchitel et al. 2005). Work on tool support for the approach was also carried out as part of an IBM Eclipse Innovation Award (IBM 2004; IBM 2005). This thesis however, should be regarded as the definitive account of this work.

## 1.5 Thesis Outline

This thesis presents an approach which considers the issues of designing web service compositions (outlined in earlier sections) by providing a mechanism to support verification, validation and generally greater understanding of the behaviour of compositions created. In chapter 2 we describe a background to web services and the efforts in progress towards compositions and other standards to evolve web services for reliable and critical service configurations. Furthermore, a review of modelling software processes and other work in modelling web service compositions is discussed. Chapter 3 details building design specifications, in the form of MSCs for compositions and their choreography, and how interaction models are synthesised from these specifications. Chapter 4 provides a guide to modelling BPEL4WS compositions in the Finite State Process (FSP) notation. BPEL4WS semantics are described by way of FSP models and Labelled Transition Systems (LTS) used to illustrate these models graphically. Chapter 5 extends the approach to include models of service choreography with multiple interacting web service compositions, from the perspective of a collaborative distributed composition development environment. The process of behaviour analysis moves from a single local process to that of modelling and analysing the behaviour of multiple processes across composition domains. In chapter 6, the verification and validation steps are described, and utilising the models of design specifications (from chapter 3), modelling BPEL4WS processes (from chapter 4) and elaboration of these models for choreography (from chapter 5), examples are given to implement, translate and generate complete system models for checking with properties specified for verification. Furthermore, validation is illustrated in the form of animated and interactive process models represented back to designers and implementers of the process. Chapter 7 describes an example implementation of the approach steps in a tool, providing mechanical automation of the steps, and linking these steps with results back to designers, implementers and users of the process. Chapter 7 also considers how the approach has been

applied to a service development life cycle through a sample case study. The result of this case study is used in a discussion and evaluation on the findings and contributions contained within this thesis. Finally, Chapter 8 provides a conclusion on our work carried out, a view of anticipated future work and closing remarks. Overall consideration is also given how the contribution of this thesis aids in the broader areas of related research, and how it is believed further research will evolve and in which directions this may take. Figure 1-4 outlines the chapters and subtopics of this thesis.

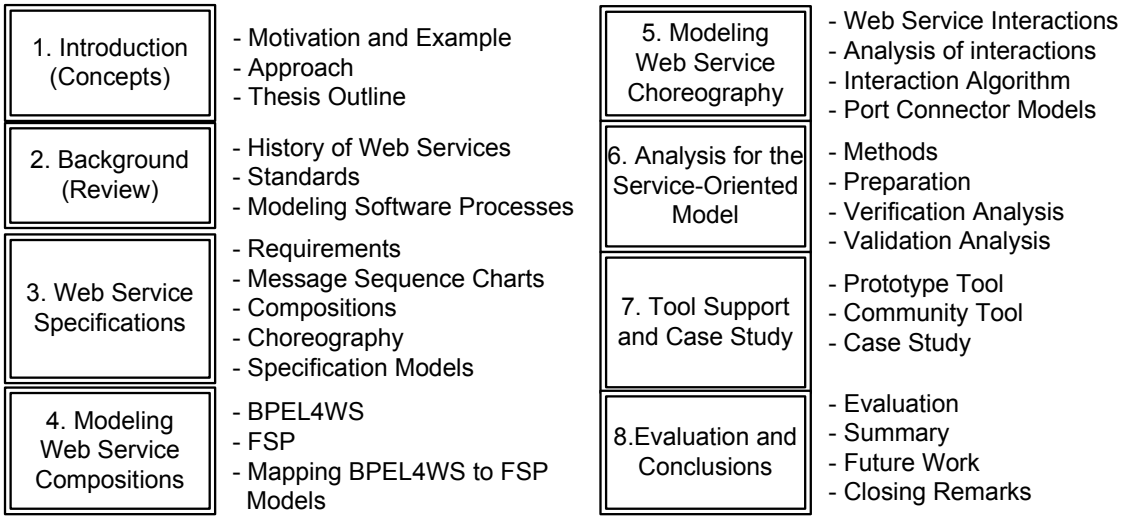



**Figure 1-4 Chapters and subtopics of thesis**

# Chapter 2

# Background

*"By examining the history of distributed computing, we can see how web services are a consequence of a natural evolution."* (Wrox 2003)

---

In this chapter, we describe the evolution of distributed computing and give an account of web services, their compositions, choreography and modelling distributed systems in general. We focus in more detail on the issues in what these systems are, how we analyse such systems and describe the areas of research that this work builds upon. Related work is also reviewed as part of a background to this topic area.

## 2.1 Evolution of the Computing Network

In the early 1990s, few had heard of Tim Berners-Lee's World Wide Web (Berners-Lee 2000), and, of those that had, many fewer appreciated its significance. However, since the 1970s computers had been increasingly connected to the Internet, and transferring specific data loads among computers was commonplace amongst Defence and Academic IT infrastructure (Leiner, Cerf et al. 2002). Yet the Web brought something really new: the perspective of viewing the whole Internet as a single information space, where users accessing information could move seamlessly and transparently from machine to machine by following related information links. A similar shift in perspective is currently underway, but the focus now is with application programs. Although distributed computing has been in use for as long as there have been computer networks (Roberts and Wessler 1970), it's only recently that applications that draw upon many interconnected machines as one vast

computing medium are being deployed on a large scale. The basis for making this possible are new standards for protocols of distributed computing built upon existing internet standards and that are designed for *programs interacting with programs*, rather than for *people interacting with browsers*. The move away from an environment where applications are deployed on individual machines or Web application server, is to a world where applications are composed of pieces (called *services)* and that are spread across many different machines, where the services are aimed at interacting seamlessly and transparently to produce an overall solution. While the consequences of this change could appear not dramatic, it's also possible that they could be as profound as the introduction of the Web. The computing industry is introducing new *Web service* frameworks that exploit this new architecture. Sun Microsystem's SUNOne (Sun 2001) and Microsoft's .NET (Microsoft 2001) are two such frameworks. In this work we concentrate on how this shift of focus highlights the need to have concrete methods to verify and validate solutions built for this new application architecture style. We begin to consider what aspects this involves by looking at the history of distributed computing and architecture styles.

## 2.2 Evolution of Distributed Computing

Distributed Computing become popular with the difficulties of centralised processing in mainframe use. With mainframe software architectures all components are within a central host computer. Users interact with the host through a terminal that captures keystrokes and sends that information to the host. In the last decade however, mainframes have found a new use as a server in distributed client/server architectures (Edelstein 1994). The original PC networks (which have largely superseded mainframes) were based on file sharing architectures, where the server transfers files from a shared location to a desktop environment. The requested user job is then run (including logic and data) in the desktop environment. File sharing architectures work well if shared usage is low, update contention is low, and the volume of data to be transferred is low. In the 1990s, PC LAN (local area network) computing changed because the capacity of the file sharing was strained as the number of online users grew and graphical user interfaces (GUIs) became popular (making mainframe and terminal displays appear out of date).

The next major step in distributed computing came with separation of software architecture into 2 or 3 tiers. With two tier client-server architectures, the GUI is usually located in the

user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients. Processing management is split between the user system interface environment and the database management server environment. The two tier client/server architecture is a good solution for locally distributed computing when work groups are defined as a dozen to 100 people interacting on a LAN simultaneously. However, when the number of users exceeds 100, performance begins to deteriorate and the architecture is also difficult to scale. The three tier architecture (also referred to as the multi-tier architecture) emerged to overcome the limitations of the two tier architecture. In the three tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, messaging middleware, or application servers. The middle tier can perform queuing, application execution, and database queries. For example, if the middle tier provides queuing, the client can deliver its request to the middle layer and disengage because the middle tier will access the data and return the answer to the client. In addition the middle layer adds scheduling and prioritization for work in progress. The three-tier client/server architecture has been shown to improve performance for groups with a large number of users (in the thousands) and improves flexibility when compared to the two tier approach.

Whilst three tier architectures proved successful at separating the logical design of systems, the complexity of collaborating interfaces was still relatively difficult due to technical dependencies between interconnecting processes. Standards for Remote Procedure Calls (RPC) were then used as an attempt to standardise interaction between processes. As an interface for software to use it is a set of rules for marshalling and un-marshalling parameters and results, a set of rules for encoding and decoding information transmitted between two processes; a few primitive operations to invoke an individual call, to return its results, and to cancel it; provides provision in the operating system and process structure to maintain and reference state that is shared by the participating processes. RPC requires a communications infrastructure to set up the path between the processes and provide a framework for naming and addressing. There are two models that provide the framework for using the tools. These are known as the computational model and the interaction model. The computational model describes how a program executes a procedure call when the procedure resides in a different process. The interaction model describes the activities that take place as the call progresses. A marshalling component and a encoding component are brought together by an Interface

Definition Language (IDL). An IDL program defines the signatures of RPC operations. The signature is the name of the operation, its input and output parameters, the results it returns and the exceptions it may be asked to handle. RPC has a definite model of a flow of control that passes from a calling process to a called process. The calling process is suspended while the call is in progress and is resumed when the procedure terminates. The procedure may, itself, call other procedures. These can be located anywhere in the systems participating in the application.

Figure 2-1 summarises the evolution of Distributed Computing architecture styles discussed. Web Services have repositioned the distributed computing architecture from further splitting the architecture into a domain of separate standards for remote procedure calls (RPC), interface definitions, and component technology, limiting the definitions and messages exchanges to be used only by that of XML specifications. These are marketed as a service-oriented, interaction based, architecture.

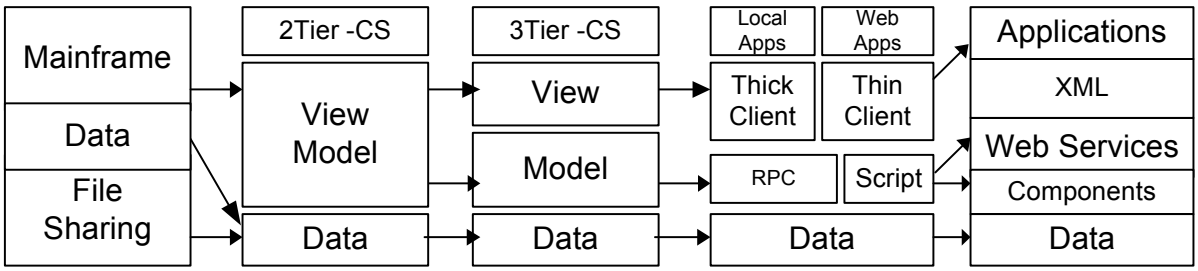



**Figure 2-1 Evolution of Distributed Computing Architecture Styles**

## 2.3 Web Services Architecture

A Web Service Architecture (WS-A) is a web component architecture that aims to address the service oriented requirements mentioned previously. A commonly used definition of a Web Service is taken from the W3C official Description Working Group specification (W3C-WS 2002) is; ***"A web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols."***

At a *conceptual* level, Web Services are pieces of functionality, hosted on internet enabled application servers that have the ability to be invoked for and respond to instructions carried

out in compliance with an XML messaging standard. The basis for web services is that they provide easily accessible interfaces to methods contained within application resources, and correspondingly identified uniquely on that hosted system by a uniform resource identifier (URI) (Berners-Lee, Fielding et al. 1998). Each resource offers one or more methods to be called upon by a message handler. The handlers receive messages as documents of instructions or parameters (depending on the web service messaging standard used), detailing which resource and data should be passed in order for that method to carry out its responsibilities. At a *physical* level, Web Services are built from a stack of emerging standards, some of which are specified in (Box, Ehnebuske et al. 2000; Arkin, Askary et al. 2002; Curbera, Goland et al. 2002; W3C-WSCI 2002; Christensen, Curbera et al. 2003). An example stack is illustrated in Figure 2-2. The layers cover descriptions for web service data, format, messages, interface, orchestration (compositions), transactions and coordination (choreography). Each W3C or OASIS standard is an implementation of an XML Schema for a set of permissible elements for defining each of the layers. As each standard is merely a "template" for describing web service implementations and their partners, an actual implementation engine is required in each case to support execution or interpretation of the XML implemented.

| | Layer | Standards (W3C/OASIS) |
|---|---|---|
| Compositions | Policy | WS-Policy, XACML |
| | Choreography | WS-CDL, WS-CI |
| | Transactions | WS-Transaction |
| | Orchestration | BPEL, WSCI |
| Services | Interface | WSDL |
| | Message | SOAP |
| | Format | XML Schemas |
| | Data | XML |

**Figure 2-2 Web Services Standards Stack**

## 2.4 Web Service Behaviour

Whilst Web Services themselves are components with a clearly defined interface (based upon the standards mentioned in 2.3), the architecture lacks verification and validation of process behaviour in the composition and coordination of these services to the requirements and behaviour of users (or clients) of these services. This is in a similar way to that of analysing the impact of customisation and personalisation of web sites, in which the main

problems associated with reusable web sites is that they are written without significant analysis of the potential use-cases and the needs of various clients (Bonett 2001). We describe what constitutes a web service's behaviour as; ***"A web service's behaviour is defined by the set of activities behind that service and mapping those activities to message exchanges."***

Web Service behaviour analysis consists of analysing two aspects of web service architecture style. The web service formally exhibits its identity and permissible interactions through definition in the Web Service Description Language (WSDL), which we describe in 2.4.2. However, within the implementation for a web service the behaviour of its interactions is defined. The coordination of a service's behaviour is formed from the basic operations of service invocation, replying to a service or receiving the reply from a service and this forms the basis for service analysis for its interaction behaviour. Standards elaborate the specification of how, what and when these interactions can occur. These standards can be aligned with that of software process analysis areas, as illustrated in Figure 2-3

.

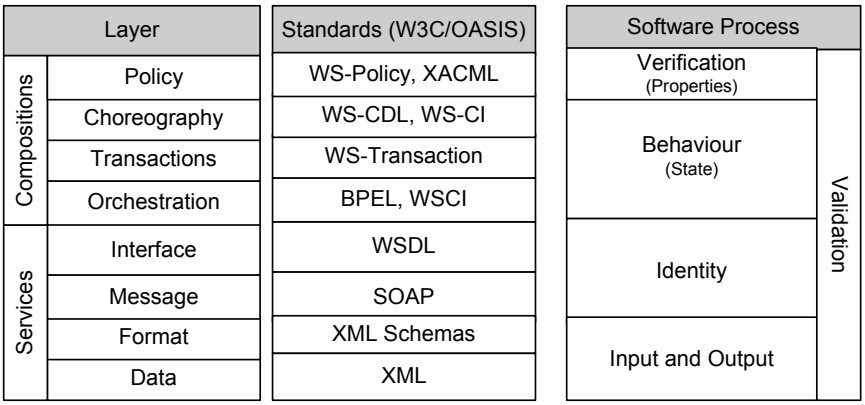



**Figure 2-3 Web Services and Software Process Elements**

In essence, to analyse web services we must consider what is defined by and behind the service interface, and consider those activities that comprise the process that the service offers. Web Service Behaviour Analysis can be described as; ***"Web Service Behaviour Analysis considers analysing the set of activities behind a service (a composition), and together with service interactions (choreography), provides an end-to-end view that models the role of each individual process in the choreography and the activities performed by each role."*** The theme of Web Service behaviour is used throughout our work, and is the

basis for understanding what may be observable from a web service implementation and a partnered composition process. The behaviour aspect of web services also encourages a view of the problem domain for the web services architecture.

### *2.4.1 The Problem Domain*

In Chapter 1, we scoped our work given a particular problem domain. An appropriate definition of problem domains for web service compositions needs to address both local and global service requirements. A definition of a conventional problem domain is given in (Subramanian 1993) as "A problem domain is given by a set of possible states of a physical world, and a set of actions that can be executed sequentially to change the state of that world**"**. As we are interested in describing problem domains with respect to web services to support a problem domain for one or many user requirements, our definition for a web service problem domain extends the conventional definition by specifically stating that the domain consists of services and that multiple interested groups are interested in or have control over these services. A web service problem domain can be therefore be described as; ***"A web service problem domain is the functional area of interest, or under control, by individual or groups of (web) services hosted on the internet and accessible either locally or globally (to other service groups) to fulfil a task or a series of tasks within the function area".***

### *2.4.2 Web Service Interfaces*

The web service interfacing standard is called the Web Services Description Language (WSDL). The W3C WSDL group defines WSDL as "*...an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.*" (Christensen, Curbera et al. 2001). WSDL is the interface description for any service that follows the Simple Object Access Protocol (SOAP) (Box, Ehnebuske et al. 2000) using HTTP GET/PUT or MIME standards. At the time of writing this work, the W3C Web Services Description Working Group has released two revised Working Draft specifications for WSDL 2.0. "*Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*" describes the Web Services Description Language (WSDL) Version 2.0, an XML language for describing Web services. It defines the core language which can be used to describe Web services based on an abstract model of what the service offers. It

also defines criteria for a conformant processor of this language. "*Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns"* (Gudgin, Lewis et al. 2004) defines patterns that are intended for use with WSDL. "WSDL message exchange patterns define the sequence and cardinality of abstract messages listed in an operation. Message exchange patterns also define which other nodes send messages to, and receive messages from, the service implementing the operation. WSDL patterns are described in terms of the WSDL component model, specifically the Label and Fault Reference components." The structure of a basic WSDL document is illustrated in Figure 2-4.

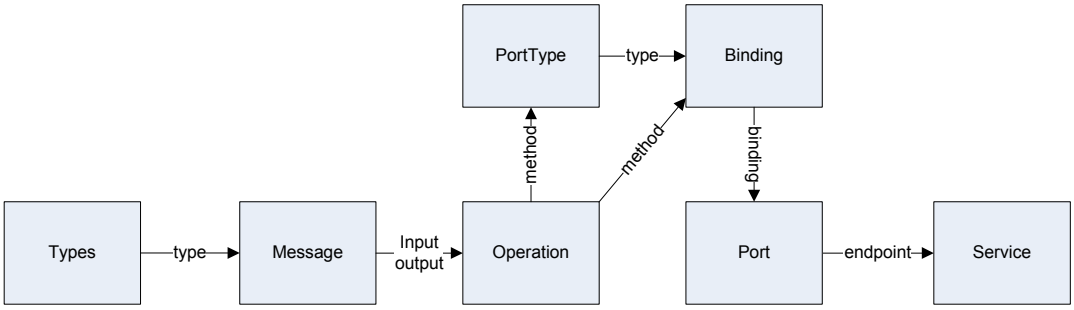



**Figure 2-4 WSDL Structure**

WSDL complements a service composition by providing an abstract process representation in terms of its interface. WSDL is broken down into seven parts, providing a declaration of an interface for message types, messages, operations (methods in traditional component terminology), port types and ports, and bindings. The topmost layer of a WSDL service description is the "service" element. WSDL is not limited to these basic elements and extensions can be specified by providing suitable element reference types to include in a document. The basic elements however, must be included as part of every WSDL document. WSDL builds on the SOAP specification for binding a service interface and operation to an actual endpoint service.

### *2.4.3 Web Service Compositions*

Web Service Compositions are formed from a singular problem domain which is "local" to problem domain owner. This has traditionally been a clearly marked boundary for a development project to work to. The difference in terms of *structured* and *anarchic* systems is described by (Chandy and Rifkin 1996). In structured system domains, the design proceeds from a specification, and there is a single entity that is ultimately responsible for

the design and implementation of the system. Anarchic systems on the other hand are collaborative application developments on the Internet which comprise many program units developed by different groups of people. For such distributed systems, no single agency assumes overall responsibility for reliability control. They define the difference in terms of ownership of the design implemented, and thus this ownership is either local or global as part of the deployment of the implementation. Our thoughts are towards this issue in relation to web service compositions, as compositions can be built locally yet aim to conform to global compositional constraints through the use of choreography and orchestration rules. This rule base is significantly involved in standard ways of both communication and domain understanding. There is currently industry work however, in having a global description language for local web service compositions. Fronting these language specifications is the Business Process Execution Language for Web Services (BPEL4WS) for process orchestration and handling fault tolerance or compensation actions. There are two themes of composition, being static and dynamic. Both static and dynamic web service compositions can collectively be described as; ***"A web service composition consists of orchestrated web services through a local process, itself potentially a service. Static web service compositions are known at design time and are bound to a composition at design time. Dynamic web service compositions are one or many compositions in which web services are not known at design time, and which are discovered or their properties resolved based upon a criteria process set at design time."***

Static web service compositions appear currently the most used web service composition style in both industry and academia (Haas 2002). They are formed by identifying manually (i.e. by human assessment) the applicability of a web service to a particular problem domain. The composition is therefore limited to the web services encompassed in the design. Static compositions are represented by known paths, known data representations and expected results as part of a formal and technical link with the web service. Dynamic web service compositions form the basis for discovery and flexibility in web service invocations. WSDL goes some way to detail the technical interface and to locate a given service (Christensen, Curbera et al. 2001), yet it does not identify what that service does, what function(s) it performs in order to fulfil the request, and neither does it suggest what level of service it will provide. Technically, dynamic interactions can be achieved through utility layers, such as the Web Services Invocation Framework (WSIF) (Duftler, Mukhi et al. 2001) which complements WSDL by providing the layer of invocation once discovering WSDL

documents. The aim is to be able to build a semi-automated service discovery and execution mechanism. If the scope of a web service composition is to include dynamic discovery and invocation, it must do so by considering how these services would be used in such an anarchic developed system and within a particular problem domain.

### *2.4.4 Web Service Choreography*

Whereas Web Service Compositions describe the local process of service orchestration, Web Service Choreography describes the observable interactions between services and their users. Choreography in general terms, and by dictionary definition, explores the wider aspects of interactions, often referenced by a similarity to arranging dance or ballet group sequences. The W3C Web Services Architecture group describe general choreography as "…***the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state."*** Whereas compositions are focused from a single service viewpoint and its interactions, choreography of a group of related services is the interaction to complete a broader scenario. Web Service Choreography is more formally described by the W3C Web Services Choreography Working group as; ***"….the external observable behaviour across multiple clients (which are generally Web Services but not exclusively so) in which external observable behaviour is defined as the presence or absence of messages that are exchanged between a Web Service and it's clients"*** (Austin 2004)**.** In a series of scenarios, the messages that flow between web services and clients (which may be web services, but also applications or even human beings) may require a specific set of interactions to occur and to group these interactions for transactional or compensational reasons. Choreography is typically initiated by an external source (a client or service) and ends with a target service or a reply to the source. Such interactions during this choreography poses questions such as; *can messages be sent and received in any order?, what are the rules governing the sequencing of messages?* And can a global view of the overall exchange of messages be drawn? *(i.e. can we verify, modify and monitor the behaviour)?*

As BPEL4WS is aimed at addressing the interactions from a local process workflow and fault tolerance perspective, it does not exhibit observable state by which other web service compositions can respond to for a global service responsibility. The element in question is that of impact of state in wider composition invocations. If the composition is invocated from a local "master" controller (as currently modelled by the BPEL4WS invocation engine)

state is not transferred between composition engines. The efforts necessary for effective workflow system analysis (Bolcer and Taylor 1998), the exploration of engineering these compositions is troublesome. In addition to state, the impact of introducing dynamic service selection in choreography increases the impact on complex engineering decisions at design time. For example, in a local problem domain, a dynamic discovery of an "Order Books Service" may be satisfied. Yet the same service hosted globally may not take into account local composition behaviour, or indeed, even synchronize with data managed by the local service. These are issues that should be recognized as a web service *design* compositional constraints and verified appropriately. Implementations for choreography standards are currently in the form of the Web Service Choreography Description Language (WS-CDL) (Kavantzas, Burdett et al. 2004) and the Web Service Choreography Interface (WSCI) (Arkin, Askary et al. 2002). Both of these specifications have been introduced as part of a service-oriented model aligned with the same W3C working groups.

### *2.4.5 The Service-Oriented Model (SOM)*

The Service-Oriented Model (SOM), illustrated in Figure 2-5, is a relationship model described by the W3C Web Services Architecture Group as created "***…to explicate the relationships between an agent and the services it provides and requests***" and that this is the base layer for building service architectures (Booth, Haas et al. 2004). The fundamental elements in the model are that of goal state (*states of some service or resource that is desirable from some person or organization's point of view)*, service (*an abstract resource that represents a capability of performing tasks*), task (*an action or combination of actions that is associated with a desired goal state*), role (*defines a set of related tasks carried out and identified by message properties*) and agents (*which are capable of and empowered to perform the actions associated with a service on behalf of its owner*). The model also serves to exhibit the relationships between elements, such as a service's relationship to provider, tasks, semantics and goal states. It also links these elements with web service choreography (described previously) and how web service *activities* are formed from and with the purpose of actions and state "goals".

The SOM model serves as a useful reference map in considering the elements of service oriented architecture and a rigorous approach to engineering them. Our focus and interests however, lay closely with the properties defined by *goal state* elements and its relationships to service, interface, semantics and choreography. Choreography is clearly a key part in the

service architecture model yet what is more interesting in this model is that a *goal* state is bridged between service task and action and does not have a direct link with choreography.

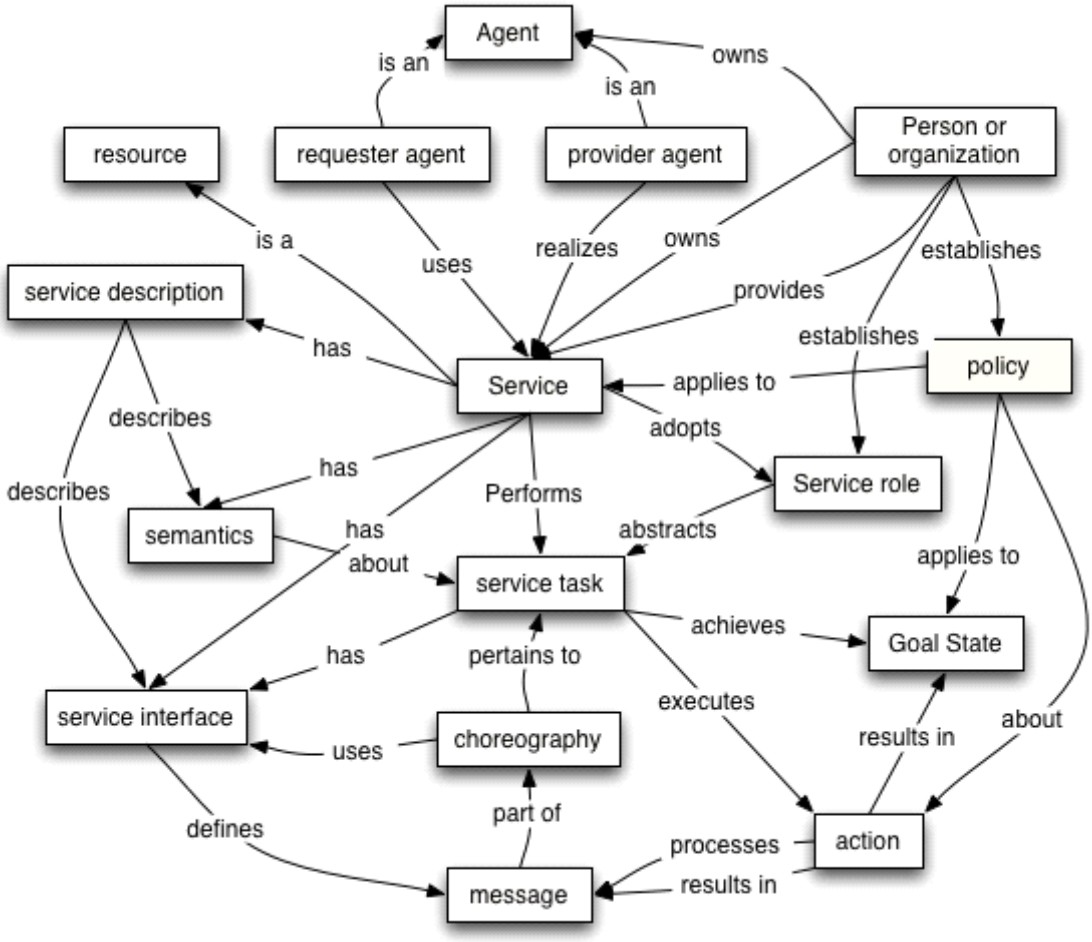



**Figure 2-5. Elements and relationships of a Service Oriented Model**

At a broader level this model poses questions such as does the architecture form a complete goal or a series of goals? Additional elements linked in the SOM include that of policy. A policy is a constraint on the behaviour of agents as they perform actions or access resources. The SOM policies are described as either *permissions* or *obligations*. We describe the service obligations at a later stage of our work.

### *2.4.6 Service Goals, Policies and Obligations*

A subset of the SOM related to our approach is illustrated in Figure 2-6. This subset is essentially built around a core set of *policies* which are monitored by agents in the architecture to support service security requirements and constrain the use of a service

through interaction rules, being pre-requisite and post-requisite actions to comply with a service's use.

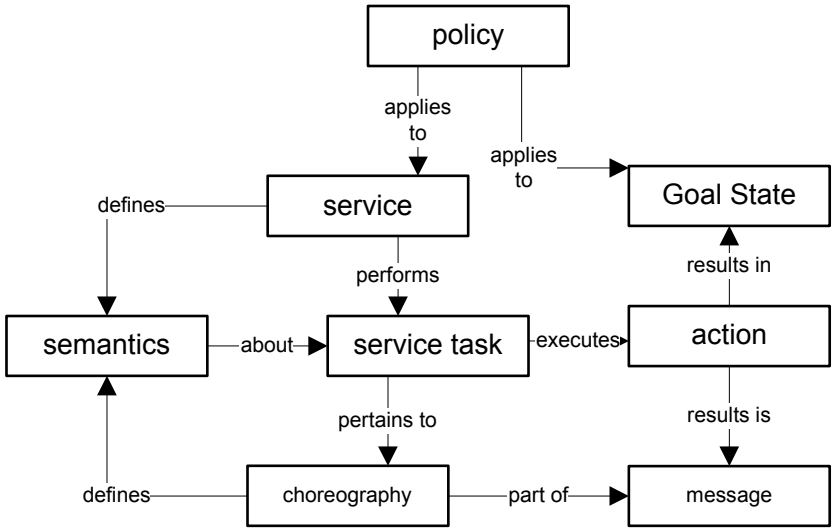



**Figure 2-6 Elements of SOM for Verification and Validation of Services**

A policy is applied to a goal state and a service. Policies are considered into two categories; that of service *permissions* and *obligations*. Permissions are a type of policy that prescribes the allowed actions and states of an agent or resource. A permission policy refers mainly to security aspects of a service model, as it is presented as more of a technical constraint rather than behavioural requirement. We therefore concentrate on the obligation category of policies as the focus of our analysis context. There appear differing definitions of an obligation in earlier work. In the PONDER language (Damianou, Dulay et al. 2001) an obligation is summarised as "*...the actions that must be performed by managers within the system when certain events occur and provide the ability to respond to changing circumstances",* whilst the W3C (WS-A SOM) defines obligations in terms of the goal states. For example, when a service provider (or agent in W3C terminology) has an *obligation* to perform some action, then it is required to perform that action. When the action is performed successfully, then the agent can be said to have *satisfied* its obligations. Not all obligations relate to actions. For example, an agent providing a service may have an obligation to maintain a certain state of readiness (quality of service policies are often expressed in terms of obligations). Such an obligation is typically not discharged by any one of the obligee's actions; although a triggered event (such as a certain time period expiring) may discharge the obligation either before (pre) or after (post) an activity in the service model. An obligation may continue to exist after its requirements have been met (for

example, an obligation to maintain a particular credit card balance), or it may be discharged by some action or event. The policy is established by implementers and results in goals and goal states from the actions in service compositions. Obligations in the services architecture may not be mandatory for every interaction scenario, yet through service analysis we are interested in the assurance that an obligation may be satisfied in a series of compositions and their choreography, if it is expected in a particular scenario. Obligations therefore become the basis for the general properties in our approach, implemented as either a safety or progress property type (which we elaborate upon in Chapter 6).

### *2.4.7 Goal-Oriented Requirements Engineering*

Goal-oriented requirements engineering has become an increasingly researched topic. There are differing definitions of goals and their related states (Lamsweerde 2001). Lamsweerde describes goals as the objectives of a system to be constructed. They are declarative statements that refer to intended properties to be assured of the system. Whilst the WS-A includes this broader definition in its standards based objectives, it currently describes goals broadly as those which complement non-functional policies (such as security, safety and other permission based roles of a policy). The goals we are particularly interested in for verification are those which are more functionality based (such as objectives of providing suitable service behaviour to its users). The W3C goal states are related to object states in these goal related works. An object state is something that an agent or other resource is interested in or exhibits; likewise a goal state is reached by the undertaking of a service's task and is monitored or reacted to by another service or agent. An example goal is that of a book ordering service, which may have a goal state in which a book has been purchased by a legitimate customer. It is important to establish the meaning of a goal state in the web services context and relate this to other goal-oriented requirements engineering definitions. The WS-A group elaborate on a goal (state) as "*a state of some service or resource that is desirable from some person or organization's point of view. Goal states are associated with tasks. Tasks are the unit of action associated with services that have a measurable meaning. Typically measured from the perspective of the owner of a service, a goal state is characterized by a predicate that is true of that state"*. In this way we build upon goal oriented requirements engineering principles by applying these concepts to web service goal verification. Our issue is to identify these goals, yet also assume a suitable representation for these goal properties to be used as input to an analysis process (Levi and Arsanjani 2002). We now consider expressing goal states and policy obligations of web service compositions,

seeking to describe the properties used in our approach after consideration of preparing web service composition models for verification and validation analysis. Formal software process analysis is a set of techniques we discuss for this analysis.

## 2.5 Software Process Analysis

So far, we have described a web service composition as that which is constructed from a series of references to web services, hosted either locally or by other parties. Within a service composition is a coordinated and orchestrated workflow of service calls, replies, decision points, and data manipulation. One goal of such compositions is to provide a process implementation language without the technical challenge of writing bespoke service connection handlers or providing differing interfaces to different users (Srivastava and Koehler 2003). With this independence however, comes an increased difficulty in concisely determining the behaviour of a web service composition as the focus is on interoperability rather than process semantics. There have also been varied efforts to leverage a Model-Driven Architecture (MDA) to constructing web service compositions, such as flow-chart specifications (Sherman, Shaffer et al. 2002), or by using the Unified Modelling Language (UML) (Hruby 1998; Gardner 2003; Iyengar 2003; Siegel 2003). With this work however, the process is modeled from a single composition, with interactions only visible from the local process perspective, and vagueness on how compositions over multiple processes may be assured. The assumption with the MDA approach to service compositions is also that the resulting implementation is assured to up-hold behaviour properties (being for example, that a reply will always be made to a partner of the service or that the result of a concurrent invocation of several services will all be synchronized once completed) (Soley 2003). This assumption however, does not provide formal evidence for a confident deployment of the composition in all scenarios. A more rigid approach is required, to assist in decomposition of system specifications where distributed compositions are organized on the basis of ownership over a particular domain of the system (Hall 2003). Basic synthesis of web service compositions from these models does not appear to provide a concrete behavioral representation that would give confidence to the implementers that once deployed, these processes are error free. A further series of steps in the web service engineering life cycle is desirable.

### *2.5.1 Software Process Models*

In (Osterweil 1987) it was asserted that “software processes are software too”, and thus could (and should) be developed, analyzed, and managed using the same software engineering methods and techniques that are applied to software. This idea implies there is a software service life-cycle that resembles the software life-cycle, involving analysis, design, implementation, and maintenance of software processes (Scacchi 2000). In contrast to software life cycle models, service process models represent a networked sequence of activities, objects, transformations, and events that embody strategies for service evolution (such as accommodating for varied interaction sequences). Service process models can be used to develop more precise and formalized descriptions of service life cycle activities. Their power emerges from their utilization of a sufficiently rich notation, syntax, or semantics, often suitable for computational processing.

Traditionally, software process models can be viewed as representing multiple interconnected task chains (Kling and Scacchi 1982; Garg and Scacchi 1989). Task chains represent a non-linear sequence of actions that structure and transform available computational objects (resources) into intermediate or finished products. Non-linearity implies that the sequence of actions may be non-deterministic, iterative, accommodate multiple/parallel alternatives, as well as partially ordered to account for incremental progress. Task actions in turn can be viewed as non-linear sequences of primitive actions which denote atomic units of computing work, such as a user's selection of a command or menu entry using a mouse or keyboard. Winograd and others have referred to these units of cooperative work between people and computers as "structured discourses of work" (Winograd and Flores 1986), while task chains have become popularized under the name of "workflow" (Bolcer and Taylor 1998). The main reason to use formal description techiques, or formal methods, is that they give us tools to analyze protocol models and other systems of interest, and find facts and errors in them, both at the specification level and at the implementation level.

One of the difficulties of using formalisms is perhaps in choosing the formalisms to use and the abstraction level of the model (Ross-Talbot 2004) (Aalst 2004), (Basten 1998). A few of the more commonly referenced software process modelling formalisms are in the form of Petri-Nets, π-calculus, and other process algebras (such as LOTOS (Bolognesi and Brinksma

1987), CCS (Milner 1980), CSP (Hoare 1985) and FSP (Magee and Kramer 1999)) . We briefly compare π-calculus, Petri-Nets and FSP notations as representative formalisms for modelling service compositions.

### *2.5.2 π-calculus*

The π-calculus is a mathematical model of processes whose interconnections change as they interact (Bergstra, Ponse et al. 2001). The basic computational step is the transfer of a communication link between two processes; the recipient can then use the link for further interaction with other parties. This makes π-calculus particularly suitable for representing where accessible resources vary over time. (Milner, Parrow et al. 1992) present an example of this through a simple resource sharing communication and transition with a server, client and a printer resource (Figure 2-7).

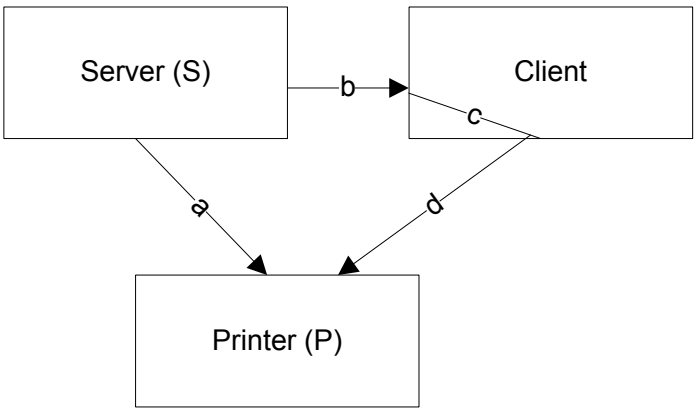



**Figure 2-7 Scenario of shared (printer) resource between server and client**

A server controls access to a printer and that a client wishes to use it. In an initial state only the server itself has access to the printer, represented by a communication link *a*. After an interaction with the client along some other link *b* this access to the printer has been transferred through the links *c* and *d*. In the π-calculus this is expressed as follows:

$$ba.S \mid b(c).cd.P \;\rightarrow\; S \mid ad.P$$

Where S is the server and P is the printer. The notation specifies "." for sequence and "|" for parallelism. The server that sends *a* along *b* is *ba.S*; the client that receives some link along *b* and the uses it to send data along it is *b(c).cd.P***.** In this example *a, b, c* and *d* are all just names which intuitively represent access rights. One constraint posed by this example model is that after the interaction occurs between server and client (providing access to the printer)

nothing else may access the printer. For this reason the π-calculus is also known as a calculus of "mobile" processes.

### *2.5.3 Petri-Nets*

Petri-Nets (Petri 1966) is a graphical directed net, and can be described in a formal mathematic language (Murata 1989), which is also suitable for modelling systems with concurrency. Petri nets provide a tool for describing systems that are characterized as being concurrent, asynchronous, distributed and nondeterministic. In graphical form, Petri Nets can be used as a visual communication aid in a similar way to that of structured design notations from traditional systems analysis and design methodologies. The language of Petri Nets however, provides a sold mathematical basis for the description and analysis of equations of state, algebraic and other mathematical models. This yields a practical notation for describing the behaviour of systems processes, such as that given for a simplified alternating bit (Paananen 1995) illustrated in Figure 2-8.

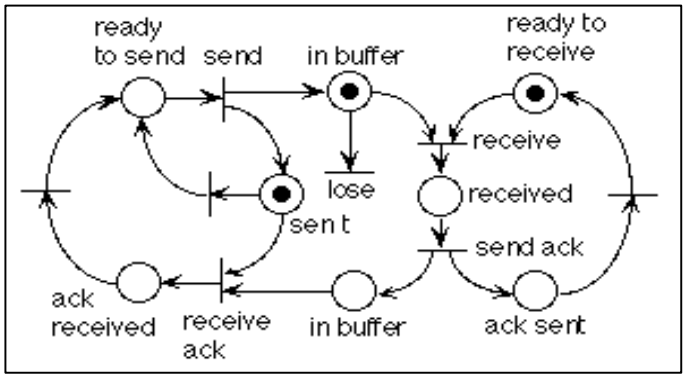



**Figure 2-8 A Petri-Net example for a Simplified Alternating Bit process**

Additionally, a planning-based role for Petri-net interaction models has now been suggested for commercial environments (Castilho, Kunzle et al. 2004). The rationale for plan-based models to describe software processes is associated with the reasoning that process enacting choices are based upon existing conditions within the current state of a process's execution (Huff and Lesser 1989). Having instantiated a project plan, role interaction nets can be used as a method of coordinating the routing of artefacts among interacting roles and as a method of tracking progress by the completion of interactions among roles. That is, this formalism can be used as an underpinning for coordinating activities in a process-driven environment.

### *2.5.4 Finite State Process*

The Finite State Process (FSP) notation (Magee, Kramer et al. 1997; Magee and Kramer 1999) is designed to be easily machine readable, and thus provides a preferred language to specify abstract workflows. FSP is a textual notation (technically a process calculus) for concisely describing and reasoning about concurrent programs. The constructed FSP can be used to model the exact transition of workflow processes through a modelling tool such as the Labelled Transition System Analyzer (LTSA) (Magee and Kramer 1999), which provides compilation of an FSP into a state machine and provides a resulting LTS. LTSA is a tool which provides a means to construct and analyse complex models of finite state process specifications. This tool, which is fully explained in (Magee and Kramer 1999), provides us with an opportunity to model workflows prior to implementation and deployment testing, and with an MSC editor and synthesis extensions (S.Uchitel and Kramer 2001) to easily model a scenario-based design specification, which can increase the expectation that process composition will provide the necessary path of invocation in all states specified (e.g. reliably by eliminating deadlock situations). With process animator extensions, the tool can also provide a facilitator in simulating workflow specifications. An example FSP for a parallel composition of concurrent invoke and receive transitions in a process is given in Figure 2-9. The composed Labelled Transition system for this system is also illustrated.

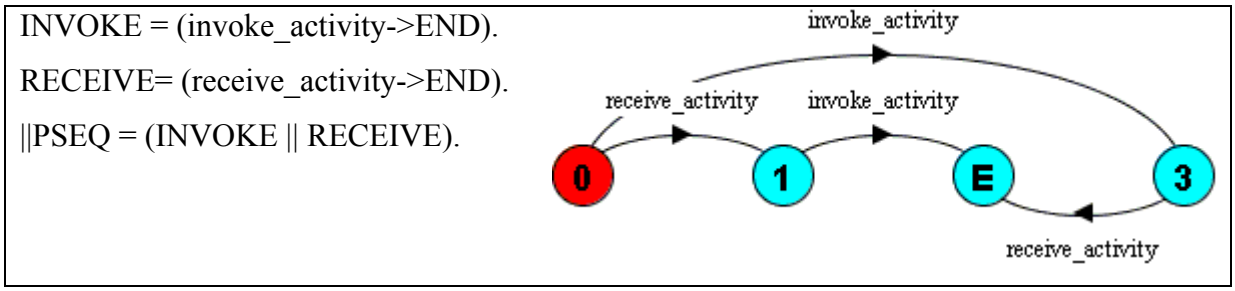



**Figure 2-9 FSP and LTS of a sequential process composition**

### *2.5.5 Comparison*

In (Ross-Talbot 2004) a comparison between several process algebras is given with a focus on the requirements for web service process modelling. The aim of this comparison is to provide criteria for modelling web service choreography (including state of resources etc) through a notation. In this article the author considers completeness (a complete set of semantics and maturity of notation), compositionality (providing process composition operators and modelling), and parallelism (a key aspect of formalism which must be fulfilled

for accurately modelling web service choreography). Our view is closely related to views of Ross-Talbot in that comparison; however, we distinguish our comparison by the additional need to have extensive tool support and experience available to support implementations of web service compositions in that formalism (away from any specific web service standard). We also do not place as strong an emphasis on resource modelling capabilities. Whilst this may be useful for future work, our initial requirements place value on completeness, compositionality, parallelism and extendable tool sets. As the FSP model is formalised, provides the necessary operators for composition behaviour (in similar set of operations to business workflow processes) and is supported by an easily extendable plug-in framework we therefore believe that FSP models provides a sufficient process formalism to support our requirements for this work.

## 2.6 Review of Related Work

Related work on defining specifications, composition implementations and verification and validation for web services has followed closely with the advancements of recent standards in describing such web compositions. Although specifications have emerged with little formal explanation of their underlying semantic representations (Aalst, Dumas et al. 2003), the research community has focused its attention on modelling web service processes and conversations. The methods of modelling fall into two categories, firstly using a design notation (such as features of UML) and to then generate a composition skeleton in the form of a BPEL4WS process and WSDL (interface) documents. Secondly, process algebras have been used to model the processes and interactions of service compositions and their choreography specifications. Here we describe the related work to ours, and where our work is positioned alongside these.

### *2.6.1 Web Service Specifications*

In (Papazoglou and Yang 2002) a general basis for service (component) specifications is stated as being that it describes all interfaces of a set of operations that are available to the service client for invocation. Although the service standards are rapidly changing and evolving, with respect to their interfaces and operations, there is some common ground on requirements-driven design specifications of web service compositions and equally some varied methods of building these process specifications. The common ground appears to be formally specifying the business requirements of the composition process required, such that

it captures the requirements for both the actors of the services and also the features of the web service interactions. In (Gardner 2003; Iyengar 2003; Mantell 2003) the authors describe an approach to specifying business processes through a subset of the UML profile (driven from a process *class* with attributes and methods). The behaviour of the interacting process classes is given using an activity graph. Whilst these works show partnered processes working together, it is unclear how multiple scenarios of each process would be specified In (S.J.Woodman, D.J.Palmer et al. 2004) however, the authors provide examples in UML Sequence Diagrams and Activity graphs, yet refer back to building requirements in a process algebra (in this case the π-calculus) to represent the concurrent and alternative paths possible in a composite web service specification. Additional work, acting as a bridge between model-driven based approaches (such as UML) and directly being specified in a process algebra, has been discussed in (Pistore, Roveri et al. 2004). The authors use an extended version of the TROPOS methodology, featuring a modelling framework proposed in (Yu 1997) to capture business requirements and then generate BPEL4WS source code from these requirements. In (Hamadi and Benatallah 2004; Yi and J.Kochut 2004; Yi and Krys.J.Kochut 2004) Petri net-based models are used to specify the semantics of web service specifications, their compositions and the communication between services. A direct mapping is mentioned between service and Petri-net yet no examples were given for this. With these works however, the key point that appears to be a disadvantage is that of alternative scenarios of the service specifications in design. For example, in each of these works it is possible to design abstract specifications yet they appear to represent single scenario processes only. A wider view of the *alternative* behaviours of the process must also be captured to gain greater assurance of a compositions use once deployed.

### *2.6.2 Modelling Web Service Compositions and Choreography*

The second set of research in the analysis of web service compositions has been undertaken by modelling these implementations directly in a process algebra. One of the earlier proposals for formal analysis of composition implementations was given in (Nakajima 2002). In this the author suggests that due to the nature of the software assets (the compositions in this case) being deployed to the internet, that the risk of a bug in such a composition impacts are much greater than that of conventional system deployments. The author of this work has also provided analysis of compositions in terms of those implemented in the Web Service Flow Language (WSFL) (Leymann 2001), which is one of a group of specifications that have been used to create BPEL4WS, and implements a

mapping between WSFL and Promela (the language of the SPIN tool) (Nakajima 2002). The work provides a useful reference point on mapping XML schemas (as web service specifications are defined in XML). Since the BPEL4WS specification has only recently become a standard (at the time of writing this thesis), one of the earlier works attempted on mapping BPEL4WS to a process calculus was reported in (Koshkina 2003) . The author introduces an extended process calculus, named "BPE-Calculus" which aims at concisely describing BPEL4WS processes in a notation similar to that of other process calculus, such as CCS (Milner 1980; Milner 1989). The calculus is then compiled into a Labelled Transition system (LTS). The authors of the BPEL-Calculus state that the disadvantages of other methods to model BPEL4WS are that their reported results are difficult to trace in the end tool. This point is useful to consider how our approach can cater for usability of results back to the end-user from that of the common mechanisms used by formal verification toolsets. In (Ferrara 2004; Salaun, Ferrara et al. 2004) web service specifications are described in the Language of Temporal Ordering Specifications (LOTOS). The authors extend the common mapping theme between the algebra and BPEL4WS by providing rules for a two-way process. They also confirm however, that due to the expressive and flexible structure of LOTOS, the mapping from LOTOS to BPEL4WS clearly does not preserve the structure of a process. We can learn from this that the abstraction necessary to perform modelling is not compatible in a two-way style, and that to perform this, additional resources (such as a resource map) would need to be included to "fill" the gaps between process algebra and implementation specification. Alternatively, (Hamadi and Benatallah 2004; Yi and J.Kochut 2004; Yi and Krys.J.Kochut 2004) use Petri net-based models to represent web service composition flows. In (Hamadi and Benatallah 2004) the work also defines a "web service algebra" (a grammar in BNF-like notation). However, there is a little coverage of how this maps to current standard web service composition languages (such as BPEL4WS or WS-CDL).

There has been some work on providing formal semantics for web service composition languages. In (Ankolekar, Burstein et al. 2002), the mark-up and semantics for DAML-S (another web service composition specification proposal) is described. They describe the notion of a "semantic web" as a series of Web resources that provide services, which effect some action or change in the world, such as the sale of a product or the control of a physical device. The semantic web should enable users to locate, select, employ, compose, and monitor Web-based services automatically. Whilst in (Duan, Bernstein et al. 2004)

BPEL4WS abstract processes (to describe the interface between processes) are analysed and semantics given on the construction of BPEL4WS implementations behind this. BPEL4WS and DAML-S are similar attempts at a standard for workflow of services, however, BPEL4WS focuses more on business web service orchestration whilst DAML-S is more generic in terms of any web based service or object (Seeley 2003). Additionally in (Woodman, Palmer et al. 2004) the authors present an extension to the WSDL specification (discussed in section 2.4.2), to describe the interactions between web services. This is then in turn mapped to π-calculus processes and sequencing formed using its operators. Tasks are represented as processes and dependencies linking the tasks, represented by channels (representing data dependencies in conditional linking). As BPEL4WS extends WSDL with an abstract process (which we describe further in Chapter 4), this mapping is aimed more at the choreography level (where the inner process of a service is not directly observed).

In terms of choreography and web service conversations, work on asynchronous web service communication has been described in (Fu, Bultan et al. 2004; Fu 2004d), with an example focus on the BPEL4WS specification reported in (Fu, Bultan et al. 2004b). A formal specification framework is described to analyse the conversations proposed by the asynchronous communication channels utilized on the internet. Interestingly, we shall show later in this thesis, that BPEL4WS provides a pseudo-asynchronous interaction model (whereby an invocation is sent, and then a separate receive activity formulates the link of call and reply). The technique proposed appears more useful for modelling general web service communication, rather than that of compositional specifics. Both the work on asynchronous and BPEL4WS interaction modelling is achieved through the use of Guarded Finite State Automata (GFSA) which enables data dependencies to be modeled alongside process transitions. In (Brogi, Canal et al. 2004) the authors describe an approach to formalizing conversations, by way of mapping the WSCI standard (as discussed in section 2.4.4) to CCS for web service choreography descriptions. The technique is similar to that of formalizing compositions by way of mapping each of the actions and data parameters between two or more partnered services in choreography. The conversation is traced by modelling the web service invocations with that of the receive and reply actions of the partnered service. The authors call for a common view of representing both composition and choreography models, such that fluid design and maintenance of individual specifications is not detrimental to the development effort. They additionally claim that other work in this area, including our work in (Foster, Uchitel et al. 2003a) does not provide support for channel adaptors (those which

link services interactions together). This was the case until we published our port connector work in (Foster, Uchitel et al. 2004a). Unfortunately, at the time of writing this thesis the WSCI specification work appears to have ceased, whilst the work on WS-CDL has continued with support from academia (Carbone, Honda et al. 2005; Carbone, Honda et al. 2005) in modelling the choreography specifications (in this case, in π-calculus).

### *2.6.3 Verification and Behaviour Analysis*

Compositional verification of concurrent systems has been explained in many articles including (Lynch and Tuttle 1987; Abadi and Lamport 1993; De-Leon and Grumberg 1993; Clarke, Grumberg et al. 1994a; Clarke, Grumberg et al. 1994b; Abadi and Lamport 1995; Hailpern and P.Santhanarn 2002). In most of these works a model of the behaviour exhibited by concurrent systems is created. However, the verification context is defined in different terms depending on the work reviewed. For example in (Hailpern and P.Santhanarn 2002) the context is very much on equivalence verification of implementations against that which is specified in requirements and their design specifications. The list of related work for verification of process models is extensive; therefore we concentrate on web service verification work in this thesis.

In (Koshkina 2003), which was also described previously in terms of modelling, the authors discuss verification of compositions for checking the existence of deadlocks, livelocks (i.e. a test of liveness) and sequencing constraints in service conversations. This is just one example of the general themes for properties used in web service composition and choreography verification. Some variations exist between web service verification works, largely focusing in the modelling notation used for mapping and the context at which the end-user is interested in verification of the service offerings. For example, for web service conversations, the approach to verification is to have two sets of conversation protocols (Fu, Bultan et al. 2004b). Alongside our work, the implementation protocol concentrates on BPEL4WS, whilst the design representation is different being modelled in a Guarded Finite State Automata (GFSA). The approach translates the BPEL4WS to GFSA and then both sets of GFSA to the Promela language. Intermediate analysis is performed on the BPEL4WS for synchronisation compatibility (a Cartesian product) on conversation appropriateness of interacting BPEL4WS processes. Whilst in (Fu 2004d) the same authors discuss a verification method of asynchronous communication for the broader web service conversations patterns (where there are long lasting, decoupled interactions between

services). The authors use data dependencies to check the possible values and alternative paths of execution by a form of message analysis (using the interface descriptions for the message document types being passed between partners). This is quite a different approach to other work which abstracted data dependencies out of the implementations and enumerate possible values that effect reasoning of conditional execution paths in a composition. In (Hamadi and Benatallah 2004) the analysis approach again consists of describing both the actual business process and equivalent specification. However, the defined web services algebra is used to specify both, and as such limits the use for practical purposes against actual standard implementations. In (Pistore, Roveri et al. 2004) verification is considered against a business requirements model, represented in the Tropos language.

Aside from deadlock checking, there have been some attempts to reason about the behaviour of web services in terms of compatibility (a sub-set of analysis of choreography specifications). In (Foster, Uchitel et al. 2004a) we described the synchronisation of interactions between partnered web service compositions using the notion of port connectors. We described an algorithm that constructed these port connectors and provided models which could be checked to detect if the necessary interactions had been specified to fulfil a certain property of the services requirements. Additionally in (Brogi, Canal et al. 2004) a similar reasoning is undertaken but for web service choreography specifications by considering the local and global choices (which relates to the local and global problem domains discussed in section 2.4.1) of web service choreographies in interactions. This work does not consider the broader choreography properties (such as verifying transaction, compensation or fault tolerance behaviour) of partnered services, and as such is therefore similar to those described which focus on interaction compatibility.

### *2.6.4 Tool Support and Case Studies*

Several tools have been reported as part of the formalising and modelling work discussed in previous sections. In (Fu, Bultan et al. 2004), a tool for analysing and verifying interactions among web services is supported. The mechanism is based upon conversation modelling (interactions of messages passing between processes) using both BPEL4WS and a formal conversation representation (in the form of GFSA) to describe verification properties to be analysed. The GFSA is then translated to the Promela language, which can then be verified in the SPIN tool (Holzmann 1997; Holzmann 2003). In (Koshkina 2003) the BPE-Calculus is incorporated into the Concurrency WorkBench (CWB) tool, a tool which is fully described

in (Cleaveland, Parrow et al. 1993; Stevens 1999). The CWB presents a new view to analyse the compositions specified the BPE-Calculus. We believe that the end-user (i.e. the web service engineer) would ideally require that the view of the verification tool or process algebra compilation is hidden from the development process, which encourages us to develop the verification toolset to support automated, background compilation of BPEL4WS processes and present findings back in the way in which design specifications were constructed (e.g. in MSC form).

To this date, our research in verification and validation of web service compositions has not yet encountered many real-world projects that have adopted the principles set out in these works, although we present one such case study in Chapter 7. There are some adoptions however; in the modelling of specifications whilst these specifications are being agreed (for example the W3C Choreography Working Group has adopted an adapted form of π-calculus to reason about WS-CDL descriptions). We believe that real-world examples and testing of these approaches will provide the necessary confidence in tools and support for undertaking the verification and validation techniques proposed.

### *2.6.5 Summary of related work and our approach*

As with some, but not all of the related work, our approach builds on two representations for the design and implementation of web service composition and provides a single framework for both verification and validation of the compositions. If the verification is performed using just one representation of the web service composition (i.e. either design specification or implementation) then there is naturally a limited offering of assurance in terms of if the behaviour was deployed without an equivalence check. A preferred approach is to offer both validation and verification at any time in the engineering of the service compositions, such that sufficient analysis results are determined by the nature of the changes in either design or implementation. We believe that both of these brought together, provides the level with which to assure compositions prior to deployment.

Our choice of MSCs for the design specification of web service interactions aims at providing a sufficient notation to describe the synchronised interactions between web service compositions. The implementation is assumed to be in BPEL4WS (due to it being a standard and with commitment from several leading industry organisations) and translated to an intermediate representation. The design representation is also synthesised to an intermediate

representation, on the basis of the behaviour modelling work described in (Uchitel and Kramer 2001). We believe this is easier for designers to construct the various scenarios of the system without learning a complex algebra or other process oriented language to specify this in. The design and implementation representations are then translated into the Finite State Process (FSP) notation to provide behaviour models for trace equivalence analysis, and to check further model properties by way of either safety or liveness queries. With a pair of behaviour models, verification can be performed on equivalence, and checked using safety or liveness properties. Validation can also be measured by animated or simulated process executions. In the LTSA tool, the FSP process algebra can be compiled into an LTS. With this further representation, other verification and validation modules can be reused. For example, in (Chatley, Kramer et al. 2003) the LTSA tool is extended to support a web-based interactive (step-by-step) process simulation.

## 2.7 Summary and Discussion

In this chapter we have described a background to web service architectures and how the principles of software process modelling can assist web service engineers to verify that web service compositions are an accurate implementation of specifications (given as a set of properties or obligations of interest). We have described how the Service-Oriented Model (SOM) specifies aspects of web services architecture that provide a framework for us to consider the verification and validation of web service composition scenarios. By application of policies between services and goal states (in the form of obligations), we have the necessary basic information to check these obligation properties of models in service compositions. There is a clear need to support engineering tasks for web service compositions, by applying semantic checking and simulated workflow animation to validate workflow semantics prior to deployment. BPEL4WS currently lacks the ability to test or simulate the correctness of workflow sequences. It is desirable that a composition framework should include checks that for example, a composition actually terminates and that the required execution path (end to end) is possible in all defined scenarios (Karamanolis, D.Giannakopoulou et al. 1999). These checks provide greater assurance before deploying one or many compositions in a local global domain which is connected to a wider, global, domain of partnered clients and services. To begin our approach to providing this assurance; the first consideration is focused on the need to specify the design of such compositions thoroughly and in a method easily accessible to non-system implementers. In

the next chapter, we discuss our approach to designing web service compositions using a scenario-based technique.

# Chapter 3

# Web Service Specifications

*"I don't paint things, I paint only the difference between things…"* (Matisse 1908)

---

It is an interesting concept that Henri Matisse approached his work not by centralising on objects in view, but on how these objects interacted with each other in the scene being depicted. A similar view can be taken with the services of a web services architecture. Through modelling their interactions from the perspective of multiple interaction scenarios, a complete view of components behaviour can be specified in different sequences. In this chapter, we describe how to model a series of interacting web service compositions using Message Sequence Chart (MSC) standards and a scenario-based approach to describing multiple use-cases for services.

## 3.1 Specifying Web Service Compositions

In section 2.4.3, it was discussed that a web service composition orchestrates a series of web service interactions, whilst also carrying out some internal functions for a service goal. Specifying these interactions means defining a local process (in similar way to building a system workflow) but limiting compositions to service interactions and without detailing the human interaction as part of wider system goals. As an illustration of a composition, we describe a scenario whereby a process for selecting the "lowest loan offer" is selected from

two loan providers. A credit rating is also obtained as part of an initial decision making step. Figure 3-1 illustrates this process and service composition. Three services are partnered with this composition, being; a credit rating service and two loan provider services.

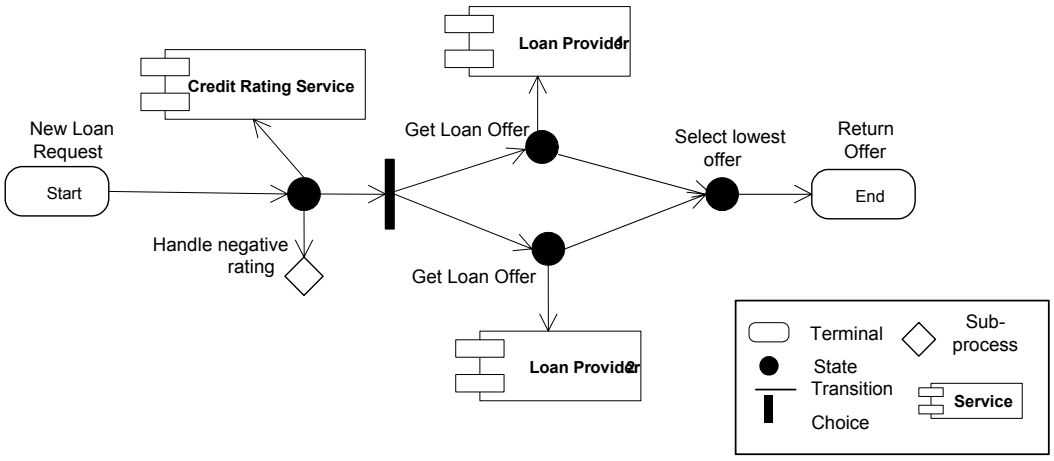



**Figure 3-1 Example service composition for a new loan request**

The example composition above is relatively simple to analyse by human observation, yet larger, more complex compositions, encourage an analysis of how different scenarios are catered for in the complete system model (Magee and Kramer 1997). For example, if a credit rating is obtained which does not provide a positive result then an alternative path of execution is undertaken to that of if a positive result is obtained. The implications of designing a process given such situations encourages a designer to explore such requirements from a wider perspective, possibly using techniques such as the Soft Systems Methodology (SSM) (Checkland 1982; Checkland 1990). Using SSM, a "rich picture" is built to consider all aspects of Clients (beneficiaries of the system), Actors (those who "play" out a series of scenes), Transformations (actions of Actors, transformations and process that take place), World-view (influences from within and outside the environment), Owners (those of own the domain of interest) and lastly the Environment (political, legal, technical and other aspects of the process domain). A Rich Picture for the loan service example, illustrated in Figure 3-2, highlights some of these aspects of the wider impact of a composition implementation. These aspects are difficult to represent in a single process design diagram. Consider each of the viewpoints of the actors in the service composition (such as client or provider) and how the behaviour exhibited by the process. Emerging from this picture is a set of scenarios, or "outlines or models of an expected or supposed sequence of events" from the perspective of each actor. It is worthy to note that whilst previous compositional work (using traditional

distributed component techniques) has also considered sequencing of components in this way – web service compositions emphasise a more global nature of exposure through standard techniques in discovery, interoperability and interfacing. We now consider how the use of a scenario approach aids elaboration towards a rigorous method of specifying Web Service composition models.

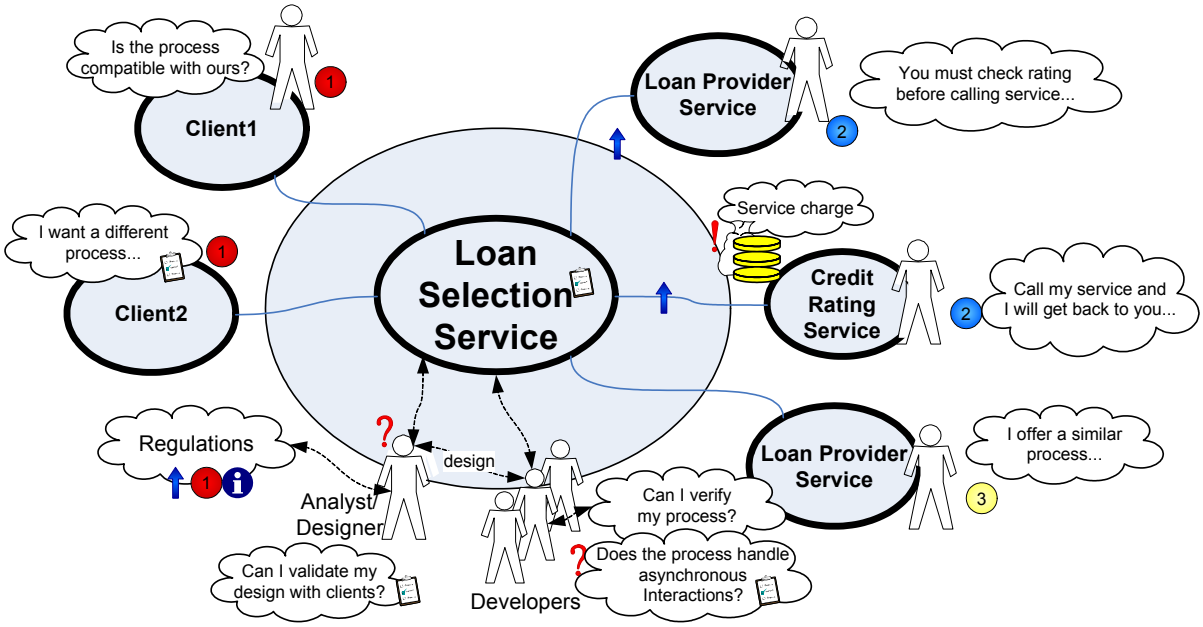



**Figure 3-2 A Rich-Picture of viewpoints in a loan selection service composition**

## 3.2 The Scenario Approach

The scenario based design approach has been a popular technique to capture user requirements by way of story telling (Jacobson, Rumbaugh et al. 1999). This method provides a concise yet simple tool for "*painting a picture*" of how actors (clients), components and messages are composed together to complete one or more system goals. It has commonly been used in the past for actual interaction by system users (Graubmann 2003), the actors can also represent any agent or service that interacts with the system being described by way of activities. For clarity, an example initial requirement of one textual representation of the Rich-Picture illustrated previously could be as described in Figure 3-3. Scenario based design has aids in the form of various system design modelling standards. The messages and their sequences that pass between components in a process can be described by way of a Message Sequence Chart (MSC) (ITU 1996) or similarly Sequence Charts in the Unified Modelling Language (UML) (OMG 2002). MSCs are part of building a set of scenarios of partial system behaviour (Uchitel 2003). The ITU MSC specification as

defined in (ITU 1996), forms MSCs as being part of two structures, that of Basic Message Sequence Charts (bMSCs) and High-Level Message Sequence Charts (hMSCs). Scenarios can be expressed in terms of positive scenarios (those messages which are expected) and negative scenarios (those messages which are not expected) in a composition. Here we use only the former for positive scenarios, to illustrate scenarios of web service compositions.

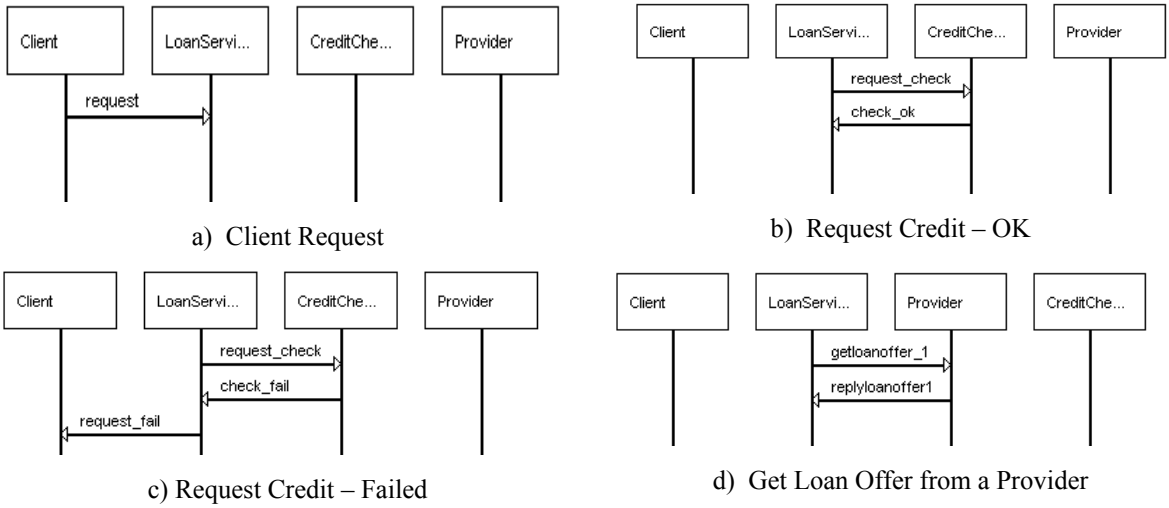
A loan selection service accepts a request for a new loan. The loan selection service first checks whether the credit rating of the client applying is suitable. If the rating is suitable, then the loan selection service contacts at least two loan providers. Each loan provider provides a rate and the loan selection service determines the lower rate of the two replies. Only when both loan provider replies are made, does the service calculate the lowest offer and then return this to the client.

**Figure 3-3 Example scenario of a loan offer service composition**

### *3.2.1 Basic Message Sequence Charts (bMSC)*

Basic MSCs (bMSCs) are useful for describing compositional activities, illustrating partial system behaviour by defining sequences of messages between components, and providing information of interaction state. Messages are passed between the "actors" (known as components in MSC terminology) of the objects that interact in the scenario arena. When diagramming a description of a concurrent or distributed system, we list the interactions between the system's environment and external systems, labelling the interactions scenario-by-scenario. Two terms are important here: interaction and interaction scenario. An *interaction* refers to a specific sequence of events that happens among participating entities. For example, one interaction might be a temperature gauge (an external system) sending a message to a system controller. The message would be one interaction. At more detailed levels, interactions involving internal components (subsystems) of the system are also included in the system description. An *interaction scenario*, on the other hand, details a specific group of interactions that form an episode (one of the parts into which a scenario is divided) and often stands for the possible event sequences in that episode. For example, a series of interaction scenarios would be the acceptable interactions between a pressure gauge, controller, and a valve that make the valve open when the pressure is too high. Each interaction scenario is often classified as either desirable or undesirable. Ideally, the implemented system should meet all the desirable interaction scenarios and none of the

undesirable ones. Figure 3-4 illustrates a set of example scenarios for the loan selection service composition example.




a) Client Request

b) Request Credit – OK

c) Request Credit – Failed

d) Get Loan Offer from a Provider

**Figure 3-4 Example bMSCs for scenarios in the loan selection service composition**

### *3.2.2 High Level Message Sequence Charts (hMSC)*

The bMSC and similar UML sequence charts are sufficient to describe the *interactions* of Web Service Composition designs. In web service composition design, the scenario based approach provides a suitably detailed notation to establish a specification of the required interactions in a series of scenarios for a given problem domain. We can use this notation to provide an abstract description of what the required behaviour of the web service compositions should be and leverage this into the specification approach such that individual service interaction models may be synthesised. Prior to this synthesis however, a set of scenarios for a web service composition requires further sequence detail as to how the individual scenarios may be composed. A higher contextual level of interaction may be specified using the hMSc (high level Message Sequence Chart) technique.

A hMSC is a collection of nodes connected to each other through transitions. Each node is either a start symbol, an end symbol, a rectangle enclosing a reference to a bMSC or another hMSC, a hexagon enclosing a condition, a small empty circle denoting a connection point, or a parallel frame containing two or more hMSCs that act in parallel. To focus on the choreography aspects, components are not shown in an hMSC as they have no meaning to the high-level sequence. Conditions in an hMSC are global in the sense that they apply to all the entities and indicate a global system state. This technique links together the basic

(scenario) message sequence charts and prescribes the expected possible paths in execution of one or more compositions. Figure 3-5 depicts a high level message sequence chart for the scenario given earlier. Whilst this is a simple example, it illustrates how a series of bMSCs may form the possible paths of sequence through a composition of possible message sequence scenarios.

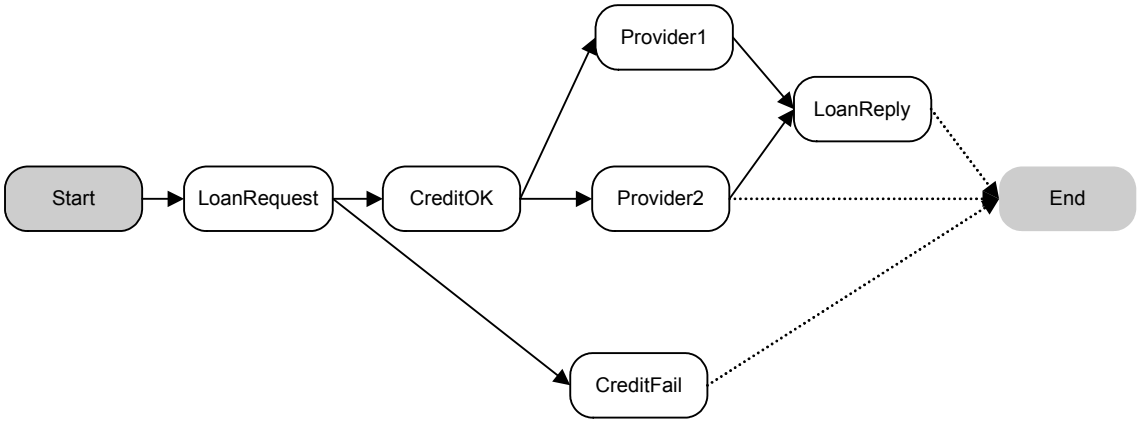



**Figure 3-5. High Level MSC (hMSC) for Loan Selection Service composition**

## 3.3 MSCs, Compositions and Choreography

### *3.3.1 Mapping MSCs elements to Web Service Composition Behaviour*

A web service composition design can be seen as a composed process consisting of various scenarios which when combined together, provides a complete set of sequence paths describing all possible paths through a service composition design. We relate the concepts of scenarios to web service compositions using a mapping between the elements of message sequence charts and those in building standards based web service composition. Each of the elements of the MSC (defined by ITU) is described in relation to elements of web service compositions and web service choreography (defined by OASIS for BPEL4WS and W3C for WS-CDL respectively). Table 3-1 lists each of the MSC elements as they relate to these two standards.

### *3.3.2 Web Service Compositions as MSCs*

A *Message Sequence Chart* describes the message flow between instances and partners of the composition. One message flow describes a partial behaviour of a system.

**Table 3-1. Web Service Compositions and Choreography as MSC Elements**

| MSC Element | ITU Definition | Web Service Composition | Web Service Choreography |
|---|---|---|---|
| hMSC | Describes high level sequence of partial behaviour (bMSCs). | Links several compositions or episodes together to form complete system behaviour. | Defines sequence of compositions and overall system behaviour in wider context. |
| bMSC | Describes a partial behaviour of a system between instances. | A Composition MSC is used to describe a single participant view of the overall message exchange. This provides a" skeleton template" for the composition as a workflow process, as seen from one participant. | A Choreography bMSC is used to describe a multi-participant view of the overall web service collaboration and message exchange. |
| Instance | Name blocks, processes or services of a system | *Partners* of composition, as seen from the local process perspective. Specific process name may also be included in the instance title. | *Partners* in choreography. Local Partner services or compositions are created upon a signified request. |
| Messages | A relation between an instance output and an instance input. message exchange can be split into two messages for input and output events. | *Communication* between composition "partners" or internal process activities. Messages are mapped to activities in the local composition. (e.g. service invocation, conditional branching). | *Communication* between all "partners" in choreography. Messages are related directly to an abstract process of web service calls between services and the compositions as a service. |
| Conditions | Restrict the traces that an MSC can take. Setting conditions describe global or non-global conditions. Guarding Conditions restrict behaviour depending on event values. | A web service composition may sequence its events to only be allowed if its current state permits this. The composition therefore imposes a guarded condition on the process activities. | A web service choreography may allow interaction only if a current state permits this. The composition therefore imposes a guarded condition on the partner interactions. |
| Gates and Environment | Gates represent interface between MSC and its environment. | All interactions in web service compositions are through standard web service calls. Thus, gates are another form of instance in this case. | Choreography tracks the sequence of messages involving multiple parties, where no one party truly "owns" the conversation |
| Instance creation | Creation of a instance in message sequence chart scenario | Creation of *s*ervice process or interaction. Creation may also represent the lifetime of a unique instance of a composition. | Instance creation of choreographed services is determined by the instance creation of local services or compositions |
| Instance stop | Termination of an instance in current scenario (by itself) | Termination of service process or end of interaction. Invocation may be asynchronous; therefore the partner process has control of termination. | Services are potentially long-running instances however, a service process has clearly defined start and end markers |

The message sequence chart is limited to describe only interactions and activities in a web service composition; therefore the chart can form a skeleton for the interaction constructs in a web service composition process generation. Figure 3-6 illustrates the mapping of message sequence chart elements to web service composition elements.

*Instances* may be blocks, processes or services of a system. In addition to service name, the specific instance process name may also be included. In a web service composition, instances are referred to as partners of the composition. Partners may be clients or other web services.

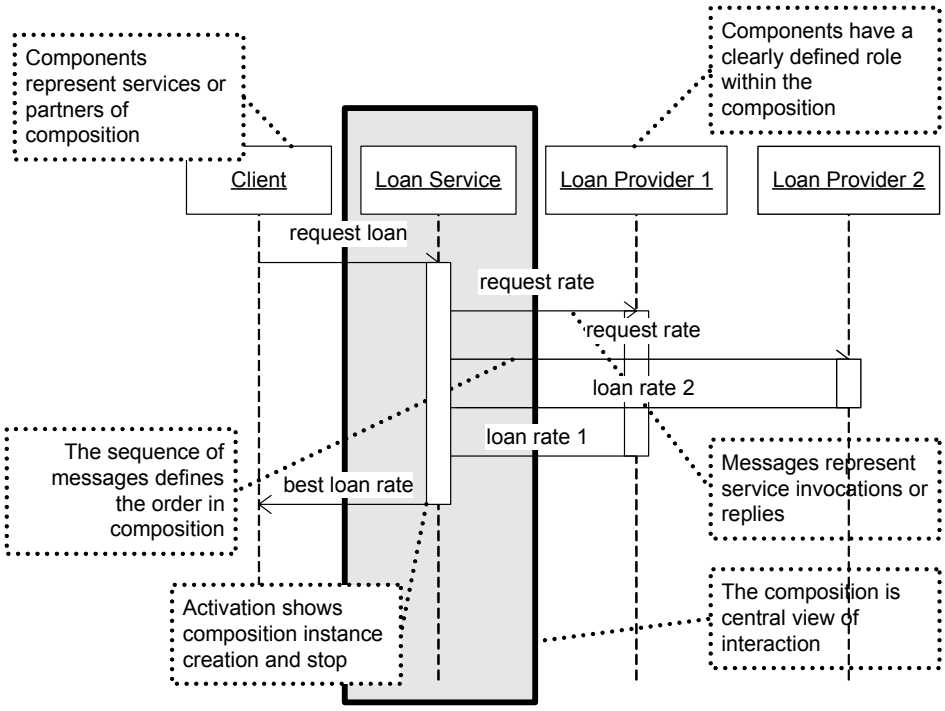



**Figure 3-6. Basic MSCs and Web Service Composition elements**

Partners can interact either by consuming service messages, in the form of SOAP operations, or by actively requesting within one or more compositions. *Messages* represent the relation between an instance output and an instance input. Additionally, MSCs can specify messages from the environment (via a gate) or be found. For web service compositions, a gate is assumed to always be a partner of the composition (a client or service). Additionally, messages may also represent events (timed) or other internal process. *Conditions* can be used to restrict the traces that an MSC can take, or the way in which they are composed into hMSCs. There are two types of condition: *setting* and *guarding* conditions. Setting conditions set or describe the current global system state (global condition), or some local state (conditions). In the latter case the condition may be local, i.e. attached to just one instance. Guarding conditions restrict the behaviour of an MSC by only allowing the execution of events in a certain part of the MSC depending on their values. This part, which is the scope of the condition, is either the complete MSC or the innermost operand of an inline expression or a branch of an hMSC. Web Service compositions can equally impose

conditions on sequence of interactions with partners, and locally in the flow of execution in its process. *Gate*s (points of interacting with the Environment) and *Environment* elements of Message Sequence Charts are not required in a mapping to web service compositions. Whilst they hold value for the environment specification in an MSC chart, standard links in a services environment is always undertaken by messages (i.e. interacting with other instances). The *Instance creation and stop* element of MScs represent that a service has a finite duration. Instance creation can be achieved by another instance. Whereas conventionally no messages or events may be made on an instance which has not been created, messages between service components may create an instance of that service composition. If not directly specified, it is assumed that an instance is created on activation of the entire scenario. Alternatively, an instance may terminate, although an instance may only terminate itself. It is assumed in web service compositions that services will be available (and thus created) prior to any scenario being carried out. However, a web service composition may listen for a creation message (to start a given scenario) which may indirectly create another instance of a composition process.

### *3.3.3 Web Service Choreography as MSCs*

In section 3.3.2, we described how message sequence charts can be used to describe web service compositions and the mapping of elements between them. Web Service choreography extends the use of using bMSCs in specifying the web service behaviour by considering how to elaborate for multiple web service compositions or services interacting in a wider enterprise scenario. The compositional view can still be used, however rather than concentrating on a single service the story of multiple services (or compositions) is described. Figure 3-7 illustrates how a bMSc may be used to describe elements of WS Choreography. As choreography describes interactions based upon multiple-viewpoints it is therefore also related to the using higher level message sequence charts to specify the sequence of interactions in various scenarios of Choreography interfaces Describing the collective flow of scenarios (specified in bMSCs) also forms part of the choreography design. Although designing web service choreography using MSCs is similar to building web service compositions, the subtle difference is that of coordination across multiple compositions and scenarios.

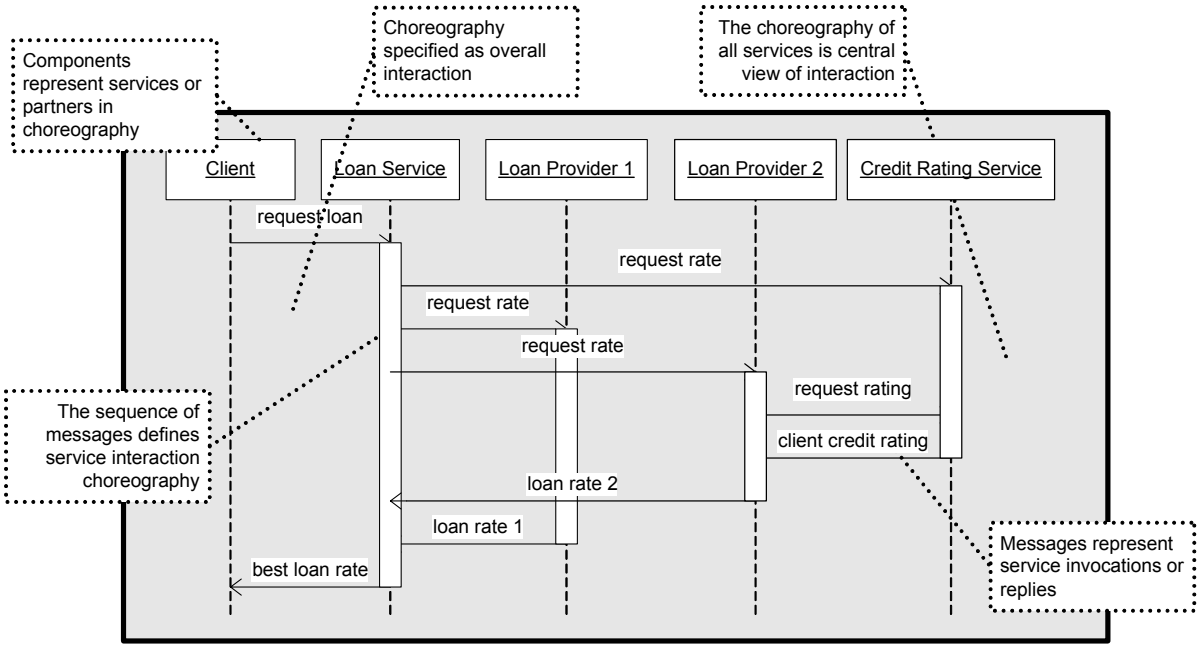



**Figure 3-7 Basic MSC and Web Service Choreography elements forming a Collaboration Group**

## 3.4 Synthesising MSCs to Labeled Transition Systems

MSCs are visual aids to design requirements specifications for web service compositions, yet their combined behaviour (as a set of partial stories in a complete composition behaviour model) is still difficult to analyse by human observation. The process of synthesising these MSC scenarios to Labelled Transition Systems (described in 2.5) provides a way to computationally and mechanically analyse these scenarios to determine whether the behaviour specified is desirable given a complete system behaviour model. A formal syntax and semantics for MSCs is described in (Uchitel and Kramer 2001; Uchitel 2003), whilst a corresponding algorithm to synthesise MSCs to a LTS based upon these definitions is described in (Uchitel, J.Magee et al. 2001).

In theoretical computer science, a state transition system is an abstract machine used in the study of computation. The machine consists of a set of states and transitions between states. State transition systems differ from finite state automata in several ways: In a state transition system the set of states is not necessarily finite, or even countable. In a state transition system the set of transitions is not necessarily finite, or even countable. In a state transition system, transitions do not form a function, but a relation between the states, and therefore, there may be zero or more than one transition out of a given state, with the same input. State transition systems with a finite number of states and transitions can be represented as

directed graphs. There are at least two basic types of state transition systems: "labelled" (or LTS for *labelled transition system*) or unlabelled. The net result of performing this synthesis of MSCs to an LTS is to "*help correct, elaborate and refine scenario-based specifications by way of experiments and replaying them…*". The algorithm we reuse in this work takes as input a series of MSC scenarios, and builds an LTS for each of the components involved. An intermediate step is performed, building finite state machine models in the FSP notation, which are in turn compiled to a architectural LTS model (Figure 3-8).

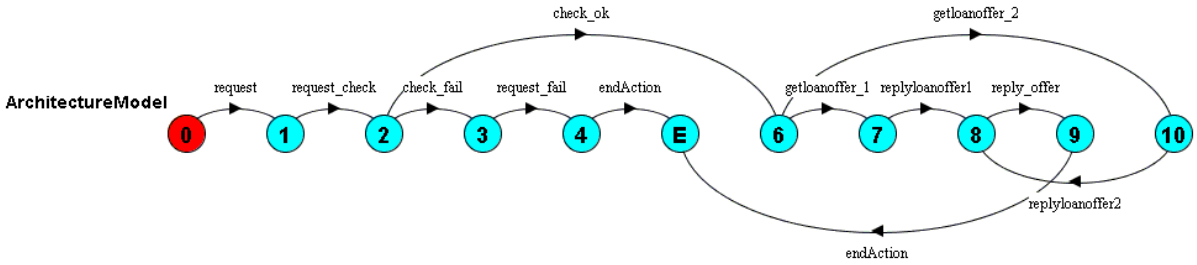



**Figure 3-8 Architectural Model LTS of Loan Selection Composition**

Represented as LTSs, the web service compositions built in the scenario approach appear quite different. The model produced in this synthesis provides a complete view of a system modal process, providing concise inspection of all possible paths and thus behaviour of the composition. Equally, from an abstract view, each of the actors in the process can be analysed for interaction. For example, to see the steps of activities that the Loan Service process exhibits we can compile only the Loan Service into an LTS and similarly analyse.

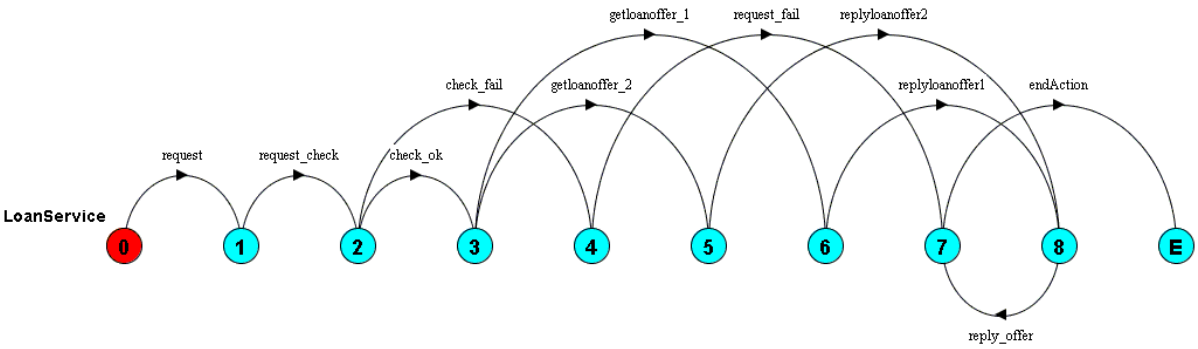



**Figure 3-9 LTS for Loan Selection Service Process**

## 3.5 Summary and Discussion

In this chapter we have described the notion and practical steps to designing web service compositions and choreography by means of using a scenario based approach. The

technique uses Message Sequence Charts to describe the interactions both within and outside the local process environment, depending on the viewpoint taken. The synthesis of these Message Sequence Chart scenarios provides a model of the complete system interaction design. Initial analysis can be performed on the design to ensure that different scenarios are catered for, but also to provide a sufficient analysis specification as input to verification against a process implementation. We consider the implementation of these design specifications in the following chapter. In terms of the approach discussed in section 1.2, we have introduced the highlighted parts as illustrated in Figure 3-10.

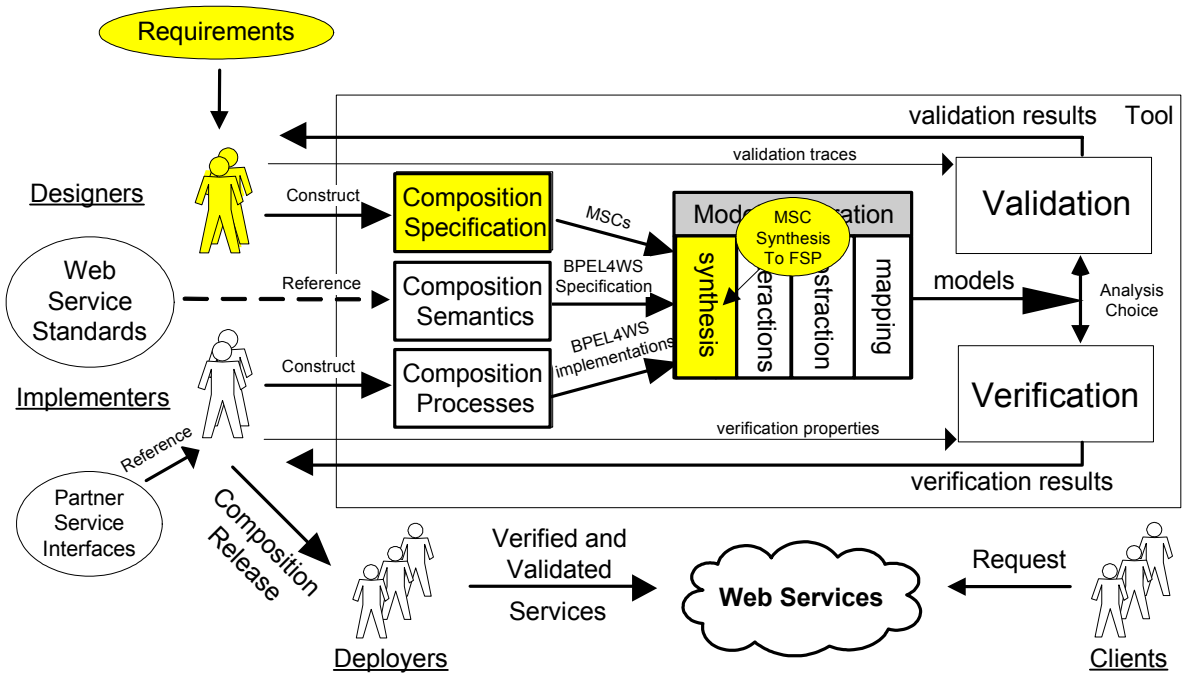



**Figure 3-10 Elements of the approach discussed in chapter 3.**

# Chapter 4

# Modelling Web Service Compositions

*"Nothing goes by luck in composition. It allows of no tricks. The best you can write, will be the best you are."* (Henry D. Thoreau, 1817-1862)

---

In Chapter 3 we discussed an approach to specifying web service compositions for the purpose of design and synthesising these design specifications to software process models. In this chapter we present a mapping of web service composition implementations (in the form of BPEL4WS processes) to that of processes described in the Finite State Process (FSP) notation. The mapping is required to provide explicit process representation behind that of the BPEL4WS constructs activities and other process definitions. The specification of BPEL4WS is analysed and each construct, and their related formations, described by way of the FSP semantics. To further illustrate the mapping described here, we utilise (by way of graphical representation) the corresponding Labelled Transition Systems (LTS) for each FSP model generated.

## 4.1 Modelling BPEL4WS Processes

### *4.1.1 Overview of BPEL4WS*

The Business Process Language for Web Services (BPEL4WS) (Curbera, Goland et al. 2002) is a standard for specifying and executing service orchestration processes against a domain of web services. The ability to develop a standards based process execution notation emerged from earlier efforts of several competing specifications including the Web Service Flow Language (WSFL) (Leymann 2001), XLANG (Thatte 2001) and BPML (Arkin 2002). The desire is to form a standard way of coordinating a uniformed and collaborative

mechanism to support multi-service interactions for a business or other process. This is seen as a critical element of making web services viable for wide spread use. The BPEL4WS standard is based on XML and is defined as being a layer above the WSDL (described in section 2.4.2). As we have discussed in previous chapters, distributed systems integration requires more than the ability to conduct simple interactions by using standard protocols. The potential of Web Services as an integration platform will be better achieved when applications and business processes are able to integrate their complex interactions by using a standard process integration model. The interaction model that is directly supported by WSDL is essentially a stateless model of synchronous or uncorrelated asynchronous interactions. Models for business interactions typically assume sequences of peer-to-peer message exchanges, both synchronous and asynchronous, within state enabled, long-running interactions involving two or more parties.

### *4.1.2 BPEL4WS Processes and Business Protocols*

To define such business interactions, a formal description of the message exchange protocols used by business processes in their interactions can be implemented for BPEL4WS as a public process and added to the WSDL document. The definition of such public *business protocols* involves precisely specifying the mutually visible message exchange behaviour of each of the parties involved in the protocol, without revealing their internal implementation. There are two good reasons to separate the public aspects of business process behaviour from internal or private aspects. One is that businesses may not wish to reveal all their internal decision making and data management to their business partners. The other is that, even where this is not the case, separating public from private process provides the freedom to change private aspects of the process implementation without affecting the public business protocol. The viewpoint of interaction analysis in this work however, is aimed at the designers and implementers of the compositions and it is therefore necessary to analyse the private process and how its logical parts define the behaviour of what the composition execution may actually perform. A private process is implemented in BPEL4WS using a series of XML constructs (described later in this chapter) for basic logic, structural and concurrent activities in a separate document. This document forms the executable process which we use as source for private process modelling. In the later stages of analysis in our work, both public and private processes are collated to support a complete interaction analysis.

### *4.1.3 Private Process Structure*

A basic process in BPEL4WS 1.1 is defined as a root element, consisting of one or more child elements describing partners, variables, correlationsets, faultHandlers, compensationHandlers and core process sub-activities (as illustrated in Figure 4-1).

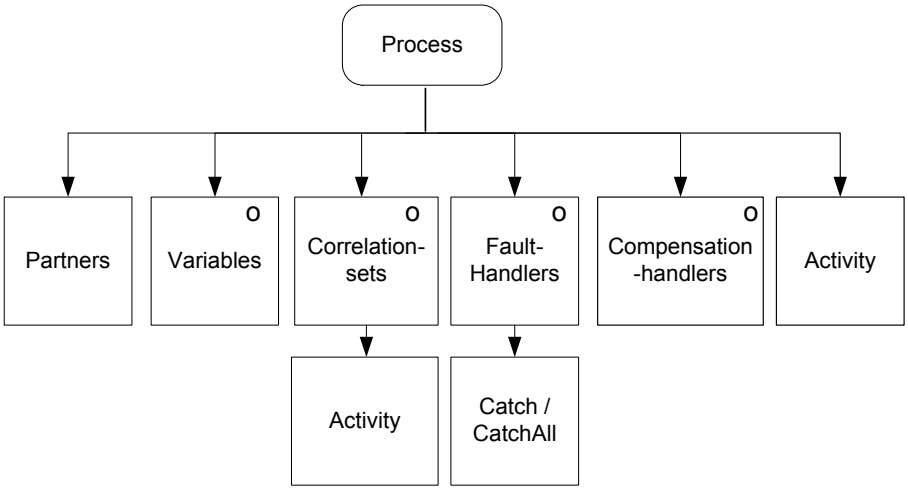



**Figure 4-1 Basic BPEL4WS Process Structure and Activity Groups**

A BPEL4WS document is split into five areas of process definitions. The first area of process elements covers process declarations. These do not directly influence the behaviour of the process, but are used to declare process wide declarations (such as partners, data containers used by the process etc). The declarations are used as reference in later stages of our modelling approach to associate interaction models between processes and its partners. The other areas of a BPEL4WS process specify the actual workflow of the process (e.g. interactions, conditional logic etc). Firstly, a set of constructs are used to specify basic interactions in the process (such as service invocation or receiving a request from a partnered service). We describe these activities as "primitive" in that they perform basic operations on behalf of the process. The second set of constructs defines the order in which activities are carried out. Such activities provide sequences of execution or concurrent activity execution. The remaining sets of activities are formed from the traditional structured programming language concepts (such as structured sequencing, iteration etc) and for specifying specific fault tolerance and compensation handling activities. We specifically group the conditional constructs of BPEL4WS, which we name "guarded process activities", as they utilise a similar representation to that of guarded transitions in process algebras. Guards define the flow of control in a computation, and how they may be translated under refinement is central to the formal treatment of models in model-checking techniques (Pavlovic and Smith 2002).

The areas of a BPEL4WS process discussed here and associated activities are summarised in Table 4-1.

**Table 4-1 BPEL4WS Process Definitions, Activity Sets and Activity Constructs**

| Element | Description | Section |
|---|---|---|
| **Common Process Declarations** | | **4.1** |
| Process | Root process element – marks the start of process | 4.1.1 |
| Partners | The "actors" in the process | 4.1.2 |
| Variables | Specify variables for message data | 4.1.2 |
| Correlation sets | Correlating activities between partner and process execution | 4.1.3 |
| **Primitive Activities and Constructs** | | **4.2** |
| Invoke | Messages sent to partner services | 4.2.1 |
| Receive | Messages received from partners | 4.2.1 |
| Reply | Messages conveyed back to partners | 4.2.1 |
| Wait | Wait for specified period | 4.2.2 |
| Empty | No actual operation, used for synchronisation of activities | 4.2.2 |
| Terminate | End process at place in process | 4.2.3 |
| **Structured Activities** | | **4.3** |
| Sequence | Sequential execution of activities | 4.3.1 |
| Flow | Concurrent message transitions | 4.3.2 |
| Links | Transitional conditions between constructs | 4.3.3 |
| **Guarded Activities and Variable Abstraction** | | **4.4** |
| Assign | Message data that requires manipulation or evaluation | 4.4.2 |
| While | Process iterations | 4.4.3 |
| Switch…Case | Test and branch conditions | 4.4.4 |
| Pick (OnMessage) | Event driven selection of transitions | 4.4.5 |
| **Fault Tolerance and Compensation Handlers** | | **4.5** |
| Scope | Scope sub-process of activities for compensation | 4.5 |
| FaultHandlers | Define either global or scope fault handling | 4.5.1 |
| Throw | Throw a specific fault at point in process | 4.5.2 |
| Compensate | Force compensation of scope or enclosing scopes | 4.5.3 |

BPEL4WS also has a structure for standard-attributes and standard-elements for all activities used in a process. Standard-attributes are provided to give each element instance a name, whether an activity join condition is applicable in this instance and whether a join failure is suppressed for this activity or not. The Standard-Elements are related to links – in that target and source link elements can be used to designate respective link origin or destination. A simplified structure representation of each of these constructs is illustrated in Figure 4-2.

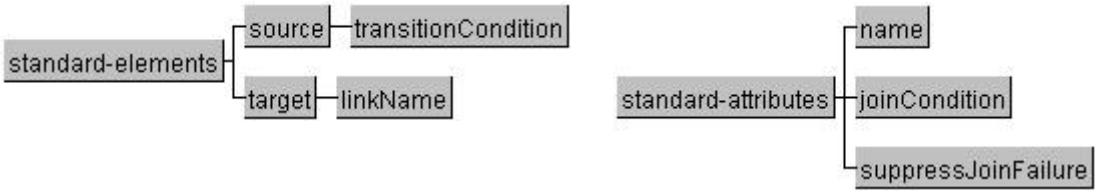



**Figure 4-2 Standard Transitional Attributes and Elements Tags of BPEL4WS Activities**

### *4.1.4 Mapping BPEL4WS Processes to FSP*

To analyse these processes, we firstly use the FSP notation to build a model of the semantics of a BPEL4WS process. Although we could build representations in Labelled Transition Systems (LTS) directly, we have already discussed (in Chapter 3) how these representations work well with few states and that they become impractical when defining larger LTSs. For this reason, (Magee, Kramer et al. 1997; Magee and Kramer 1999) proposed a simple process algebra notation called Finite State Processes (FSP) to textually specify LTSs. FSP is a specification language with well-defined semantics in terms of LTSs, which provides a concise way of describing LTSs. Each FSP expression *E* can be mapped onto a finite LTS. FSP introduces several operators, including an action prefix, choice, recursion, an end state, sequential and parallel composition and equivalence minimisation (a more detailed list is given in Appendix A). A summary of the semantics for FSP are listed below.

If x and y range over actions, and P and Q range over FSP processes, FSP introduces the following operators:

- **Action prefix** "`->`": `(x->P)` describes a process that initially engages in the action `x` and then behaves as described by the auxiliary process `P`.
- **Choice** "`|`": `(x->P|y->Q)` describes a process which initially engages in either `x` or `y`, and whose subsequent behaviour is described by auxiliary processes `P` or `Q`, respectively.
- **Recursion**: the behaviour of a process may be defined in terms of itself, in order to express repetition.
- **End state** "`END`": describes a process that has terminated successfully and cannot perform any more actions.
- **Sequential composition** "`;`": `(P;Q)` where `P` is a process with an `END` state, describes a process that behaves as P and when it reaches the `END` state of `P` starts behaving as the auxiliary process `Q`.
- **Parallel composition** "`||`": `(P||Q)` describes the parallel composition of processes `P` and `Q`.

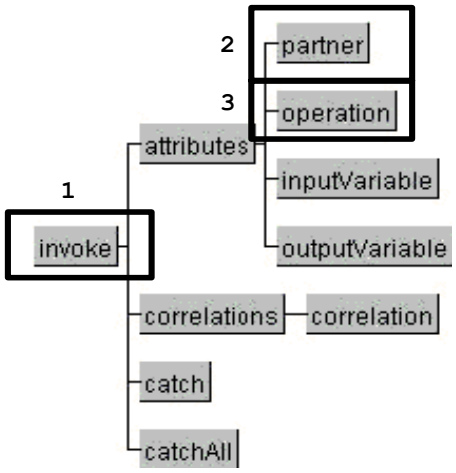
- **Trace equivalence minimisation** "`deterministic`": `deterministic P` describes the minimal trace equivalent process to `P`. If no terminating traces are proper prefixes of other traces, then it also preserves `END` states.
- **Strong semantic equivalence minimisation** "`minimal`": `minimal P` describes the minimal strong semantic equivalent process to `P`.

## 4.2 Mapping Primitive Activities

The basic activities in the BPEL4WS specification are formed from the constructs of Invoke, Receive, Reply, Terminate, and Wait. The Receive, Reply and Wait activities are basic transition activities without additional complexity, however, the Invoke and Terminate activities hold additional properties, whereby invocations can be designated with either an expected reply or without. Terminate is used to abruptly end a process, and is specified with an "eventual" end to running activities once initiated.

### *4.2.1 Label Abstraction of Web Service Interactions*

In BPEL4WS, an activity consists of a parent element name and a series of attribute elements. The activity that is modelled can be formed using a representative activity label structure associated with the transition of the activity in a model. There are differences however in the elements that are used to build meaningful activity labelling depending on whether it is an interaction activity or an interaction dependency activity (such as links, conditional activities or fault/compensation handling). The basic structure used for the activity process in our modelling is firstly the construct name (e.g. invoke), secondly the partner name (i.e. from the role designated in process) and thirdly the operation name (e.g. "getOrders"). The three main interaction activities of invoke, receive and reply for message communication are equally modelled, in that activities are labelled with the same syntax but with different corresponding construct name. This labelling structure is illustrated in Figure 4-3. Other constructs in BPEL4WS also have a scheme applied to constructing appropriate labelling. However, we shall see that this naming convention changes when we introduce connectivity between interacting partners and their processes in section 5.2.1. Further elaboration is also required to synchronise these interaction activities between compositions, which is discussed in chapters 5 and 6.




Service Interaction Activities Labels:

*<constructname_partner_operation>*

- Invoke construct.

  `invoke_buyer_neworder`

- Receive construct.

  `receive_seller_neworder`

- Reply construct.

  `reply_seller_confirmation`

**Figure 4-3. Basic Service Activity Labelling**

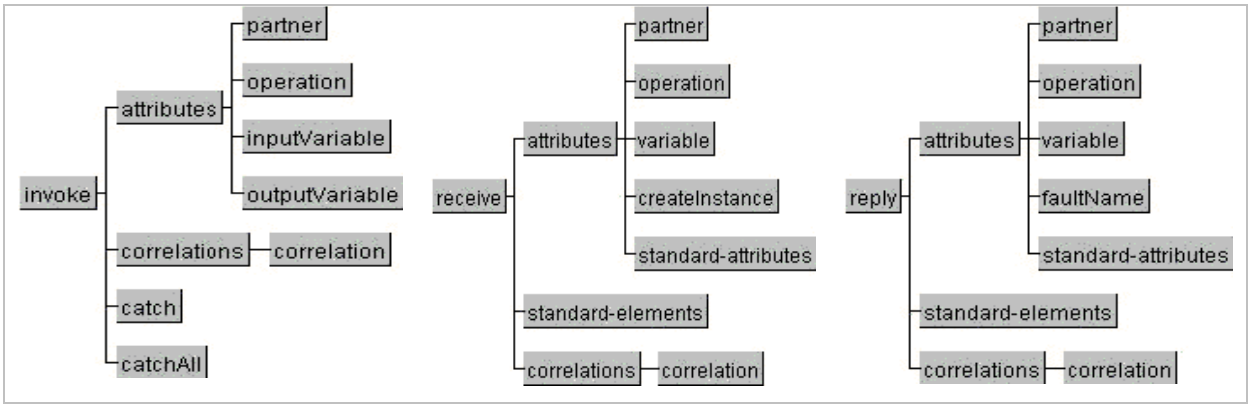
### *4.2.2 Invoke, Receive, Reply*

The invoke activity specifies that a request message is to be sent to a designated partner of the process. The style of invocation can differ depending on whether the activity is specified to be completed with a reply (i.e. a reply from a partner service operation), or is expected to be invoked without a operation reply (commonly known as a "request only invocation"). The latter provides scoping the invocation only for the additional fault or compensation handling actions. We consider the difference in invocation style, in terms of service choreography, in Chapter 5. The invoke activity may also contain optional correlation, fault tolerance and compensation activities. These are also discussed separately in a later section.

The Receive activity specifies that a message is to be received from a designated partner of the service. The receive activity has an optional "*createinstance*" attribute which, if set to true, sets a process creation point on the message received, as a form of process trigger. The semantics of this are unclear in the BPEL4WS specification, as the result of having multiple process creation points is highly non-deterministic. In our work, we are concentrating on static analysis of these interactions and therefore assume that the process is created at the first activity specified in the process document.

The Reply activity specifies that a reply message is to be sent back to a designated partner of the service. This is specified on the assumption that a receive activity has successfully been carried out earlier in the process. We discuss compatibility checking of the interaction between invoke, receive and reply activities, and process partners in detail in Chapter 5. The forms of invoke, receive and reply activities are illustrated in Figure 4-4. The semantics of

invoke, receive and reply can be represented in FSP using a simple process transition. In FSP a transition is represented by a process name and a labelled transition followed by an END process marker. The synchronisation and concurrency of these basic activities enriches the semantics of how these activities affect process execution. We consider this enrichment in section 4.3. To illustrate the semantic mapping of these activities we present each in Figure 4-4, as BPEL4WS forms, FSP and graphically as a series of LTS processes.




BASIC ACTIVITY BPEL4WS

```
<invoke partner="p1" operation="o1" />
<receive partner="p2" operation="o2" />
<reply partner="p1" operation"o1" />
```

BASIC ACTIVITY FSP

```
INVOKE =   (invoke_p1_o1 -> END).
RECEIVE =  (receive_p2_o2 -> END).
REPLY =    (reply_p1_o1-> END).
```

BASIC ACTIVITY LTS

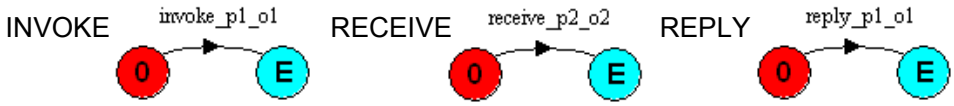



**Figure 4-4 Invoke, Receive and Reply constructs and mapping to FSP**

### *4.2.3 Wait and Empty*

The wait activity specifies that a process be suspended for a duration or until a certain expression is satisfied (for example a date or time has passed). Valid expressions are evaluated as either Boolean (true or false), as a duration (counter), a deadline (date and time) or the result of an assignment expressions. Our semantics concentrate on static analysis and we translate the *duration* and *deadline* types of conditions to literal values. The *empty* activity is used as a "placeholder" to provide activity linking synchronisation. The semantics for this are elaborated upon in section 4.3.3. Therefore both the wait and empty activities can also be represented in FSP with a process transition as with the other basic activities discussed previously.

### *4.2.4 Terminate*

The terminate activity specifies that a process be immediately terminated. As specified in BPEL4WS, a terminated event in a process has varying effects on the activities currently being executed. In this mapping, we have assumed that a termination will cause an immediate "STOP" state of the process. Therefore, the terminate activity in BPEL4WS can be semantically mapped in FSP by using process composition, synchronised on a choice of either performing an activity of a set, or the terminate activity itself. Figure 4-5 illustrates this mapping with an example of a concurrent execution of terminate with two other activities. At any point in the process, a process termination may occur.

TERMINATE BPEL4WS

```
<flow>
 <invoke partner="p1"
     operation="o1" />
 <invoke partner="p2"
     operation="o2" />
 <terminate />
</flow>
```

TERMINATE FSP

```
INVOKE1 = (invoke_p1_o1->END).
INVOKE2 = (invoke_p2_o2->END).
||FLOW1 = (INVOKE1 || INVOKE2).
set ACTSET = {invoke_p1_o1,invoke_p2_o2}
TERMS = (ACTSET->TERMS | terminate->END).
||TERMINATE = (FLOW1|| TERMS).
```

TERMINATE LTS

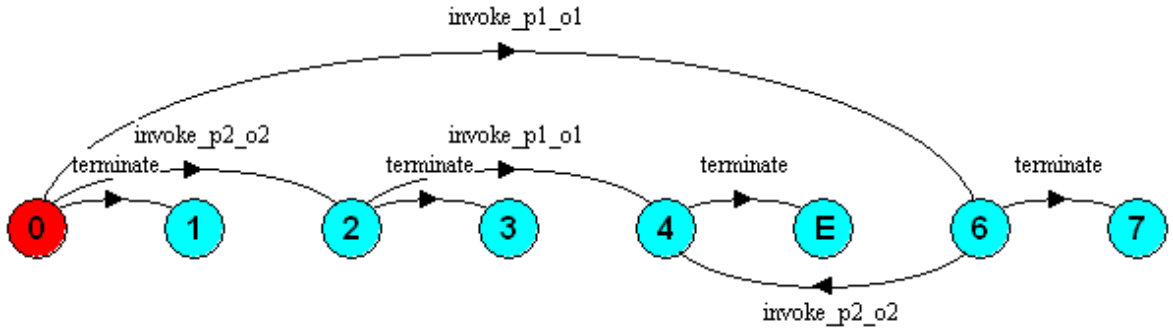



**Figure 4-5 Example mapping of terminate activity as LTS process**

Note that in BPEL4WS, the sequence and flow constructs are used to denote these scopes yet are considered to be activities in their own right.

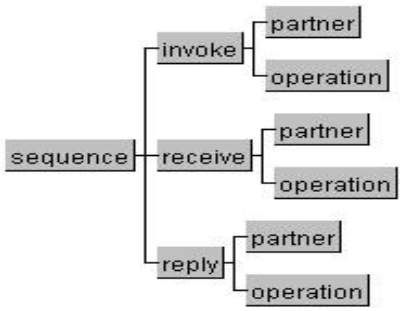
## 4.3 Structured Activities

In this section of the semantics mapping between BPEL4WS and FSP, we consider the transition of activities scoped within a sequence or concurrent order of execution. In modelling terms, these are known as sequential or parallel compositions. Modelling the possible paths of execution in sequential or parallel compositions provide rich details of the behaviour that the activities carried out could be undertaken in when the process is executed.

### *4.3.1 Sequences of Activities*

The sequence activity construct is used to scope a sequence of activities in the order they given in that sequence scope. For example, a BPEL4WS sequence form (illustrated in Figure 4-6) defines a sequence of activities that firstly invokes a partner service, receives a message from a partner service and then replies to the partner in sequence. A sequence of activities in FSP is described using the notion of a sequence composition. This is represented between the activities of the sequence using the **;** operator. The sequence activity may include any other BPEL4WS activity (including other sequence or flow activities). For example, the sequence mapping from the form illustrated in the figure previously, is illustrated in Figure 4-6. The mapping to FSP generates three processes (in this case invoke, receive and reply activities) and a process defines the sequence of these activities using the FSP sequence operator.

SEQUENCE Construct




SEQUENCE BPEL4WS

```
<sequence>
 <invoke partner="p1" operation="o1">
 <receive partner="p2" operation="o2">
 <reply partner="p1" operation="o1">
</sequence>
```

SEQUENCE FSP

```
INVOKE1 = (invoke_p1_o1 -> END).
RECEIVE1= (receive_p2_o2 -> END).
REPLY1  = (reply_p1_o1 -> END).
SEQUENCE = INVOKE1; RECEIVE1;
           REPLY1; END.
```

SEQUENCE LTS

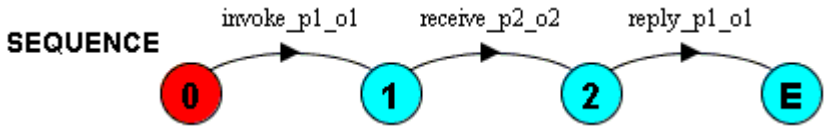



**Figure 4-6 Sequence construct and mapping to FSP**

### *4.3.2 Concurrent Activities*

Concurrent activities in BPEL4WS are scoped using the *flow* construct. The semantics for the scope are defined as that the flow completes when each activity contained within the flow scope has completed. Each activity is executed concurrently; however, both the flow

and any activity within the flow scope may have external dependencies outside of the flow scope, specified in the links section of each activity. The construct of the *flow* activity is formed as illustrated in Figure 4-7. A flow activity creates a set of concurrent activities directly nested within it. It further enables expression of synchronization dependencies between activities that are nested directly or indirectly within it. The link construct is used to express these synchronization dependencies. As with most other constructs in BPEL4WS a link has a name attribute, although in a *flow* this attribute name is used to identify the link in other parts of the process. All the links of a flow activity must be defined separately within the flow activity. Source and target elements of an activity are used to link two activities. We consider the semantic mapping of *Linked transitions* in the next section of this chapter. The basic flow activity in FSP is represented using the parallel composition operator, which is ||. Parallel composition in FSP is defined as: If P and Q are processes then (P || Q) represents the concurrent execution of P and Q. This satisfies the concurrent execution of activities in the BPEL4WS flow construct. Figure 4-7 provides an illustrated example of a concurrent set of BPEL4WS activities for invoking, receiving and replying.

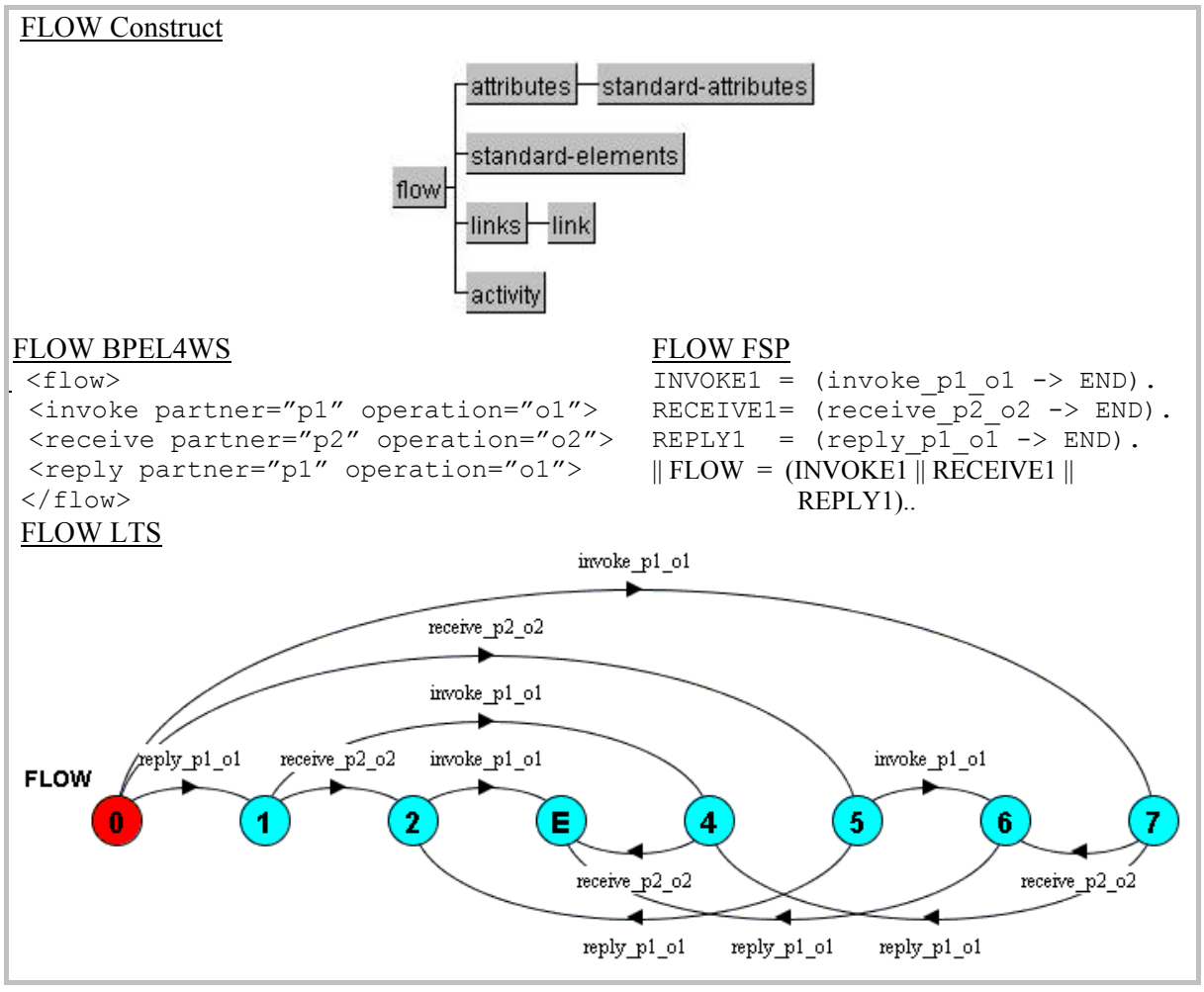



**Figure 4-7 Flow construct and mapping to FSP for concurrent activities**

### *4.3.3 Linked Transitions*

Linked transitions are pre and post activity execution conditions. They are used to determine when activity transitions can be made given the requirement that other activities have successfully completed. The source of a link must provide a source element specifying the link's name and the target of the link must specify a target element specifying the link's name. The source activity may also specify a transition condition through the transitionCondition attribute of the source element. If the transitionCondition attribute is omitted, it is deemed to be present with a value of "true". Every link declared within a flow activity MUST have exactly one activity within the flow as its source and exactly one activity within the flow as its target. The source and target of a link MAY be nested arbitrarily deeply within the (structured) activities that are directly nested within the flow, except for the boundary-crossing restrictions. Target and source links are represented in FSP as transition processes that are sequenced prior to or post the linked activity process, depending on whether it is a source or target link respectively. As an example, Figure 4-8 illustrates the target and source links that are used in an approval invocation service request in a “loan approval” process (as part of the example requirements scenario discussed in section 3.1). Here, an “invoke_p1_o1” operation is only carried out after an approval is received and assessed, with links guarding the operation transition.

LINKS BPEL4WS

```
<sequence>
 <invoke partner="p1"
         operation="o1">
    <target linkName="receive-to-
             approval"/>
    <target linkName="assess-to-
             approval"/>
    <source linkName="approval-to-
             reply" />
 </invoke>
</sequence>
```

LINKS FSP

```
LINK1 = (receive_to_approval->END).
LINK2 = (assess_to_approval ->END).
||TARGETLINKS = (LINK1 || LINK2).
LINK3 = (approval_to_reply ->END).
||SRCLINKS = (LINK3).
INVOKE1 = (invoke_p1_o1 -> END).
SEQUENCE  = TARGETLINKS; INVOKE1;
            SRCLINKS; END.
||LINKS = (SEQUENCE).
```

LINKS LTS

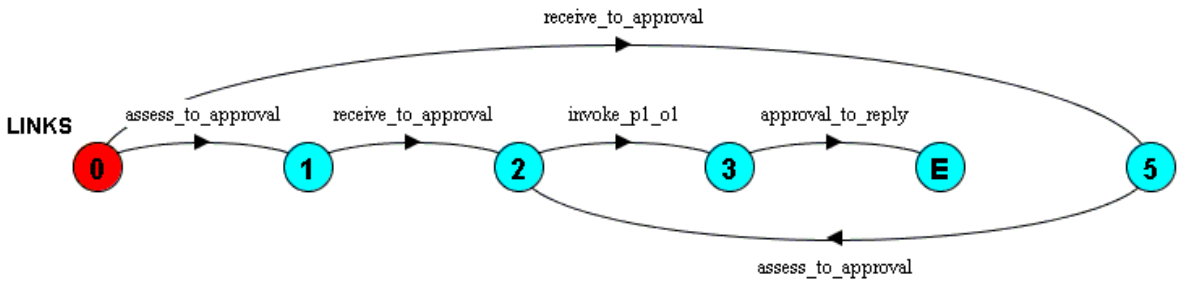



**Figure 4-8 Mapping link Semantics for part of a loan approval process**

Whilst the actual declaration of a link is simply referred to as a label, the representation of a process activity provides synchronisation of dependencies against other processes. Additionally the transitionCondition can be given for source links (as either a constant or variable Boolean expression). This condition is represented in FSP as a guarded process activity.

## 4.4 Guarded Process Activities

The basic and structural activities in BPEL4WS do not require in-depth knowledge of data handling between partners and the state of a process, however, the conditional activities of the specification require expression evaluation to determine specific paths of execution. For example, a *while* activity (based upon the traditional iteration programming language construct) contains a condition expression, which is evaluated against one of several value types defined in BPEL4WS. All those activities, including the *while* activity, which require expression evaluation for conditional structured processing are described in this section, being mapped to FSP using the guarded action and process variable concepts.

### *4.4.1 Variable Abstraction and Guards*

A key concept in BPEL4WS, for both message data and conditional processing flow, is the use of messages variables. Variables (formally known as containers in earlier specifications) hold data between services invocations and replies, and are also used for a source of values in semantic expressions for some activities. Variables take the form as illustrated in Figure 4-9;

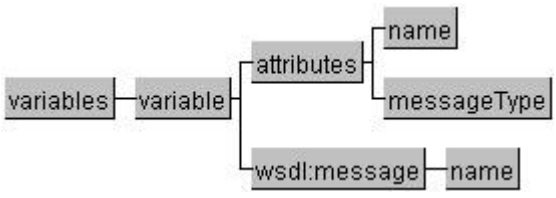



**Figure 4-9 The Variable form in BPEL4WS**

Variables are closely linked with message definitions and the abstract protocol (via the WSDL associated with the process). Although variables are essentially type-less, the message format that variables hold are defined in the WSDL and contain type information as part of the WSDL XML schema specification. A representation of the semantics of variable

use in BPEL4WS is based upon a *read-write* model in a similar style to the "get or set" that used in traditional programming language variables. In FSP, a variable can be associated with a set or a range of integer values. To model the reading and writing of variable expressions an action label is used to represent a read or a write respectively. Each variable used within the BPEL4WS may be modelled using this mechanism. The range of values set for a variable can be determined during mapping, as activities access and assign values to variables. The FSP code illustrated in Figure 4-10, defines such a read-write variable model.

There is clearly a limitation imposed on the mapping of variables, in that we are mapping on the basis of a static data analysis and representation. To provide some flexibility in determining how the values of variables affect the process execution path, we add further mapping to enumerate static values within the process. For example, in the conditional guarded activities a "Switch..case" path is determined by which path of activity is undertaken. By representing possible process path outcomes based upon each *case* value evaluation and each variable read-write mapping, all possible paths can be analysed. The enumeration of variable values, from expressions, is discussed further in the following sections, as referenced with the *assign* activity. An example variable model, in this case for the variable of sellerInfo and a variable part named "askingPrice", is illustrated in Figure 4-10. The corresponding use of this model within further BPEL4WS activity mapping would assume that the variable is used twice, as the value range is set to 0..1 (such as the transition for askingPrice.write[0] or askingPrice.write[1]).

**/* FSP code for read-write variable model for a variable A with range VR. */**

```
range VR = 0..1
VARIABLE(A=0) = VARIABLE[A],
VARIABLE[i:VR] = (write[j:VR]->VARIABLE[j]|read[i]->VARIABLE[i]),
VARIABLE['null] = (write[j:VR]->VARIABLE[j]|read['null]->VARIABLE['null]).
```

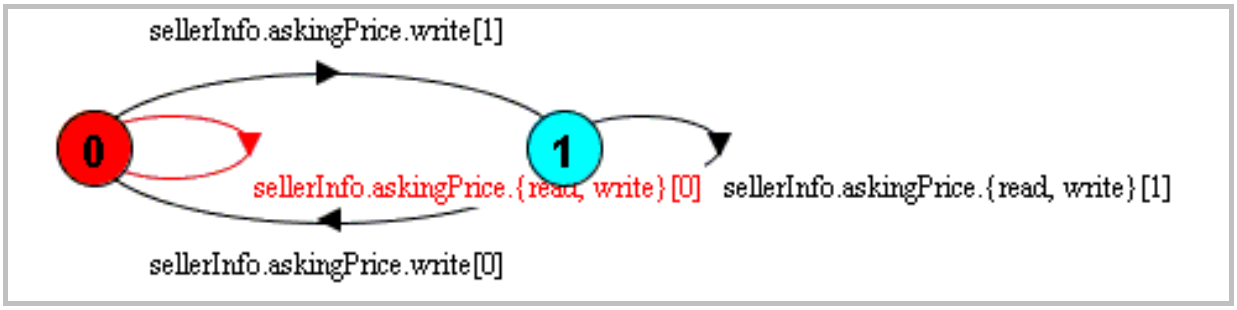



**Figure 4-10 Read-write Models for BPEL4WS Variables. FSP (top), LTS (bottom)**

With the variable and condition mapping defined, all of the following guarded activities may be mapped with synchronisation against one or more variable definitions.

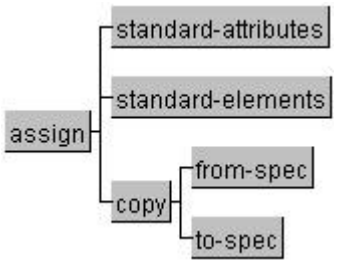
### *4.4.2 Assign*

The assign activity reads a value *from* a source data element and writes *to* another data element. The source may be a literal value, a variable, a partner, a variable property or a value expression. For the purpose of mapping the assign activity to FSP, we reuse the read-write model described previously, as the basis for assign activity processes. In BPEL4WS, the assign activity takes the form illustrated in Figure 4-11. The assign activity can also be linked for pre or post-conditional activities. For each assign statement in the BPEL4WS implementation, an associated read or write is mapped to a process activity with a parameter representing the variable value assigned. This may be an enumeration of an expression or a literal value. For each variable in BPEL4WS, the read and write of variables used in assign statements is referenced. In Figure 4-11, a BPEL4WS example illustrates how an assign is used to copy the expression "yes" to the approvalInfo variable. The assign statement is linked to an "assess-to-setMessage" link declaration, which is sourced as a pre-condition elsewhere in the complete BPEL4WS implementation. The assign statement also declares a source condition for another activity. The corresponding FSP for the assign activity is generated with an enumeration of the possible values that a condition may evaluate in the scope of the process. This is achieved by having a lookup table created and appended to as each assignment is made to a variable. Regardless of the type of the from specification of the assign activity, an enumerated value is given to that from spec. The enumeration is created in the sequence that values are assigned, and a FSP write activity (as part of the read-write model) represents the assign copy action for that assignment.

### *4.4.3 While*

The while activity provides a construct to perform iterative execution of activities until a Boolean condition is evaluated to true. The while activity is represented in FSP in two parts. Firstly using the variable expression evaluation described at the beginning of this section. The second part is to represent recursion and use the FSP *if....then…else* with alternative process transitions depending on whether the evaluation is true or false. For example, if a condition is evaluated using an expression which considers whether a variable contains a "yes" value, the "yes" value can be associated with a integer value and compared with the actual value currently stored in the model guard variable.

ASSIGN Construct Form




ASSIGN BPEL4WS

```
<assign>
   <target linkName="assess-to-
           setMessage"/>
   <source linkName="setMessage-to-
           reply"/>
   <copy>
       <from expression="'yes'"/>
       <to variable="approvalInfo"
           part="accept"/>
   </copy>
</assign>
```

ASSIGN FSP

```
<…variable range and process for enumerated yes
    and no  - range 0 ,1…>
LINK1 = (assess_to_setMessage->END).
||TARGETLINKS = (LINK1).
LINK2 = (setMessage_to_reply->END).
||SRCLINKS = (LINK2).
ASSIGN1 = (approvalInfo.accept.write[0] ->
          END).
ASSIGNSEQ = TARGETLINKS; ASSIGN1;
          SRCLINKS; END.
||ASSIGN = (ASSIGNSEQ).
```

ASSIGN LTS

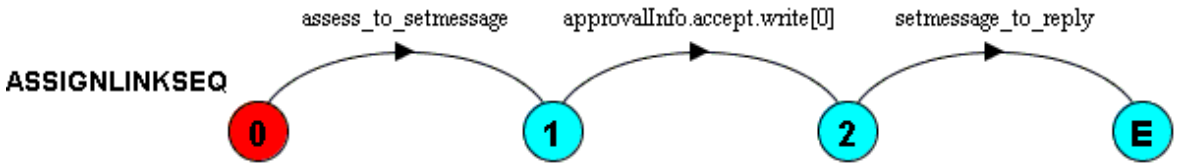


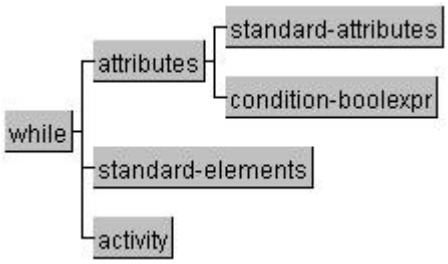

**Figure 4-11 Assign construct and FSP mapping**

The activities defined within the while activity are mapped as per the sequence construct detailed in section 4.3.1. The example, illustrated in Figure 4-12, has an iterative body of receiving a message and replying to the service message. The corresponding FSP mapping, illustrates how the conditional expression that "exp = yes" is mapped to a read process on the variable "exp" and the processes are synchronised in a parallel composition model.

### *4.4.4 Switch..Case*

The Switch..Case structured activity supports conditional activity selection in a pattern that occurs quite often. The case branches of the switch activity are considered in the order in which they appear. The first branch whose condition holds true is taken and provides the activity performed for the switch. If no branch with a condition is taken, then the otherwise branch is taken. If the otherwise branch is not explicitly specified, then an otherwise branch with an empty activity is deemed to be present.

WHILE Construct Form




WHILE BPEL4WS

```
<while condition ="exp = 'yes'">
  <sequence>
      <receive partner="caller"
          operation ="echo"
          variable = "request" />
      <reply partner="caller"
          operation="echo"
          variable="request" />
  </sequence>
</while>
```

WHILE FSP

```
<…variable range and process for enumerated true
     and false for while - range 0 ,1…>
||WHILEEXP = exp:WHILE_variable.
set WHILEEXP_alphabet =
{exp.{read,write}.[Range]}
WHILEEVAL = (exp.read[i:Range]-
>WHILEEVAL[i]),
WHILEEVAL[i:Range] = if (i==0)
then SEQ1; WHILEEVAL else END.
RECEIVE1 = (receive_caller_echo -> END).
REPLY1 = (invoke_caller_echo -> END).
SEQ1 = RECEIVE1; REPLY1; END.
WHILESEQ = WHILEEVAL; END +
{WHILEEXP_alphabet}.
||WHILE = (WHILESEQ).
```

WHILE LTS

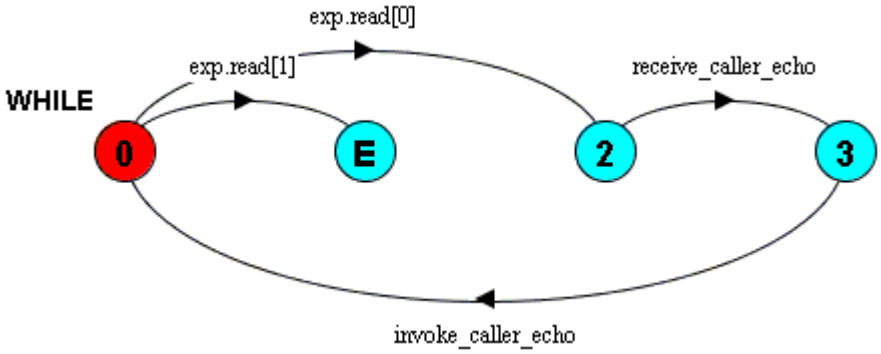
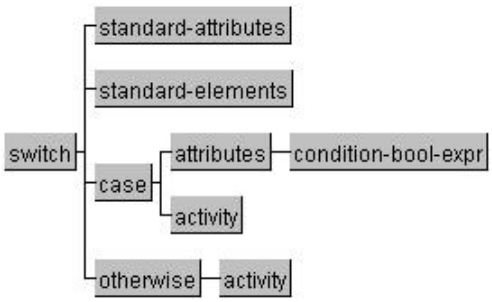



**Figure 4-12 While construct and mapping to FSP**

To map the Switch activity, we can again utilise the variable FSP read-write model. An evaluation process is defined for each case defined in the body of the switch. The conditional evaluation is identical to the while condition explained in section 4.4.3. The mapping includes the requirement that if the Switch activity contains an "otherwise" condition (i.e. that if none of the case statement conditions evaluates to true) then this activity is executed. The formation of this evaluation is then in a sequence of cases, or if none of these results in true, then the "otherwise" activity path must be followed. Activities defined within each of the case clauses, is mapped as a sequential composition. Figure 4-13 illustrates a "Switch..Case" mapping to FSP, for a choice of whether a negotiation outcome

results in a success or a failure. The case condition (the value of sellerInfo variable), is evaluated and either the process CASE1 or OTHERWISE activity is executed.

SWITCH Construct Form




SWITCH BPEL4WS

```
<switch>
  <case condition="bpws:getVariableData('sellerInfo', 'askingPrice')">
    <assign> <copy>
         <from expression="'yes'"/>
         <to variable="approvalInfo" part="accept"/>
    </copy> </assign>
  </case>
  <otherwise>
    <assign> <copy>
         <from expression="'no'"/>
         <to variable="approvalInfo" part="accept"/>
    </copy> </assign>
  </otherwise>
</while>
```

SWITCH FSP

CASE1EVALC= (sellerInfo.askingPrice.read[i:MPRange]->CASE1EVALC[i]),
CASE1EVALC[i:MPRange] = if (i==0) then CASE1; END else OTHERWISE; END.
||CASE1EVAL = (CASE1EVALC).
SUCCESSASSIGN = (negotiation.outcome.write[0] -> END).
||CASE1 = (SUCCESSASSIGN).
FAILEDASSIGN = (negotiation.outcome.write[1] -> END).
||OTHERWISE = (FAILEDASSIGN).
MARKETPLACESWITCH = CASE1EVALC; END.

SWITCH LTS

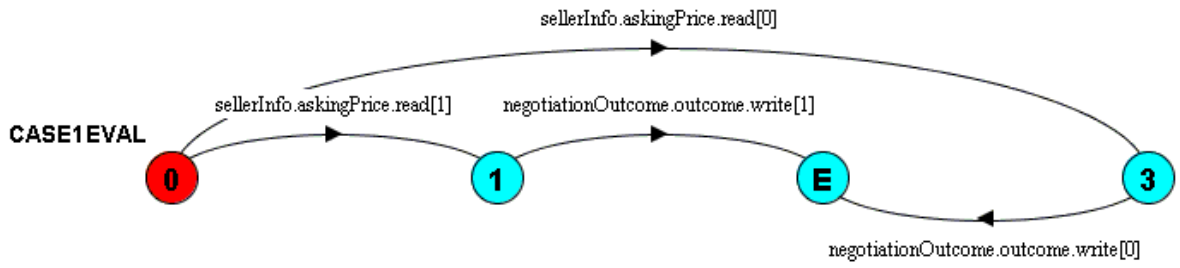


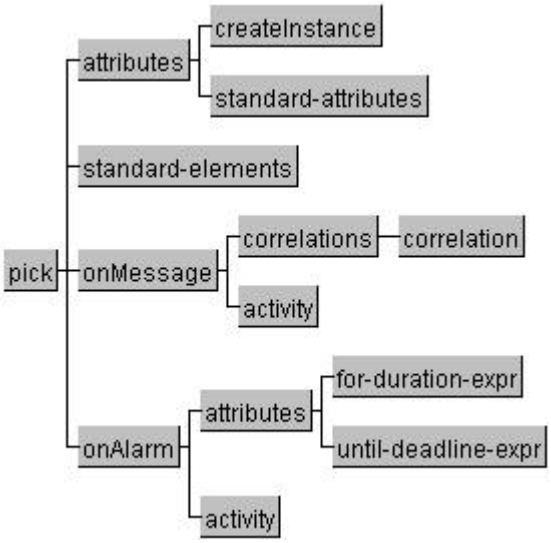

**Figure 4-13 Switch/Case construct and mapping to FSP**

### *4.4.5 Pick..onMessage*

The pick activity awaits the occurrence of one of a set of events and then performs the activity associated with the event that occurred. The occurrence of the events, as defined in the BPEL4WS specification, is often mutually exclusive (the process will either receive an acceptance message or a rejection message, but not both). If more than one of the events occurs, then the selection of the activity to perform depends on which event occurred first. If the events occur almost simultaneously, there is a race and the choice of activity to be performed is dependent on both timing and the actual implementation of a runtime BPEL4WS engine. The Pick..onMessage construct takes the form illustrated in Figure 4-14. Each of the event types (either an *onMessage* or an *OnAlarm*) is triggered in BPEL4WS using a choice of event received. In FSP, this corresponds directly to the notion of process choice, being specified using the FSP operator of |. In addition to standard-element and attribute mappings (section 4.1.2), the mapping provides a choice of any OnMessage or OnAlarm activity. The choice within a *pick* activity is then on the activities within each event, and this is mapped using the sequence mapping discussed in section 4.3.1. Each of these sequences is placed in a "wrapper" pick process, delimited by the FSP choice operator.

For example, in an ATM logon process, the event triggers of *logon* and *disconnect* can be modeled as an onMessage transition processes. The pick sequence for each of these events is illustrated as FSP code in Figure 4-14. Notice that there are two cases of the write process transition for the process variable *connected.value.write*. As discussed in section 4.4.1, the variables possible values are enumerated based upon the types and number of assignments that the process defines. In this case the *logon* is enumerated as condition 0 and *disconnect* enumerated as 1. Therefore, further process activities which may read this variable would check for logon (0) or disconnect (1) using the enumerated lookup table generated. In the case of the model presented here, we define the possible paths of a pick activity, for any logon or disconnect received by the process. Figure 4-14 also illustrates a sample of Pick..OnMessage BPEL4WS, the FSP and a pick model as a graphical LTS. Note that we do not model the timed or triggered event itself. We abstract this and provide a process which represents an alternative possible path through the process model. Therefore, it can be used to aid the validation and verification of if a particular event occurs, which path would be chosen.

PICK Construct Form




PICK BPEL4WS (sample)

```
<pick createInstance="no">
   <onMessage partner="atmFrontEnd" operation="logoff"
     variable="logoffReq">
      <assign><copy><from expression="false()"/>
        <to variable="loggedon" part="value"/>
       </copy></assign>
   </onMessage>
   <onMessage partner="atmFrontEnd" operation="disconnect"
     variable="sessionMsg">
      <assign><copy><from expression="false()"/>
        <to variable="connected" part="value"/>
        </copy></assign>
   </onMessage>
</pick>
```

PICK FSP

```
ATM_ONMESSAGE_DISCONNECT = (disconnect->END).
DISCONNECT = (connected.value.write[1] -> END).
ATM_ONMESSAGE_DIS_SEQ = ATM_ONMESSAGE_DISCONNECT; DISCONNECT; END.
ATM_ONMESSAGE_LOGON = (logon->END).
LOGON = (loggedon.value.write[0] -> END).
ATM_ONMESSAGE_LOGON_SEQ = ATM_ONMESSAGE_LOGON; LOGON; END.
||PICK = (ATM_ONMESSAGE_DIS_SEQ || ATM_ONMESSAGE_LOGON_SEQ).
```

PICK LTS

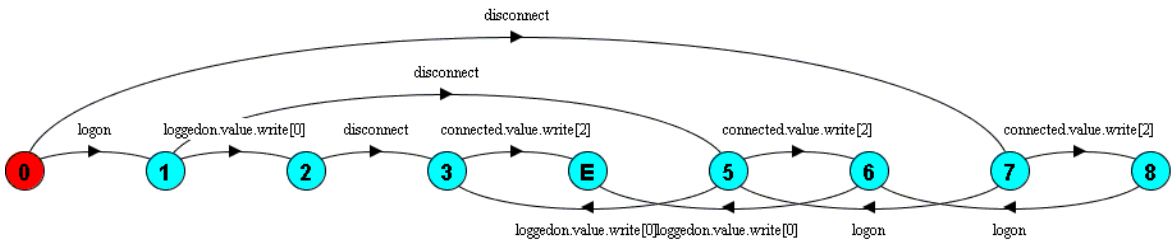



**Figure 4-14 FSP Code for Pick..OnMessage event model for an ATM Logon**

## 4.5 Fault and Compensation Handlers

In BPEL4WS, fault and compensation handlers can be defined for one or many activities in activity *scopes*. Scopes may also contain nested scopes. In the case of compensation handlers, they may also be specified "inline" to an activity, which means that an activity has its own individual compensation routine. A fault may be raised on the receipt of a message, or defined by the BPEL4WS engineer to be specifically thrown by way of the BPEL4WS *throw* activity. Compensation handlers define part of the process behaviour that is meant to be reversible, for example, if a request is made to place an order, the compensation action may likely be to cancel the order.

Compensation routines in BPEL4WS are self-contained blocks of code that are scoped the same as fault handlers, yet are only executed on the completion of a scope or on completion of a particular activity. Fault handling in BPEL4WS is provided for two levels. Firstly, the composition (process) may declare fault handling scopes that are focused on specific fault handling for a series of activity blocks, a "local scope". In this scope, specialised fault handling can be undertaken for events that occur in specific areas of the process. Alternatively or additionally to local fault handling, global handling routines may be specified that react to certain events at any point in the composition process. Such fault handling can be a useful way to trap errors and respond to partners if a complete process must be completed in a usual reply, and any specific faults are not known or captured to a local scope. Therefore, this level of fault handling is at a "global scope". For nested scope handlers, when a fault is raised the innermost enclosing scope is executed. The BPEL4WS specification discusses how fault and compensation handling refers to data variables, with compensation handling having access to a snap-shot of the process data variable state. Also, fault handlers may receive one of many fault types raised. In our work we concentrate on the possible behaviour exhibited by faults raised and if compensation routines are undertaken in the event that a fault is raised in both local and global scopes.

### *4.5.1 Modelling Fault Handling*

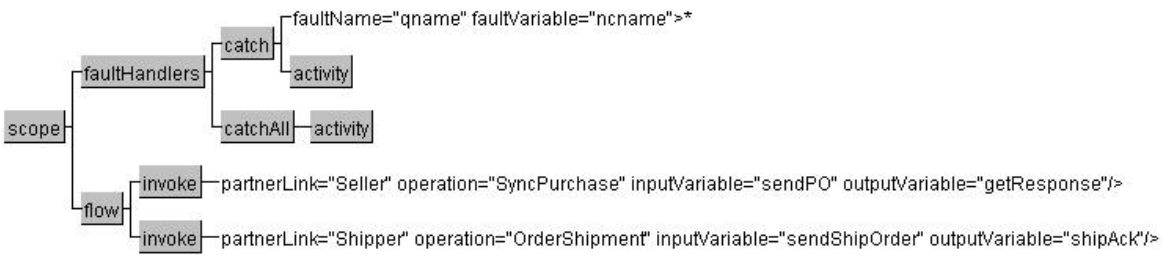
To model a fault handler we firstly build a representation of the fault Handler in BPEL4WS by mapping the activities of the handler to FSP. Any activity described earlier in this section can be mapped as part of this modelling exercise. If an error occurs inside a scope, a particular type of fault is generated. A fault can also be thrown for a scope from code, if the

BPEL4WS engineer adds this to the process code. An interesting characteristic of fault handling in BPEL4WS is that if a fault is raised, then the remaining activities in the scope are automatically terminated. An example form of a fault handler, within a scope is given in Figure 4-15. The scope activity defines the start of the mapping, with two composition processes modelled as part of the fault handling and normal execution set of activities. Firstly, the activities are built as described by each element in the set, in our example previously; this would be a set of two processes executed as a sequential composition process. Secondly, the fault handling process is modelled as a choice of fault types that could potentially be raised. This is modelled in the same way as a Switch..Case activity, described in section 4.4.4. The example in Figure 4-15, is given as sample BPEL4WS, FSP model and is modelled as a graphical LTS. As the semantics of BPEL4WS includes this "scope termination" once a fault has been raised, then this is very similar to the example of the *terminate* activity we have discussed in section 4.2.4. The differences being that a choice of processes is carried out depending on the fault type caught, and that the termination does not finish the composition, but only the scope. This is handled through the fault handling process being composed with the wider process and the scope end transition synchronising with the next activity in the composition process. The BPEL4WS specification also provides a *catchAll* mechanism to provide faultHandling for any fault that is not caught by the choice of fault handling specifically defined. We are limited in modelling this mechanism as a simple alternative (again as an additional case in a list of choices) such that an unknown fault is considered an alternative path of execution (e.g. *faultxraised*).

### *4.5.2 Throw*

The throw activity is used to signal an internal fault explicitly. As each fault is named uniquely within the process (given a qualified name), the throw activity specifies this name such that a fault handler sequence is identified and appropriate action is undertaken as a result of the fault signal. To illustrate the semantic mapping of faults thrown, we provide the structure of the throw activity, the BPEL4WS form and the corresponding FSP model in Figure 4-16. The FSP model composes the activity of a throw, with that of the action of an individual fault in the faultHandler model discussed previously. This synchronisation of these activities causes the scope to terminate immediately following the faultHandler process, as it would if caught in from a fault raised by the BPEL4WS process implementation engine.

FAULT HANDLING Construct




FAULT HANDLING BPEL4WS (example)

```
<scope>
    <flow>
        <invoke partnerLink="Seller" operation="SyncPurchase">
        <invoke partnerLink="Shipper" operation="OrderShipment">
    </flow>
    <faultHandlers>
        <catch faultname="fault1" faultVariable="fault1var" />
        <catch faultname="fault2" faultVariable="fault2var" />
    </faultHandlers>
</scope>
```

FAULT HANDLING FSP

```
range Fault_IntRange = 0..1
FAULTHAND1 = (fault.read[i:Fault_IntRange]->FAULTHAND1[i]),
FAULTHAND1[i:Fault_IntRange] = if (i==0) then DOFAULT1; END else if (i==1)
then DOFAULT2; END.
DOFAULT1 = (fault1raised->END).
DOFAULT2 = (fault2raised->END).
INVOKE1 = (invoke_seller_SyncPurchase->END).
INVOKE2 = (invoke_shipper_OrderShipment->END).
ACTSEQ = INVOKE1 ; INVOKE2 ; END.
||ACTIVITIES = (INVOKE1 || INVOKE2).
set ACTSET = {invoke_seller_SyncPurchase,invoke_shipper_OrderShipment}
TERMS = (ACTSET -> TERMS | fault1raised -> END | fault2raised -> END).
||FAULTMON = (ACTIVITIES || TERMS || FAULTHAND1).
```

FAULT HANDLING LTS

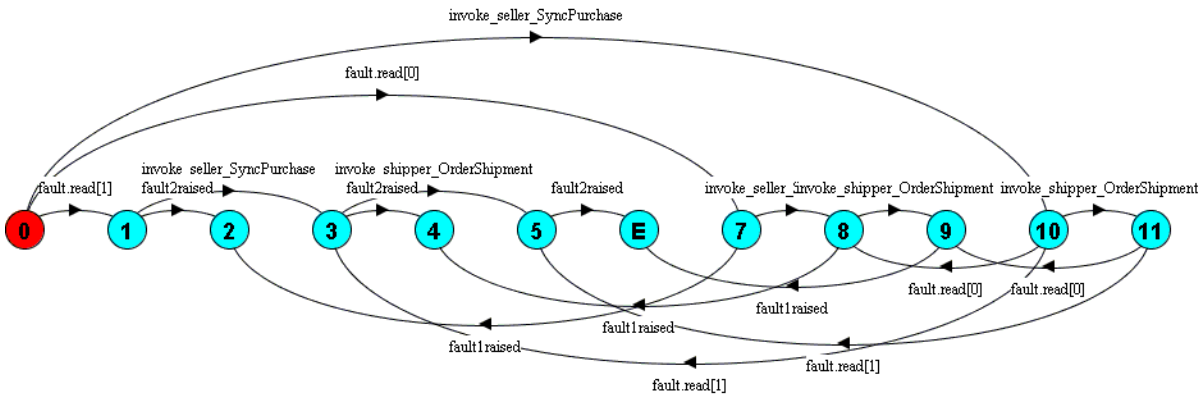
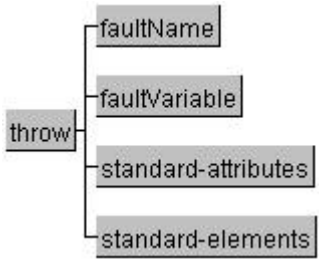



**Figure 4-15 LTS of Fault Handler and normal execution activity scope**

THROW Construct Form




THROW BPEL4WS (sample)

```
<scope>
  <flow>
    <sequence>
      <invoke partnerLink="Seller" operation="SyncPurchase">
      <throw faultName="fault1raised"
             faultVariable="faultvar"></throw>
    </sequence>
    <sequence>
      <invoke partnerLink="Shipper" operation="OrderShipment">
      <throw faultName="fault2raised"
             faultVariable="faultvar"></throw>
    </sequence>
  </flow>
  (see faultHandler BPEL4WS example in Figure 4-15)
</scope>
```

THROW FSP

```
range Fault_IntRange = 0..1
FAULTHAND1 = (fault.read[i:Fault_IntRange]->FAULTHAND1[i]),
FAULTHAND1[i:Fault_IntRange] = if (i==0) then DOFAULT1; END else if (i==1)
 then DOFAULT2; END.
DOFAULT1 = (fault1raised->END).
DOFAULT2 = (fault2raised->END).
INVOKE1 = (invoke_seller_SyncPurchase->fault.read[0]->END).
INVOKE2 = (invoke_shipper_OrderShipment->fault.read[1]->END).
ACTSEQ = INVOKE1 ; INVOKE2 ; END.
||ACTIVITIES = (INVOKE1 || INVOKE2).
 set ACTSET = {invoke_seller_SyncPurchase,invoke_shipper_OrderShipment}
TERMS = (ACTSET -> TERMS | fault1raised -> END | fault2raised -> END).
||FAULTMON = (ACTIVITIES || TERMS || FAULTHAND1).
```

THROW LTS

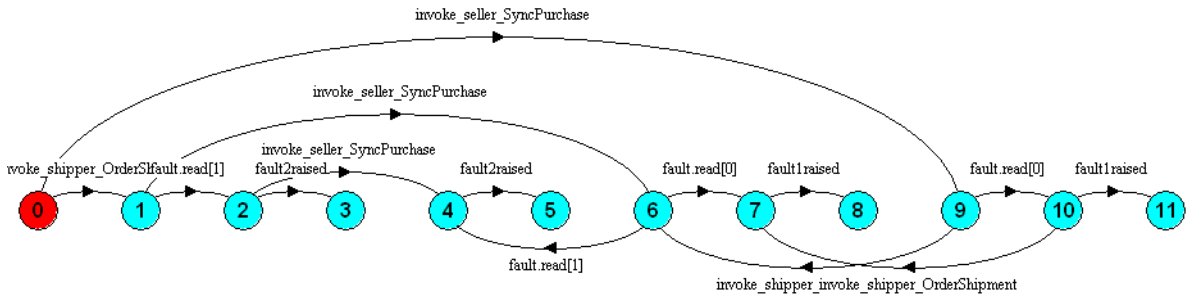



**Figure 4-16 LTS of Throw activity model and process synchronisation**

### *4.5.3 Modelling Compensation Handling*

To model compensation handling we can build a composition process for the compensation handler in a given activity or as part of an activity scope and then combine the compensation composition process with that of the normal execution activity process or activity scope. As a set of compensation activities is only carried out at the end of an activity or a scope of activities, the composition of both activity groups yields a behavioural choice of execution paths in the BPEL4WS composition model. A compensation handler takes on the forms as illustrated in Figure 4-17. An example inline compensation handler in BPEL4WS is also illustrated as its form in Figure 4-17. To map the compensation handlers for inline activity compensation, we build two processes in FSP, one representing the normal executed activity and the other the compensation activities undertaken if compensation would be undertaken. For the example given, this means a single process for the invocation of the Seller service operation "SyncPurchase" and then a process for the compensation activity of invoking the Seller service operation "CancelPurchase".

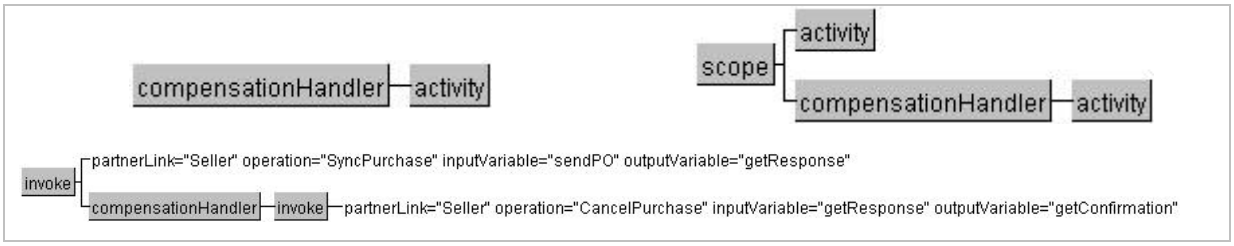



**Figure 4-17 Compensation Handlers as inline (top-left) or scoped (top-right) and activity (bottom)**

Clearly, the compensation action here is to cancel the purchase made previously. The model of this compensation needs to consider the choice of undertaking compensation or not. In this way, we can reuse the mapping FSP for the Switch..case activity (described in section 4.4.4). With this conditional transition included in the model for the inline compensation we illustrate a composition choice path with a BPEL4WS sample, mapping to FSP and an LTS in Figure 4-18. The other form of compensation handlers is as part of a *scope*. We provide another example of a compensation handler, inside a scope of activities, in Figure 4-19. The scope is considered to be the start point at which the compensation process is composed along with the normal execution process. In this case, the two normal invoke activities, for the seller and supplier service operations, are composed with an alternative set of invoke activities and conditionally undertaken following normal execution as with the inline

example previously. The yielding model from mapping this scope of activities is given in Figure 4-19.

COMPENSATE (inline) BPEL4WS

```
<invoke partnerLink="seller" operation="SyncPurchase">
    <compensationHandler>
        <invoke partnerLink="seller" operation="CancelPurchase" />
    </compensationHandler>
</invoke>
```

COMPENSATE (inline) FSP

```
COMPENSATE = (compensate.read[i:TRUEFALSE_variable]
->COMPENSATE[i]),
COMPENSATE[i: TRUEFALSE_variable] = if (i=='true') then
COMPENSATE_INVOKE; END else END.
COMPENSATE_INVOKE = (invoke_seller_CancelPurchase -> END).
INVOKE = (invoke_seller_SyncPurchase -> END).
INVOKE_SEQ = INVOKE; COMPENSATE; END.
||COMPENSATEEXAMPLE = (INVOKE_SEQ).
```

COMPENSATE (inline) LTS

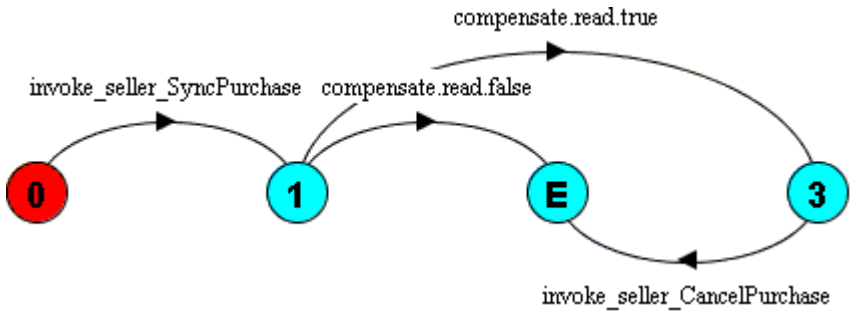
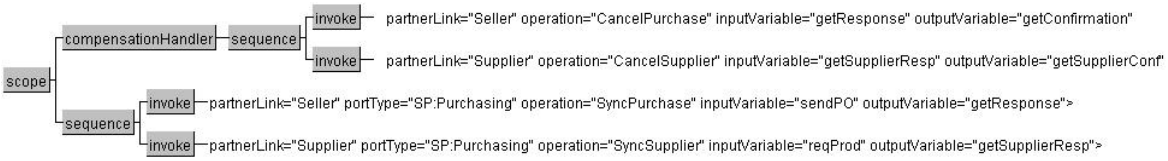



**Figure 4-18 Compensate (inline) choice of execution paths**

Lastly, the compensation handler process may be synchronised with the *compensate* activity, if directly specified in the BPEL4WS process code by the BPEL4WS Engineer. Note that the default behaviour of BPEL4WS process compensation is to execute the scoped compensation handler. If the scope is part of a nested scope, the engineer may instruct the process to execute a compensation routine out of the current scope. Such an activity is modelled through synchronisation of the compensation handler process at the given point in the process mapping, using an alternative compensation process name.

COMPENSATE (scope) Construct




COMPENSATE (scope) BPEL4WS

```
<scope>
    <sequence>
        <invoke partnerLink="Seller" operation="SyncPurchase">
        <invoke partnerLink="Supplier" operation="ReqProd">
    </sequence>
    <compensationHandler>
        <sequence>
            <invoke partnerLink="Seller" operation="SyncPurchase">
            <invoke partnerLink="Supplier" operation="CancelSupplier">
        </sequence>
    </ compensationHandler >
</scope>
```

COMPENSATE (scope) FSP

```
COMPENSATE = (compensate.read[i:TRUEFALSE_variable]->COMPENSATE[i]),
COMPENSATE[i: TRUEFALSE_variable] = if (i=='true') then
COMPENSATE_INVOKE; END else END.
INVOKE3 = (invoke_seller_CancelPurchase->END).
INVOKE4 = (invoke_supplier_CancelSupplier->END).
COMPENSATE_INVOKE = INVOKE3; INVOKE4; END.
INVOKE1 = (invoke_seller_CancelPurchase->END).
INVOKE2 = (invoke_supplier_CancelSupplier->END).
INVOKE_SEQ = INVOKE1; INVOKE2; COMPENSATE; END.
||COMPENSATESCOPE = (INVOKE_SEQ).
```

COMPENSATE (scope) LTS

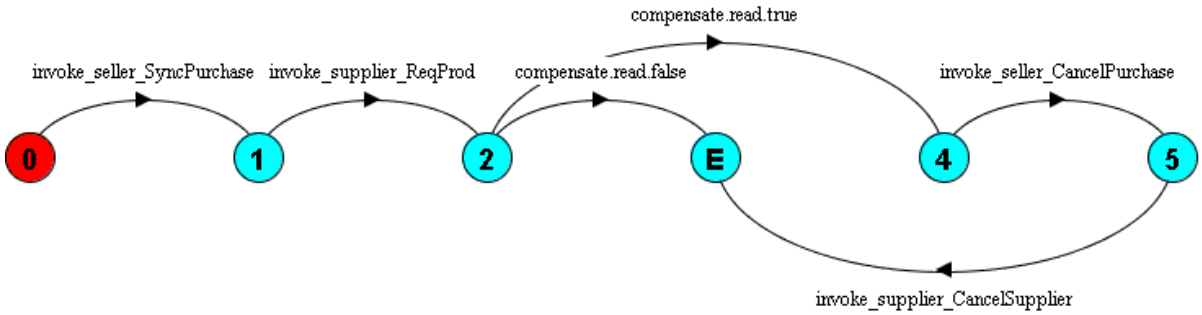



**Figure 4-19 LTS of scoped compensation handler activities**

## 4.6 A Complete Example

A complete example of modelling a web service composition process is taken from the BPEL4WS source samples provided by a BPEL4WS engine implemented called "*BPWS4J*"

(Curbera, Duftler et al. 2004). The example, an elaborated version of the "loan approval process" (as part of the example requirements scenario discussed in section 3.1), utilises a variety of the mapping concepts discussed in this chapter. We present the process (with original source available as part of the tool discussed in Chapter 7) as a series of mappings for each of the activities and variable definitions in this example, and discuss the resulting composition from the view of a compiled LTS. The example consists of a single concurrent activity (a flow element), with its child elements providing the loan approval process from a series of linked message exchanges between the process partners for a customer, a loan assessor and a loan approver. The process behaviour begins with an initial receive activity (with its form illustrated in Figure 4-20), whereby a customer (partner) requests a loan approval and this activity is marked as the process begin point. The request consists of various information, amongst which, one is the actual amount requested for the loan approval. The activity also has two source *transitionConditions* (described in section 4.3.3). The conditions are evaluated for either the request amount being equal to or over 10000, or less than 10000. An alternative of this strategy of linking would have been to include a *Switch..Case* activity block immediately following the *receive* activity and an expression evaluation used on the "equal to or greater than 10000" condition. The model produced by this activity is composed of three processes. Firstly, there is the simple transition process representing receiving the message itself. Secondly, there is a guarded transition activity for the conditional link source to assess the request for loan approval (a "receive_to_assess" transition) and thirdly, there is a guarded transition activity for the conditional link source to approve the request for the loan (a "receive_to_approval" transition). These processes are illustrated in Figure 4-20.

The first link target upon receiving the request for loan approval is the activity connected to the *receive-to-assess* linkName. This activity is the invocation of the operation "check" provided by the service partner "invokeAssessor". The form of this invoke activity is illustrated in Figure 4-21. In the mapping, the invoke activity itself also creates a process transition, however, the semantics of the link target sub-element of the invoke activity means that a process transition must also be created to provide a synchronised process composition on the invoke activity and only when the guarded source link transition, "receive_to_assess", has taken place. The invoke also has two source links, which are again mapped to guarded process transitions. The resulting models are illustrated as LTSs in Figure 4-21.

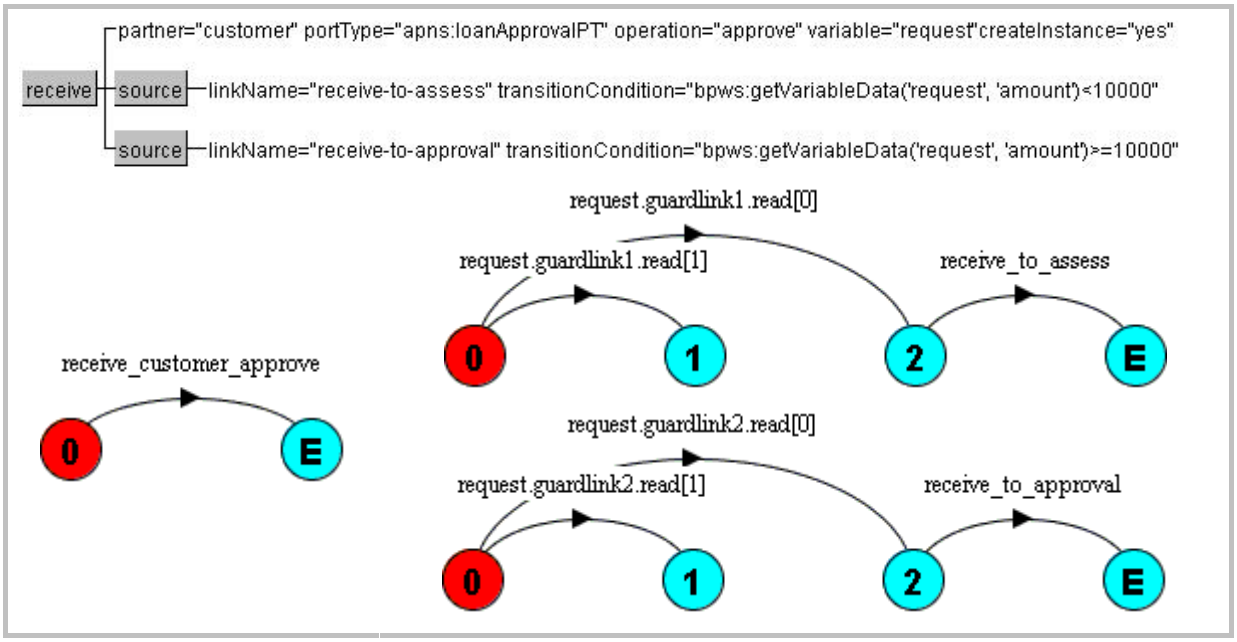



**Figure 4-20 LoanApproval models (bottom) produced from Linked Receive Activity (top)**

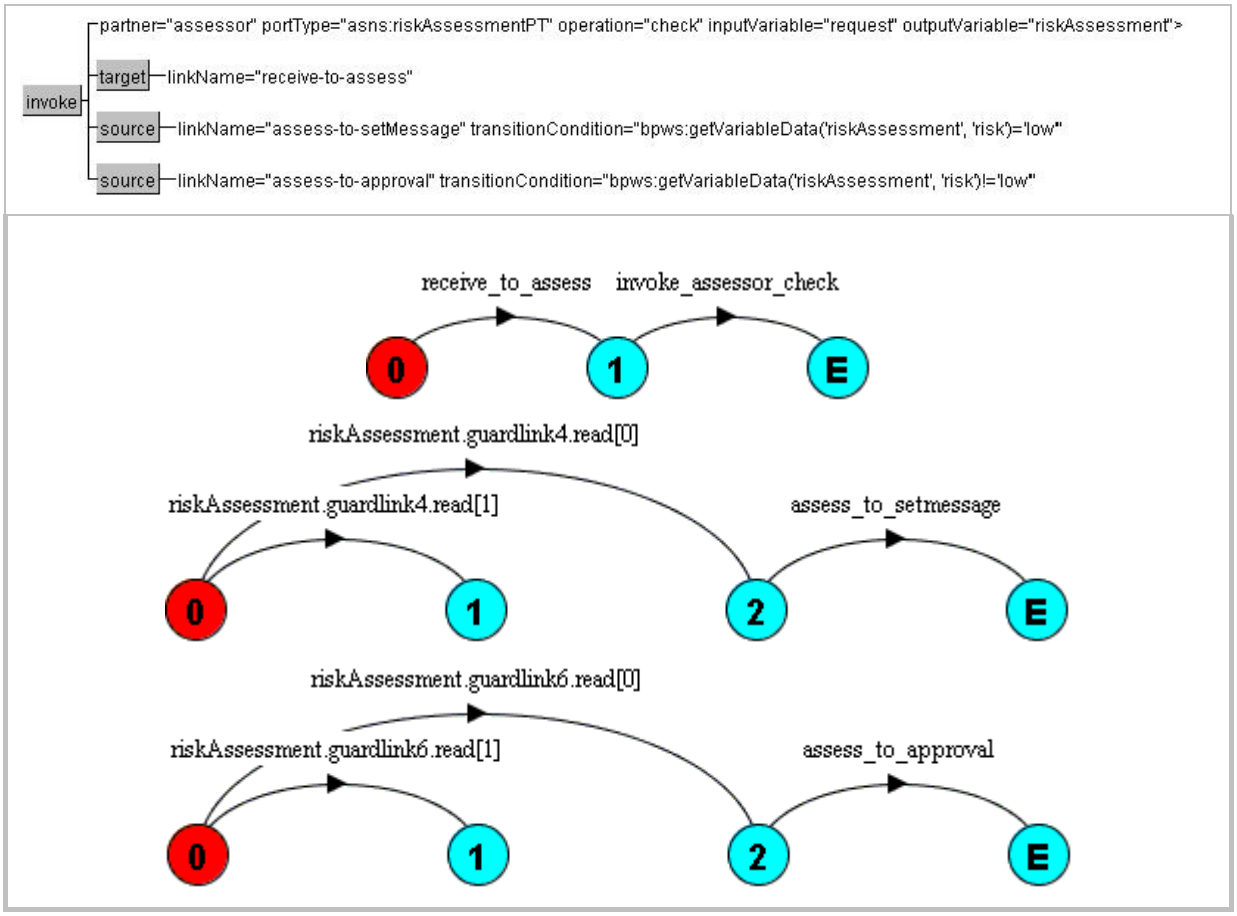



**Figure 4-21 LTS for models produced from InvokeAssessor Activity**

The second link from receiving a request for loan approval is the invokeApprover activity. In a similar way to that of the invokeAssessor activity, the invokeApprover activity also has target links, however, the invokeApprover draws upon source links which are guarded as

both part of the receive and invokeAssessor activities. The invokeApprover also has a source link of an "approval_to_reply". The form of this activity is illustrated in Figure 4-22. Again, the basic invoke activity is modelled as a simple process transition, however, the two target links (namely for "receive_to_approval" and "assess_to_approval") can occur in any order, yet they both must occur for the invoke activity transition to be undertaken. Therefore firstly, a parallel composition process is created for the target links and the invoke transition, secondly, a sequence process with two transitions is created for the invoke transition and the source link transition. The resulting models are illustrated as LTSs in Figure 4-22.

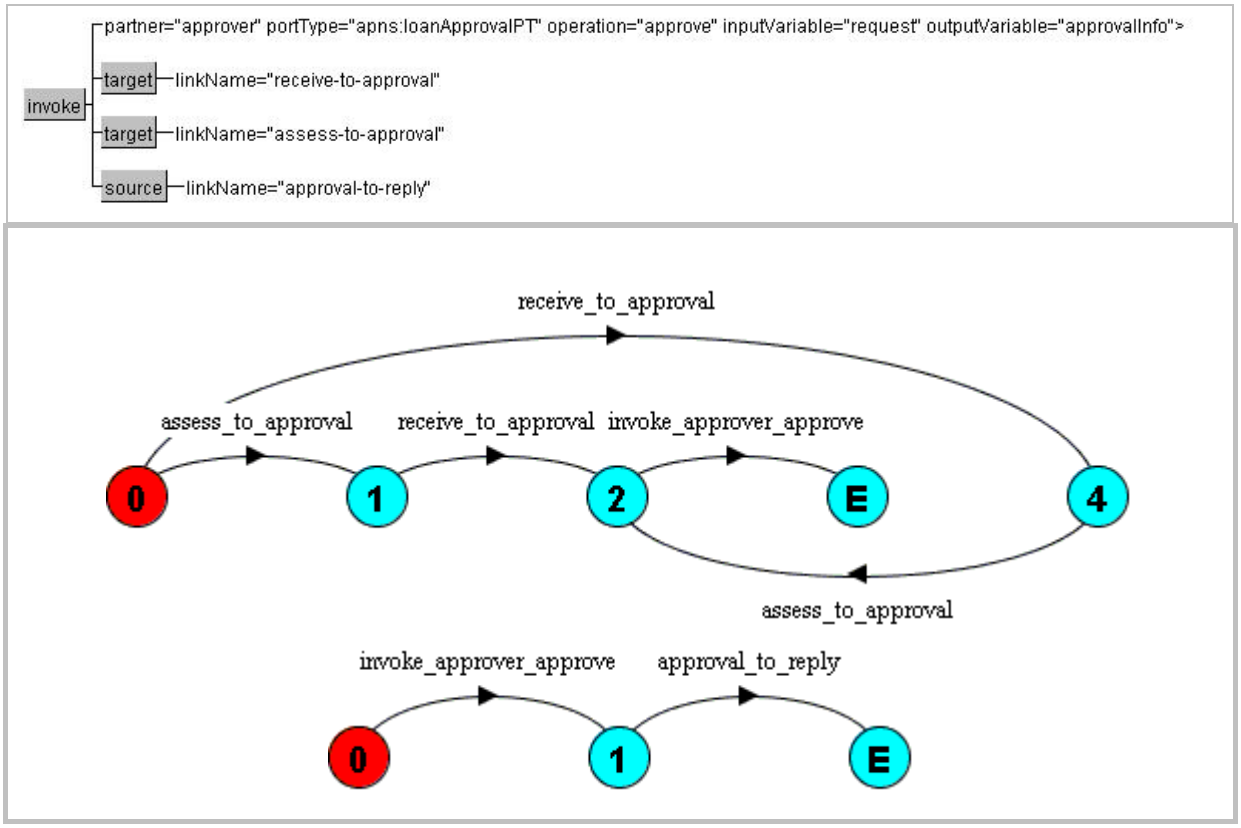



**Figure 4-22 LTS for models produced from InvokeApprover Activity**

The result of the invoke approver activity is a source linked transition to the target linked activity of replying to the loan approval request. However, the reply activity is also linked to the result of the assessment activity via an intermediate activity of assigning a value for the reply. In this case, the activity is linked as a target of the "assess_to_setmessage", which was created as part of a transitionCondition from the invokeAssessor activity earlier in the mapping. The assign activity copies the value "yes" to the reply message if the invoke activity for invokeAssessor returned as value of "low" (risk) in its reply message. The form of this assign activity is illustrated in Figure 4-23.

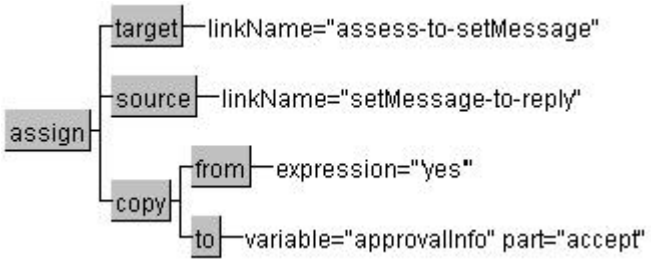



**Figure 4-23 Assign activity to set reply message content**

The mappings of this assign activity forms three processes. Firstly, the process of the target transition "assess_to_setMessage" is created and assigned to this process composition. Secondly, a write process transition is created to enumerate the assigning of the "yes" value to the message reply variable of "approvalinfo" and its variable part of "accept". The "yes" value is given an enumerated value of 0 as this is the first assignment to this variable part. Thirdly, the continuation of the transition link between assessor result and reply is given by another link transition process of "setmessage_to_reply". The sequential process composition from the mapping is illustrated in Figure 4-24.

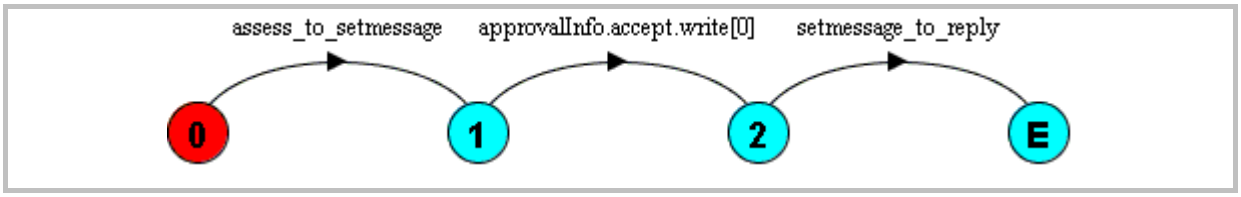



**Figure 4-24 LTS for models produced from Reply Message Assign Activity**

The last activity to be mapped in this web service composition example is that of the reply to the original receive for a loan approval. The reply activity is a target of both "setMessage_to_reply" (from the assign activity) and "approval_to_reply" (from the approver activity) respectively. The form of the reply activity is illustrated in Figure 4-25. The mapping for the reply activity therefore is similar in model form to invokeApprover activity, in that two target link transitions could occur in any order. Therefore, the parallel composition process model is mapped such that the "setmessage_to_reply" and "approval_to_reply" can occur in any order before a process transition for the actual reply to customer is undertaken. This is illustrated in Figure 4-25. The final step of modelling the web service composition is to link all the process compositions (i.e. the individual process models produced by the mapping from BPEL4WS to FSP) into a complete system architecture and to request the process *FLOW* element scoping the activities through a parallel composition.

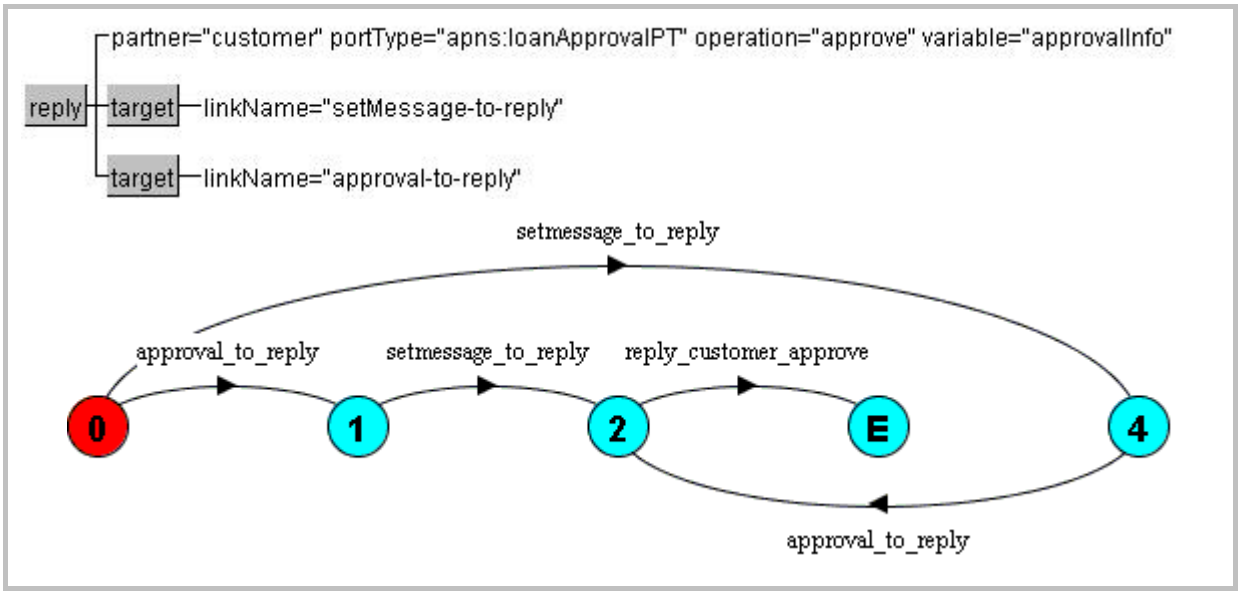



**Figure 4-25 LTS for model produced by mapping of reply activity**

In FSP, this architecture model is defined as follows and is illustrated in Figure 4-26.

**/* FSP code for architecture model of web service composition mappings. */**

||ARCHMODEL = (LA_RECEIVESEQ || LA_INVOKEASSESSORLINKSEQ || LA_ASSIGNLINKSEQ || LA__INVOKEAPPROVERLINKSEQ || LA__REPLYLINKSEQ).

## 4.7 Assumptions and Limitations

Amongst the assumptions in our semantic mappings of BPEL4WS to FSP, we have considered that a process lifecycle begins at the first receive activity specified in the process document. The possibility of multiple start points as part of a series of receive activities (discussed in section 4.2.2) would affect the order in which activities are executed. Related to this is also a limitation on modelling the correlation attribute of activities, which are used to match returning or known clients to interact in long-running processes (in a message to correlation linking). We have not implemented a synchronisation of such events, but we anticipate these mappings would be evolved to consider this in our future work. Our mapping is also currently limited in the translation of variables, in that we are mapping on the basis of a static representation (to values enumerated based upon occurrence of conditional variable comparisons). To provide some flexibility in determining how the values of variables affect the process execution path, we add further mapping to enumerate static values within the process. The mapping does not consider translating event handling, as part of an activity scope. Such a mapping would however, take a form similar to the fault and compensation handling although the semantics behind event handling are much more towards a time based simulation basis.

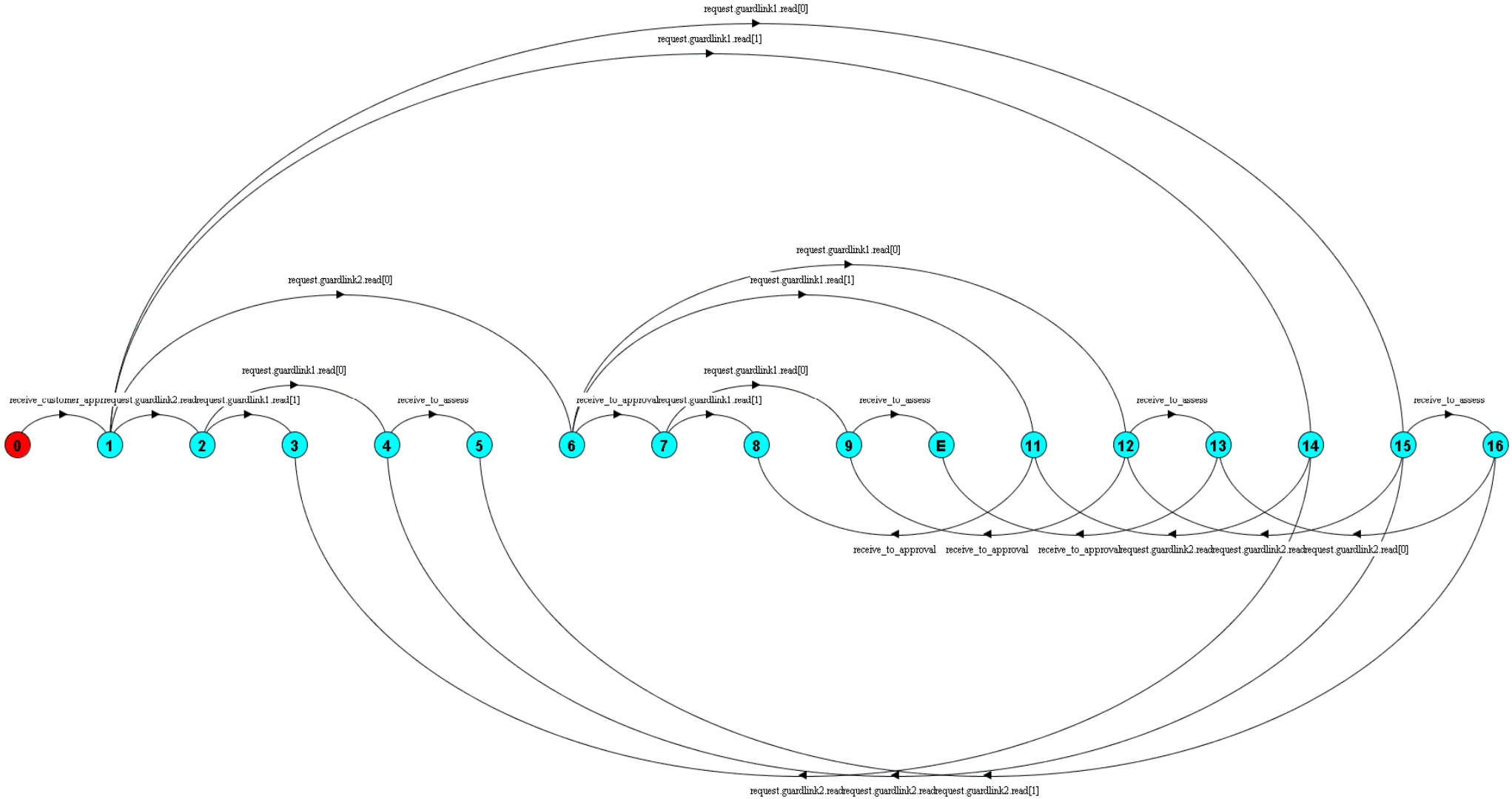



**Figure 4-26 Architecture Model of Loan Approval Web Service Composition**

We are seeking to evolve the methods described here to ease these limitations and provide a closer representation of a BPEL process model. Our evaluation of modelling BPEL4WS implementations (section 8.1.2) also discusses the implication of a new standard (with few runtime implementations) and the impact of observing a standard compliant BPEL4WS engine's behaviour against our own translation from specification to FSP.

## 4.8 Summary and Discussion

In this chapter we have described the semantics of BPEL4WS by way of mapping each of the BPEL4WS constructs to the FSP algebra and building a model of the process behaviour. With these mapping rules, we have described a modelling approach of a process defined for a single web service composition, however, this modelling is limited to a local view, or in other words, it can only be used to model the behaviour of a single process. In the next chapter we further the semantic mapping to include web service composition interactions through modelling web service conversations and their choreography. In terms of the approach discussed in section 1.2, we have introduced the highlighted parts as those illustrated in Figure 4-27.

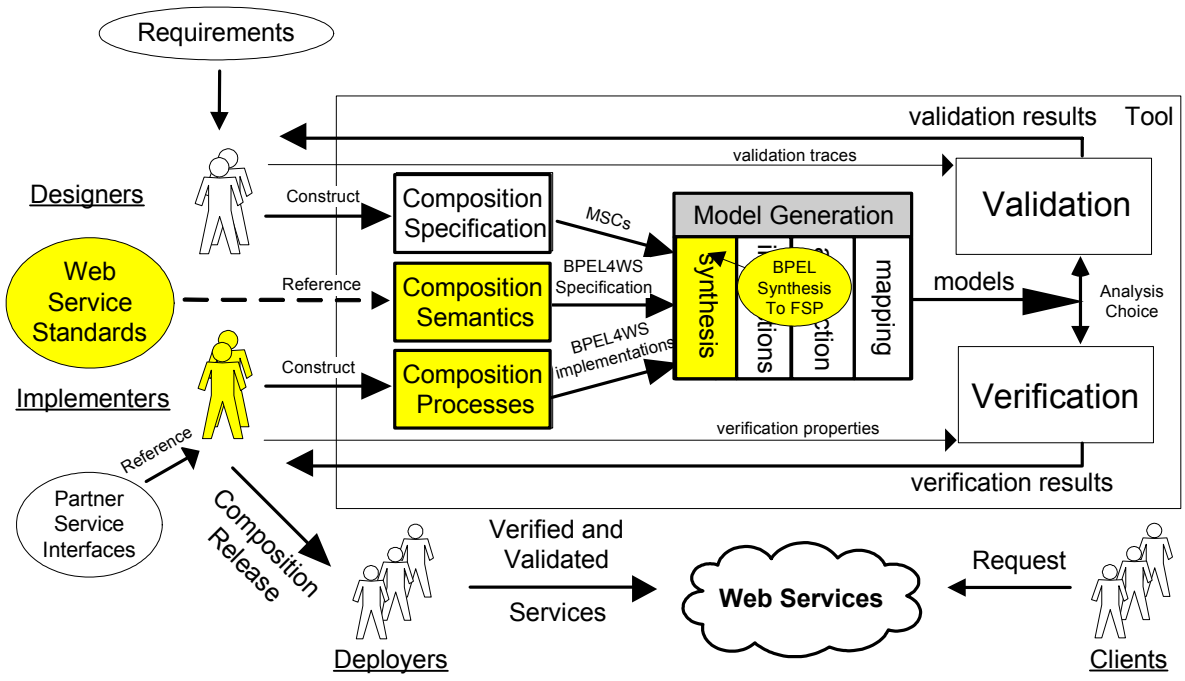



**Figure 4-27. Elements of approach discussed in chapter 4**

# Chapter 5

# Modelling Web Service Choreography

*"If the constituent parts can be understood, the reasoning goes, some insight into the whole will follow….."*
(Sanmay Das, "Modelling Complexity - Agents of Creation", The Economist, 2003)

---

In chapters 3 and 4 the design and implementation of web service composition interactions was discussed and models were produced to provide a formal representation of the behaviour specified. These models are useful to describe individual compositions; however, an elaboration of modelling is required to represent the behaviour of interacting compositions across partnered processes. A series of compositions in web service choreography needs specific modelling activities that are not explicitly derived from an implementation. In this chapter, we describe this elaboration of models to support a view of interacting web service compositions extending the mapping from BPEL4WS to FSP discussed in Chapter 4, and including web service interfaces (WSDL) for use in modelling between services.

## 5.1 Web Service Interactions and Choreography

In the previous chapters of this thesis, we have described how the process view of a web service (also known as a *composition*) takes focus of how a service interacts with other services as well as its own internal processing steps. Whilst the other participants in a composition may simply be seen as another service, their compositions are equally important when describing the form of a *conversation* between participants in web services architecture. The connective expression of these conversations provides the benefit of designing collaborative compositions or services in a given business scenario, yet as we discussed in Chapter 2, the choreography of these interactions is required to understand

exactly what state a conversation is in, and on a broader view, how multiple conversations (as part of a scenario) are coordinated by state and goals. In essence, web service choreography defines a kind of policy for "rules of engagement" in conversations implemented between partners in web services architecture, alongside operational state transfer for interactions indirectly between services (i.e. variable passing). It defines the global interactions necessary to be controlled and enforced when two or more services are interacting to fulfil a goal in a scenario. In this sense, the designer of partnered service choreography may specify global service interactions such that they form a policy of how the services should interact to complete a common goal. The concept is illustrated in Figure 5-1, where a defined choreography specification (as a set of scenarios) is used as a policy for coordinating several service partners.

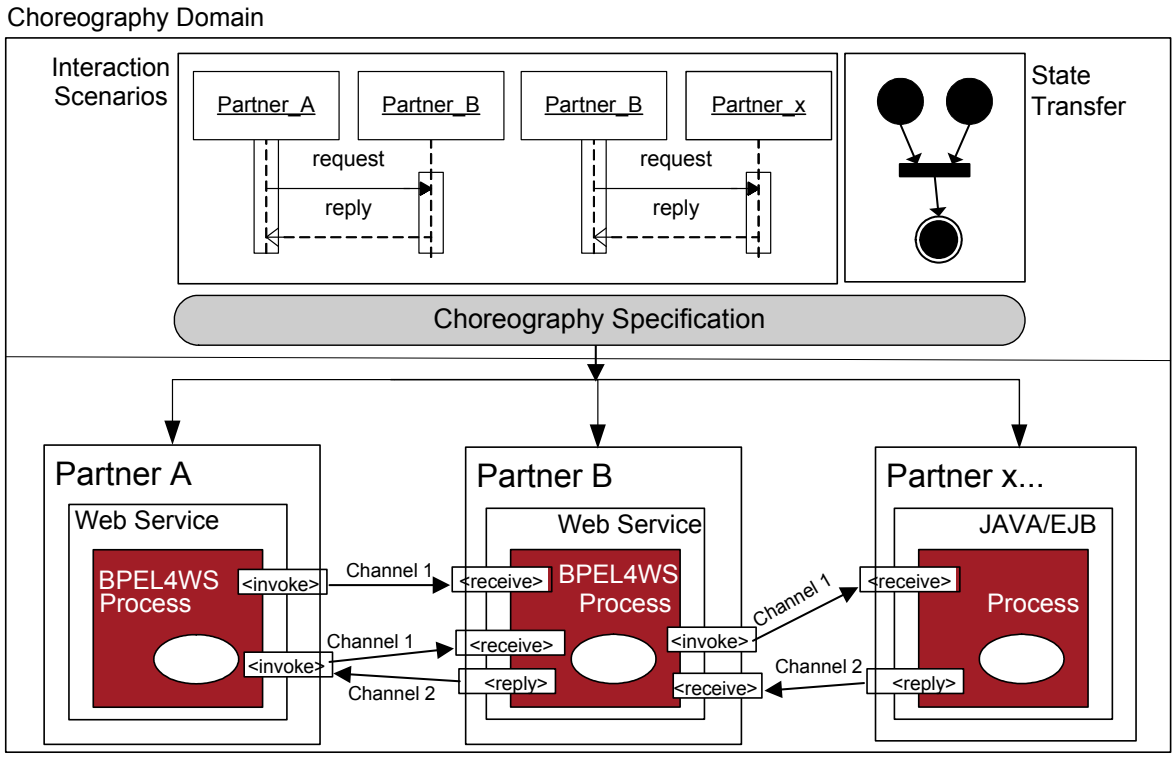



**Figure 5-1 View of multiple service compositions interacting and choreography layer**

Choreography describes the peer-to-peer collaboration between participants with designated roles in service interactions. Choreograph makes use of interaction and activity notation to define the relationships, which represents message exchanges between two web services participants. Web Service Choreography differs from Web Service compositions in that it is a global view of interactions; in other words, it is a view of a series of process compositions that interact to fulfill a common goal. Web Services Choreography explores the

relationships, constraints and *liveness* of a series of interactions between two or more partners in a wider goal and with a standard (such as Web Service Choreography Description Language (WS-CDL) (Kavantzas, Burdett et al. 2004)) the aim is to provide a technology independent method of describing these through state collaboration and awareness at a higher level than individual service processes. WS-CDL suggests providing a design specification for choreography, which in turn is used for *defining a contract* for composition interactions. It is anticipated that a designer would use a choreography editor which would generate WS-CDL and then either a centralized or distributed engine would carry out the necessary monitoring, reporting and rule-enforcement as part of the choreography specification.

In this chapter we seek to further our modelling of web service interactions through two viewpoints. Firstly, we examine the *interactions* within the choreography layer of web service compositions collaborating in a global goal. Secondly, through further behaviour analysis, we model the interaction sequences built to support multiple-partner conversations across enterprise domains and with a view of wider goals. We limit the scope of choreography analysis to that of interactions between compositions, and discuss the future work needed on state collaboration analysis in the evaluation sections of this thesis.

## 5.2 Modelling Web Service Interactions

### *5.2.1 Service Conversations*

Interactions of objects are carried out as part of a conversation. A conversation however, is defined differently depending on the context it is used within. For example, for a verbal conversation a definition is "*the use of speech for informal exchange of views or ideas or information*", applied to interacting service components there are several definitions, however the W3C consortium defines a conversation as "*...interfaces or public processes supported by a service. They differ from interfaces as defined by CORBA IDE or Java interfaces because they also specify the possible ordering of operations,*". A composition interacting with other compositions or services employs the use of a web service conversation protocol (Banerji, Bartolini et al. 2002; Fu, Bultan et al. 2004b). The W3C Web Service Conversation Language (WSCL) (Arkin, Askary et al. 2002) defines a standard for describing a conversation in terms of documents, interactions and transitions. Whilst a document is the body of a message used in conversation, interactions model the exchanges of

documents between two or more participants. A transition specifies an ordering between a source interaction and a destination interaction. A conversation defines how interactions can start and end depending on the goal of the conversation. For example, between a customer and an ordering service, there may be several interaction scenarios including a "Login" scenario and a "Purchase" scenario. A conversation therefore, also specifies the order in which these scenarios could occur (in a similar way to that of the hMSc in the design specification specifies the possible sequences of bMSCs discussed in Chapter 3).

To model these conversations in the context of web service compositions we perform an analysis process on all the implementation processes and use an algorithm as part of this analysis to semantically check and link partner process interactions. The algorithm uses as input partner service interfaces (in the form of a WSDL document) and the implementation models created in the initial implementation synthesis. The output of the composition modelling is a list of composition mapping requirements (as input to the mapping stage discussed in later sections) and information on non-interaction activities encountered and unmatched partner process references. These sub-actions of our approach are illustrated in Figure 5-2.

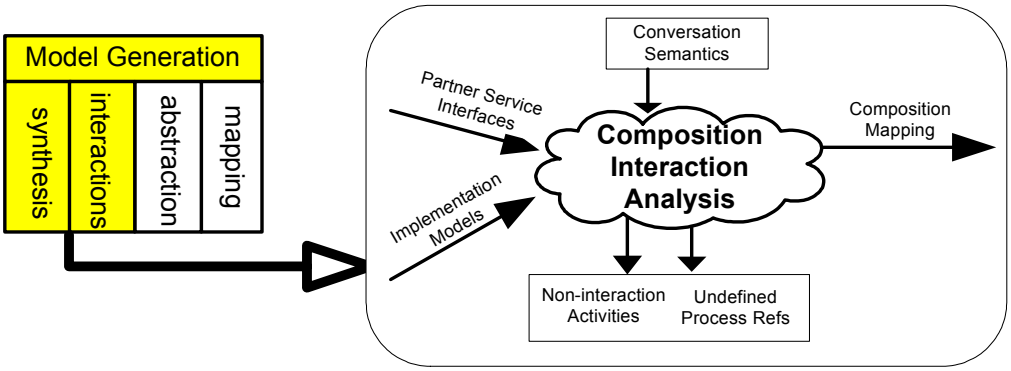



**Figure 5-2. Composition Interaction Analysis Sub-Action Diagram**

### *5.2.2 Service Partners and Roles*

A service conversation consists of a number of service partners and a service partner is considered in two ways. Firstly, the partner's service has a process role in the choreography of the service scenario (e.g. to provide a book ordering service). Within that, or another service process, a partner of a service may be considered to have one or many roles depending on what behaviour the partner's service provides. For example, a service partner

in a BPEL4WS composition may be labelled “Vendor”. This partner can be designated with one or many roles, such as in this case, both a “Seller” and a “Shipper”. The link to the partner and a list of their roles is defined by the client of the service composition. Therefore the service partner role semantics are defined locally to the process. The role indicator is used primarily to distinguish what the business process is referencing as part of the collaborative business service (for example, that the invocation from a buyer is in the buy context of conversation with another service acting as a seller).

### *5.2.3 Linking Composition Interactions*

As part of the BPEL4WS specification, abstract processes (described as part of the WSDL interface) can be defined which hide the private implementation of interactions within the process. These are not directly executable, but they can indirectly impose behaviour compliance upon private processes executed by the BPEL4WS engine server. Abstract processes may assist in execution however, as a BPEL4WS engine server validates and assures public protocol conformance of executing processes. Whilst abstract processes assist in this way, we scope this approach to the design of the core (private) process to capture the actual interaction and process of each BPEL4WS implementation, whilst using the abstract process as a reference point to link compositions together through the semantics of interface ports (described later). The core semantics of BPEL4WS, as discussed in (Aalst, Dumas et al. 2003; Khalaf, Mukhi et al. 2003; Foster, Uchitel et al. 2003a) and by way of translation to FSP in Chapter 4, describes how the language provides interactions of web service compositions. The interfaces are a key link between compositional partners in collaborating service scenarios. To model interacting web service compositions there is clearly a need to elaborate our analysis of implementations by linking compositional interactions based upon:

- activities within the process
    - identifying invocation style (rendezvous or request only)
    - identifying and recording the points at which interaction occurs
- the abstract interface
    - linking between the private process activities and the public communication interface declared in the abstract WSDL service description

To model the semantics of linking interactions between processes requires a mapping between activities in each of the processes translated (using the translation rules described in

Chapter 4) and building a *message port connector* for each of the interaction activities linking invoke (input) with receives, and replies (output) and with the returned message to an invoke. In addition to the executable process of BPEL4WS, the specification's abstract process defines PartnerLinkTypes, which are used as a class or type of relationship between a web service partner's invocation and a corresponding receiving service partner port. These link types are then referenced in composition implementations to distinguish service interactions between two or more partners. For example, the linking and dependency of service partners, roles, service interface (WSDL) and the executable composition (BPEL4WS), along with where a modelling port connector is positioned is illustrated in Figure 5-3.

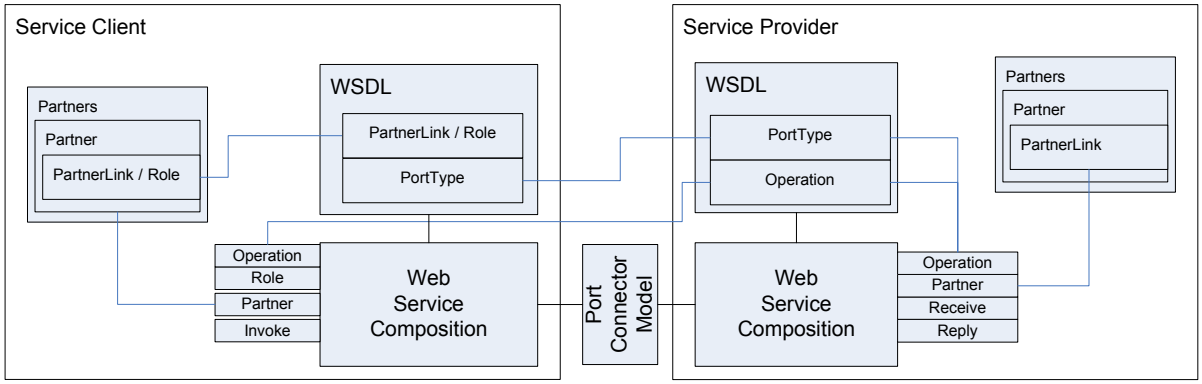



**Figure 5-3. Service Partners, PartnerLinks and Roles in Composition Linking**

In BPEL4WS, a partner link type characterizes the conversational relationship between two services by defining the "roles" played by each of the services in the conversation and specifying the portType provided by each service to receive messages within the context of the conversation. Figure 5-4 illustrates the structure of a partner link type declaration. BPEL4WS utilises the extensibility mechanism of WSDL (as in version 1.1). To define partnerLinkType as a new definition type to be placed as an immediate child element of a <wsdl:definitions> element in all cases. This allows reuse of the WSDL target namespace specification and, more importantly, its import mechanism to import portTypes. For cases where a partnerLinkType declaration is linking the portTypes of two different services, the partnerLinkType declaration can be placed in a separate WSDL document (with its own targetNamespace). The services with which a business process interacts are modeled as partner links in BPEL4WS. Each partner link is characterized by a partnerLinkType. More than one partner link can be characterized by the same partnerLinkType. For example, a certain procurement process might use more than one vendor for its transactions, but might

use the same partnerLinkType for all vendors. Figure 5-4 also illustrates the basic syntax of a partner link type declaration.

The fundamental use of endpoint references is to serve as the mechanism for dynamic communication of port-specific data for services. An endpoint reference makes it possible in BPEL4WS to dynamically select a provider for a particular type of service and to invoke their operations. BPEL4WS provides a general mechanism for correlating messages to stateful *instances* of a service, and therefore endpoint references that carry instance-neutral port information are often sufficient. However, in general it is necessary to carry additional instance-identification tokens in the endpoint reference itself. A partner link represents a conversational relationship between two partner processes; relationships with a business partner in general require more than a single conversational relationship to be established. To represent the capabilities required from a business partner, BPEL4WS uses the `partner` element. A partner is defined as a subset of the partner links of the process, as shown in the example below. Partner definitions are optional and need not cover the entire partner links defined in the process. From the process perspective a partner definition introduces a constraint on the functionality that a business partner is required to provide. In this way, the same partner (e.g. a Vendor), may provide two roles in a set of interactions within a process. For example, a vendor could be the "seller" and the "shipper". Figure 5-4 also illustrates the syntax of a partner element.

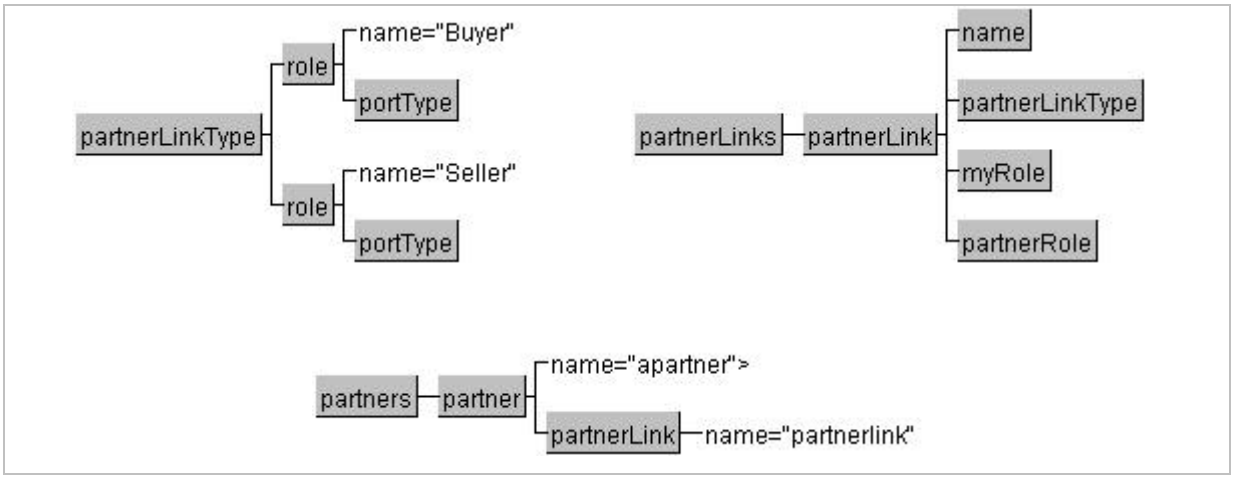



**Figure 5-4. PartnerLinkType, PartnerLink and Partner construct forms**

### *5.2.4 An Interaction Modelling Algorithm*

The physical linking of partnerlinks, partners and process models is undertaken as follows. For each invocation in a process, a messaging port is created. BPEL4WS defines communication in a synchronous messaging model. BPEL4WS process instance support in the specification specifies that in order to keep consistency between process activities, a synchronous request mechanism must be governed. The synchronous model can be formed by the following process.

```
For each composition process
     For each process invoke service activity
       Get invoke activity local partner
       Lookup partnerlink using local partner
       Get porttype using partnerlinktype
       For each process interface definition
            Lookup porttype using activity portype
            Store matching partner
            Lookup partner operation
       End For
       If invoke activity is in rendezvous style
           Add invokeoutput action to activity model
           Build reply-invokeoutput port
       End If
       Build invoke-receive connector partner labelling
    End For
End For
```

For every composition process selected for modelling we extract all the interaction activities in this process. As mentioned previously, interaction activities are service operation invocations (requests), receiving operation requests and replying to operation requests. In addition to an invocation request, we also add an invocation reply to synchronise the reply from a partner process with that of the requesting client process. The list is then analysed for invocation requests, and for each one found a partner/port lookup is undertaken to gather the actual partner that is specified in a partnerlink declaration. To achieve this, a partner list is used and the partner referenced in the invocation request is linked back to a partnerlink reference. The partnerlink specifies the porttype to link operation and partner with an actual

interface definition. To complete the partner match, all interface definitions used in composition analysis are searched and matched on porttype and operation of requesting client process. This concludes the partner match. A port connector bridge is then built to support either a simple request invocation (with no reply expected) or in "rendezvous" style, building both invoke/receive and reply-invokeoutput models. This supports the model mapping. The sequence is then repeated for all other invocations in the selected composition process, and then looped again for any other composition processes to analyse. We therefore specify an algorithm that will enable mechanical linking between activities, partners and process compositions. The algorithm is illustrated in Figure 5-5 with a flow diagram. The algorithm supports a mechanical implementation of linking composition processes together based upon their interaction behaviour. Two build phases are required as part of the algorithm, being that of building a reply-invokeoutput port and invoke-receive connector between partnered processes.

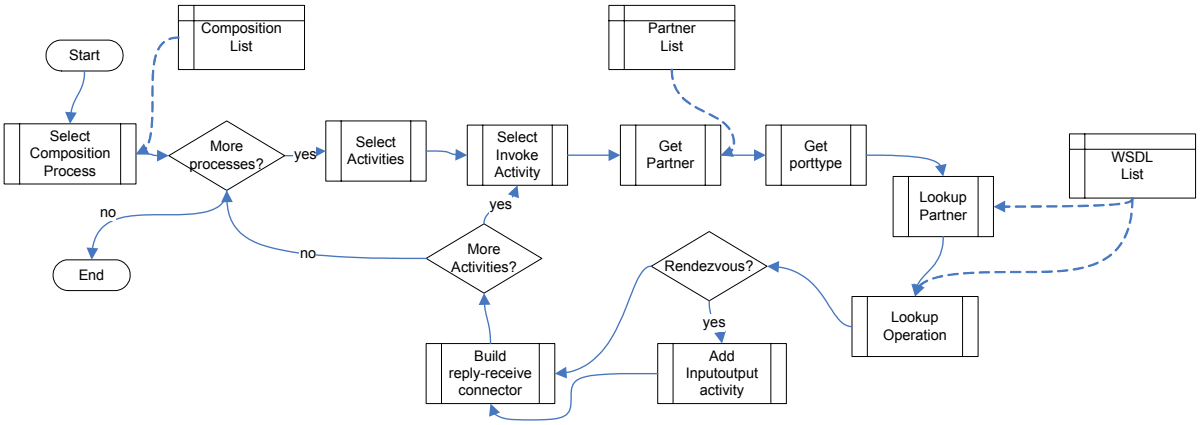



**Figure 5-5. Flow-chart of algorithm for Modelling Composition Interactions**

In summary, the algorithm described provides a port connector based implementation of the communication between two partner processes. Where multiple partner communication is undertaken in a composition, a port connector is built between each instance of a message (and optionally a reply if used in rendezvous interaction style). When each process is being translated in the synthesis step of this work's approach, the viewpoint changes, but the activities are synchronised in parallel. For example, an invoke is received by a partner process. The receive activity is viewed, when translating the partner process, as a new connector, but synchronised with the invoke connector of the calling process. We elaborate on how these connectors are formed in the following subsections.

## 5.3 Building Interaction Models

The activity of building port connectors for our integration mapping is based on the basic concept of message passing in the formation of web service composition communication. Messages can be sent directly to their destination partner process or indirectly via some intermediate entity. Modelling these different types of messaging style has been considered in (Magee and Kramer 1999). The essence of this work is that messages are passed through *channels*. A channel *connects* two and only two processes, in which a single process sends to a channel and a destination process can receive from a channel. The term connector is used to symbolise that a one-to-one channel is used in process synchronisation. A connector is the implementation between port and channel, in that a sender port is connected to a sender-receiver channel. An example between two processes is shown in Figure 5-6.

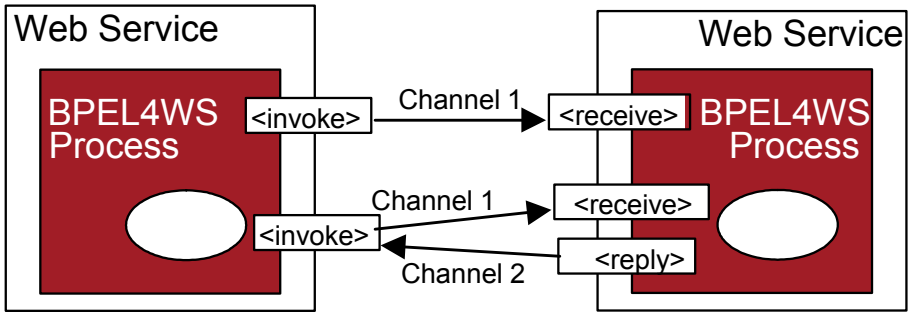



**Figure 5-6 Web Service Composition and Port Channels**

Applied to the context of web service composition processes, there are two views that process connection can take. Let us consider this by example, viewed from either between the composition processes or externally at the higher level web service communication.

### *5.3.1 Composition Process Interactions*

With the view of composition process actions, a process makes a request by passing an invocation message to a partner process. A composition process is sequenced such that it either expects a message from a sender at a particular state or it rejects a request if this violates the process sequence (Figure 5-7). The sender process will block execution of its current process *thread* of the invocation until a response is received. The process may continue processing if the invocation is part of a concurrent execution thread (for example see Flow statement of BPEL4WS in section 4.3.2). As described in the beginning of this chapter, in BPEL4WS there are two styles of invocation being that of rendezvous or invoke

only. The latter does not expect a partner service reply to be synchronised with the invocation, rather, it may suggest that the invoking process will expect a reply at a later part of the process execution. This would be synchronised on a receive activity with a partner process invocation.

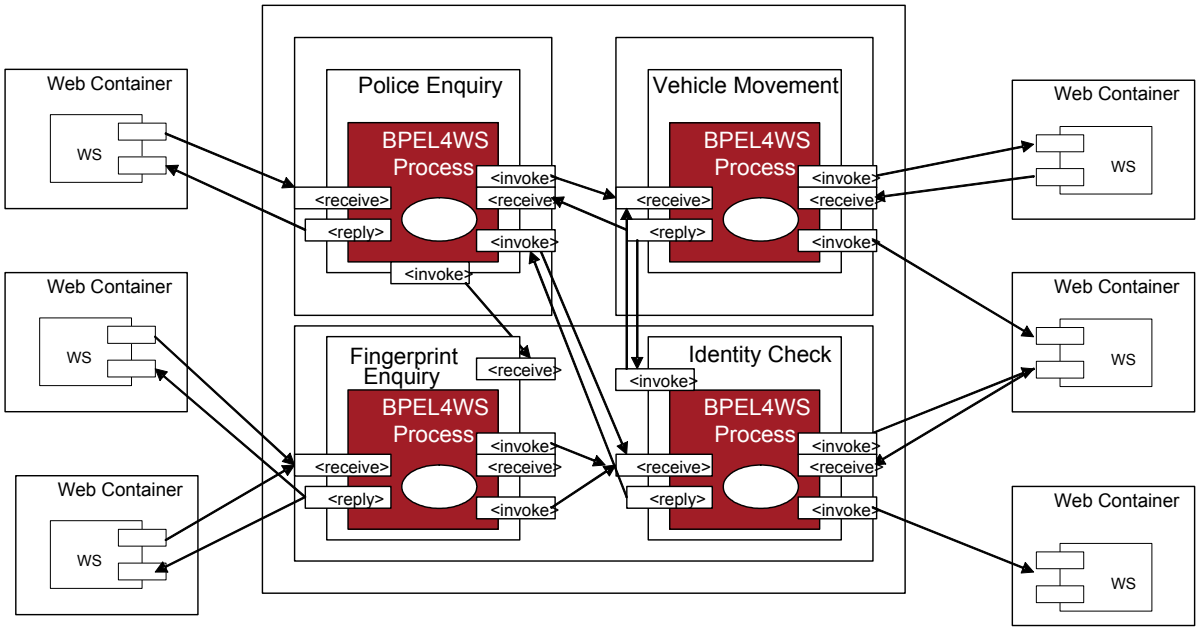



**Figure 5-7. View of Multiple Web Service Compositions Interacting in a Police Enquiry Scenario**

### *5.3.2 Connecting a Set of Processes*

To consider synchronising the connector models between composition partner processes we firstly examine how an unsynchronised communication model is represented. Individual processes can be modelled as a set of partner interactions, with an inner process defining its internal behaviour around these partner interactions. A process model commonly used to illustrate partnered compositions is that of a business Market Place, where a buyer and seller negotiation interactions are maintained and evaluated through a core process. An example definition of requirements for these set of services is described with a related context diagram illustrated in Figure 5-8. Transition semantics are labelled using the construct name (invoke or receive), *partner* (seller) name, partner process name (marketplace) and by the *operation* being requested (e.g. offer a product). It is worthy to note that this is an extended labelling scheme from that of which was described in section 4.2.1. These provide us with a set of labelled process transitions, such as "*invoke_seller_marketplace_offer*". If there is more than one invocation in the seller process, then this can be sequentially numbered. Equally, the receive activities in a Marketplace process example gives as an example translation of "*receive_seller_marketplace_offer*".

The marketplace provides three stages to a negotiation. Firstly, a product may be either offered or requested. The message is passed from the seller or buyer role respectively, and is received by the marketplace service. Once a request is received, the marketplace instantiates a new transaction and awaits for either a seller or buyer to offer or request a similar product. This process matches a seller to a buyer. A seller cannot be matched to another seller, and equally a buyer cannot be matched to another buyer. When a match is recognized, the second stage is undertaken. The second stage of the negotiation is to receive initial prices from the partners, for when satisfied, allows the workflow to proceed to the third stage. The third stage provides an iterative negotiation of prices, with each partner able to specify a price and then place agreement as to whether a deal is made or terminated.

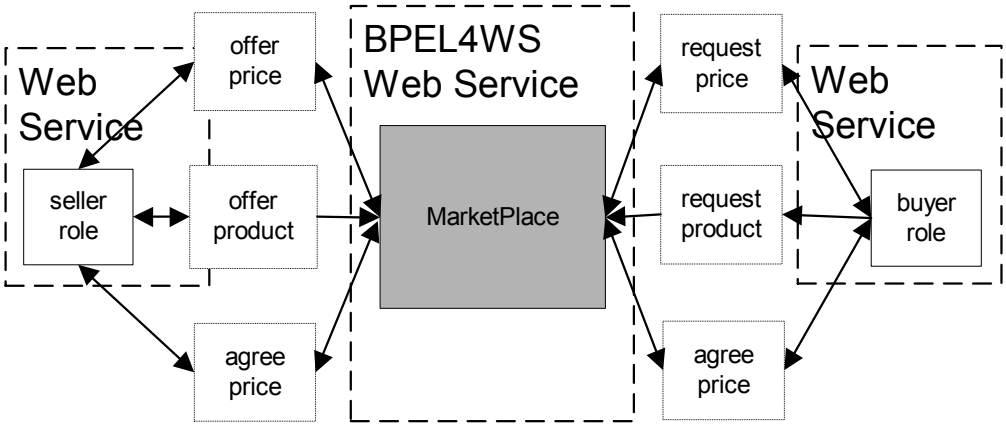



**Figure 5-8 Scenario and Diagram for a MarketPlace Service Composition**

The composition generates three process compositions, one for each of the partner processes in the scenario. The FSP for these processes is illustrated in Figure 5-9. Note that here we have simplified the FSP generated, by including only the activities for interaction and not the logic for conditional selection of when a price is agreed.

### *5.3.3 Messaging Port Connector Models*

To build connected composition interactions, port connector channels are used for each of the invocation styles between two or more partnered compositions. The algorithm is used from the viewpoint of a process composition at the "centre of focus", that is, the one in which initial process analysis is being considered. The interface of subsequent partner interactions is used in the algorithm to obtain a link between two partners and an actual operation. For example in Figure 5-10, two BPEL4WS process interact using both a request only invocation (Channel A) and a Rendezvous style (Channel A and B).

```
/* FSP code for seller service composition */
      // SELLER PROCESS: FLOW MODEL OF TWO OFFERS
      SI1 = (invoke_seller_marketplace_offer->END).
      SR1 = (invoke_seller_marketplace_reply->END).
      SI2 = (invoke_seller_marketplace_offer2->END).
      SR2 = (invoke_seller_marketplace_reply2->END).
      SSEQ1 = SI1; SR1; END.
      SSEQ2 = SI2; SR2; END.
      ||S_BPELModel = (SSEQ1 || SSEQ2).
/* FSP code for buyer service composition */
      // BUYER PROCESS: SEQUENCE MODEL OF A REQUEST AND REPLY
      BI1 =(invoke_buyer_marketplace_request->END).
      BR1 =(invoke_buyer_marketplace_reply->END).
      BSEQ1 = BI1; BR1; END.
      ||B_BPELModel = (BSEQ1).
/* FSP code for marketplace service composition */
      // MARKETPLACE PROCESS: RECEIVING AND REPLYING TO BUYER/SELLER
      MPSI1=(receive_seller_marketplace_offer-> END).
      MPSR1=(reply_seller_marketplace_offer->END).
      ||MPSBR = (MPBSI1 || MPBSR1).
      MPBR1=(reply_buyer_mp_request -> reply_seller_mp_offer -> END).
      MPBSEQ1 = MPSBR; MPBR1; END.
```

**Figure 5-9. FSP Code for Buyer and Seller Interactions with a MarketPlace Process**

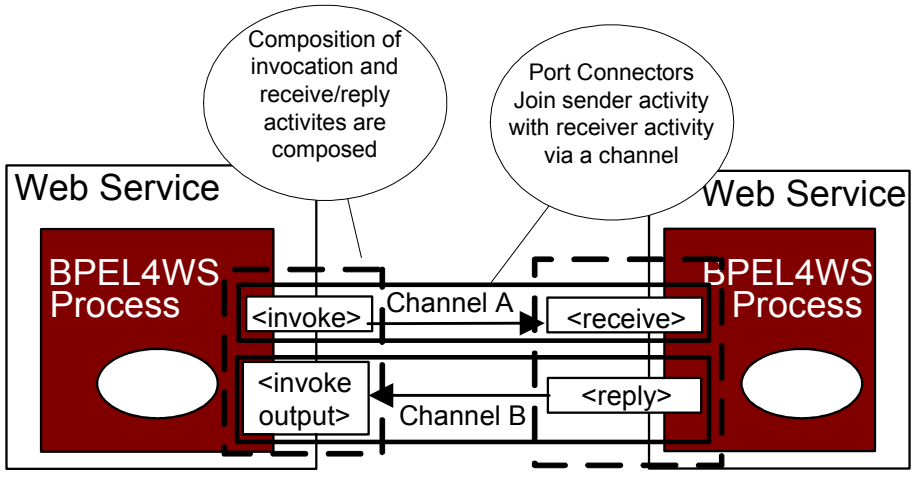



**Figure 5-10 Channels and Interaction Activities of Web Service Compositions**

Our model of interactions using channels is based upon the interaction state and not on the messaging architecture used for transport. In this way, we do not consider synchronous against asynchronous messaging models for modelling the communication flow between

compositions. The model produced from analysis of the compositions is from the viewpoint of the composition performing as part of a role in choreography. This makes the model an abstract view of interactions for the purpose of linking invocations and not on the actual order of messages received by the process host architecture (synchronous and asynchronous messaging models for web services can be referred to in (Fu, Bultan et al. 2004)).

#### 5.3.3.1 Request only invocation (Channel A)

Web Service compositions specified with the invoke construct (see section 4.2.2) and only an *input* container attribute declare an interaction on a request only basis (there is no immediate reply expected). More generally this requirement is for a reliable message invocation without any output response from the service host (other than status of receiving the request). The message synchronisation for this port model is listed below with an example number of messages ranged from 1 to 3. The model for this is illustrated as an LTS in Figure 5-11.

**/* FSP code for request-only service invocation model */**

```
range MSG = 1..3 /* no of msgs */
CHANNELA = (invoke_input[v:MSG] -> receive[v] -> CHANNELA).
||REQONLY_PORT = (CHANNELA).
```

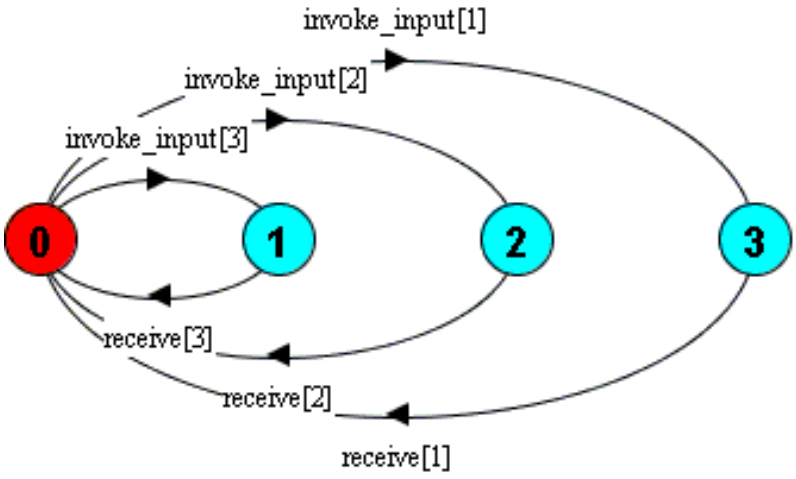



**Figure 5-11 LTS of Model for Request Only Port Connector**

The process "ChannelA" is defined with two transitions, that of an invoke_input and a receive, followed by a recursive transition back to the start of the process. Note that this

model simply defines a *send* and *receive* mechanism, whereby the transitions are labelled invoke_input and receive respectively.

#### 5.3.3.2 Rendezvous style invocation (Channels A and B)

"Rendezvous" (Request and Reply) invocations are specified in BPEL4WS with the <invoke> construct, with both *input* and *output* container attributes. To model these types of interactions, we use a generic port model for each process port. A synchronous messaging model in web services compositions (such as BPEL4WS) requires an additional activity of an "input_output" to link a reply in a partnered process to that of the caller receiving the output of the invoke, however, this is necessary only if the invocation style is that of rendezvous. The message synchronisation for this port model is listed below with an example for a single message composition in Figure 5-12.

**/* FSP code for rendezvous service invocation model */**

```
range MSG = 1..1 /* one message example */
CHANNELA = (reply[v:MSG] -> invoke_output[v] -> CHANNELA).
CHANNELB = (invoke_input[v:MSG] -> receive[v]-> CHANNELB).
||RENDEZVOUS_PORT = (CHANNELA || CHANNELB).
```

A corresponding model for this port connector is illustrated as an LTS in Figure 5-12.

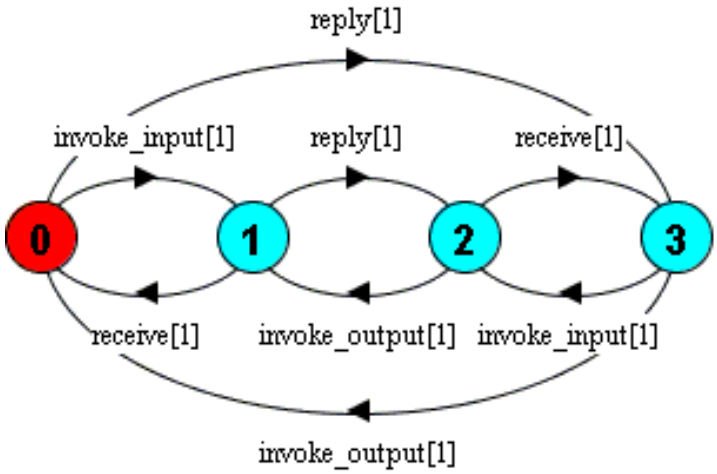



**Figure 5-12. LTS of Model for Synchronous Rendezvous Port Connector**

5.3.3.2 Mapping Process Activities to Port Connectors

The next step in the port connector modelling process is to map the activities of the BPEL4WS process to the port connector activities. This is achieved using the semantics of BPEL4WS for the interaction activities discussed earlier and replacing the port connector activities appropriately. The invoke activity in BPEl4WS is mapped from the client process to the invoke_input action of the port connector – this represents the initial step of a request between web service partners. The associated receiving action of the BPEL4WS partner process is mapped to the receive activity in the port connector. The reply from the partner process to the client process is mapped to the reply in the partnered process. Both receive and reply activities in the BPEL4WS are discovered as part of the interface analysis described in section 5.2.4 . Figure 5-13 lists the mapping explained here.

| WS Interaction | Port Action | BPEL4WS Action (example) |
|---|---|---|
| Invoke (Client) | Invoke_input | `invoke_seller_marketplace_offer` |
| Receive (Partner) | Receive | `receive_seller_marketplace_offer` |
| Reply (Parnter to Client) | Reply | `reply_marketplace_seller_offer` |
| | Invoke_output | `output_marketplace_seller_offer` |

**Figure 5-13. Mapping Activities Between Port Connector and BPEL4WS for A Seller and Marketplace Example**

With both of the invocation model types, the connection interaction for invoke activities in BPEL4WS can be modeled effectively using transition links for send, receive and reply processes in FSP. The task of modelling the invocation process and port is completed by using the re-labelling feature of FSP linking the appropriate activities between process and port. At this point the interactions are mapped into a connector yet it is still an unsynchronised set of activities if aligned with the main BPEL4WS process models. Therefore we need to compose these activities with those specified in the BPEL4WS processes. To complete the modelling of the compositions, we specify an architecture model composing the previous models for seller, seller port, marketplace, buyer port and buyer processes. An example of applying this mapping to the Seller and Marketplace processes described earlier produces the FSP code as illustrated in Figure 5-14.

.

```
/* FSP code for building model of port connector in marketplace example */
 range MSG = 1..1
 PORT_INVOKE=(invoke_seller_marketplace_offer[v:MSG]
 ->receive_seller_marketplace_offer[v]->PORT_INVOKE).
 PORT_REPLY=(reply_marketplace_seller_offer[v:MSG]
 ->output_marketplace_seller_offer[v]->PORT_REPLY).
 ||PORT_MODEL = (PORT_INVOKE || PORT_REPLY).
/* mapping process and port connector transitions for request only conversation */
 / {invoke_seller_marketplace_offer / invoke_input,
   {receive_seller_marketplace_offer / receive).
/* mapping process and port connector transitions for rendezvous conversation */
  / {reply_marketplace_seller_offer / reply,
   {output_seller_marketplace_offer / invoke_output).
/* code for composition of processes and port connectors */
 ||CompArch = (Seller_BPELModel || Seller_Port || MP_BPELModel
 || Marketplace_Port).
```

**Figure 5-14 FSP Code segments for mapping activities**

A partial set of interactions is modelled in Figure 5-15 as an LTS model for the seller and buyer with the Marketplace process. The labels in this model have been shortened for presentation purposes and clarity. The result of providing a complete model of interactions, as part of a conversation, is that a choreography specification can potentially be used as input to a verification process. This verification process can highlight inconsistencies on a series of compositions and their interaction behaviours implemented. We term this type of verification as “compatibility checking”, in the way that focus can be placed on how the compositions interact and if they are suitable to be used for choreography as part of a business process. Although the choreography specifications are still being created, such as in the case of WS-CDL, these additional features will be key part in how choreography will be undertaken.

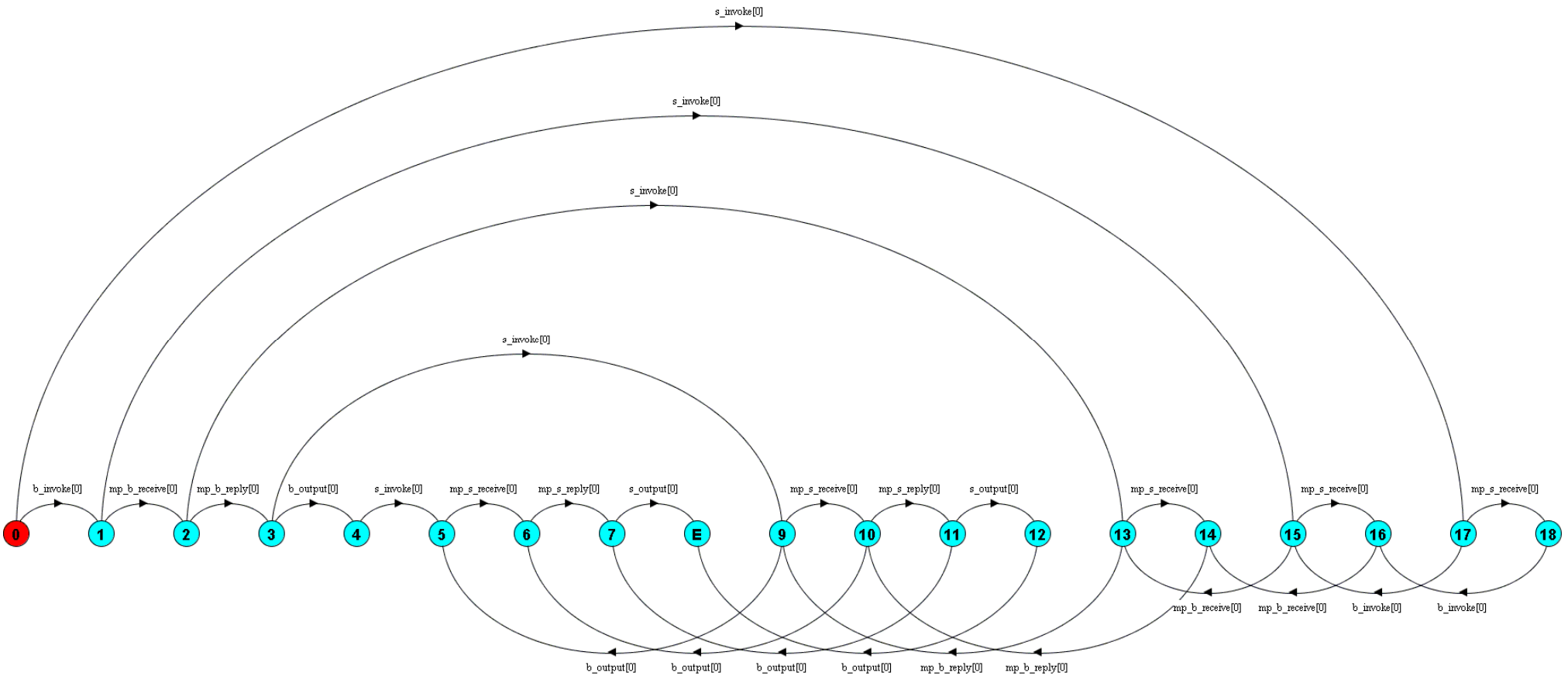



**Figure 5-15 LTS for Partial Set of Interactions between Seller, Buyer and Marketplace Compositions**

## 5.4 Summary and Discussion

In this chapter, we have described an elaboration of composition models to support a view of interacting web service composition processes extending the mapping from BPEL4WS to FSP discussed in Chapter 4, and introducing web service interfaces for use in modelling between services. The ability to model these conversations is important to discovering how web service interactions fulfil a choreography scenario and if the conversation protocol implement (by way of interaction sequences) is compatible with that of partnered services. In essence, our view of modelling has moved from analysing a local process, or in other words a single composition, with that of other services and their interactions. With both the local behaviour and mappings between compositions defined, we now have a sufficient model to perform analysis of service interaction for behaviour properties. The approach to verifying and validating these properties is discussed in Chapter 6. In terms of the approach overview discussed in section 1.2, we have introduced the highlighted parts as illustrated in Figure 5-16.

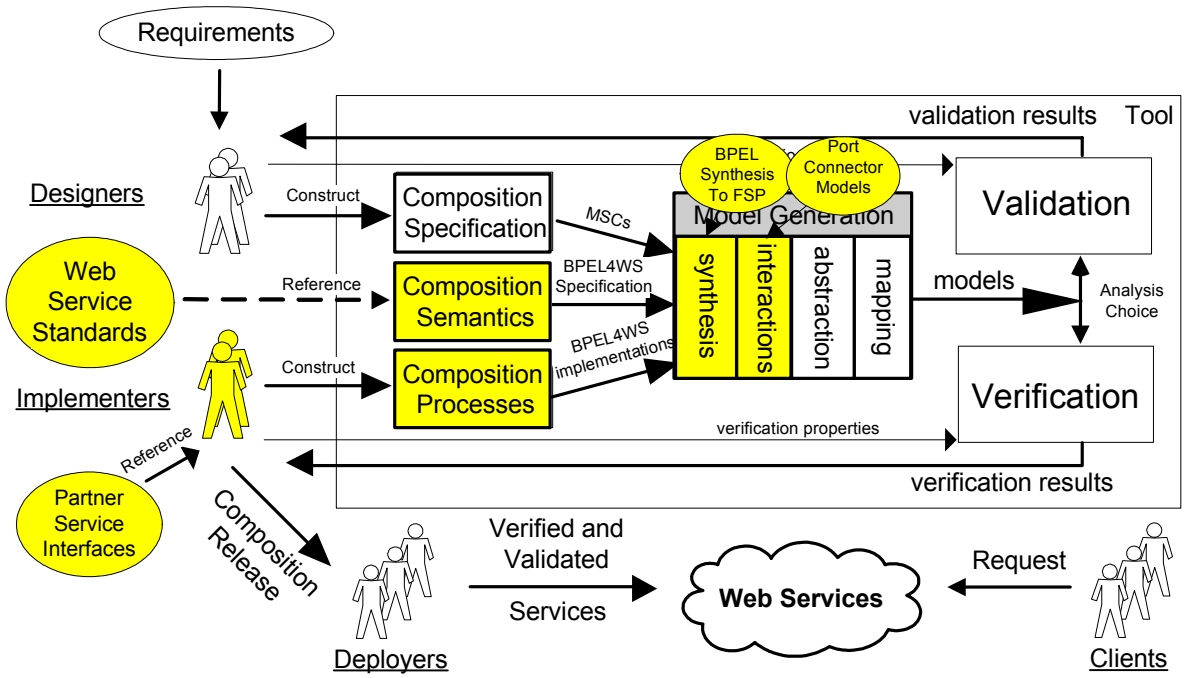



**Figure 5-16. Elements of the approach discussed in chapter 5.**

# Chapter 6

# Analysis for the Service-Oriented Model

*"Contrariwise," continued Tweedledee, "if it was so, it might be, and if it were so, it would be; but as it isn't, it ain't. That's logic!"*

(Lewis Carroll, Alice In Wonderland)

---

In the previous chapters we have described an approach to design, implement and model web service compositions with respect to their specification processes and interactions. These models provide a representation that can be used to perform verification and validation analysis using formal model checking techniques. In this chapter we discuss this analysis and how software process model checking techniques are applied to the web service composition models to assist designers and implementers assess the correctness of compositional behaviour. This chapter brings together the models of design and implementation for a service-oriented model and evaluates their behaviour in the form of obligations.

## 6.1 Analysis of Web Service Compositions and Choreography

### *6.1.1 Approach to Analysis of the SOM*

In the introduction to our work, we described verification of processes to be used to identify parts of the web service composition's behaviour that have been implemented incorrectly, or

perhaps have unforeseen interactions. This aims to satisfy such questions as; does the implementation match the requirements and was the *process built correctly*? Additionally, we described validation as a mechanism to clarify the understanding of requirements against that of the implementation and that the result of validation is to ensure that the *right process was built*. In chapter 2 we described the policies and goal states defined of the SOM (section 2.4.5) related to service objectives and obligations, and why the analysis of web service compositions (and its related use in choreography) facilitates the construction and general engineering of web services. The ability to perform verification and validation between implementations and design, and within the process compositions themselves, is a key requirement of the web services architecture specification (Booth, Haas et al. 2004). We portray this analysis through an approach to compare the design specification models against that of implementation models (and vice-versa), and report back on obligations specified by either service designer or implementer that result in implied scenarios or progress violations (as illustrated in Figure 6-1).

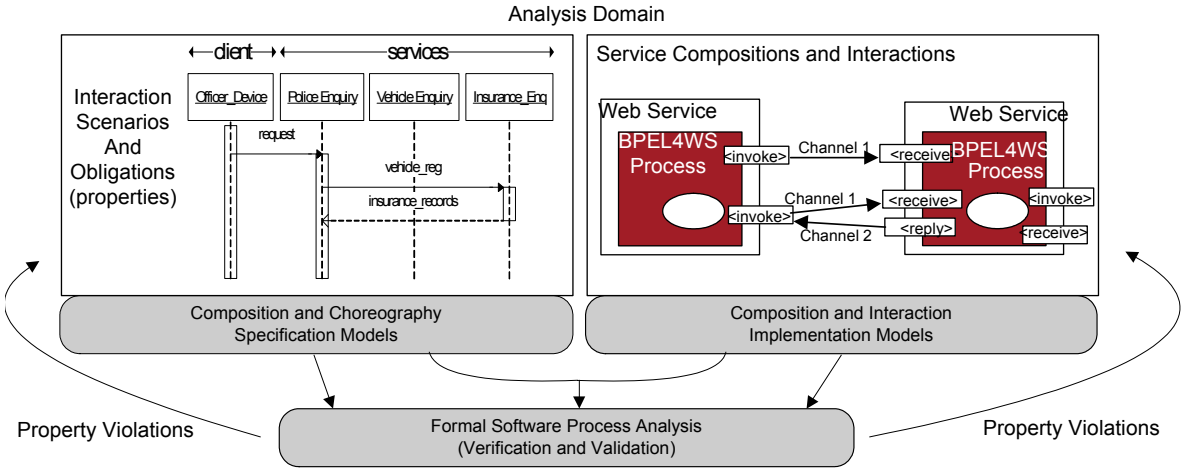



**Figure 6-1 Approach to analysis of Service Specifications and Implementation Models**

The use of MSCs to specify obligations of a service composition or choreography provides an accessible method for end-users to describe the sequence of interactions necessary to fulfil their requirements. Whilst there exist other suggestions for describing web service policy standards through such work as WS-Policy (Bajaj, Box et al. 2004), which provides a general framework to specify constraints on web service communication, at the time of writing this work there is little on specifying interaction constraints in policies for decisions about anything other than security and resource access control permissions. Perhaps the nearest exposure of obligations as part of a general web service policy framework is the eXtensible Access Control Markup Language (XACML) (Anderson, Nadalin et al. 2004)

which defines pre- and post conditions in terms of activities that a service participant must have carried out prior to or after requesting a service. Due to the lack of available standards in this area, we assume that a general policy of desirable interactions can be defined sufficiently in a design specification (such as we used MSC specifications in Chapter 3). Additionally, for the purpose of our verification approach, we assume that a stated policy is to be upheld within the choreography model (regardless of duration of existence) relating to the interactions permissible in a given scenario of a businesses goal. Collectively gathering scenarios and building a set of specifications forms a choreography verification set with which properties may be defined to analyse how implementations fulfil the set of specifications given in these sets.

As an example, if a service choreography is defined between three different services and their tasks, the properties we are interested in is that the implementation has observable behaviour which satisfies the choreography rules and requirements. We abstract away from the storing and passing of state values between choreography scenarios – however, the interaction state (whether a service request, request received or reply is given) is considered for state transition analysis. The process in the web service context is that of a Web Service Composition's tasks (or more generally seen as any service's set of tasks). For verification we analyse how web services, a service's tasks (inner process), and actions specified in service compositions (referred to as activities in our models) fulfil several areas of analysis. Equally we also provide a mechanism for confirmation of the behaviour exhibited by a modelled composition, and allow the analyst or engineer to validate that the processes created are indeed the required behaviour for a solution to the requirements. In this section we discuss the techniques for both of these methods and provide examples of undertaking them.

### *6.1.2 Techniques used in the Analysis*

Verification is achieved through the use of formal software process model checking techniques, but we evaluate specific topics of our approach for web service compositions by wrapping and applying these techniques under the notions of deadlock freedom and safety and progress property analysis. Firstly, we can check the behaviour of a composition or choreography is *deadlock free* (i.e. that there are no states with no outgoing transitions). In a finite state model (such as the models we produced from design specification and implementations in chapters 3 and 4), a deadlock state is a state with no outgoing transitions

in these models. A process in such a state can engage in no further actions. The deadlock states we are interested in are those that arise from a parallel composition of concurrent activities in a single composition, a number of interacting compositions and one or many compositions against that of their design specifications. This analysis can be performed by input of a series of processes and using a parallel composition to build an architecture model. A breadth-first search of the model is then performed and trace results can be obtained of the activities taken from the start state to the state at deadlock. An example of a deadlock state in web service choreography is that two services are waiting to receive a message from each other. The processes of these services are clearly in a deadlock situation where one is awaiting the other, and will never transition past this state.

Secondly, we can use *safety property* checking techniques to determine if given model properties are satisfied in one or many compositions. Safety properties are distinguishable from deadlock states in that they result in an error state – identified uniquely within a trace of the given model analysed. For example, if a safety property is composed with a given model, a safety check will result in error if the property is not preserved in the composed model. Safety properties used on complex systems are usually better stated as what is required, rather than stating what is not required (Magee and Kramer 1999). Thirdly, we can use *progress properties* (one of several liveness property analysis types) to assert that whatever state a process is in, it is always the case that a specified activity will eventually be executed. Progress is the opposite of *starvation*, the name given in a concurrent programming situation in which an action is never executed. Progress properties are simple to specify and are sufficiently powerful to capture a wide range of liveness problems in concurrent processes.

For validation we provide additional mechanisms for designers to validate web service composition design specifications through simulation and animation. Assertions are used to identify properties for service interactions in a simulation of the composition, again from the model built in previous chapters. Animation is also provided, whereby designers are able to walkthrough scenarios of the composition, and selectively choose different paths of execution to check requirement scenarios are fulfilled in the given design or implementation. To perform direct process analysis we use model checking techniques (such as deadlock, safety and progress properties) to specify the checks we wish to perform against process models. Whilst deadlock and safety can be performed generally (through direct instruction

to an analyser) safety and progress can also be applied subject to those properties of interest or required by an end-user (for example, to directly assert whether a system can perform a series of activities or that the system exhibits to necessary behaviour to complete and fulfil a property). These more "end-user" properties are considered related to policy verification and validation. Preparation for property checking using such concepts is discussed in the next section of this chapter.

## 6.2 Preparation for Analysis

Whilst the model synthesised from the MSC design of a web service composition (illustrated in chapter 3) is focused on service interactions, the implementation may also include additional representation in the form of conditions and constraints (also known as *links* in BPEL4WS). The naming scheme of the MSC message interactions is also likely to be differing to that of the implementation specification naming standards for interaction activities. It is necessary to abstract these additional representations away from the implementation by hiding or mapping them in the model composition. The common elements of the models produced for both the design and implementation of web service compositions are the interaction activities. In essence, our preparation focuses on abstraction, applying a concise labelling scheme to the implementation specification, hiding implementation specific activities which are not based upon direct interaction messages and identifying a mapping between activities specified in the implementation and the design. These collectively fall under some common abstraction preparation activities.

### *6.2.1 Types of Preparation Activities*

The techniques to apply for this abstraction against our earlier models is categorised under the area of abstraction with reference to model checking. Abstraction is used to reduce the complexity of a model by including only the parts of the system necessary for the issues being investigated (Frantz 1995). Engineers and scientists routinely use abstraction in problem solving. Amongst the abstraction techniques used, the following are prominent in these works. *Variable elimination* removes parts of a system that are not relevant to the properties and behaviour to be demonstrated or proven (Heitmeyer, Kirby et al. 1998; Bharadwaj and Heitmeyer 1999). Irrelevant variables can be identified by looking at dependencies and then removed. *Enumeration* is a technique that represents the range of the values of a continuous variable as a set of abstracted terms. The general approach is to partition the range of the variable into a set of subparts. *Reduction* is a technique that

decreases the size of individual parts of a system while preserving relevant characteristics needed to verify the behaviour of the system. The reduction choices are made based upon what behaviour is to be investigated; this is the modelling perspective. Performing abstraction by using non-determinism involves allowing arbitrary choices at decision or transition points in a model. In this technique, the details in the logic used to make a choice among alternatives are ignored. *Grouping* is an explicit many-to-one mapping of variables or entities into a single descriptor. The issues the model is being used to explore as described in section 6.1 guides the grouping. The goal of this technique is to group entities into a smaller set or to regroup entities to facilitate modelling and analysis. *Decomposition* is a technique for systematically partitioning a system into structural or functional components. While this approach is not traditionally considered an abstraction technique, it is effective in helping to make decisions about what is needed in a model and what is not and in understanding individual components of a system and their interrelationships.

### *6.2.2 Preparation for Composition Abstraction and Mappings*

Our approach to modelling has at its core; synthesis, mapping and abstraction. We describe the model abstraction steps for web service compositions as consisting of a series of tasks given input from the synthesis of initial design and implementation models and specifications and semantics of the processes modeled. The composition model preparation is illustrated in Figure 6-2.

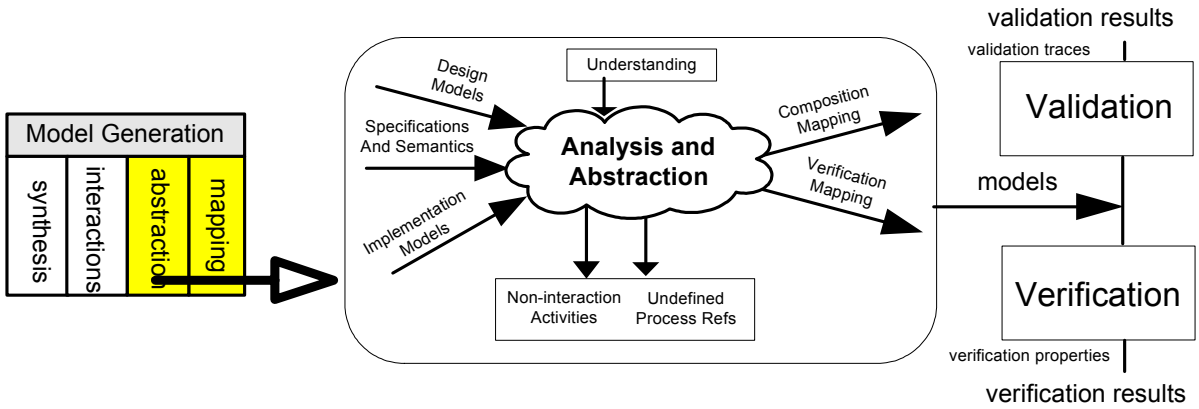



**Figure 6-2. Behaviour Refinement through Analysis and Abstraction**

The process of abstraction is decomposed in to analysis of non-interaction activities in implementation (reduction), joining actions between design and implementation through specification semantics (grouping) and linking interactions between compositions through service interface models (further mapping).

The output of the abstraction steps are enhanced models including mapping information that can be used for joining composition process models, reducing the activities which are considered in the behaviour analysis and verifying models of design against implementation models. The inputs to behaviour abstraction are the models from synthesis of a design and models from the translation of implementation discussed in the previous chapter. In addition to these models, the composition standards specifications and design model semantics are passed on through to assist in analysis of the elaboration techniques discussed later in this chapter. The service composition standards are formed from those used in the BPEL4WS specification - providing syntactic standards and suggesting semantics of how BPEL4WS processes are defined, implemented and formed and provide information on those activities which are candidates for reduction. The WSDL specification also provides semantics of service binding and linking with BPEL4WS partner interface extensions (as described in Chapter 5). We now discuss the steps in this abstraction within our approach, by firstly considering the refinement of translated models with reference to the semantic information from the specifications.

### *6.2.3 Sample Scenario for Verification and Validation*

In this chapter we refer to a simple yet illustrative composition example, being a simple service which receives a message from a client to log a message by way of an audit provider. The audit is an additional service which is invoked from an echo service provider. The initial requirements for this composition example are given in Figure 6-4.

**The Echo-Audit composition is carried out by three partners. The Echo client requests a message to the Echo service provider. In turn, the request is processed and an Audit Service is requested by the Echo service provider to record the message. The Audit service logs the message and returns a status to the Echo service provider. The Echo service provider acknowledges the Audit request. This can occur before the Audit Service returns the actual Audit status. The Echo client can continue with other actions once the reply is given from the Echo service provider.**

**Figure 6-3 Example scenario of activities in a Message Auditing Service Composition**

To consider how the design and implementation models generated through the modelling as described in chapters 3, 4 and 5.would be analysed for the example scenario given, we begin

by considering what the models exhibit that is not required in the analysis, such that we have a refined model.

## 6.3 Refining Composition Behaviour Models

### *6.3.1 Reduction of Implementation Specific Activities*

The BPEL4WS specification includes a subset of a traditional programming language, having structured and variable statements to define conditional process flow. Whilst some activities can be treated as primitive processing directives (such as assignments between variables in the process), they cannot be removed completely from the implementation process model as a different interaction model would be produced (e.g. when the next path of a variable expression is evaluated). This part of the preparation for verification specifies which activities can be abstracted to be verified in the BPEL4WS models, for the analysis of process interactions. These are identified as transitions in the BPEL4WS that are concerned with assignments, switch conditions, end actions and initiators. This reduction does not change the behaviour of the model, but hides the activities from the set used in model analysis. The alphabet of the model produced from the BPEL4WS translation will include assignments, switch statements and other conditional processing which is unlikely to be specified in a design. A sample alphabet of a BPEL4WS model produced prior to abstraction is illustrated in Figure 6-4.

```
Process:
   echostring_echoslt_echo
Alphabet:
   {invoke_audit_echo,invoke_audit_echo_reply,
   invoke_echoprovider_echo,invoke_echoprovider_echo_reply,
   receive_caller_echo,reply_caller_echo,
   assign.client.message,assign.audit.message,
   assign.audit.result, assign.client.result}
```

**Figure 6-4. Composition Implementation Alphabet before Reduction**

Notice that the alphabet includes activities starting with the invoke, receive, reply and assign constructs. In addition to the re-labelling operator (used in Chapter 5), the FSP language includes a *hiding* operator to conceal (and "reduce") a set of activities in a specified model so that they are not witnessed a process trace. To specify this in FSP, the \ operator is used.

The hiding for the Echo_Audit example is given as follows to illustrate abstracting the assignment constructs.

```
/* FSP code for hiding non-observable activities in BPEL4WS composition
model */
    \ {assign.client.message,assign.audit.message,
    assign.audit.result, assign.client.result}.
```

### *6.3.2 Grouping Design and Implementation Activities between Models*

The models produced as part of the design specification and BPEL4WS implementations may have different process alphabets. An alphabet in terms of a finite state machine is the string of symbols that are read by a machine. In machine state transition, the next state is determined using the current state and the symbol next in the alphabet (or the empty string). When the machine has finished reading, it is said to be in an accepting state, otherwise it is said to reject the string. Clearly, where activities are thought equivalent in the alphabet sets, these must match between design and implementation. The designer's choice of labelling interactions may have different forms of expression to describe each of the required activities as part of a service process composition requirement specification. In a similar way to mapping activities for interaction port connectors, as described in chapter 5, we are required to provide a mapping (or "bridge") between the activity alphabets compiled in the translation from the BPEL4WS source implementation to the message sequence chart model produced as part of the design composition steps. The mapping can be achieved by re-using the notion of re-labelling. FSP supports relational re-labelling. The relation operator applies a relation to a process, which can result in replacing many labels with a single label and replacing one label with multiple labels. The re-labelling relation is defined by a set of pairs. Each pair takes the form *newlabel/oldlabel*. Sets of labels and the replication construct permit the concise definition of the re-labelling relation. Re-labelling occurs before a parallel composition of processes – therefore before we combine the implementation with design processes (this is discussed in section 6.4.1).

An example of re-labelling activity labels from the synthesis of the MSC specifications and translation of BPEL4WS to FSP is given in Table 6-1. This example shows how the receive, invoke and reply activities contained within a partner process are mapped to the alphabet of

activities defined in a service composition design. The result is a model that both MSC and BPEL4WS reflect the same alphabet.

**Table 6-1 Mapping Activities as part of model abstraction**

| MSC Action | BPEL4WS FSP Action | Source Code for FSP Mapping |
|---|---|---|
| call_echo | receive_caller_echoprovider_echo | /{call_echo/ receive_caller_echoprovider_echo, call_audit/ invoke_echoprovider_audit_echo, reply_audit/reply_audit_echoprovider_echo, reply_echo/reply_echoprovider_caller_echo }. |
| call_audit | invoke_echoprovider_audit_echo | |
| reply_audit | reply_audit_echoprovider_echo | |
| reply_echo | reply_echoprovider_caller_echo | |

The tasks of abstracting implementation specific activities, labelling appropriately in service interaction activities, hiding implementation specific activities and mapping between implementation and design composition models provides a typical set of characteristics in abstracting models for model verification and validation (Gluch, Cormella-Dorda et al. 2001; Engels, Kuster et al. 2003).

### *6.3.3 Building an Architecture Model for Analysis*

A final preparation activity to perform analysis for verification and validation requires that these models are composed together to produce a model which represents a minimal, deterministic representation (Erdogmus 1997). A minimal model means that a trace in the original process leads to an END state if and only if the trace leads to an END state in a determinised process. The step combines the original synthesis of the composition design MSCs, the translation from BPEL4WS to FSP and the abstraction mappings described previously. An architecture model is produced by composing these models and mappings together. The models are tagged prior to compilation so that the FSP compiler performs the necessary abstractions. Figure 6-5 illustrates an example of FSP code for composing an architecture model for composition analysis. Note that the Client, Provider and Audit BPEL4WS architecture models have previously been translated and mapped with port connectors to produce the "BPEL_ArchitectureModel" process. The FSP code here composes the architecture models of MSC and BPEL4WS, with additional abstraction (a subset of the MSC and BPEL4WS activity mappings).

```
/* FSP code for architecture models of MSC and BPEL4WS compositions, specifying
mapping between models */
    ||MSC_ArchitectureModel = (Client || Provider || Audit).
    || BPEL_ArchitectureModel = BPELModel /
        {invoke.echo.provider/invoke_client_echo}.
    ||ArchitectureModel = (MSC_ArchitectureModel || BPEL_ArchitectureModel).
```

**Figure 6-5 FSP code for Refined Composition Architecture Model**

## 6.4 Analysis of Composition Behaviour Models

In this section we describe the verification part of the approach, which considers analysing the architecture models of the compositions produced in section 6.3, and specifying properties of interest to provide greater assurance that compositions hold the required behaviour. The essence of the tasks performed here is to give greater assurance of compositions correctness, through equivalence checking of implementation and design models (termed *trace equivalence*). The behaviour model abstraction tasks that have been discussed in previous sections of this chapter, illustrate how we deduce what is not required to be "observed" or "witnessed" in the trace equivalence verification – i.e. we specified parts that differed between design and implementation, and thus reduced the set of observable activities from the combined model. The result of which is used in model checking (in order to prove properties of the compositions) consisting of; *liveness* (paths contain positive transitions), *safety* (paths do not contain negative transitions), and specific compatibility reasoning in the implementation models. For example, that two partnered processes have sufficient *liveness* that the interfaces between the processes are fulfilled. The following inputs are required to support the verification process, and form the inputs to this step in the approach. Firstly, the original implementation and design models accompanied by the mappings through interaction analysis, and mapping through abstraction, are required as the model input to verification. Secondly, a set of analysis properties to check are required as pre-requisites to *checking* the models. These are discussed in the following sections as to the format and nature of these properties. We consider verification in terms of trace equivalence (checking the implementations against specification models), interface compatibility – ensuring that interactions following the web service conversation standards and that interaction activities are suitably placed in processes to provide progress of a process, and thirdly a mechanism to check general properties in terms of both safety and progress. The

output from this verification is a set of trace results which can be used to determine the cause of why any property violations occur, or indeed if a successful result is obtained. The steps described here are illustrated in Figure 6-6.

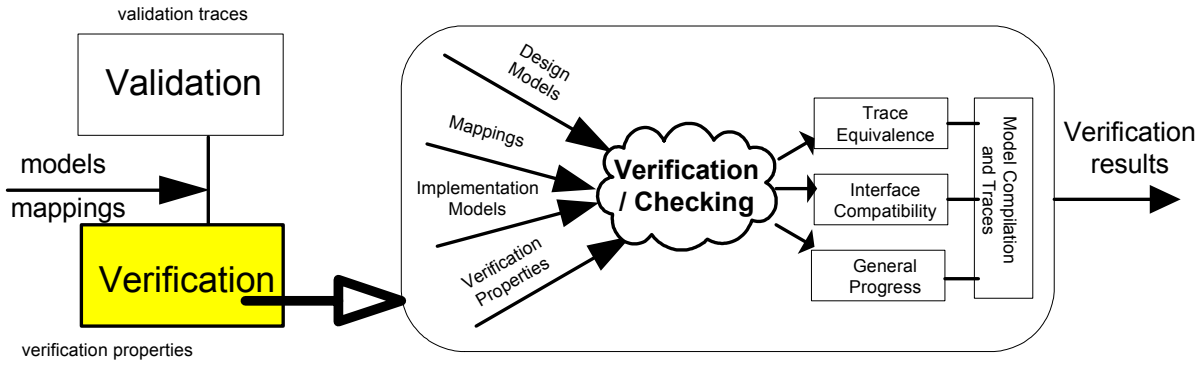



**Figure 6-6 Approach for Verification Analysis of Composition Models**

### *6.4.1 Composition Design and Implementation Equivalence*

The primary role of verification in our approach is to assist in checking whether the implementation of web service composition requirements and the related designs are equivalent. Equivalence verification has been reported in various themes. Amongst these are strong and week equivalence (Milner 1989), and traces and failure-divergence equivalence (Hoare 1985) are commonly referenced. In the context of this thesis we leverage trace equivalence to support our approach upholding the requirement that a composition that is tested equivalent provides *at* least the necessary behaviour to fulfil the test specification. In summary, the main property that will be considered in *trace equivalence* is that the BPEL4WS implementation exhibits the necessary behaviour to fulfil the requirements that are described in the MSC design.

The essence of this verification is to prove that a property exists in the composition modelling of combined implementation and design models. In section 6.3.3 we built a combined architecture model of the implementation and design models, specifying additional abstraction rules based upon our understanding of the composition environment and implementation semantics. This is used as source for the trace equivalence verification. Furthermore, any additional behaviour can be fed back to the implementer as counter examples. It is also the case that by definition of trace equivalence, the MSC design can be checked against that of the implementation. However, this may appear less useful in the

design approach of web service compositions, but essentially this also provides a technique for future re-engineering and checking against existing compositions where the implementation is the initial requirements in focus. In summary, the equivalence verification may also be used to check that a MSC design specification exhibits the behaviour of a BPEL4WS implementation**.** The requirements to trace equivalence verification of web services composition design and implementations are listed in Table 6-2. We continue to use the Echo-Audit web service composition example (specified in section 6.1) for ease of following the approach steps, and to illustrate how this verification is undertaken with the composition models. We begin with the synthesized model of the MSC design. From the Echo-Audit composition example, we build a scenario which provides a sample sequence of interactions between client and the services (Provider and Audit). In this scenario we describe a call to the provider (call_echo) and it's reply back to the client (reply_echo), and a call to the audit service (call_audit) and it's reply back to the provider service. Given the initial requirements specification back in Figure 6-3, the designer would construct a series of bMSC to specify required design components and interactions. The resulting bMSCs that would be produced by this exercise would be similar to those illustrated in Figure 6-7.

**Table 6-2 Trace Equivalence Verification Requirements**

| Approach Item | Product | Role in verification |
|---|---|---|
| Design Specifications | Message Sequence Charts | Input model for requirements of service composition |
| Implementations | BPEL4WS processes | Input model for service implementation |
| Abstractions | BPEL4WS process refinements | Refine BPEL4WS model for verification if process contains non-interaction elements |
| Mappings | Mapping activities | Map activities between input models |
| Properties | Equivalence property | Verification properties |

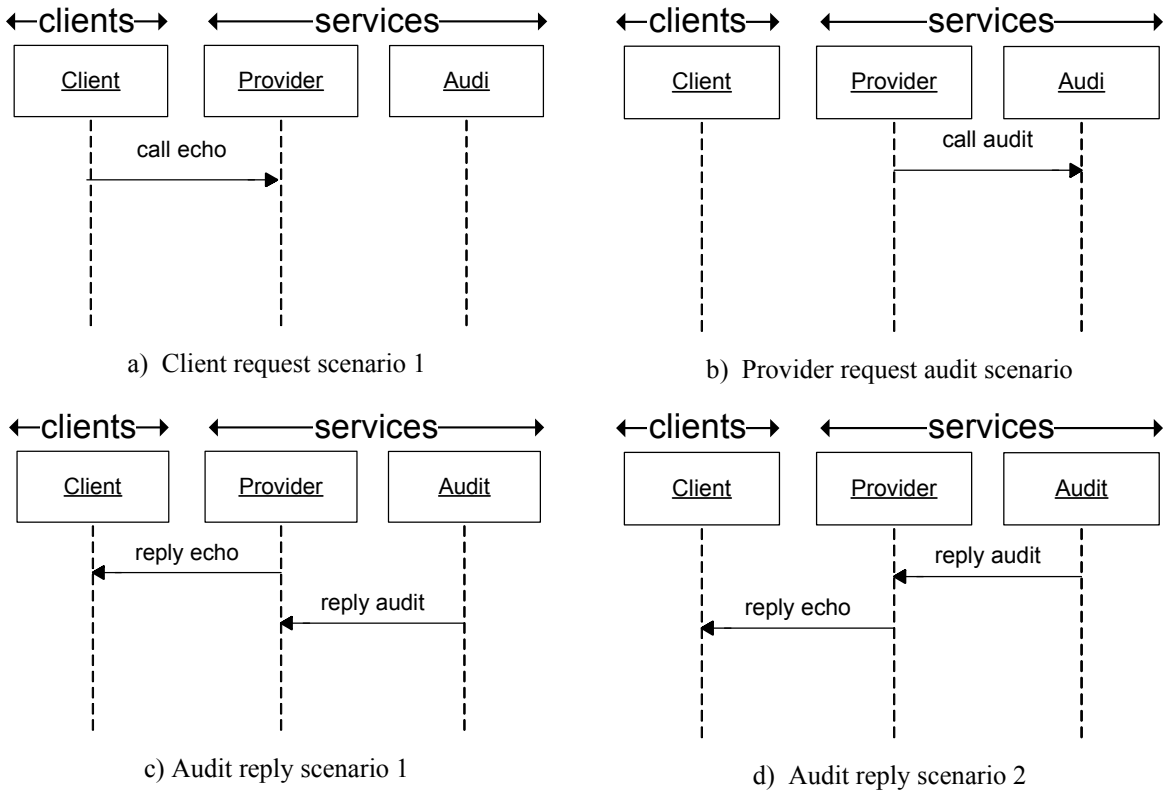



**Figure 6-7 bMSCs for scenarios in the echo-audit service composition**

A corresponding hMSC (Figure 6-8) would also be specified to sequence the scenarios constructed earlier. In this example, the initial scenario is sequenced as 1) client request (to provider), 2) audit request (from provider to audit service), either the audit service replies to provider and then provider to client, or alternatively the provider replies to client and audit to provider.

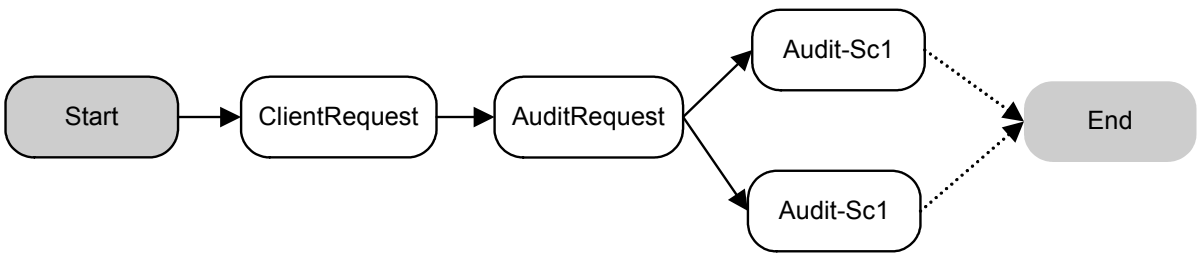



**Figure 6-8 hMSC of echo-audit service composition**

We introduce the notion of choice here to consider alternative paths of execution in the composition example. A labeled transition system is then generated from this model, using

the technique described in section 3.4. The model represents the finite state machine of the design built as messages between three components (that of requester, provider and audit). A graphical view of the LTS for this model is illustrated in Figure 6-9.

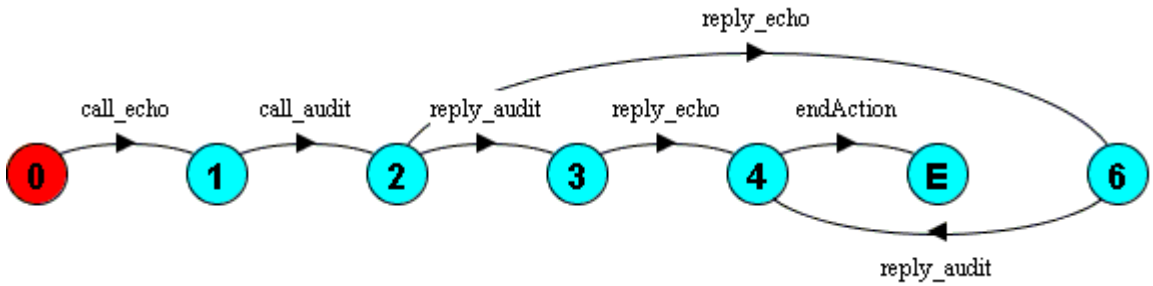



**Figure 6-9 LTS model for MSC scenario Echo-Audit Composition**

Each component in the MSC specification represents a service in the web service composition. Notice however, that the synthesis for this MSC model includes an *endAction* transition. The endAction transition provides a synchronized way of ensuring that the process terminates between components in the bMSC and a sequence of processes in the hMSC. We are now required to implement such a service architecture, however, for this example we illustrate building this using just one service composition and its interactions. A good example for verification is the provider service (effectively a coordinator service between client and audit). Note that the provider is a good example as it includes receiving, invoking and replying to services, all the types we can model in the MSC design specification. The structure of the BPEL4WS process used to implement this service is given in Figure 6-10.

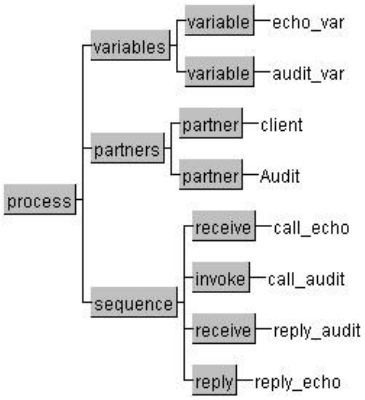



**Figure 6-10 BPEL4WS Process Structure for Provider Service in Echo-Audit Composition**

A labeled transition system is then generated from this BPEL code, using the technique described in Chapter 4. The full source for this service example is given in the appendix of the thesis. The model represents the finite state machine of the BPEL process as activities between the client and audit services. A graphical view of this model is illustrated in Figure 6-11.

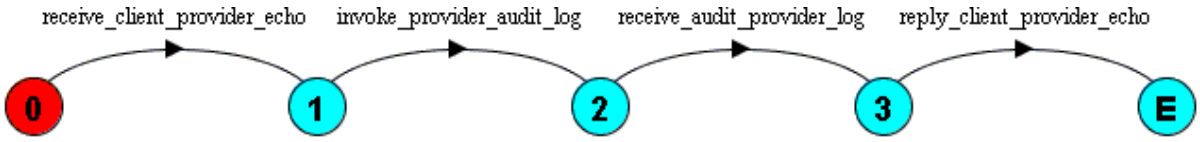



**Figure 6-11 LTS model for BPEL4WS Provider Service Activities in Echo-Audit Composition**

A brief comparison of the models shows that there are clearly differences between the possible paths in the MSC over that defined by the BPEL4WS implementation. The main difference is clearly that in the specification, the reply to a client (*reply_echo*) and the reply from the audit service (*reply_audit*) could happen concurrently. In other words, the design specifies that either execution path of replies could occur. Also notice that the activities are labeled differently (in this case, we have singled out the provider service and the viewpoint has changed from an invoke activity by the client, to a receive activity by the provider). This is where we use the mapping discussed in 6.3.2. The mappings group the activities in these models and assign the semantics of implementation from that of the design and requirements. The mappings (listed in Table 6-3), are applied to the LTS model of the BPEL4WS process. The resulting model has the same structure as the original BPEL4WS process, but with the naming scheme applied from the MSC designs. This model can then be used for trace equivalence verification. This mapped model is illustrated in Figure 6-12.

**Table 6-3 Mapping from MSC design to BPEL4WS activities for Echo-Audit Composition Example**

| MSC Action | BPEL4WS Action |
|---|---|
| call_echo | receive_client_provider_echo |
| call_audit | invoke_provider_audit_log |
| reply_audit | reply_audit_provider_log |
| reply_echo | reply_provider_client_echo |

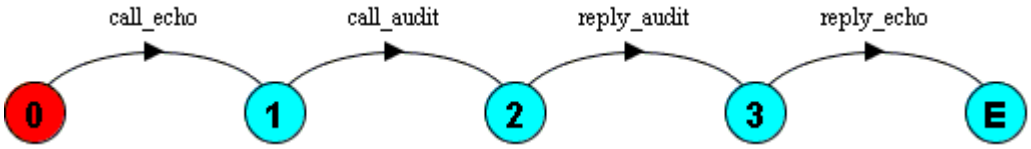



**Figure 6-12 LTS model for BPEL4WS Provider Service mapped to MSC activities**

Although the task of comparing models is easier in simple processes, more complex processes require an in-depth and time-consuming comparison. Model checking can then be performed to formally test that a BPEL4WS implementation provides the necessary activities to meet the MSC specification, through a model trace. Additionally, the aim is to provide as much a mechanical verification as possible, so that observation is not required by human eye in larger more complex processes. Although we have built models of both MSC and BPEL4WS activities, we are interested in the minimal trace equivalence in both these models. To specify this in FSP, we use the *deterministic* operation on the given MSC model and include abstracting the *endAction* transition as it is not included in the BPEL4WS model. A complete analysis model is created by combining these models together to form an ArchitectureModel, as discussed in section 6.3.3. We are also required to specify the property we are seeking to trace in the software model analysis. The properties of interest for verification are specified in FSP using the *property* function of the FSP language. The property of verification in this case is on equivalence checking of the BPEL4WS implementation against that of the MSC specification (our main property stated earlier). The property function creates a process by assignment of another process. For equivalence, we assign the property "*BIS_MSCBPEL"* as the deterministic MSC model giving a complete representation as the requirement for our verification check as the MSC specification.

The final activity in constructing the model for equivalence is to produce a parallel composition of the property model and the BPEL4WS model. This is achieved using the standard parallel composition operator of FSP. An example FSP code for checking trace equivalence of a MSC LTS model and BPEL4WS LTS model is given in Figure 6-13. The result of performing a trace operation on this composition is that of checking each and every transition of the MSC against that of the BPEL model and reporting on any violations where one has a transition that the other does not exhibit. For example a sample trace from checking the equivalence of the Echo_Audit service and MSC design given previously is listed in Figure 6-14.

**/* FSP code for equivalence checking of MSC and BPEL4WS compositions, with the property that BPEL4WS implementation should uphold activities of MSC design */**

**MSC_ArchitectureModel = *MSC composition model FSP***

**BPEL_ArchitectureModel = *BPEL composition model FSP* + *mappings*…**

**deterministic ||DetMSC = MSC_ArchitectureModel \ equ.**

**property ||Bis_MSCBPEL = DetMSC.**

**||CheckBPEL = (Bis_MSCBPEL || BPEL_ArchitectureModel).**

**Figure 6-13 FSP code for equivalence verification of BPEL4WS against MSC models**

**/* Trace run example from Equivalence Verification of MSC over BPEL4WS */**

**Trace to property violation in**

**Bis_MSCBPEL:**

**receive_client_provider_echo**

**invoke_provider_audit_echo**

**reply_audit_provider_echo**

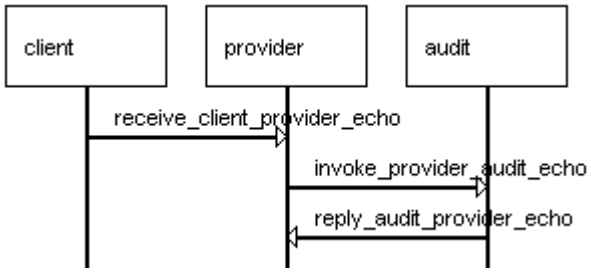



**Figure 6-14 Trace run example of trace equivalence of MSC and BPEL4WS models**

The reason for this property violation is that the *reply_echo* activity is permitted in the specification before the *reply_audit* activity but the BPEL4WS process model does not allow this (in equivalent mapped actions). Studying the BPEL4WS source for this example, and highlighting the implementer's decision to only allow a sequence of activities for the reply_echo and reply_audit is the source of this problem. The implementer can correct this issue by adding a concurrent activity execution statement (the *flow* element in the BPEL4WS source) or changing the existing sequential execution statement to concurrent. Similarly, a check of the specification for the composition against that of the implementation can be undertaken. We recode the FSP to have as source the BPEL4WS model as the property, and the MSC as the model to check. The modified code is given in Figure 6-15.

**/* FSP code for equivalence checking of MSC and BPEL4WS compositions, with the property that MSC design should uphold activities of BPEL4WS implementation */**

**deterministic ||DetBPEL = BPEL_ArchitectureModel \ {endAction}.**

**property ||Bis_BPELMSC = DetBPEL.**

**||CheckBPEL = (Bis_BPELMSC || MSC_ArchitectureModel).**

**Figure 6-15 FSP code for equivalence verification of BPEL4WS against MSC models**

We provide more detailed examples of equivalence verification of web service compositions as a result of a case study in Chapter 7. With the results gathered in equivalence checking, and primarily checking that the implementation fulfils the design specification provided against client requirements, the implementers can be assured that the web service composition process (using a standard such as BPEL4WS) will exhibit the behaviour necessary to fulfil these requirements. By way of behaviour equivalence verification it is not the purpose however, to guarantee that the messages passed between services is suitable to carry out the operations specified. This requires data analysis and possibly a run-time verification for checking its suitability (e.g. as expressions are evaluated within a process). Equivalence verification in the context of web service compositions ultimately allows the designer and implementer to compare interactions between models produced from their works. In an increasingly distributed environment and as design and implementation professions are frequently separated, the use of this activity provides early results in the success of executing such compositions.

### *6.4.2 Compatibility of Service Composition Interactions*

As we discussed in Chapters 3 and 5, compatibility verification is an important aspect of behaviour requirements between different clients of compositions. Clients will likely anticipate different behaviour depending on their individual requests and therefore the composition must be tested against various scenarios to reflect these different sequences of activities. There is also an assumption that a web service composition will work in any process environment (not just the original development domain). A greater level of *assurance in compatibility* can be given if interacting services are checked whether a composition exhibits the correct behaviour for its own use. Web Service compositions can also be seen as the implementation layer of a multi-stakeholder distributed system (MSDS) (Hall 2003). An MSDS is defined as; "*a distributed system in which subsets of the nodes are designed, owned, or operated by distinct stakeholders. The nodes of the system may, therefore, be designed or operated in ignorance of one another, or with different, possibly conflicting goals*". The focus is on interaction with multiple parties and the behaviour could be somewhat ad-hoc depending on the requirements of the partner services. However, three basic levels of compatibility for component compositions have been previously reported in (Larrson and Crnkovic 1999). These are defined as *interface*, *behaviour* and *input-output (data)* compatibility. Whilst input-output data compatibility is of interest, it is not the main focus of this verification work. We would however, expect a related growth of data analysis

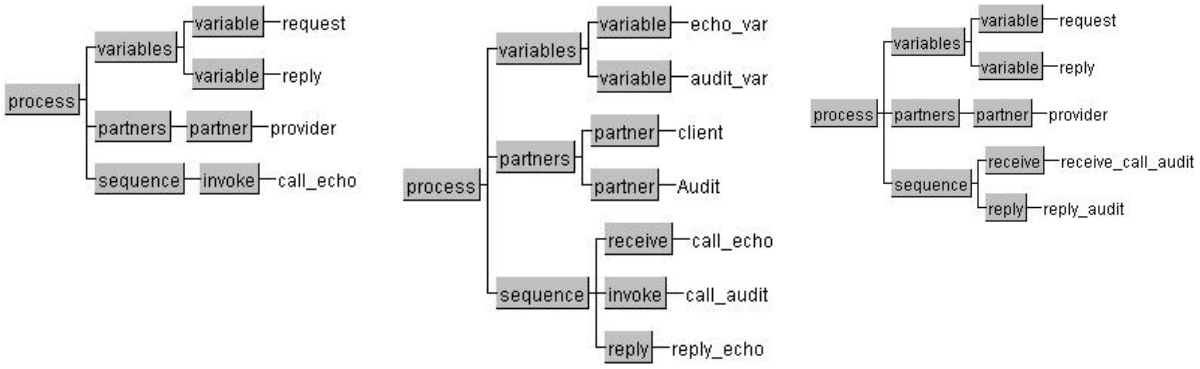
work to monitor and analyse service messages. We now apply the first two of the concepts discussed for compatibility, and describe interface compatibility specifically for web service compositions as; **the activity of correlating invocations against receiving and message replies between partner processes, such that invoke, receive and reply activities are synchronised.**

In our compatibility checking, the focus is currently only on the implementations and does not introduce another specification standard for compatibility in choreography (although this may be added at a later stage). Given a series of service implementations (in the form of BPEL4WS processes) the approach elaborates on the interaction mappings between processes and further inputs from port connectors between interaction activities in these processes. The requirements to compatibility verification of web services interactions are listed in Table 6-4:

**Table 6-4 Compatibility Verification Requirements**

| Approach Item | Product | Role in verification |
|---|---|---|
| Implementations | BPEL4WS processes and partner interface definitions (WSDLs) | Input model for service implementations and interface activities |
| Abstraction | BPEL4WS process refinements | Refine BPEL4WS model for verification |
| Interactions | Service interaction models | Translate BPEL4WS into a model representation (FSP) and assign related interface activities from partners (WSDL) |
| Mappings | Mapping activities | Map activities between input models |
| Properties | Safety properties for compatibility checking | properties to analyze those compatibility requirements are held for conversations where specified interactions exist |

We expand on our example from the trace equivalence verification by providing all three processes in BPEL4WS. The structure of these processes takes the form as illustrated in Figure 6-16. The process for performing compatibility verification focuses on the interactions between processes, rather than comparing activities between design and implementation (although this can be performed as a secondary verification step). The essence of the verification relies on our choreography modelling, as discussed in Chapter 5. For each BPEL4WS process there is a corresponding service interface defined. This is in the format of a WSDL document.




**Client Service Process** **Provider Service Process** **Audit Service Process**

**Figure 6-16 BPEL4WS Process Structures for Services in Echo-Audit Composition Example**

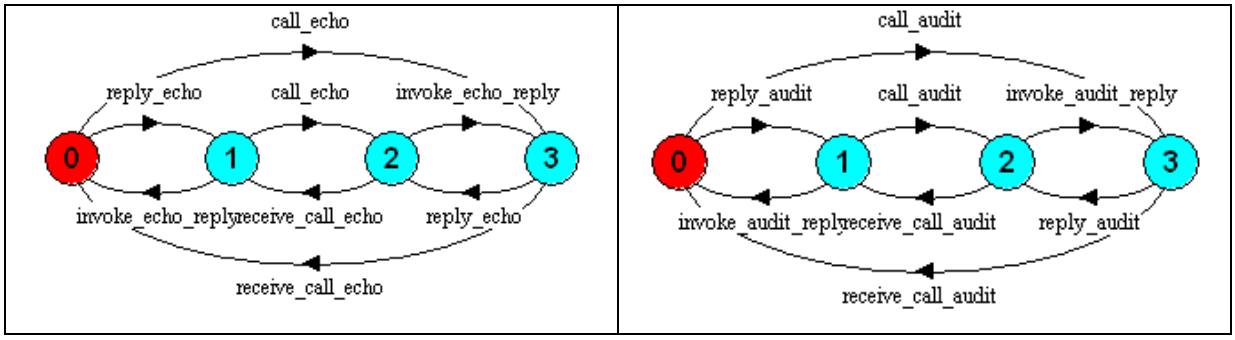
The set of BPEL4WS processes and WSDL interface definitions are translated and abstracted, and the algorithm for building port connectors (chapter 5 section 5.2.4) is performed for each invocation or reply in a given process. Thus, an iterative modelling exercise is undertaken starting with the first process input, and finishing with the last in the set. A sample port connector for the modelling of interactions between client and provider services in our example is listed in Figure 6-17.

**/* FSP code Client-Provider port connector model and mappings */**

```
CLIENT_ECHO_PORT_REPLY =
  (reply_provider_client_echo->
   output_provider_client_echo->CLIENT_ECHO_PORT_REPLY).
CLIENT_ECHO_PORT_INVOKE =
  (invoke_client_provider_echo->
   receive_client_provider_echo->CLIENT_ECHO_PORT_INVOKE).
||CLIENT_ECHO_PORT = (CLIENT_ECHO_PORT_INVOKE || CLIENT_ECHO_PORT_REPLY).
||CLIENT_PROVIDER_PORT_MAPPING = (CLIENT_ECHO_PORT)
/{reply_provider_client_echo/reply,output_provider_client_echo/input_output,invoke_
client_provider_echo/invoke,receive_client_provider_echo/receive}.
```

**Figure 6-17 FSP code for Client-Provider port connector model**

For the client, provider and audit processes, the port connector interaction models (between client and provider, and provider and audit) are listed in Figure 6-18. Compatibility verification is the trace result of a parallel composition of input BPEL4WS models and the port connector models. We perform a safety deadlock check on this new composition architecture model to ensure that each of the interaction activities are resolved in the port connector models. The FSP code for this parallel composition model is listed in Figure 6-19.




**Client-Provider Port Connector** **Provider-Audit Port Connector**

**Figure 6-18 Port Connectors for Services in Echo-Audit Composition Example**

```
/* FSP code for parallel composition of BPEL4WS service models and ports */
||CompositionModel =
   (CLIENT_BPELModel || CLIENT_PROVIDER_PORT_MAPPING ||
    PROVIDER_BPELModel || PROVIDER_AUDIT_PORT_MAPPING ||
   AUDIT_BPELModel).
```

**Figure 6-19 FSP code for parallel composition of BPEL4WS services and port connectors**

Again, it is possible to *witness* errors in building compositions correctly for partnered processes only if the process is of an applicable size to an observer's comprehension. The model checking facilities of deadlock analysis provides us with a suitable compatibility checking mechanism to check larger process compositions. If, for example, an invoke activity in one service process does not have a suitable reply in a partnered service process, a deadlock trace may give the result listed in Figure 6-20.

**/* Trace run example from compatibility verification of BPEL4WS interactions */**

**Trace to DEADLOCK:**

**invoke_client_provider_echo**
**invoke_provider_audit_log**
**reply_provider_client_echo**
**invoke_audit_log_echo**
**output_audit_provider_echo**

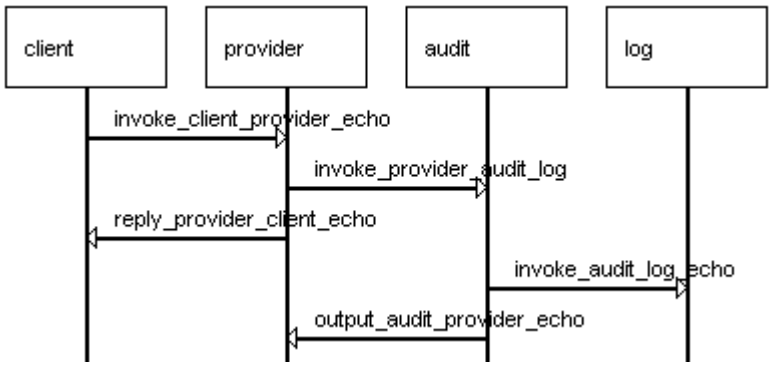



**Figure 6-20 Deadlock example of compatibility verification BPEL4WS and partnered services**

The *output_audit_provider_echo* activity built in the port connector between provider and audit services has detected that a path to *reply* to the provider has not been modelled (and as

such terminates before the provider reply is received). This indicates that a reply activity has not been given in the audit service, or was omitted in error. The BPEL4WS engineer can then revisit the BPEL4WS implementations and adjust accordingly. Repeat tests may exhibit further interactions that violate the standards of web service conversations or indeed, assure that the composition interactions are compatible.

In summary, compatibility verification provides a BPEL4WS engineer and any partnered BPEL4WS engineers to check the suitability of service conversations in composition implementations. This is important in two ways. The first is that a process may be required to *behave* in different ways for differing partner interactions, and that other service processes in the same domain have to be capable of interacting appropriately to fulfil there own progress and liveness. Secondly, in system decomposition (where a business or other process is split into several services) the engineers can check that responsibility to carry out a task has been split safely between services and that the complete goal is fulfilled by a complete trace model. Generally however, we expect this technique to be used with more emphasis on the first case, providing engineers with a safety check of collaborating services with partnered services (where the engineers may exist in different problem domains). Compatibility verification is undertaken by the input of a series of service compositions, service interfaces and through the generation of service port connectors. A safety check of deadlock analysis results in success if no deadlock (or a trace to deadlock) is detected as a violation in compatibility.

### *6.4.3 Other Properties*

Our third set of verification checking is more general than the previous two. By specifying particular properties of interest, engineers can check whether a web service composition can reach a particular state in terms of its *obligations* in more general cases (over that of individual scenarios used in section 6.4.1). This assists in building reusable SOM architectures, for which a policy states obligations in which web service choreography may be undertaken. We describe the model checking techniques for general properties of the composition models under two different types in our approach, categorised as;

- *Safety* – providing assurance that the composition is checked for partial correctness of transitions for a given property within the model, e.g. that a partner service invocation is always logged following an failure

- *Progress* – providing assurance against starvation of progress in the composition, such that, whatever state the composition is in, an activity will always be executed e.g. that a reply is always sent back to the original requester.

For both property types, we can reuse the model building steps described for trace equivalence (Table 6-2 - excluding the design specification requirement) and compatibility checking (Table 6-4). The building step requirements for including one or many processes is dependent on the source in question, or in other words, whether it is that the property must be tested on one composition or over a choreographed domain of processes. In this section we simplify the examples by concentrating on one composition to illustrate how each of the property checks are carried out in analysis.

In *safety analysis* of the compositions, we are seeking to assist the engineer to specify properties (or activities in the composition) that should be upheld in the composition. For example, the engineer may want to revisit the requirements for the service to be provided and note a series of conditional processing dependent on a sequence of activities having been carried out. In the Echo-Audit example, we can give a simple example that the *Provider* process must request the audit log of each request made to it (the LTS model for the provider service process was illustrated in Figure 6-11). To model check this and perform a safety analysis we can use the FSP syntax of **property** to describe the safety property of interest in our model. A safety property defines *a deterministic process that asserts that any trace including actions in the alphabet of the process P, is accepted by P*. The property syntax for the audit after request requirement is listed in Figure 6-21.

```
/* FSP code for safety property that a request to the provider will be logged by request
to the audit service */
        property REQUESTAUDITCHECK =
                (receive_client_invoke_echo->invoke_audit_echo->END).
        ||PROVIDER_BPELArchitectureModel =
                (PROVIDER_BPELModel || REQUESTAUDITCHECK).
```

**Figure 6-21 FSP code for safety property that a request to log a client is made**

Compiling the *PROVIDER_BPELArchitectureModel* in this model, composed with the safety property process provides the expected reassurance of no violation to our property, as the model built earlier clearly provides this activity following receiving a request from a

client. If however, the engineer was interested in whether a log was made prior to receiving a client request, then we could reverse the order of the property and rerun the model compilation. The result of this produces a model which includes the error state (Figure 6-22). Notice how the model reflects that each state is in violation of the property, which causes an immediate transition to the error state (identified by a state transition of -1). By iteratively specifying key properties of the requirements and safety checking the composition, the engineer can be given greater assurance in releasing the composition for deployment.

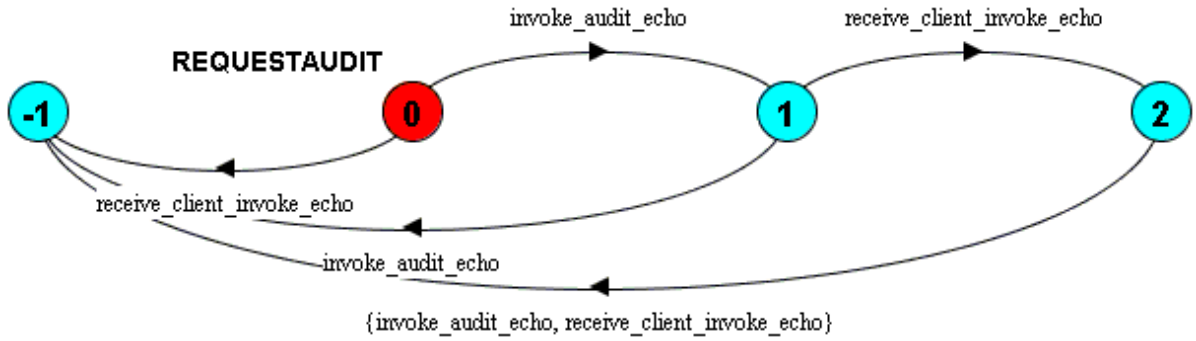



**Figure 6-22 LTS model of a violation of a safety property in the Provider Service Composition**

It is important to note that safety properties are usually given as those properties required by the composition, rather than those that are not. In this way, it is a much simpler and shorter task to perform, as it may be a tedious task to try and consider all the possible undesirable behaviours of a process rather than those which are easily identified as required.

Progress analysis is similarly specified by activity properties, but the focus is on those properties which will eventually happen (such as the example given previously, that a reply will always be given back to a requestor to a service). In FSP, the syntax for defining progress properties uses the **progress** keyword. A progress P = $\{a_1,a_2 \ldots a_n\}$ defines a progress property P which asserts that in an infinite execution of a target system, at least one of the actions $a_1,a_2 \ldots a_n$ will be executed infinitely often. This definition allows us to specify a range of progress properties, with the condition that at least one must be upheld in a service composition. By way of example, we use the provider service model to check whether a reply to client to always given. A progress property for this requirement is listed in Figure 6-23.

```
/* FSP code for progress property that a reply is always given to a client */
        progress ALWAYSREPLYCLIENT = {reply_provider_client_echo}
        PROVIDER_BPELArchitectureModel =
                (PROVIDER_BPELModel || REQUESTAUDITCHECK).
```

**Figure 6-23 FSP code for progress property that a reply to a client is always made**

A repeat of compiling the *PROVIDER_BPELArchitectureModel* in this model, composed with the progress property specified again provides the expected reassurance of no violation to our property, as the model built earlier provides this activity as a terminating state. If however the process did not include a reply to the client, then the result of compilation produces a model which includes a set of terminal actions (Figure 6-24).

```
/* FSP code for progress property that a reply is always given to a client *
Progress violation:
ALWAYSREPLYCLIENT
Trace to terminal set of states:
    receive_client_provider_echo
    invoke_provider_audit_log
Cycle in terminal set:
Actions in terminal set:
  {output_audit_provider_log}
```

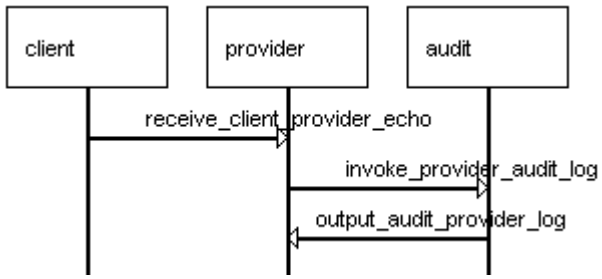



**Figure 6-24 FSP code for equivalence verification of BPEL4WS against MSC models**

The trace of the provider composition implemented in error, shows that there is no reply activity transition prior to the terminal state in the model, and that the terminating state of receiving a reply from the audit service is the last activity in the process. Clearly, an engineer can use this information to investigate the process and in this case, add an activity to reply to the client before the process ends. The task here may involve iterations of checking the composition, obtaining results of progress checks and adjusting the composition's activities, until the checks yield satisfactory results.

## 6.5 Validation Analysis of Behaviour Models

In the previous sections of this chapter we have described how to fulfil the requirements of the verification aspect of our approach using model checking techniques for equivalence implementations against design, safety and progress properties being upheld within the model and the specification and checking of general action activities using policy declarations. The other feature that facilitates a rigorous approach to engineering web service compositions is the provision and undertaking of validation of the composition for both designer and potentially, also to service clients. In the latter, it is to give assurance that requirements have been met and for the designers, that they have specified appropriate activities for the composition to be implemented. Modelling techniques (from boxes-and-arrows diagrams to logical formalisms) with varying degrees of analytical support are offered to assist requirements engineers in these tasks. The objective, in these "late-phase" requirements engineering tasks, is to produce a requirements document to pass on downstream to the developers, so that the resulting system would be adequately specified and constrained, often in a contractual setting (Yu 1997). This is particularly important as the service composition may be released in to a wider enterprise domain and reused by potentially a unknown number of clients, where there has been no previous verification with other compositions that already exist.

In this section we describe how this validation is undertaken using the model created as part of verification (but with the purpose of validation) and how features of simulation and animation can assist the designer in the requirements engineering issues discussed previously. The validation approach consists of taking as input the same inputs that were used in verification analysis, but with the exception that a set of validation properties are known by the end-user who is undertaking the validation, rather than a machine processable set used in verification. Therefore the inputs are the design models, implementation models and mappings. The process of validation is split between a focus of animation, simulation or an interactive trace. The output of the validation process is a set of results related to the validation properties used (for example, a general property would be that the behaviour defined in the models provides sufficient behaviour to support the initial requirements given). The sub-actions of our approach are illustrated in Figure 6-25.

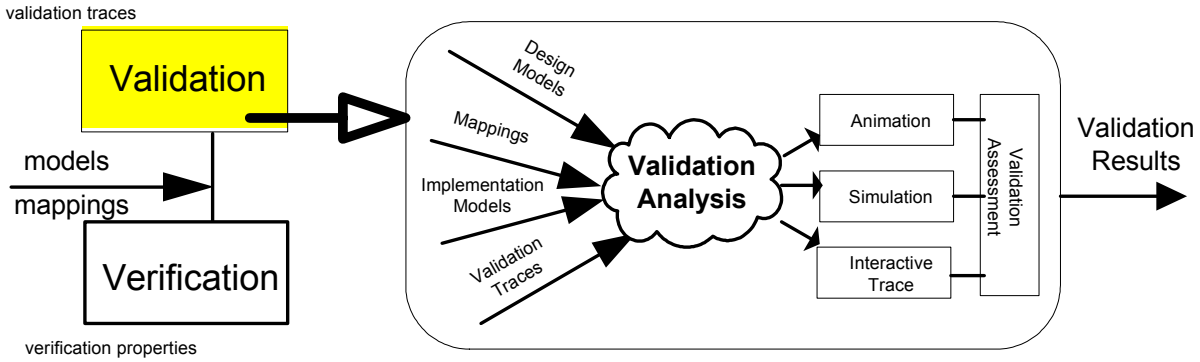



**Figure 6-25 Approach for Validation Analysis of Composition Models**

Other approaches have introduced the notion of validation of web service compositions against requirements using different source representations, such as in (Pistore, Roveri et al. 2004). The basis of these works is to have a high-level business requirements model. We believe however, that it is more important to have an *easily definable* business requirements model (such as in our Message Sequence Charts) to be able to validate and verify requirements for web services and their compositions. The basis of generating behaviour models for validation however is the same compared with these works.

### *6.5.1 Composition Validation through Animation*

Simulation is described in (Balci 1994) as "*the process of constructing a model of a system which contains a problem and conducting experiments with the model (on a computer) for a specific purpose of experimentation to solve the problem"*. In addition simulation software specifically has been described in (Schumaker 1999) as *"a software package that re-creates or simulates, albeit in a simplified manner, a complex phenomena, environment, or experience, providing the user with the opportunity for some new level of understanding."*. These propositions brought together, fulfil simulating compositions through software process models, and enabling its validation through simulation software. Our aim is to provide a simulation feature in the approach such that web service compositions can be analysed by requirement and BPEL4WS engineers for the purpose of validating that the behaviour given in designs and implementations meets stakeholder requirements. Indeed, requirements engineers must not only elicit and document requirement scenarios, but also validate that these are indeed what stakeholders want (Nuseibeh and Easterbrook 2000). The technique of simulation through animation is an effective validation technique, whereby in its simplest form, stakeholders can step through sequences of events dictated by a behaviour model

(Uchitel, Chatley et al. 2004). We begin by illustrating this through animation of the Echo_Audit composition example used in the verification sections of this chapter.

As we have shown previously, the Echo_Audit composition may come in two forms, either in that of a design or implementation. The assumption here is that if an implementation composition is used as source for validation, then it has been verified against a design prior to validation being carried out. Ideally, the requirements would be validated prior to any implementation being carried out. However, for the purpose of post-implementation validation, such as future client assessment for use within their own processes, either source is suitable for validation depending on the needs of the situation in focus. The behaviour model is a key output of design or implementation synthesis, and is the source model used in validation animation. Given a composition behaviour model, the client may step through the sequence of events exposed by this model. An example validation sequence performed using the Animator feature of the LTSA tool (described in the next chapter) is illustrated in Figure 6-26.

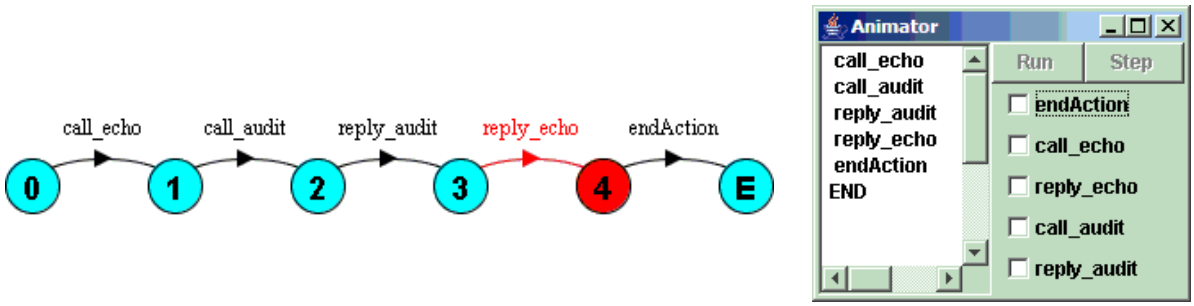



**Figure 6-26 A sample validation of a sequence using LTSA Animator function**

Here, the user steps through each state of the composition design (from state 0, labelled call_echo in this example, to state E – the end state). Whilst the sample given is just one sequence, more complex compositions may introduce alternative paths to the end state (such as the model presented earlier in Figure 6-9). In the earlier loan selection composition example (introduced in Chapter 3) there are several alternative paths of execution to the end state of the composition. For example, a "credit check" may result in either a state of *check ok* or *check failed.* The initial assumption is that if a credit check fails, then the entire request fails. If however, the credit check is ok, then the full sequence of the composition is carried out. Using animation, the designer can validate that these two possible sequences are acceptable and fulfil the requirements from path start to path end, in the service composition

to be provided. A sample validation sequence in the position just prior to the credit check choice is given in Figure 6-24. Continuing to validate this composition, introduces two paths as described previously. Either the result can lead to a *check_fail* or a *check_ok*. Sample animation steps are listed in Figure 6-28 for both paths.

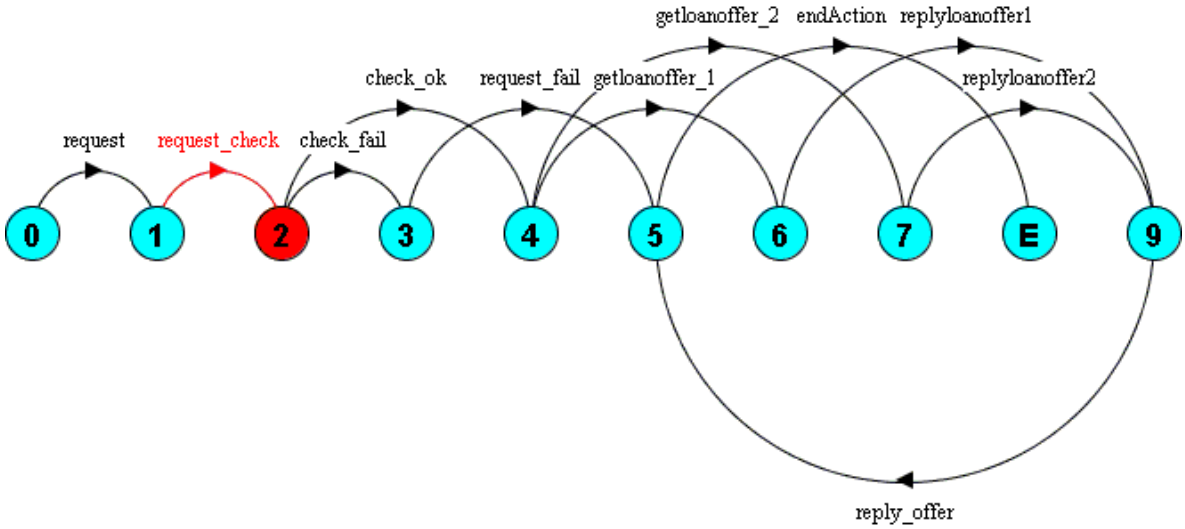
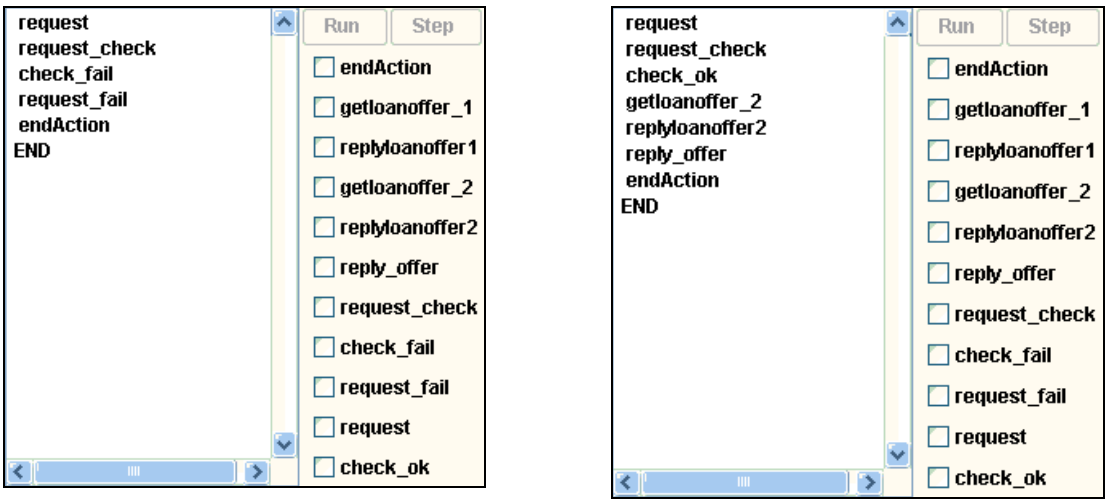



**Figure 6-27 A sample validation of alternative paths using LTSA Animator function**




***Credit Check Failed path*** ***Credit Check OK path***

**Figure 6-28 The alternative paths available using LTSA Animator function**

If we assume the credit check is successful (i.e. that the check_ok is the next transition to occur), then there is a clear issue with these animated traces of the composition. At state 4, either getloanoffer_1 or getloanoffer_2 activities may take place. This is presented as a choice to the user, but upon selection of either one of these activities, the composition selects

that given path and does not enforce that both must eventually complete in order for the composition to give a selective reply to the user. The composition is lacking in that these alternative activities must both happen, but not necessarily at the same time. In this case the designer may revisit the original composition design, and address these issues by adjustment of the MSC (e.g. checking scenarios and adapting the higher sequence diagram).

## 6.6 Summary and Discussion

In this chapter we have discussed the verification and validation steps, and how software process model checking techniques are applied to the web service composition models to provide trace results back to designers and implementers, and to facilitate the overall objectives of a service-oriented model and its goals, policies and obligations. We have shown that through verification, greater assurance can be given on the implementation of a web service composition before it is deployed in to a distributed environment. This rigorous approach to verifying and validating compositions prior to deployment is a key objective of our work, but to assess this we need to examine a case study from industry and evaluate how effective it is to web service practitioners. In terms of the approach overview discussed in section 1.2, we have introduced the highlighted parts as illustrated in Figure 6-29.

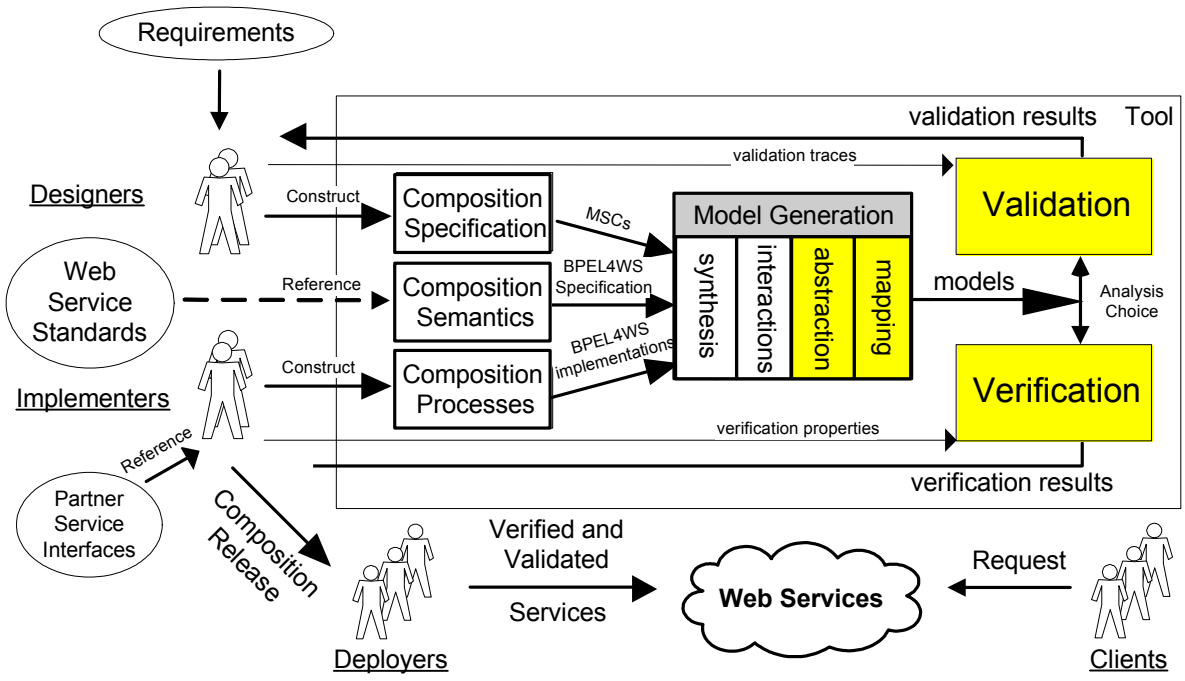



**Figure 6-29. Elements of the approach discussed in chapter 6.**

# Chapter 7

# Tool Support and Case Study

*"I view the problems created by Technology as simply opportunities for new tool making…"*
(from Tools Are The Revolution, Kevin Kelly, 2000)

---

In the previous chapters of this work, we have described an approach for designing, implementing, modelling and verifying web service compositions with respect to their behaviour. The undertaking of the approach has until this point, been suggested in a manual way. In other words, each step has been described as though it would be carried out by hand. We now present a tool which provides an implementation of this approach and features functionality to offer an interface for this design, implementation and verification mechanisms on compositions, in an integrated development environment (IDE). As a base for evaluating the approach and the tool, we also provide a case study of using the approach for a real-world industry project.

## 7.1 Tool Support

The tool requires a composed set of modules to mechanically provide the steps necessary to implement the approach described in earlier chapters. These modules can be expressed individually with regards to the design (Message Sequence Charts) models, implementation (BPEL4WS), specifying model abstraction and mappings, and executing the verification and validation steps. It is only when they are brought together however, that they ideally assist in ease of iterative design and implementation process. The integrated tool (Foster 2003b), which we call LTSA-WS is built upon the Labelled Transition System Analyser (LTSA) written by Jeff Magee in Java. The LTSA has, since its introduction, been expanded with a

plug-in framework to support various modelling design specifications, including work by Robert Chatley and Sebastian Uchitel on Message Sequence Charts (LTS-MSC) (S.Uchitel, R.Chately et al. 2003), and also Animation with Web Page Simulation enhancements (Chatley, Kramer et al. 2003). It is a well known and structured tool to encourage contributions by further applications. Our LTSA extensions were implemented in two phases. The first concentrated on functional requirements, building upon the current plug-in framework of the LTSA tool suite, and interacting with other plug-ins. The second phase considered broadening the application of the tool by migrating the LTSA-WS plug-in across to the Eclipse Integrated Development Environment and more specifically, encouraging its review by peers of the Eclipse community alongside their other BPEL4WS works. The Eclipse Innovation programme hosts a central research community, awarding those with ideas and projects to contribute to Eclipse. In addition to the community benefits of migrating to Eclipse, our work also demonstrates that our tool is flexible to be moved into different development environments. We now describe the key features of each of these phases.

### *7.1.1 Tool Architecture*

The LTSA-WS plug-in architecture (Figure 7-1) is built in the commonly known model-view-controller pattern.

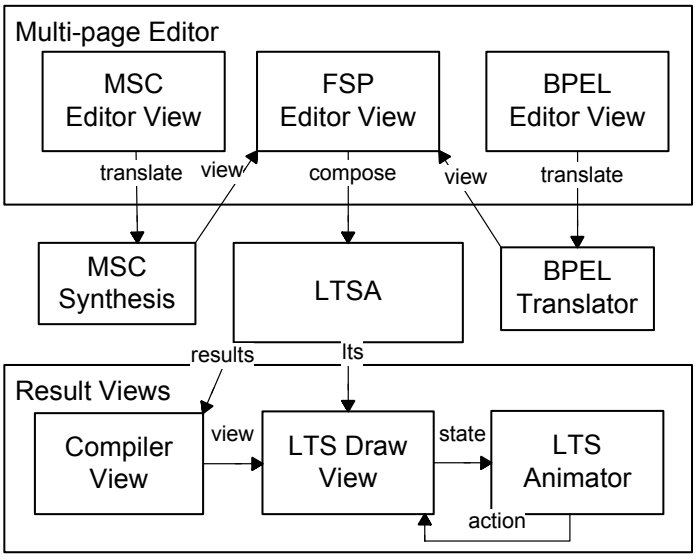



**Figure 7-1 LTSA-WS Tool Component Architecture**

The architecture of the tool consists of two models. Firstly, the BPEL4WS XML source code is used as the model for standard XML editing. The BPEL4WS source is also parsed to

provide useful editor functions, such as content outline and syntax highlighting. Parsing is also performed upon restore or save actions, whereby the translation function is called to view activities specified in the composition. The BPEL4WS engineer is able to build one or many web service compositions which aids in integrated enterprise service decomposition. For each composition selected, the engineer can either translate a single composition (by way of a mechanical implementation of translation rules described in Chapter 4) or compose multiple compositions for choreography and translate them in to FSP (as described in Chapter 5). The translation module is written as an independent module (itself potentially a web service), which takes as input one or more BPEL4WS implementations and in turn, traverses the source building a representation model in FSP. Multiple composition translation includes interaction mapping by using a mechanical implementation of the algorithm discussed in Chapter 5, to model partner links between services invoke, receive and reply actions. In addition, the composition design specifications (discussed in Chapter 3) in the form of MSCs can be synthesized to FSP models and included in the composed model. To enable this, a visual mapping table is available to the engineer to link activities in design and implementation models (as discussed under abstraction in Chapter 6). Results of checks provide implementers and designers with useful details such as missing interaction cycles (e.g. a missing receive or reply action). Checks are undertaken by the main LTSA function module. An output view summaries actions undertaken by the LTS compiler, and reports on property violations, such as deadlock, liveness or other safety properties (discussed in Chapter 6).

### *7.1.2 Initial Prototype as Plug-in for LTSA*

The initial prototype was written as an extension plug-in to LTSA, with a web service implementation, abstraction, mapping and translation interface. The plug-in adds a tab to the plug-in views of LTSA allowing an implementer to specify a series of web service compositions in BPEL4WS and then by the selection of a menu or action bar item, mechanically translate the composition into the FSP notation. A single composition may be edited at a time; however, managing a series of compositions is supported in a listed project. The LTSA-MSC, included as another plug-in into the framework, provides a designer with an editor pane to build the composition design specifications and features actions to mechanically synthesise them into FSP. Both the hMSC specifications (Figure 7-2) and the bMSC specifications (Figure 7-3) can be described.

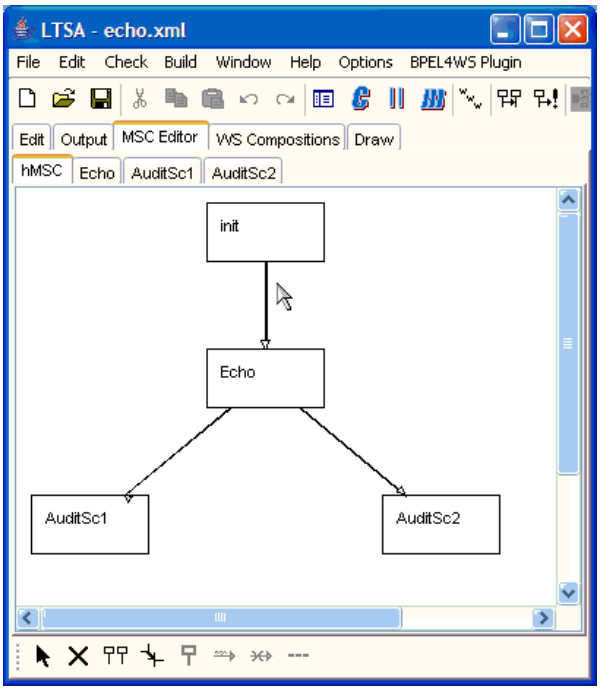



**Figure 7-2 LTSA-MSC: hMSC**

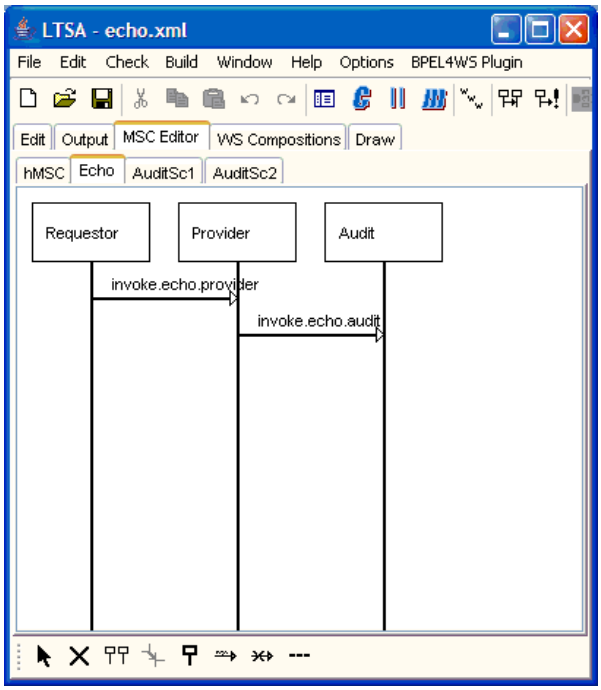



**Figure 7-3 LTSA-MSC: bMSC**

The LTSA-WS plug-in enables the mapping of these MSCs with the BPEL4WS compositions by way of the engineer selecting a FSP file for one or a series of compositions. This selected model is included at the point of translation, and abstractions and mappings included as part of the translation. This action provides the pre-requisites to verification and validation of design against implementation (and vice-versa as discussed in Chapter 6). The BPEL4WS interface, illustrated in Figure 7-4, provides a view of these model representations to the BPEL4WS engineer. Through the click of a button (or selection of a menu item), the engineer can generate the FSP model for the process currently being edited. Additionally the interface (WSDL documents) to this composition (used in the interaction modelling between partnered compositions) is edited in a sub-pane of the view. The mappings are also listed in a sub-pane (to the middle-left of the editing pane), such that when translation is invoked – the translator automatically includes any mappings specified (to hide or re-label appropriately). Two lists (for a series of compositions and WSDL documents) are in sub-panes below the mapping view. This enables the engineer to manage and specify the modelling of multiple-compositions and to automatically instruct the translator to include port connector mappings in the models produced by translation. The aim is to provide a single view (with multiple sub-panes) such that both compositions and interfaces are easily managed.

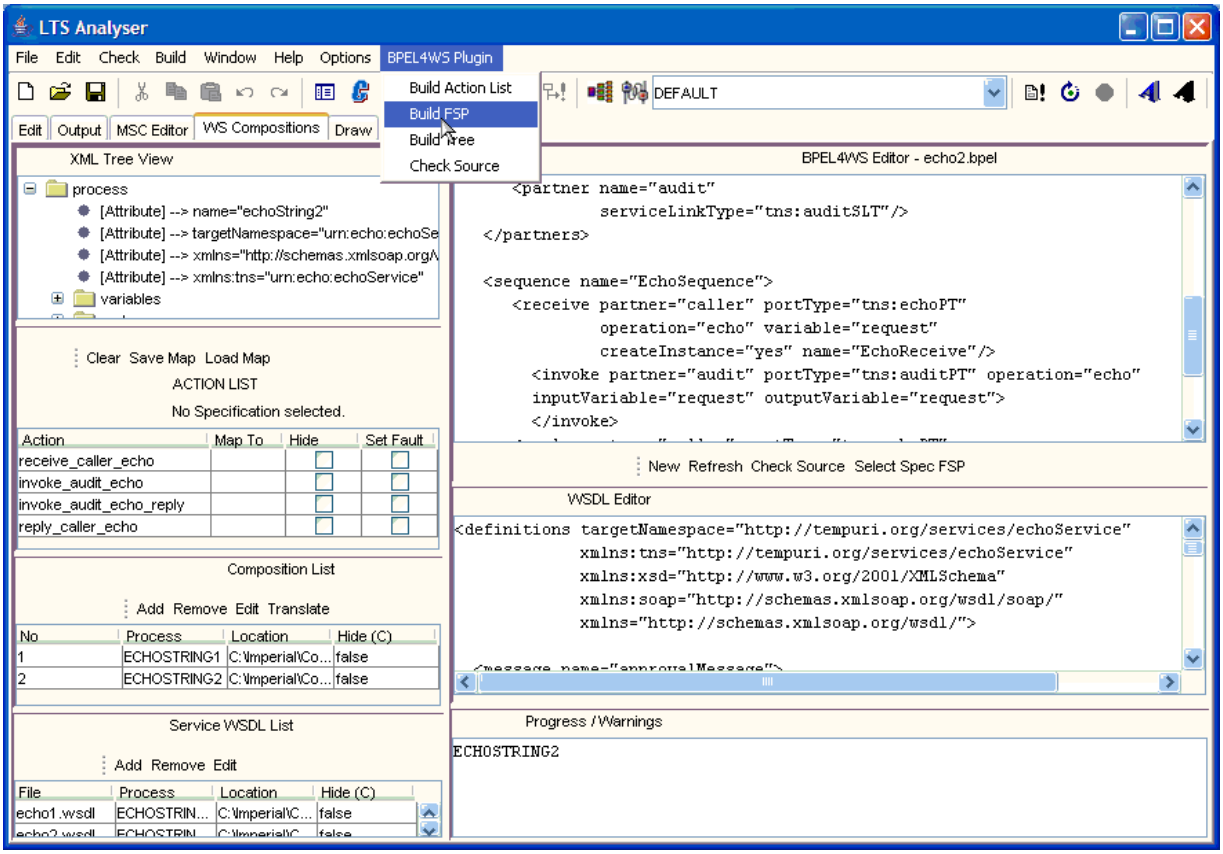



**Figure 7-4 LTSA-WS Interface and LTSA plug-in framework**

When the user selects that the BPEL4WS is translated to models, an FSP editor view is automatically presented to the engineer (Figure 7-5), so that any additional verification properties may be defined. The Safety Check (Verification) option informs of any deadlocks or errors (Figure 7-6).

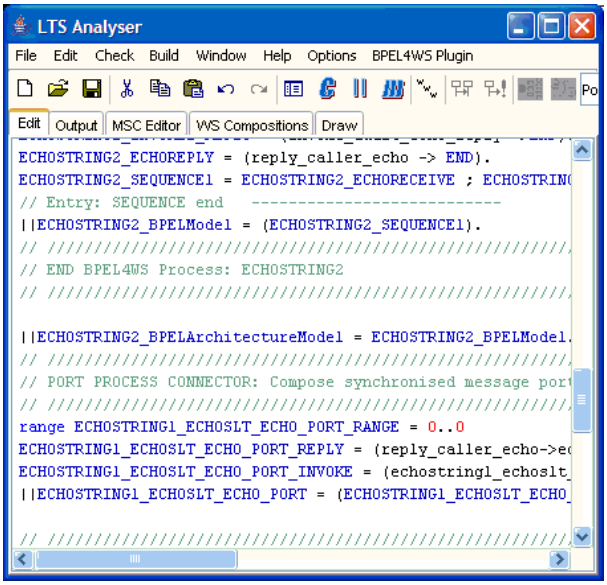



**Figure 7-5 LTSA-WS: FSP**

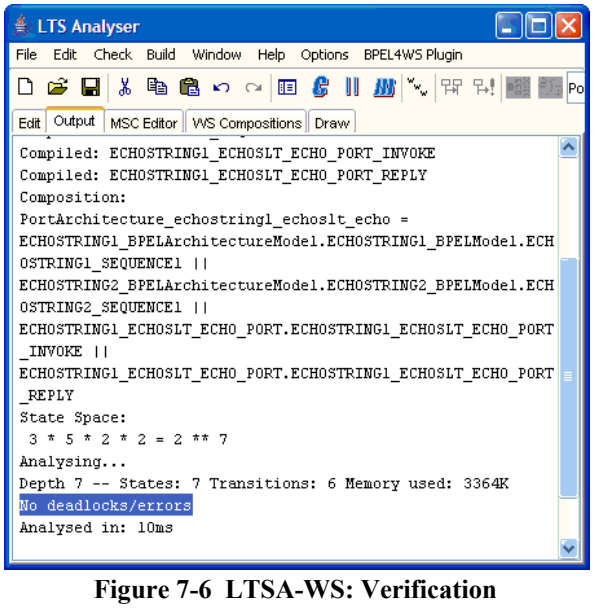



**Figure 7-6 LTSA-WS: Verification**

Additionally, validation (as described in chapter 6) is undertaken using the Animator extension to the LTSA tool. Figure 7-7 illustrates stepping through a BPEL4WS composition using the animator action selection features.

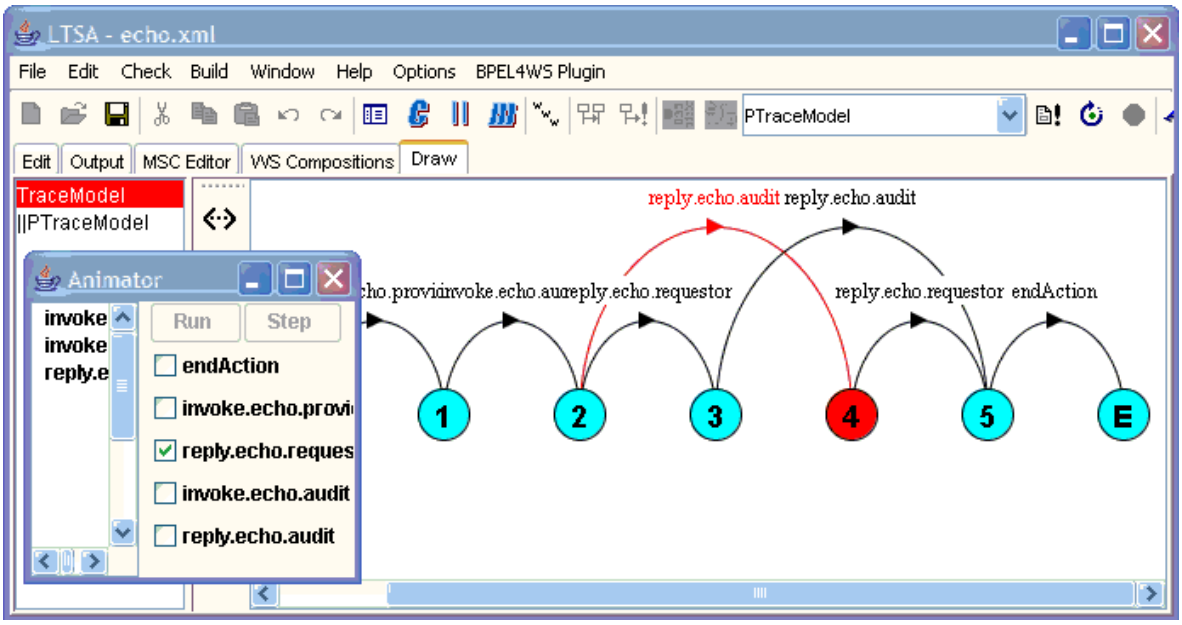



**Figure 7-7 LTSA-WS: Validation and Animation**

### *7.1.3 Migrating the tool to the Eclipse Environment*

Using the Eclipse framework opens the potential to link the tool with a network of other Eclipse plug-in contributions and aims to simplify the number of different, bespoke tools used in software engineering as a whole. Indeed, amongst these contributions are commercial BPEL4WS graphical editors (we currently only provide a basic XML editor), although the reader is invited to browse plug-in web sites as the list of contributors is continuously expanding. To migrate the prototype plug-in to the Eclipse environment consisted on rebuilding the model, views and controller pattern using the Eclipse Plug-in development environment. The plug-in views described here are based upon a migration of the Java modules from the original prototype. Core LTSA Java modules could be successfully imported into the Eclipse plug-in development environment however; rebuilding view modules has required some changes, particularly when moving the LTS Draw view from the Swing/AWT API to SWT API (the GUI API used by Eclipse). Aside from these differences however, current view migration has mapped conveniently onto the standard views provided by the Eclipse framework (Editor, Outline, Console etc). A view of the integrated Eclipse plug-in is illustrated in Figure 7-8.

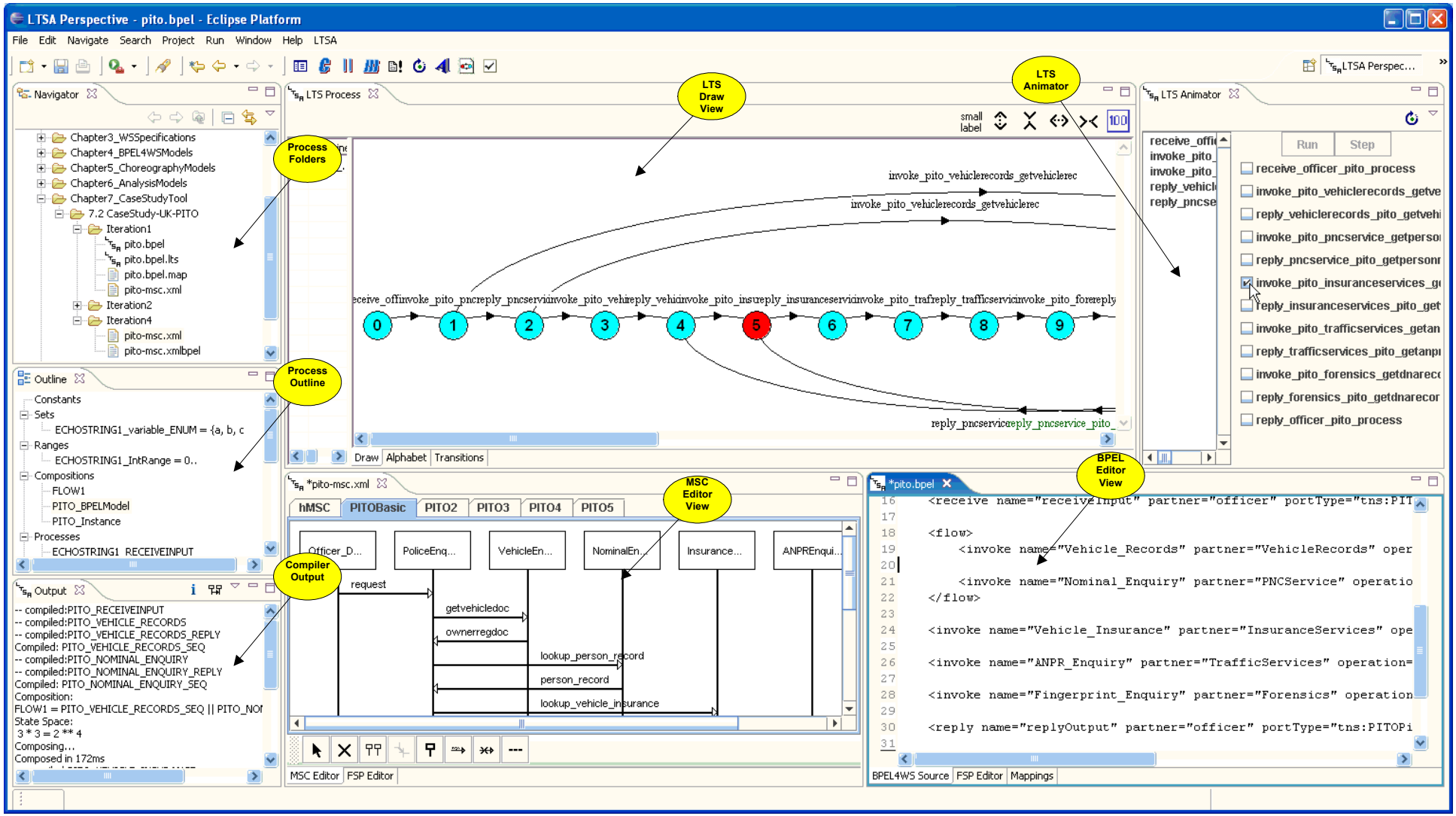



**Figure 7-8 Web Service Composition Development with LTSA-Eclipse**

## 7.2 Case Study: UK National Police IT Web Service Compositions

### *7.2.1 Introduction*

Our industry case study is taken from a national development currently underway by the Police IT Organisation (PITO) in the United Kingdom. PITO provides information technology and communication systems to the police service and criminal justice organisations in the UK. PITO's vision is "to be a trusted and valued partner in the delivery and operation of information and communication solutions to meet the needs of the police service and its partners and stakeholders". In this way, one of the highest priorities for PITO is its ability to provide secure, reliable, and available services on demand and to provide highly accurate information as a part of the processes carried out to fulfil service requests. In this project the view is to consolidate distributed national police services and to form a set of core processes by which the national police force may use without directly connecting to separate data sources in the process. Our work runs alongside reported findings so far in the progress of the project, detailing the consideration of moving to a service-oriented architecture and its quality of service provision and expectations (Hu 2003; Hu 2004). Our contribution is to assist in analysing the initial development of web service compositions, to support a series of different police enquiry types.

We present here a study of some scenarios described within the scope of interacting police enquiry services. We follow the approach described in our work to concisely model the interaction behaviour of the compositions built from the scenarios gathered as part of a business requirements building exercise. The scenarios given in this example case study are representative of the interactions used in providing solutions to the business requirements gained, however, due to the nature of the business of PITO and sensitivity in the detail of systems, these examples remain representative and may not illustrate exact developments.

### *7.2.2 Scope*

The scope of these compositions to date consists of a number of web services implemented to support some basic enquiry types of PITO systems. The position of PITO is to have core enquiry processes running on a central business process architecture (themselves deployed web services) which interact with other services provided by local force system owners. These central *service enquiry compositions* form key interactions to providing a distributed

yet consolidated view of the data and business process representations spread throughout the organisation and its associated forces. The initial pilot project consists of a series of web services providing functionality for; "Vehicle Enquiry" – matches vehicle details based upon enquiry search criteria, "Motor Insurance Enquiry" linking vehicle details with motor insurance details, "Nominal Enquiry" – matches person details based upon search criteria, "ANPR Enquiry" – provides primary vehicle identification given a Automotive Number Plate Recognition image pattern, and "Finger Print Enquiry" – provides DNA or Finger Print matches to Nominal details. Collectively the scope of the pilot architecture is as illustrated in Figure 7-9.

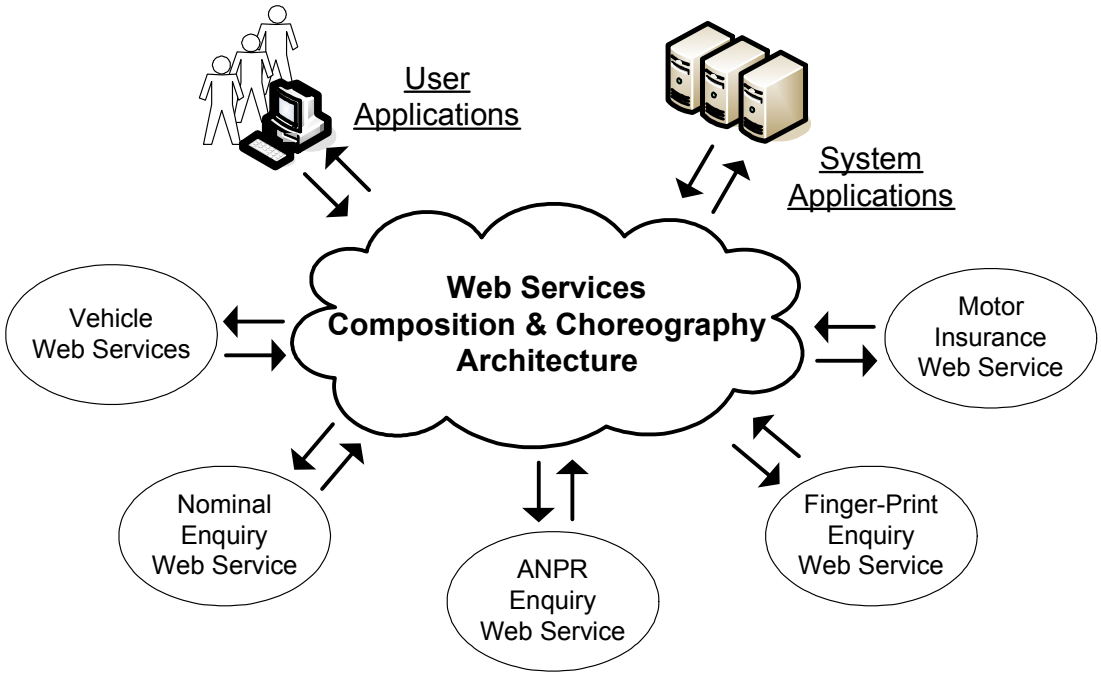



**Figure 7-9 PITO Web Services Architecture Scope**

Within this scope is an aggregation of services and the pilot project considers how these services can be combined, what the required behaviour of such compositions is, and how these compositions fulfil a goal in a series of web service choreography specifications.

### *7.2.3 Issues and Our Contribution*

Within the scope of the pilot project for the centralised service architecture is a range of two issue sets. The first, the long term strategy for PITO and the UK Police Service as a whole, has been reported in (ACPO 2002), which described how the current situation focused less on information being a service asset and that this information was often inaccessible by those

who actually need to use it. The issues were clearly focused on providing a service-orientation to the current enterprise police information technology solutions, describing the localised practice of information storing and retrieval restricting the use of data across the enterprise. The second set of issues was formed locally within the PITO service and was centered on the provision of services to support the ACPO policies recommendations. The core attribute of their issues is in service aggregation, and how service capabilities, a unified interface, co-ordinated behaviour and a combined Service Level Agreement (SLA) can be obtained through a definition of quality of service. Through our approach, we believed that some of these issues could be addressed through greater understanding and manageability of the behaviour exhibited by compositions, equally from the first time they are created and deployed, and also when compositions evolved to support elaborated client interactions and requirements. Our case study began with modelling the pilot project scenario, chosen for its breadth of service inclusion.

### *7.2.4 Requirements*

The pilot project's initial requirements were based upon the core function of facilitating a police officer's enquiry. The basis for this scenario is that when a request is made, a series of requests to various services are called to facilitate building an overall view to assist the officer make a informed decision on demand. Initially, the context of a request is built with a variety of linked information, with each request supplying the information for the next request. The scope of the sample is to address vehicle checks with that of the owner's collective *state* in the police IT network. A textual representation of this example scenario is given in Figure 7-10.

| |
|---|
| A suspicious vehicle with the number plate "xxxxxxxxx" has been identified by a police officer in Northern England. The officer launches a formal police enquiry about the vehicle including its registration record, insurance details, the registered owner's criminal records (if any) and DNA/fingerprint of the owner, as well as checking the vehicle's movement in the last 24 hours at key points in Scotland. |

**Figure 7-10 A Pilot Project Scenario for Web Service Composition in PITO**

### *7.2.5 Specification*

The specification of this scenario is built by abstracting the interaction components from the requirements scenario. In this case, the officer makes a request through some *device* (whether it is a Personal Data Assistant, Internet enabled Phone or locally via a personal computer). The interactions are then added to support the steps described in the scenario. The composition in this scenario is a police enquiry composition service. Initially, the only client in the scenario, the device component, simply has two interactions, for that of making a request and receiving a reply from the composition service. For each enquiry the police enquiry composition makes a request using key search criteria (such as vehicle registration no.). A bMSC view of this specification is given in Figure 7-11.

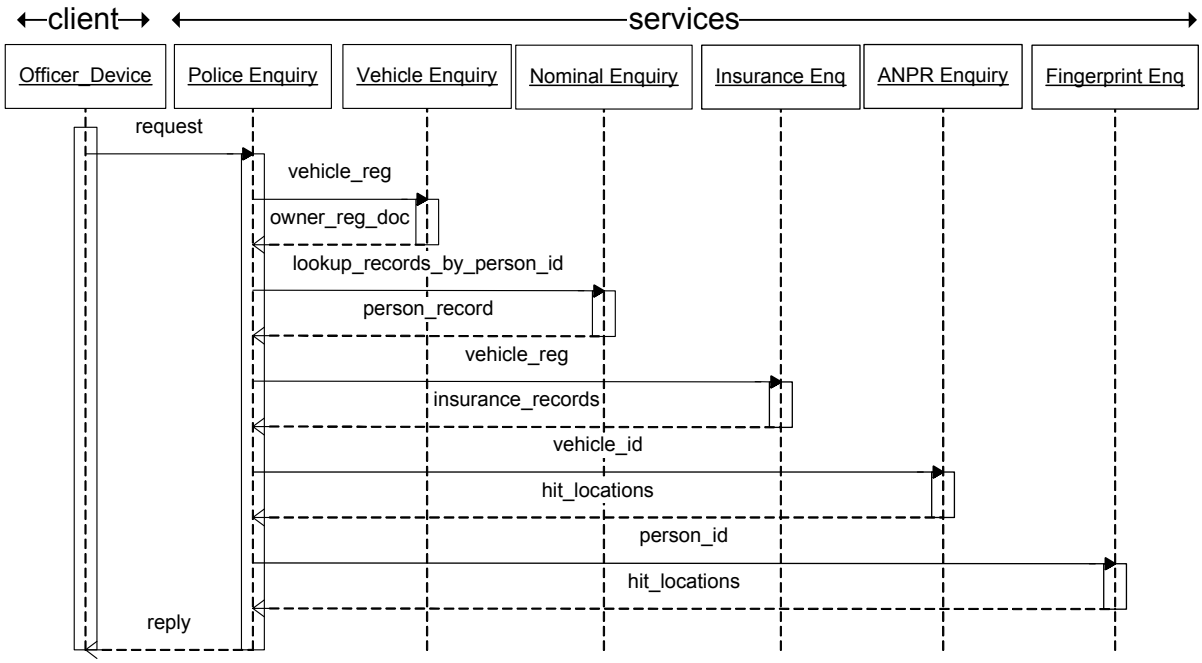



**Figure 7-11 Initial specification for a PITO police enquiry web service composition**

The specification illustrated is quite simple, in that it assumes that each enquiry is performed sequentially from a central enquiry process (i.e. the composition service) and that alternative scenarios are not possible. Studying this sample however, highlights possible areas of composition improvement through concurrent behaviour (a goal from "quality of service" in this case study), for example both vehicle records and vehicle insurance enquiries use the vehicle registration details concurrently between the vehicle enquiry and insurance enquiry components. This elaboration of requirements yields additional scenarios for the specification. The amendment to the original specification is focused on a subsequent

scenario of a permissible sequence of interactions following the initial enquiry request. A partial view of the amended parts to the original specification is illustrated in Figure 7-12.

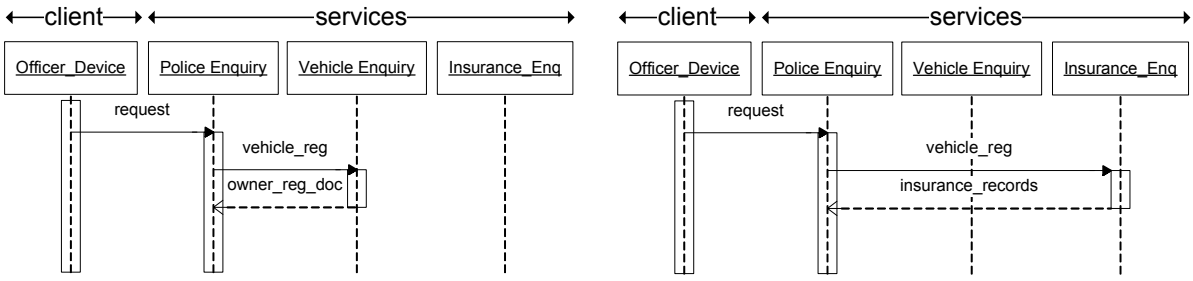



**Figure 7-12 Concurrent interactions introduced in to the PITO composition specification**

The additional specification scenarios pose a series of questions over that of the behaviour constructed initially. Firstly, how does the introduction of this concurrency effect the remaining interactions of the composition? Clearly, by introducing the possibility that two service requests can be performed concurrently suggests that a series of actions may or may not occur within the duration of these initial service requests. Furthermore, can the response from each of the service requests based upon vehicle registration exist over the entire duration of the composition? A partial answer to this question can be found by revisiting the initial specification and identifying that a vehicle id is used in another service request, that is, to the ANPR Enquiry service. Therefore, it is evident that one of the two service requests must be completed to provide the composition with sufficient detail to pass as parameters to the ANPR service call. Considering this leads a designer to enhance the specification to include the possibility of either the Vehicle Enquiry or Vehicle Insurance Enquiry replying and subsequently the ANPR Enquiry service being called. We illustrate this in further scenarios, such as that for a Vehicle Enquiry request followed by a Vehicle Insurance Request, then a Vehicle Enquiry reply, then an ANPR Enquiry request and a Vehicle Insurance reply. This is illustrated in Figure 7-13.

Further consideration of the initial specification highlights that there is another constraint required for other possible concurrent interactions. The Nominal Enquiry is requested with person identification as part of its required parameters. This identification is also taken from the result of either Vehicle or Insurance Enquiry service request.

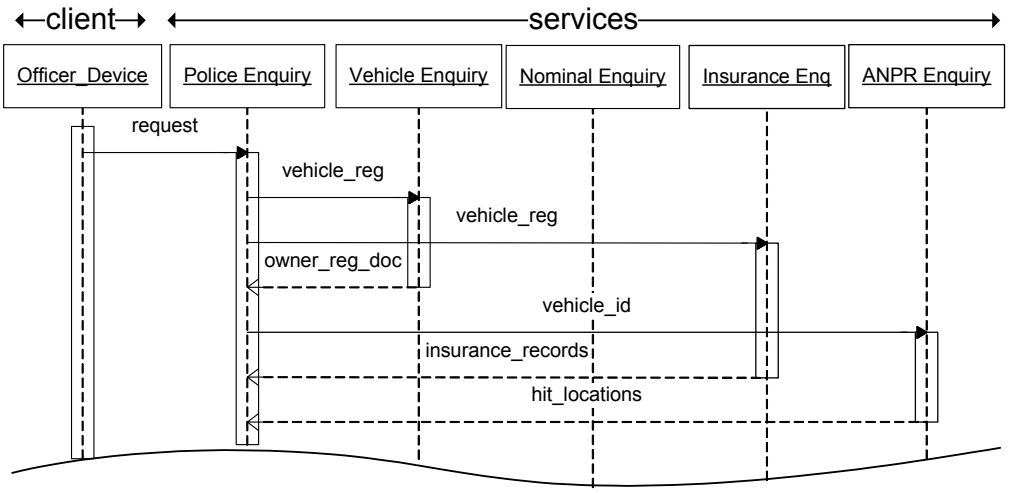



**Figure 7-13 Partial scenario for Vehicle Enquiry reply and ANPR request constraint**

The two services based upon person, being Nominal and Fingerprint enquiries, must be sequenced with this initial request if an improvement to have these performed concurrently is desired. An amended specification for the Nominal (Person) enquiry requirement is illustrated in Figure 7-14.

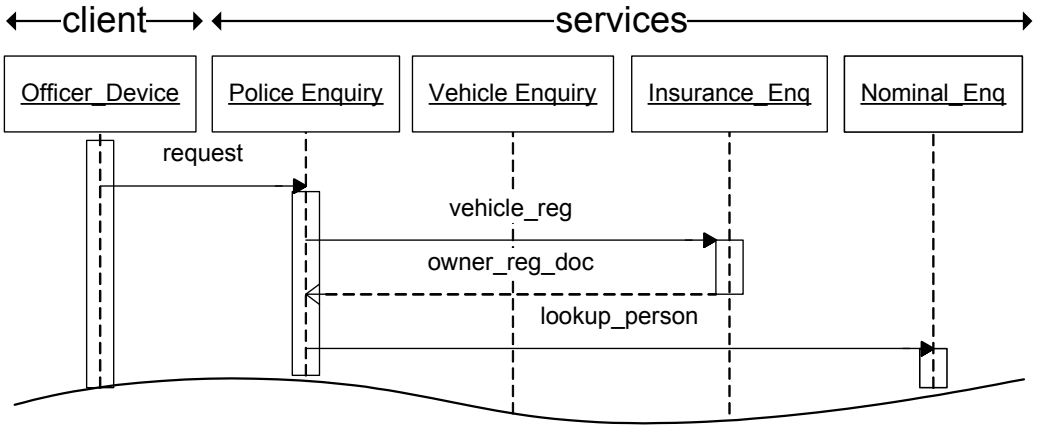



**Figure 7-14 Partial specification scenario to constrain nominal enquiry with result of insurance enquiry**

Equally we can define this constraint against that as being that the Nominal Enquiry request can not made unless the Vehicle Enquiry has been completed. This further alternative scenario is illustrated in Figure 7-15.

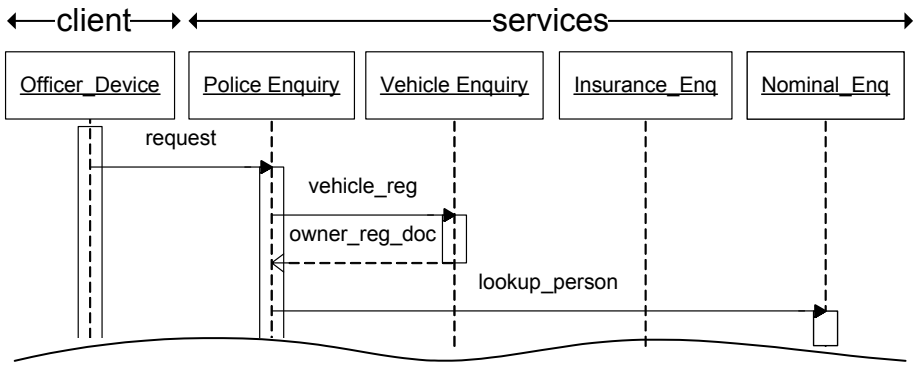



**Figure 7-15 Partial specification scenario to constrain nominal enquiry with result of vehicle enquiry**

At this stage in defining the web service composition specification, the design has moved focus from initially specifying a composition consisting of a sequence of service interactions, to reviewing the order in which interactions are made. The revised specification consists of a series of individual scenarios together with a scenario composition diagram specified in a hMSC (for the composed series of constraint scenarios). These specifications are illustrated in Figure 7-16.

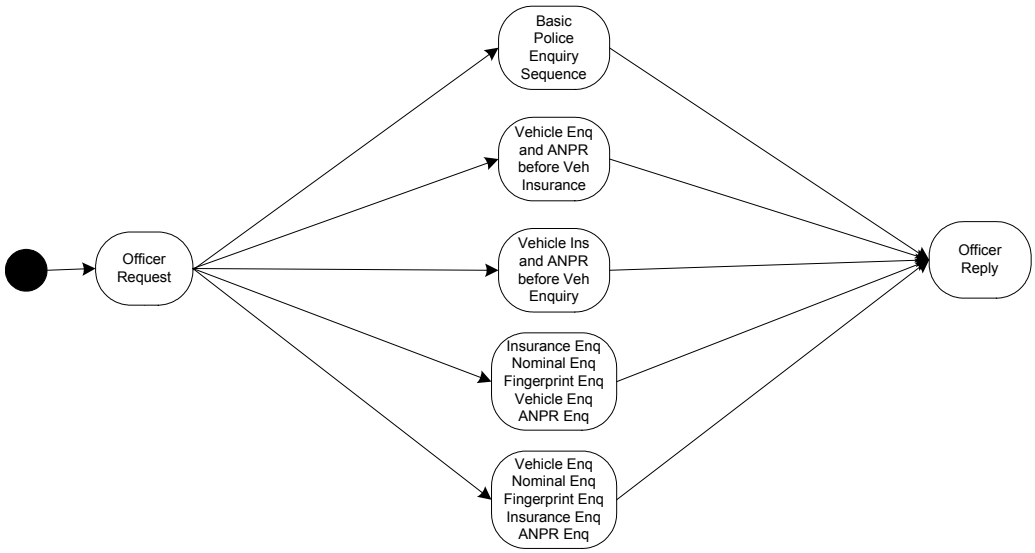



**Figure 7-16 hMSC for PITO Police Enquiry composition**

By starting with a series of basic interactions and formulating an elaboration through a higher level sequence chart, the interactions can be concurrent (where permissible) and

composed in one or many specifications. We now consider how the case study would implement such a web service composition in one or more BPEL4WS processes.

### *7.2.6 Implementation and Analysis*

In this case study, we have assumed that the implementation is undertaken in BPEL4WS and that the design specification has been generated and passed to the implementers to replicate the possible process transition paths in the implementation. The BPEL4WS process takes focus on the Police Enquiry service, the central composition in the pilot project. The process consists of a series of workflow statements described in BPEL4WS (as we illustrated in Chapter 4). To illustrate iterative development using the approach in our work, the implementation begins by building the initial process, where all the interactions are sequenced. The outline structure of this process, with service interaction activities shown only, is illustrated in Figure 7-17.

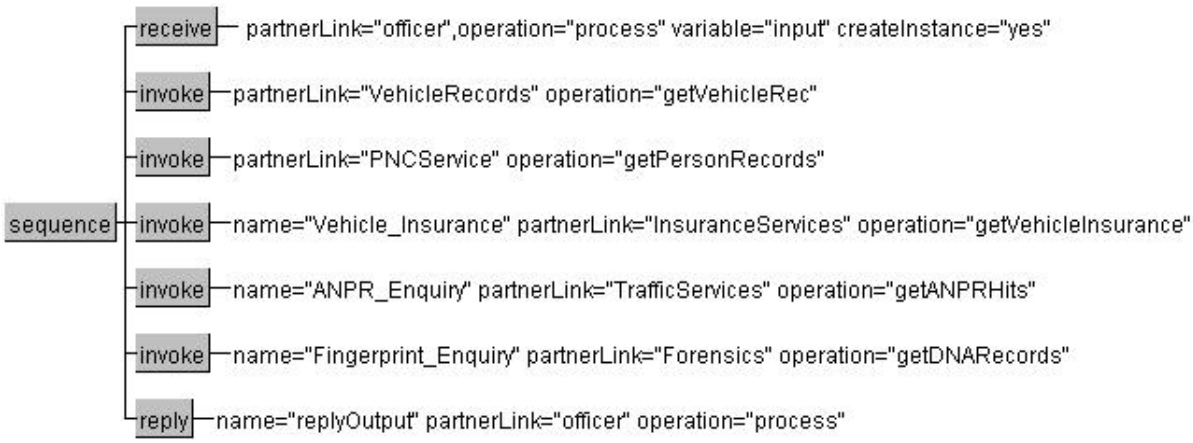



**Figure 7-17 PITO Police Enquiry Basic BPEL4WS Process structure (interactions only)**

The assignments part of the process forms the remainder of the process structure, and outlines the passing of search criteria between service invocations and replies. The initial request message part of vehicle registration is the first to be assigned, to both message variables for input to Vehicle and Insurance record enquires. We illustrate part of this assignment process structure in Figure 7-18.

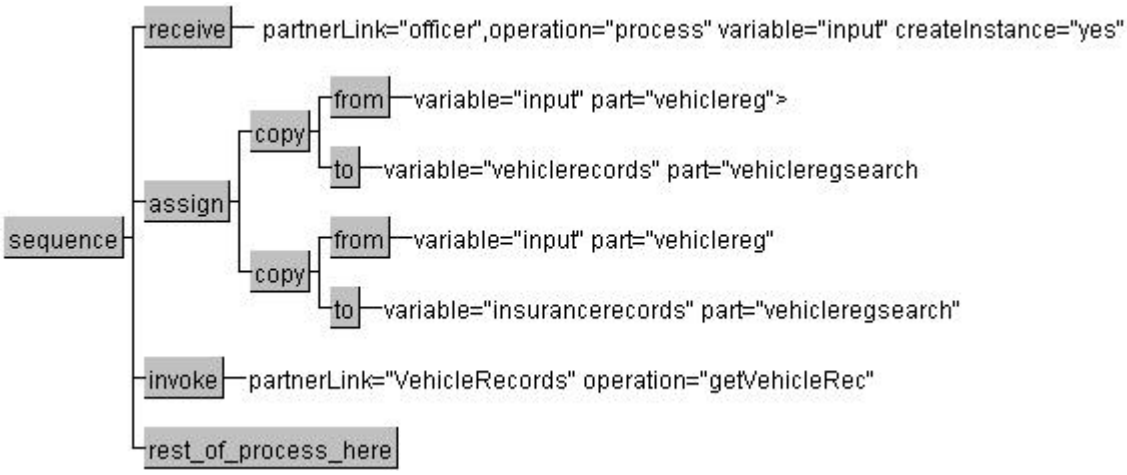



**Figure 7-18 Partial PITO Police Enquiry Basic BPEL4WS Process with assignments**

This basic process is modelled as a simple sequence of activities, as described in section 4.3.1. Each receive or invoke activity, for example the initial request by the officer, is undertaken in the order specified in the sequence. The begin and end of the process is marked by the receive, and final activity of a reply, before the process terminates. Each new request by an officer creates a new instance of the process, signified by the *createinstance* attribute on the initial receive activity. In addition to the interactions, the process composition is also modelled with variable assignment, as partially illustrated in Figure 7-19.

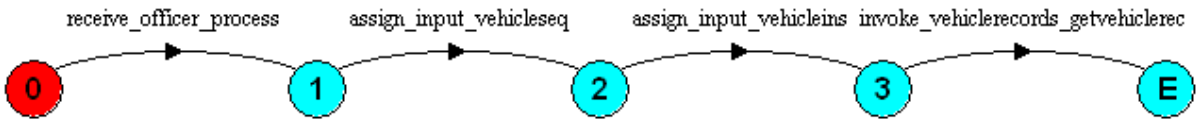



**Figure 7-19 Partial BPEL4WS Process sequence with assignments**

Whilst the engineer can attempt to perform a full trace equivalence against the specification at this point, the sequence is perhaps too trivial to expect any relationship of implementation against design. Yet even at this stage in implementation, this implementation may be partially fulfilling the specification through one such scenario (in this case the first sequenced scenario). We therefore use the trace equivalence verification method discussed in section 6.4.1 to perform such a check at this stage. The preparation of the implementation process has been discussed in section 6.3. To perform trace equivalence the non-interaction activities are marked as being non-observable in the implementation model. In this case, we use the hide operator of these activities. The tool mapping function generates the FSP code

through its compilation feature. A LTS model of this refined model is illustrated in Figure 7-20.

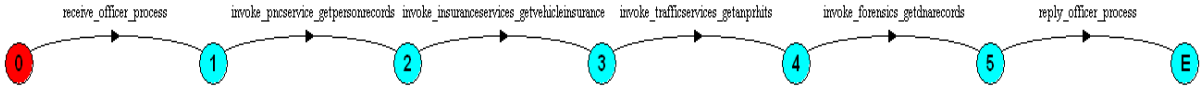



**Figure 7-20 Graphical LTS view of Police Enquiry Composition with abstraction**

The composition engineer then specifies the mappings between the interactions modelled in the composition with that of the design specification to be verified against. The mappings are selected against a list of activities presented in the tool. A summary of mappings is listed in Table 7-1. Note that the replies to the composition process are specified in this mapping as a default of the invocation name mapping with the addition of a "_reply" as a suffix. The question of reply interaction verification is not possible on a single composition and we are not able to determine when replies are actually made by a partner that has not included a process in the analysis. Therefore the equivalence is based upon the initial request (receive), invocations to other services (with either an immediate or no reply) and the reply to the initial requestor.

**Table 7-1 Mapping Activities for initial PITO Police Enquiry composition**

| MSC Action | BPEL4WS Action |
|---|---|
| request | receive_officer_process_enquiry |
| vehicle_reg_enq | invoke_vehiclerecords_getvehiclerec |
| owner_reg_doc | invoke_vehiclerecords_getvehiclerec_reply |
| Lookup_records_by_person_id | invoke_pncservice_getpersonrecord |
| Person_record | invoke_pncservice_getpersonrecord_reply |
| avehicle_reg_ins | invoke_insuranceservices_getvehicleinsurance |
| Insurance_records | invoke_insuranceservices_getvehicleinsurance_reply |
| vehicle_id | invoke_forensics_getanprhits |
| vehicle_hits | invoke_forensics_getanprhits_reply |
| person_id | invoke_forensics_getdnarecords |
| dna_hits | invoke_forensics_getdnarecords_reply |
| reply | reply_officer_process |

With the mappings, the model of the BPEL4WS process now takes the form of an LTS with label names assigned with the equivalent transition names as used in the design specification. The inputs for an initial analysis of trace equivalence of the BPEL4WS composition against that of the design specification are listed in Table 7-2.

**Table 7-2 Mapping Activities for initial PITO Police Enquiry composition**

| Approach Step | Product of step | Performed by tool, user or engineer |
|---|---|---|
| Design Specifications | Model produced as part of section 7.2.4. | Design engineer input and model produced through tool as described in section 3.4 |
| Implementations | Model produced from BPEL4WS Process | BPEL4WS process built by engineer. Tool translates to FSP model. |
| Abstractions | Refined model without assignment activities | Tool generates FSP code to provide refined model of BPEL4WS process. |
| Mappings | Composition interactions mapped to design activity labels | BPEL4WS Engineer assigns interactions to design specification using tool mapping functionality. |
| Properties | Equivalence property added to as additional process model | Tool option to run trace equivalence of BPEL4WS implementation against MSC specification. |

Using the tool, the trace equivalence is undertaken to check that the BPEL4WS satisfies (albeit partially) the scenarios covered in the specification.

```
/* Trace run of equivalence property check of BPEL4WS process over MSC */
Composition:
CheckBPEL = MSCArchitecture_Model ||
Abs_Process_BPEL.ECHOSTRING1_BPELArchitectureModel.ECHOSTRING1_BPEL
Model.ECHOSTRING1_SEQUENCE1
State Space:
13 * 13 = 2 ** 8
Analysing...Depth 13 -- States: 13 Transitions: 12 Memory used:
5104K
      No deadlocks/errors
Analysed in: 0ms
```

**Figure 7-21 Results of trace equivalence test to check BPEL4WS partially fulfils MSC specification**

Further analysis is also possible, even at this initial stage of implementation. As we are able to determine whether the BPEL4WS fulfils one scenario of the MSC specification, we are also able to describe what additional scenarios the BPEL4WS is missing. To achieve this, we swap the model that is used as the property for verification. In this case, we analyse the specification model against that of the BPEL4WS implementation model. Executing this verification yields the result listed in Figure 7-22.

/* Trace run of equivalence property check of BPEL4WS process over MSC */
Composition:
CheckMSC = BPEL4WSArchitectureModel || Abs_Process_SPEC.ArchitectureModel.FingerprintEnquiry ||
Abs_Process_SPEC.ArchitectureModel.VehicleEnquiry ||
Abs_Process_SPEC.ArchitectureModel.NominalEnquiry ||
Abs_Process_SPEC.ArchitectureModel.InsuranceEnquiry ||
Abs_Process_SPEC.ArchitectureModel.PoliceEnquiry ||
Abs_Process_SPEC.ArchitectureModel.Officer_Device ||
Abs_Process_SPEC.ArchitectureModel.ANPREnquiry
State Space:
13 * 2 * 4 * 4 * 4 * 15 * 4 * 6 = 2 ** 20
Analysing...Depth 2 -- States: 2 Transitions: 3 Memory used: 5607K
**/* Trace run example from Equivalence Verification of MSC over BPEL4WS */**
Trace to property violation in BPEL4WSArchitectureModel:

```
receive_officer_pitobasic_process
invoke_pitobasic_insuranceservices_getvehicleinsurance
```

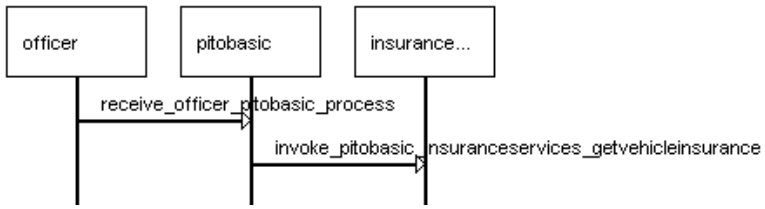



**Figure 7-22 Results of trace equivalence test to check scenarios not covered by BPEL4WS composition**

The trace that is listed has highlighted that the BPEL4WS does not exhibit the behaviour to support a Vehicle Insurance Enquiry (stated by the transition violation of a request followed by a lookup_vehicle_insurance transition). The engineer may wish to discuss this with the designer, however, with our initial knowledge of the specification's requirements, the project is keen that either service may be initiated first so that concurrency of activities may be utilised to increase performance of the composition. The BPEL4WS engineer therefore revisits the composition again and studies how this requirement can be implemented. A solution appears to be to introduce the FLOW concurrent activity execution construct in

BPEL4WS (discussed in section 4.3.2). The modified BPEL4WS process structure is illustrated in Figure 7-23.

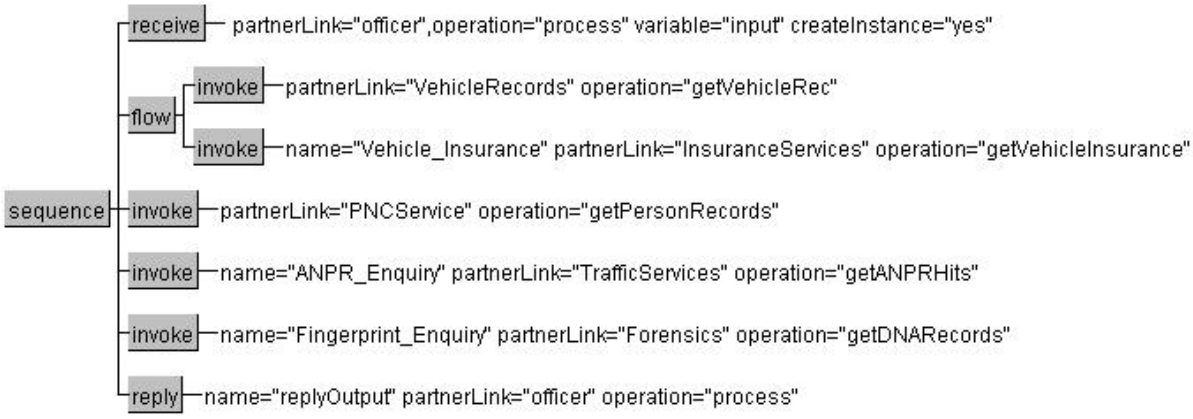



**Figure 7-23 Modified BPEL4WS Process to support FLOW of Vehicle Enquiry and Insurance Invocations**

Performing the verification against this modified process however, highlights a further issue. Notice that in the modified BPEL4WS process we have specified that the remaining interaction activities are in sequence following the completion of the concurrent flow of vehicle enquiries. This raises the violation (listed in Figure 7-24) that a Nominal Enquiry (to lookup a person record) cannot occur until both vehicle enquiries have successfully completed. This issue requires further elaboration on the BPEL4WS process to introduce further concurrency but *linked* to each of the vehicle enquiries in completion.

/* Trace run of equivalence property check of modified BPEL4WS process over MSC */

Trace to property violation in BPEL4WSArchitectureModel:

```
receive_officer_pitobasic_process
invoke_pitobasic_vehiclerecords_getvehiclerec
reply_vehiclerecords_pitobasic_ownerregdoc
invoke_pitobasic_insuranceservices_getvehicleinsurance
```

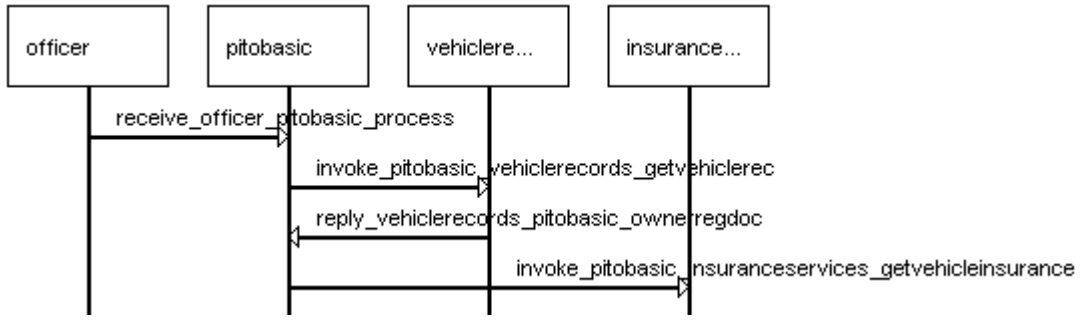



**Figure 7-24 Trace equivalence verification to check current vehicle enquiries in BPEL4WS composition**

What is now required is a link between the source of acquiring the person id (through either of the Vehicle Enquiry service invocations) and the target of performing the Nominal Enquiry (lookup_person_record). In BPEL4WS, this can be achieved through the use of *linked transitions* (discussed in section 4.3.3). The newly modified BPEL4WS process now includes this constraint on transition, by firstly placing all the partner service interactions in an additional FLOW, and then adding a source link to the invocation (and reply) of the Vehicle Enquiry and a target link (of the same name) to the invocation of the Nominal Enquiry.

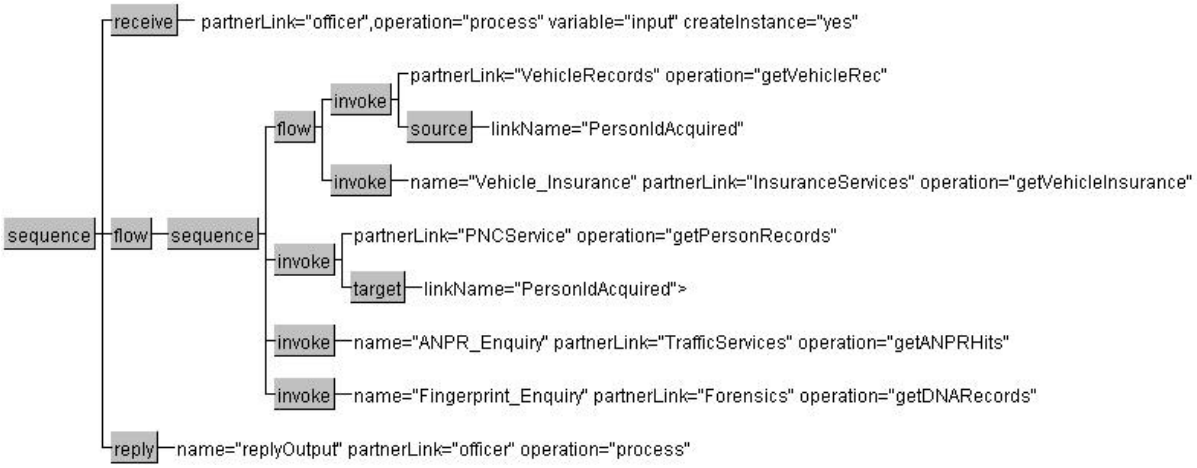



**Figure 7-25 Modified BPEL4WS Process to support LINKED transitions of Vehicle Enquiry and Nominal Enquiry**

Further analysis suggests that the linking is also required to fulfil the other set of scenarios, detailing the constraint that the ANPR Enquiry may also follow the Vehicle Enquiry (as illustrated previously in Figure 7-13). The engineer therefore adds a further source link to the Vehicle Enquiry (*VehicleIDAcquired*) and a target link to the ANPR Enquiry service invocation activity. The last link is considered against the Fingerprint/DNA Enquiry. This can only occur if the Nominal Enquiry has completed. The engineer completes the links constraints by placing a source on the Nominal Enquiry invocation and target on the Fingerprint Enquiry invocation. To complete the process, the engineer adds a FLOW wrapper to the ANPR and Fingerprint Enquiries such that either activity may commence after the Nominal Enquiry has completed. This updated process is illustrated in Figure 7-26. A final verification for trace equivalence is made on this final process, with results listed in Figure 7-27.

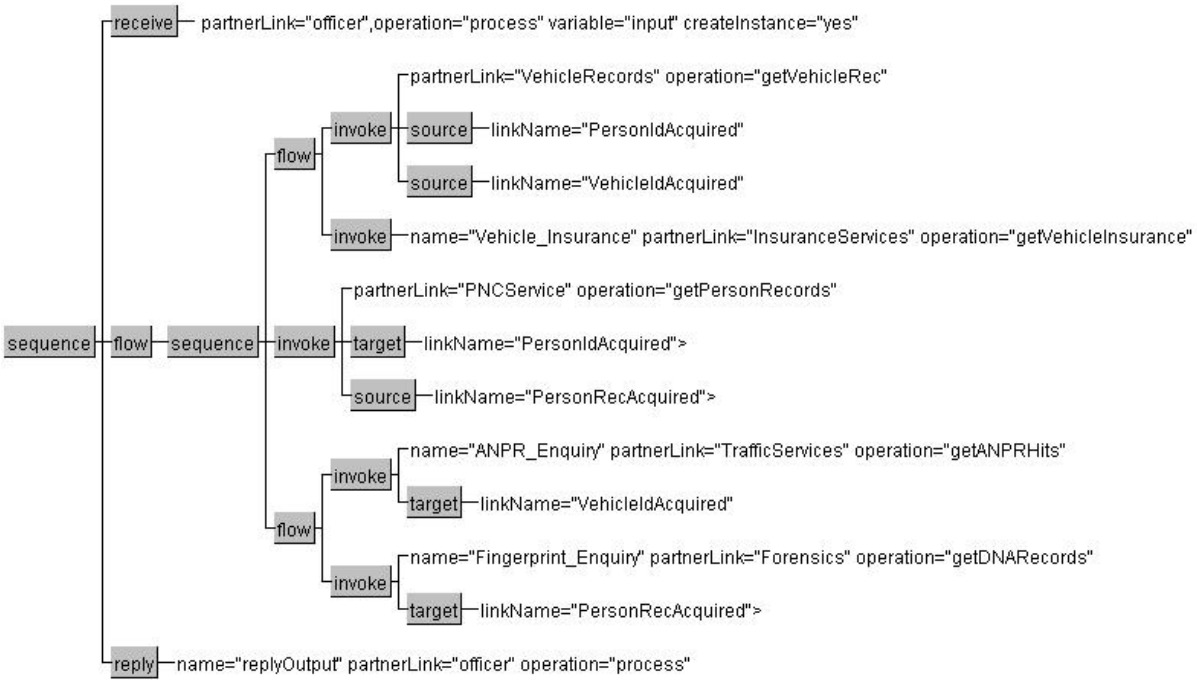



**Figure 7-26 Final BPEL4WS process for verification**

```
/* Trace run of equivalence property check of final BPEL4WS process over MSC */
State Space:
13 * 13 = 2 ** 8
Analysing...Depth 13 -- States: 13 Transitions: 12 Memory used:
5104K
      No deadlocks/errors
```

**Figure 7-27 Final BPEL4WS process verification against MSC specification**

The engineer has constructed a single BPEL4WS process for the Police Enquiry composition and has verified that the basic sequence (where all activities are completed in turn, with no initial links between activities) can be fulfilled by the process through a trace verification of the process model against the design specification model. The process can then be extended to support additional scenarios, where the invocation of Vehicle Enquiry and Insurance Enquiry could occur in parallel. Through further verification, it was established that this alone would not fulfil the additional scenarios, as a constraint was required to support the linking of Vehicle and Nominal Enquiries (of which the latter is dependent on completion of the first). These steps fulfilled the implementation of the first set of scenarios in the pilot

project case study. In a wider context, the pilot project also considerd how multiple compositions would be composed and we suggest how this is covered in choreography for the pilot project.

### *7.2.7 Choreography*

The choreography aspect of this project work focuses back on the example from our original motivation for the approach (Chapter 1, Figure 1-1). The police enquiry composition will interact with other services, themselves potentially compositions. Here we consider how these compositions can be verified together, for elaboration of scenarios in web service choreography. Addressing choreography takes us back to the designer, who may reuse existing composition scenarios to act as a source for interactions observed in those compositions. For example in the ANPR Enquiry (used as part of the vehicle movement checking requirement) the ANPR Enquiry service may consist of other traffic related service enquiries. Ideally we would assume that the service will eventually reply to a ANPR Enquiry. However, as we discussed previously, choreography provides a global view of requirements for one or more scenarios – such as in this case, that the ANPR does eventually reply to the Police Enquiry. Additionally, the engineer can gain greater confidence in the composition working alongside other partnered compositions. As choreography describes the global goal and more noticeably an understanding of a global state we introduce a third service into the PITO Police Enquiry requirements. The requirements are expanded to include an *authorisation* service which holds state of enquiry requests and provides a control on which services may be accessed in an enquiry type. The requirements are detailed in Figure 7-28.

A suspicious vehicle with the number plate "xxxxxxxxx" has been identified by a police officer in Northern England. The officer launches a formal police enquiry about the vehicle including its registration record and checking the vehicle's movement in the last 24 hours at key points in Scotland. ***Each enquiry type must be authorised and recorded at each request in the process.***

**Figure 7-28 A Pilot Project Scenario for Web Service Composition in PITO**

The Authorisation Service composition process is a key to the service choreography, by which each enquiry service must request authorisation before proceeding in the officer's request. Reusing the composition for Police Enquiry and ANPR (Vehicle Movement) composition, our domain of interest is depicted as in Figure 7-29.

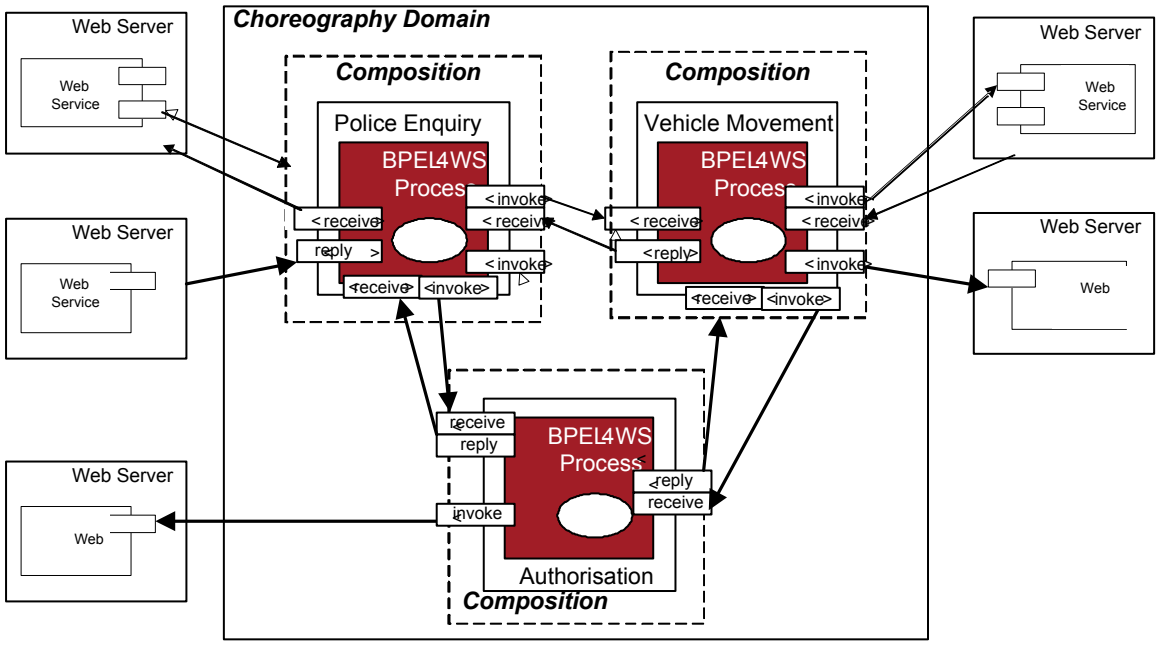



**Figure 7-29 Overview of choreography of elaborated composition scenario**

The designer specifies the choreography requirements in a further scenario. As we discussed in Chapter 3, the choreography is spread across components in the specification, with interactions occurring between multiple-parties, such that the police enquiry is not the single focus. The specification for this choreography scenario is illustrated in Figure 7-30. Notice that for each enquiry request, an invocation of the authorisation service is undertaken. The reply of a result from the authorisation is taken as the request has been granted in the current request's state. For simplicity, the designer has specified only one scenario for this choreography, and as such the behaviour of all compositions and services within the implementation of this choreography must exhibit behaviour suitable for this interaction sequence. As with the composition implementation we build the compositions supporting this requirement in BPEL4WS and then use verification, and specifically compatibility verification to ensure that the behaviour of these collaborating compositions is suitable to fulfil the requirements in the design specification.

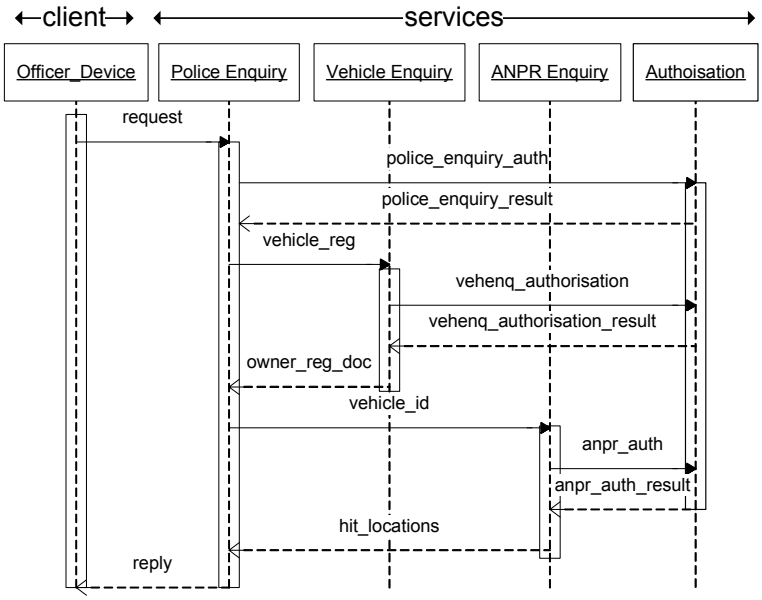



**Figure 7-30 Specification for scenario of Vehicle, ANPR and Authorisation Enquiries**

The BPEL4WS compositions consist of a Police Enquiry process, a Vehicle Enquiry process, ANPR Enquiry process and Authorisation process. Again, we have simplified the processes to support this scenario as a base for creating extended processes supporting other enquiry types. The Police Enquiry composition process is a subset of the interactions built in the process used previously. Indeed, it is the case that this additional scenario may be simply included in the current process, with the use of a SWITCH statement to distinguish which type of Police Enquiry is undertaken. However, for clarity we build a new Police Enquiry composition process outlining just the interactions in this authorisation scenario, with the distinct additions of *invoke_enquiry_auth* and *invoke_enquiry_result* activities.

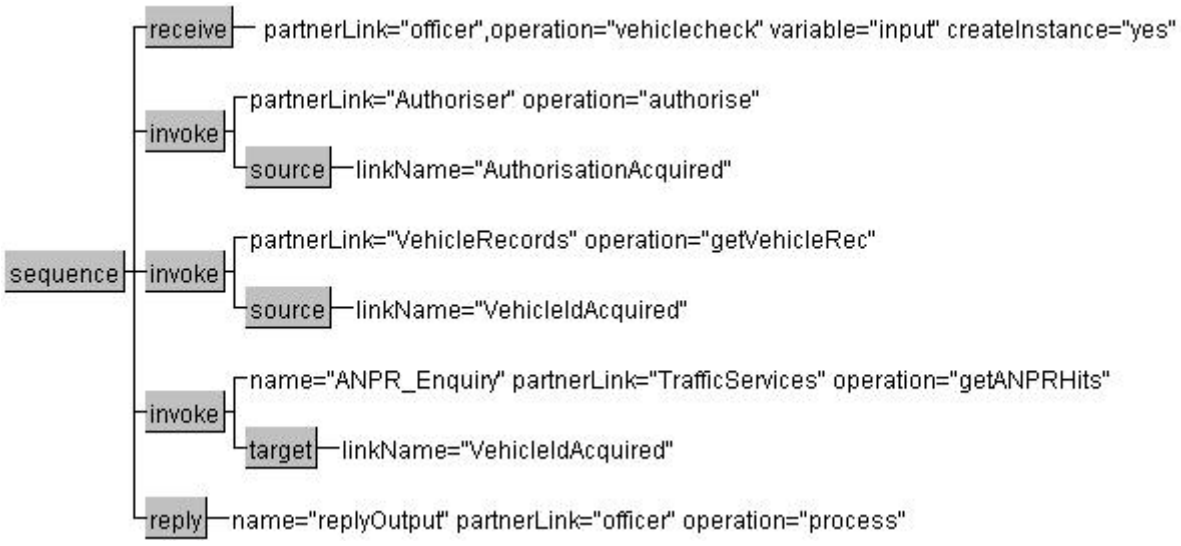



**Figure 7-31 Police Enquiry composition in Choreography example**

Similarly, the composition processes are built for Vehicle and ANPR Enquiries. In this example, these act simply as a wrapper enquiry composition, supporting the authorisation and invocation of support services to provide vehicle and plate recognition hits respectively.

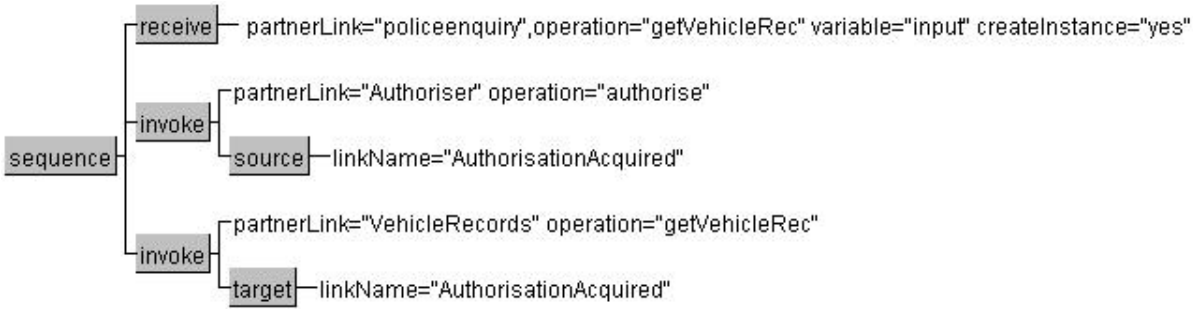



**Figure 7-32 Vehicle Enquiry composition in Choreography example**

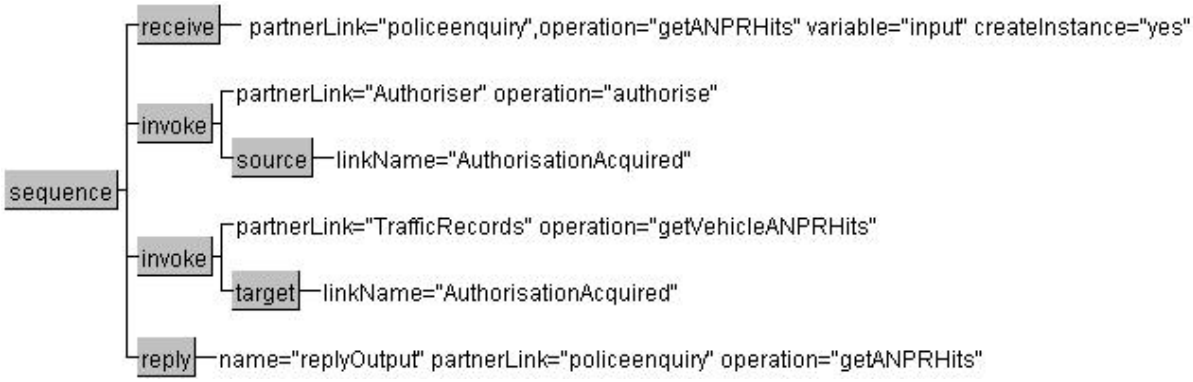



**Figure 7-33 ANPR Enquiry (Traffic Services) in Choreography example**

Lastly the authorisation process is built as a simple composition which accepts an authorisation request document, containing the enquiry type (policeenquiry, vehiclecheck or ANPRHit enquiry) and replies with whether the enquiry type is authorised or not for the given service session. This composition therefore takes the form of a sequence with a receive and reply activity only.

The process of compatibility verification is undertaken by specifying these three compositions, along with their related WSDL interface documents, in the compilation of models to analyse. The algorithm that we described in section 5.2.4 is executed and a series of port connectors built to link between the compositions. One such connector, in this case for the PoliceEnquiry and VehicleCheck compositions, is illustrated in Figure 7-34.

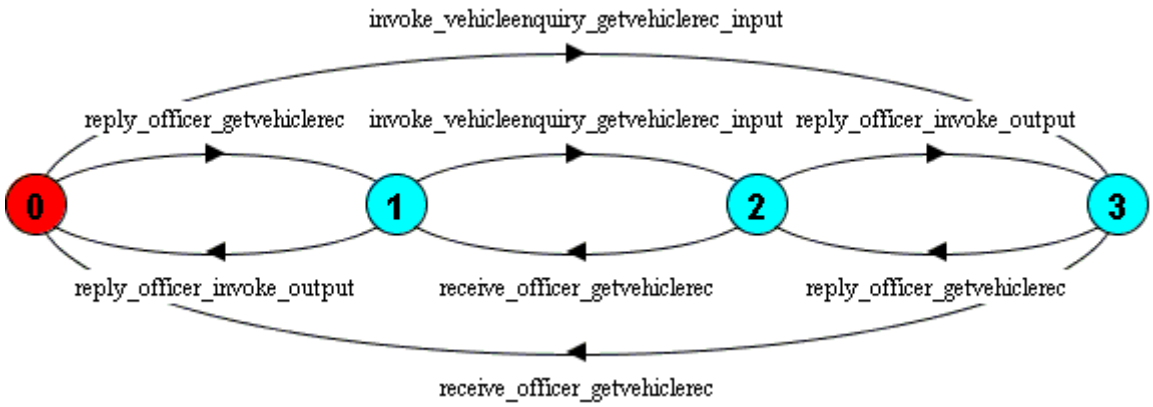



**Figure 7-34 Port Connector model between Police Enquiry and Vehicle Enquiry compositions**

Firstly, we perform a safety analysis of the choreography model, analysing it for deadlock freedom. The result of such verification is illustrated in Figure 7-35.

**/* Trace run example from compatibility verification of BPEL4WS service processes and partnered process interactions in the PITO Police Enquiry example*/**

**Trace to DEADLOCK:**

receive_officer_pitobasic_process

invoke_pitobasic_authoriser_authorise

invoke_pitobasic_vehicleenquiry_getvehiclerec

reply_pitobasic_officer_process

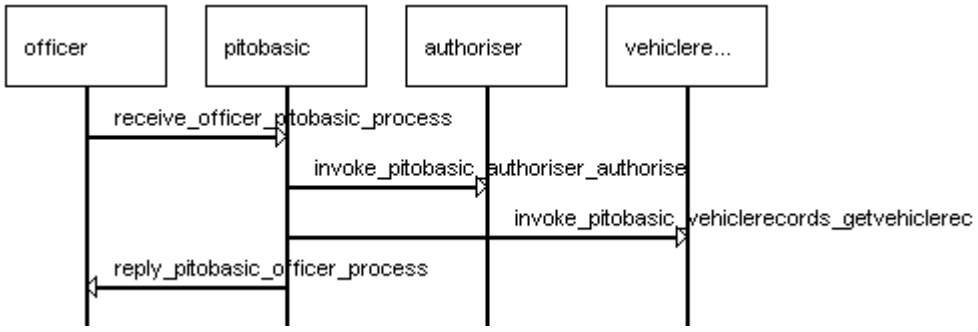



**Figure 7-35 Deadlock example of compatibility verification BPEL4WS and partnered compositions**

The reason for this deadlock is suggested in the last action of the trace – if we study the process illustrated back in Figure 7-32, we can observe that there is no Vehicle enquiry reply action specified. Consequently, the port models cannot be synchronised and a trace to deadlock is observed in the verification. To solve this issue, the engineer can add a reply and complete the model. A subsequent compatibility verification of the choreography provides

the engineer with a suitable "no deadlocks/errors found" successful result. Again, we can also perform the trace equivalence of this composed model against the specification by repeating the method described in the composition analysis example previously. Note that we have not given an example of mapping labels back to the specification here – it is assumed that as part of the composition builds, the engineer has repeated this task again. Running the trace equivalence test provides the results as illustrated in Figure 7-36.

```
/* Trace run of equivalence property check of BPEL4WS process over MSC */
State Space:
 3 * 2 * 2 * 2 * 2 * 6 * 4 * 2 * 2 = 2 ** 13
Analysing...
Depth 10 -- States: 18 Transitions: 25 Memory used: 4116K
No deadlocks/errors
Analysed in: 0ms
```

**Figure 7-36 Results of trace equivalence test to check BPEL4WS partially fulfils MSC specification**

### *7.2.8 Summary and Discussion*

In this section of our work, we have described a mechanical tool as an implementation of the approach and a series of work undertaken as part of the scope for a pilot project within the Police IT Organisation in the UK. The focus of this pilot project so far has been limited to trialling a centralised composition that sequences a series of calls to a number of web services. As part of our work, we have introduced the choreography aspect, which is an anticipated evolution of the actual pilot project adopted by PITO. As the number of compositions grows, the more difficult the observation of how the separate compositions will work together will be and specifically, what possible interactions could occur in various scenarios. Realising this through an extended example, we have aimed to illustrate that the approach and tool supporting our work can be used to consolidate these scenarios and compose models for easier assessment of combined compositional behaviour. Furthermore, the results gained from this initial case study have provided a real evaluation ground, in terms of real actors, scenarios and environments to evaluate the approach and tool away from our academic focus. This concludes describing the practicalities of the case study. We evaluate fully the effectiveness of the approach taken in this case study, the results acquired and feedback from the designers and implementers of using the tool and approach in the following sections.

# Chapter 8

# Evaluation and Conclusions

*"The medium, the process of our time – electric technology…*
*it is causing us to rethink and re-evaluate practically every thought, every action!"*
(Marshall McLuhan, Theorist and Educator, 1911-1980)

---

A key goal of the work in this thesis has been to describe an approach to designing and implementing concurrent and distributed web service compositions in a service-oriented architecture. A starting point was to examine the behaviour of components or services in this service-oriented architecture, and through design specifications, realise how developers could gain greater assurance by performing verification and validation analysis of implementations using formal software process modelling techniques. Our work has resulted in an approach which identifies key artefacts to use in this analysis, and key processes in performing verification and validation against these artefacts. The nature of web service interactions has provided the detail to which tasks these key processes must undertake to provide useful results for this assurance reasoning. In this chapter we evaluate our approach.

## 8.1 Evaluation of Approach

In this section, with provide an evaluation of the approach described in this thesis and from the case study in section 7.2. Using the results and feedback gathered by way of the case study, and through our experience of engineering web service compositions with and without the approach we evaluate the steps and their results. The evaluation of the approach is split

between the theory and its practical usage with a view of clarifying the features that make it *rigorous*.

### *8.1.1 On Design Specifications*

The first aspect of the approach we consider is that of the early requirements gathering steps and by way of describing these as specifications, how this captures the essence of interactions between web service compositions. At a level aimed at designers, in other words those who do not *build* the service components but specify *what* service components are involved in a process composition, the design specifications provide a high level description of how the services communicate and more specifically, which order the conversations made in communication occur. The ability to describe sufficient information in these high level interactions is obviously a prerequisite to being able to use models of these specifications in analysis and verification. Whilst it is possible to elaborate on interactions and specify some internal activities (such as a repetition of interaction or alternatively called *self-interaction*) this does not provide a useful mechanism to describe web service compositions or other services included in the scenarios described in a choreography domain (e.g. state management). The reasons for our use of message sequence charts are described as three main objectives. Firstly, sequence charts are a graphical tool with which provides an intuitive interface to describing the sequence of component communication. Secondly, the formation of scenarios is consistent with the protocol in which to define basic message sequence charts (bMSC) for each alternative communication sequence anticipated by the composition designer. Thirdly, by the nature of sequence chart interaction transitions, charts may be synthesised to formal models and used in further analysis through formal process modelling techniques. We have demonstrated how this technique can be used in the context of describing web service composition interactions, yet clearly, the domains in which this can be applied is not limited to this domain.

Our approach and tool support for MSCs design of web service compositions is presently limited in several ways. Firstly, the designer is unable to represent data dependencies between partners which therefore constrain interaction descriptions to the point of *a type* of message rather than a *value* of a message being passed between partners. In compositional design this is not highly detrimental to the conciseness of interactions, yet when describing choreography rules (where state and message part values are equally important) this will impact the verification approach's effectiveness in modelling conditions to analyse.

Secondly, the set of basic message charts and its higher level sequence chart can become complex to manage in itself. This complexity is exhibited when there evolves a high number of alternative scenarios to describe and sequence. If for example, a concurrent set of five interactions are permissible in a section of a composition, then the designer must describe each alternative case for invocation and reply of these five interactions. Through undertaking the case study in section 7.2, it was found that designers do not naturally think of concurrent requests and replies, highlighted by the observation that the designer considered it much greater effort to build many alternative scenarios. Yet at the same time however the designers considered the approach as an aid to a more rigorous design than had previously been undertaken.

Furthermore, at the time of writing this work, several consortiums are compiling specification languages to describe interactions, monitor and provide state transfer between choreography scenarios, yet it appears there is overwhelming support by the authors of these specifications, that describing these in a practical approach will be with a suitable sequence chart representation. Indeed, earlier work using message charts, in a similar way we make use of its notation for interaction specifications, has commented on the ability for developers (designers in this context) to easily specify what is required in scenarios. This related work has been reported on using extensions to the standard UML notation, in addition to sequence charts, to provide a level of design for the web service components themselves (Nüttgens 2003). Additional practical support for this design approach is given by Rational Software corporation in extensions to UML Specification 1.1 (RATIONALSOFTWARE 1997; Nüttgens 2003). This work however, still centralises on an approach to define core components, in a now commonly used object-oriented analysis and design of components with the focus on attributes and methods (and associated message calls to other methods). Where this approach gains in the detail of exact operations and attributes within components, it is felt that it lacks in observing the nature of service behaviour, in other words, the potentially ad-hoc conversations that may occur in architectures of partnered processes. Kept a high level, such as message sequence charts, the key interactions in one or many scenarios can be captured and used in formal verification. Additionally, other uses of MSCs, such as for negative scenarios or implied scenarios, can be captured to describe detailed constraints on interaction scenarios permissible in a given composition.

### *8.1.2 On Modelling Implementations*

Our approach to modelling implementations has focused on the standards used to compose web services, the semantics of those standards and a model representation of abstracted interactions from these composition processes. Amongst those standards, BPEL4WS appeared the most completed notation and has been reported under consideration for both academic and industry projects. We approached modelling the web service composition implementations by way of three steps. Firstly, the engineer builds the composition processes in the standard notation (in this case BPEL4WS). Secondly, a mechanical process abstracts the interactions and constructs that affect interaction behaviour from the composition processes and builds a model representing these interactions in a finite state machine representation. The semantics used to build the model are discussed in Chapter 4. The core of this *semantic translation* is based upon the semantics as defined for the Finite State Process (FSP) notation. This notation has gathered a strong user base, and has also been used in various other research projects. The ability to translate BPEL4WS to FSP is core to building the model of the composition process. Other process languages have been used in a translation similar to ours including (Wohed, Aalst et al. 2002; Duan, Bernstein et al. 2004; Hamadi and Benatallah 2004; Fu, Bultan et al. 2004b). Our approach to modelling implementations does not differ greatly to the steps described in these works. Indeed, it is encouraging to read that the workflow expressions of BPEL4WS have been translated in the same way, albeit to another process language as an end result. We have added to this translation, an interpretation of those activities which are interaction or non-interaction based, and also the wider choreography rules through process interaction mapping (as discussed in Chapter 5).

Our work has relied greatly on the ability to translate the semantics of BPEL4WS to that of FSP, and yet at the same time aiming to preserve the behaviour that would be executed by a standard compliant BPEL4WS process engine. The translation of BPEL4WS was originally based on the specification version 1.0 (Curbera, Goland et al. 2002) and then updated to include changes to naming and semantics as defined in version 1.1 (Curbera, Goland et al. 2002; Iyengar 2003). The work was, at the time of writing this thesis, wholly dependent on our view of the mapping between BPEL4WS and the FSP notation. At the time of completing this work there are emerging implementations of BPEL4WS, such as the open source BPEL engine of ActiveBPEL LLC (ActiveBPEL 2005), the IBM BPEL engine

project (Curbera, Duftler et al. 2002) and the ORACLE BPEL process manager (ORACLE 2005). A potential validation of our mapping could utilise the behaviour of these engines with respect to the logging functions of their service interactions. This is discussed further in the future work section of this chapter. There are however, already some key issues in assumptions used within this translation from the BPEL4WS specification. For example, the specification for the *Terminate* construct in BPEL4WS is officially described as that any activity in current execution will *eventually* terminate. This is not concrete enough to be able to simulate a terminate (and it's effect) for each differing construct of the specification. Our assumption has been to translate this to a STOP process in the FSP algebra but may require further interpretation. In addition, the fault tolerance and compensation sections of BPEL4WS used in this work, have not been discussed to a great extent. We are keen to expand the approach to include this, but potentially this impacts a wider scope than just the compositions and interaction choreography. The global state will be affected by any compensation action that takes place in composition. The reason for this is if either a fault is raised due to a service failure (be it technically or business process driven) then any partners of that composition will need to be notified to take appropriate actions if necessary. The modelling therefore covers a much broader scope than we have covered in this thesis to date. Further specifications, such as the WS-CDL (Kavantzas, Burdett et al. 2004) should be useful in linking with modelling choreography to determine the effects on other services not necessarily directly partnered with a composition service.

Lastly, a comment on transitional representation of data variables used within a composition process. Both compositions and choreography implementations can contain activities which are constrained by data values either returned by a interaction with other partnered services, or are dependent upon values stored within the composition or choreography itself. For example, a decision point within a composition can be represented by the *switch/case* statement pair. This evaluates an expression, and takes one of a selection of activity paths based upon the result of the expression. In our modelling, we have used an enumeration to label each possible path with a unique identifier. When these enumerations are built as part of the model, the compiled state machine builds a path for each alternate activity route. Further data analysis would bring us closer to modelling with dynamic analysis to determine actual values or possible values passing between the service components.

### *8.1.3 On Verification and Validation*

Fred Brooks observed that to achieve a dramatic reduction in development time, a new technology would have to simplify the essence of software development (Brooks 1987). The general observation was that the more a technology strikes at the essence of what makes software development difficult (duration, maintaining and cost), the better the results will be. We believe in addition to this that providing an easy to learn, easy to use and repeatable verification and validation process yields further gains against the issues in software development.

We have described a verification approach of building design models suitable for the analysis and verification of implementation processes that could be deployed into a, potentially global, distributed environment. Through our goal of facilitating greater assurance to composition engineers, these processes can collaboratively integrate into this distributed environment, and that the result obtained through verification yields suitably rich information such that engineers can adjust either design or implementation to provide greater stability to such a deployed process.

Notably, the key step of mapping activities between the models is still something that inhibits a fully mechanical approach. As we discussed in refining the service composition models for verification (section 6.3) a mapping of activities is required to perform trace equivalence of the implementation against design. This can only be achieved through a common understanding between implementation engineer and the service designer. One method to ease this mapping task is for the designer to follow a concise labelling of activities, whereby the interaction is labelled in the same style as generated in our translation whilst this does not have to be exact to the implementation labelling generation, We have also chosen to centralise focus on the trace equivalence of implementation models against design models yet the verification is not limited to this. Further work can be undertaken to provide other types of verification, such as extended safety and liveness properties, which is likely to be more useful when addressing choreography issues with partnered processes that are not necessarily linked directly to interactions in a given composition process. Other types of verification, such as fluent properties (Miller and Shanaham 1999), will provide a greater range of analysis techniques to evolve this approach.

On the validation of web service compositions, we have described this through model simulation and animation (in section 6.5). In this work we provided a simple validation mechanism illustrated through step-by-step interaction analysis of the model selected for validation. Whilst this is not exclusively reserved for design, it is felt that in the approach a designer would gain the most benefit by observing interactively what has been described in the models produced by MSC synthesis. The essence of the simulation aspect of validation is also provided through this animation mechanism. A trace run can be analysed interactively, yet there is scope of improved simulation by way of allowing the user to specify certain conditions (in terms of conditional statements) which can allow for testing of various scenarios in the environment. An example of this may be to introduce certain faults and ascertain how the composition recovers from this failure, and how the partnered compositions process is similarly affected.

### *8.1.4 On Iteration*

The principle of our approach is to provide mechanical verification in an evolutionary style of development. It is hypothesised that the steps of the approach (through design, implementation and verification) will not be undertaken in a waterfall development approach style, such as the Spiral Model described in (Boehm 1988). Moreover, it is anticipated that the style of web service composition development will be more akin to Rapid Application Development (RAD) approaches, such as the Dynamic Systems Development Methodology (DSDM) (DSDM 1995). Our reasoning for this is that with reusable components, an initial requirements baseline is considered when the first deployment occurs. The expectations of these components will quickly be exhausted as new requirements and further functionality are required by additional partners in a composition (Larrson and Crnkovic 1999; Yang and Papazoglou 2003). Thus, the approach we describe has also the thought that it must support a highly iterative process. We feel that the approach has clear boundaries in terms of inputs required and outputs gained which assists in this repetition. For example, there are two consistent sets of input criteria. Firstly, there is the set of compositional definitions (specification, implementation, interfaces and web service standards) which remains consistent in each iteration of the approach. Secondly, aligned with the compositional definitions, is a set of verification and validation properties that the composition must fulfil. We describe these as consistent in the sense that there is no fluctuation in the number of sets for input or output. It is expected that the contents of these sets will change as each iteration is undertaken.

## 8.2 Evaluation of Tool Support

For the tool support evaluation, we appropriately split the discussion into a series of criteria for evaluation taken from the work in (Clarke and Wing 1996). This criteria considers tool support from several viewpoints including; *ease of learning*, *early payback*, *efficiency of developer's time*, *increase in benefits*, *error detection*, *integrated development environment enabled*, *focus on analysis* and support for *evolutionary development*.

### *8.2.1 Ease of learning*

*"Notations and tools should provide a starting point for writing formal specifications for developers who would not otherwise write them. The knowledge of formal specifications needed to start realizing benefits should be minimal"* (Clarke and Wing 1996).
Our approach and the tool built to support it, aims at providing the following criteria for its ease of learning and carrying out verification and validation.

- The design specifications are based on the scenario approach, and the use of bMSC and hMSC sequence charts is widely undertaken and understood in industry. Where other work has concentrated on specifying formal algebraic notations for specifications, we provide a graphical interface so that the user does not have to learn these sometimes complex notations.
- The implementations are constructed either directly in the tool or through a third-party tool such that the BPEL4WS engineers are not restricted in a particular editor implementation or feature list to use our approach. There is already evidence of several BPEL4WS editors in the Eclipse development environment.
- Verification properties can be specified by reusing the approach for building design specifications. For example, the designer, in addition to building a complete set of service scenarios for equivalence verification, may also submit further safety or liveness properties by way of constructing bMSCs that specify these individual requirements.
- Validation is undertaken through an animated label transition system interface. Validation can be undertaken with two views. Firstly, the designer can animate their design specification and make initial assessment of what composition interactions should occur, in which order and in relation to other compositions for choreography scenarios. Secondly, the BPEL4WS engineer can verify implementations of composition process

interaction through the same interface. Either party can examine counterexamples generated by the verification steps in our approach through this interface.

- Results can be translated back in the format in which specifications were generated (i.e. both in bMSC scenarios). Counterexamples show were differences have been detected, and iterative checking can be used to manage each scenario and how each change made to a composition affects the set of interactions covering all scenarios.

### *8.2.2 Early Payback*

*"Methods and tools should provide significant benefits almost as soon as people begin to use them"*. (Clarke and Wing 1996).

Early payback is a key objective of the approach. As we discussed in Chapter 1, Section 1.1 in terms of motivation for this work, our aim is to support answering questions highlighted by the distributed nature of web service compositions and on consideration of the pattern by which these compositions may interact. Clearly, the benefits of using such an approach will require early feedback to the developers so that assurance can be given, in both design and implementation activities, as earlier as possible. We achieve this by separating the tool between design and implementation, and consolidate their output to provide another view for analysis. The iterative development does not necessarily suggest that service compositions will be designed and implemented in isolation; moreover, we believe that compositions will be part of collaborative developments cross-enterprise and yet still between several development teams. An example of this is from the Police Enquiry case study discussed earlier. The initial specification and composition design suggested a simple sequence of interactions between one compositions interacting with up to five partnered services. Further elaboration of the scenarios possible from that composition illustrated that the other partnered services may also be compositions. Indeed, one of the last elaborations in the case study suggested that several compositions all communicated with an “Authorizer” composition. Clearly, it can be seen that several design specifications and implementation models may be used in this case study, which are not necessarily undertaken by a single developer or engineer. By providing early feedback, in terms of verification and validation, these types of projects can resolve local differences and yet at the same time consolidate global requirements for choreography scenarios.

### *8.2.3 Efficiency*

*"Tools should make efficient use of a developer's time. Turnaround time with an interactive tool should be comparable to that of normal compilation. Developers are likely to be more patient, however, with completely automatic tools that perform more extensive analysis"* (Clarke and Wing 1996)

We have not given in depth analysis on the efficiency of our approach in this work. We have provided some examples, such as in the Police Enquiry case study and in the evaluation sections of this chapter to how the complexity of design specifications, joined with complex composition implementations yield large process machines for analysis. Further work must be undertaken to assess where efficient changes to analysis and associated algorithms are streamlined to given optimal performance. Clearly, our approach relies heavily on and is limited by, the efficiency of the underlying model checking technology. However, state of the art model checkers, such as LTSA, have proven to manage efficiently large behaviour models (Cleaveland and Smolka 1996).

### *8.2.4 Incremental gain for incremental effort*

*"Benefits should increase as developers get more adept or put more effort into writing specifications or using tools"* (Clarke and Wing 1996).

The most complex part of our approach is in determining the composite interactions in web service choreography. Our work utilizes an algorithm to link compositions and by way of elaborated design specifications, these interactions are compared with those of the choreography requirements. The developers do not have to use the approach in such a way, for example, they may simply chose to isolate verification at a single compositional level (examining one process against its interactions with other, black box, services), yet it is believed that as the developers become used to the approach that they will seek further assurance in wider, cross-enterprise solutions.

### *8.2.5 Orientation toward error detection*

*"Methods and tools should be optimised for finding errors, not for certifying correctness. They should support generating counterexamples as a means of debugging"* (Clarke and Wing 1996).

The essence of our approach is to highlight inconsistencies between interactions specified in composition implementations against that of those given in design specifications. We do not aim to clarify notational and specification semantic correctness, although that can be achieved to a degree by user validation through animation. We assume that correctness of implementations is a given attribute of the inputs submitted for observing errors against design specification scenarios, and as such, this provides orientation of our approach towards error detection rather than correctness of these artifacts.

### *8.2.6 Integrated use*

*"Methods and tools should work in conjunction with each other and with common programming languages and techniques_ Developers should not have to buy into a new methodology completely to begin receiving benefits. The use of tools for formal methods should be integrated with that of tools for traditional software development. E.g. compilers and simulators."* (Clarke and Wing 1996)

From a technical implementation perspective, we wished to provide the tool as much as a reusable service as that of which it is used to verify. In this way, we have scoped the architecture for the tool to be extendable and integrated without a presumption of which interfaces would be used to build the inputs to the tool core. In other words, the BPEL4WS engineers are free to build the compositions in any supporting editor, yet on the one condition that the output from these editors conforms with the same specification supported by our tool. We also do not believe in forcing a new methodology upon developers by way of the tool, but support various methodologies in the tasks that must be undertaken regardless of the actual steps of a methodology e.g. verification and validation can be undertaken in either design, implementation or maintenance.

### *8.2.7 Focused Analysis*

*"Methods and tools should be good at analysing at least one aspect of a system well, for example, the control flow of a protocol. They need not be good at analysing all aspects of a system"* (Clarke and Wing 1996).

In a similar way as we discussed in section 8.2.5, we believe that our focus is on providing verification analysis of composition implementations against those built as design

specifications from web service composition requirements. Validation has been discussed, but is really an additional benefit to building the core software process models. This is the core analysis that we perceive such an approach will be undertaken, yet an alternative view is that this can also lead to other forms of analysis, such as checking fluent properties (Uchitel, Chatley et al. 2004).

### *8.2.8 Evolutionary Development*

*"Methods and tools should support evolutionary system development by allowing partial specification and analysis of selected aspects of a system"*. (Clarke and Wing 1996).

By the nature of web service compositions, they may represent only one part of a service-oriented architecture. From the discussions above, it is clear that our approach provides an incremental, elaborative approach to building compositions and realising the effects of changes as they are introduced in the life-time of service. What is perhaps more interesting is that as a software engineering community we are used to hearing about lifecycles of systems, and yet the service-oriented architecture (implemented in one part through web services compositions) can be seen to avoid that practice. In other words, individual components of architecture may be removed or replaced, yet the service may still exist. Related to this, we still need to undertake further work in providing greater assurance to developers in areas such as fault tolerance, compensation and upholding choreography policies.

## 8.3 Summary of Contributions

The main contribution of this thesis is to provide an approach, which when implemented within a tool, demonstrates a mechanical verification of properties of interest to both designers and implementers of web service compositions. The use of a formal, well defined, process algebra (in this case FSP) provided a semantic mapping between the composition implementation (in the BPEL4WS specification for web service compositions), and we were fortunate to be able to leverage some work previously reported in (Uchitel 2003) for the synthesis of design specifications, in the form of message sequence charts, to the same process algebra. These two representations as models form the basis to provide further model-based verification.

Furthermore, our contribution consisted of several specific features. Firstly we built behaviour models of both design specifications and implementation processes on the basis that they modelled web service compositions, providing a guide to how this was achieved for both local and global compositions and their choreography. Secondly, we provided a guide on how to translate the semantics of the BPEL4WS specification to FSP and map implementation abstractions which preserve the interaction behaviour between services, yet also disposing of process characteristics which are not required in the analysis. Thirdly, we elaborated these models to analyse the conversations of compositions across choreography scenarios, providing both interface and behavioural compatibility verification processes. Fourthly, we collaborated with the UK Police IT Organisation to illustrate a real and practical example of how our approach may assist in web service composition development. This not only provided a ground to prove our approach and gain feedback from users, but also gains invaluable experience where currently there is a lack of reported findings in real-world situations. Finally, we have contributed a plug-in tool for both the existing LTSA plug-in architecture but also contributed to the open community through development of an equal plug-in feature for the Eclipse development environment.

## 8.4 Future Work

Fundamentally, the future opportunities from undertaking this work have been discovered through some of the limitations observed in the evaluation section of this chapter, and by the dynamic and evolving nature of the service industry and research.

Firstly, on the current approach, the method of constructing design specifications, in the form of basic message sequence charts provides further opportunity to allow message data dependencies of composition behaviour to be considered. The aspects of this that would most benefit our approach would be to observe how different message part values (between service partners) yields alternative scenarios that can be verified against implementations of compositions and choreography policies. For example, the state of a choreography enactment between several service partners can only be verified if the differing values of state are known at design time. In an order processing choreography, this would naturally include such state as “order placed” or “order could not be placed” for example. Choreography defines how this state affects partnered processes which are not necessarily involved in direct interactions. Related to this is the data representation within the

composition implementations. Future work could evolve this representation to provide expressions of data values within process algebra models.

The types of property used in verification are also open to a much broader range than suggested in this work. The aspect of goal-based objectivies of a system is a particular opportunity through the concept of checking fluent properties. Fluents are abstractions of system state specified in terms of the occurrence of events. (Miller and Shanaham 1999) informally define (propositional) fluents as follows: "Fluents (time-varying properties of the world) are true at particular time-points if they have been initiated by an event occurrence at some earlier timepoint, and not terminated by another event occurrence in the meantime. Similarly, a fluent is false at a particular time-point if it has been previously terminated and not initiated in the meantime." This type of property provides useful state analysis over the period that a service choreography is undertaken, raising the scope of verification from composition and interactions to choreography policies. Within this future work, we wish to continue describing behaviour by elaborating on the wider choreography aspects of partnered service compositions. This includes considering fault, compensation and transactional integrity within and between distributed processes. As part of this we are closely working with consortiums, such as the W3C, on their work with choreography architectures and specifications. It is anticipated that the result of their work could be incorporated into our approach to provide an extension to the choreography elements we have considered thus far. To assess our assumptions in translation of BPEL4WS semantics to that of FSP semantics, we are also seeking to provide a mechanism to check the models produced in this approach against trace runs output from BPEL4WS process engine instances. This is one way to evaluate how accurate the translation is, although consequently, there is always the question of whether the engine itself has been built to standards. We can therefore only compare expected with actual results based upon an assumption that the implementation engine and execution of a process are on best endeavours.

Secondly, the web services field is very much standards driven, and by the very nature of standards, complying means keeping up-to-date on standards released and supporting new or amended features. The expectation of this is that whilst BPEL4WS is the standard for web service compositions as of this date, newer alternatives may superceed BPEL4WS. We believe the principles applied in this work however (for verification and validation of processes), will remain consistent but will require updated work in the translation of these

notations to software process models. Furthermore, we are not independent of other closely related work in the techniques used in our approach. For example, the support in scenario-based elaboration and implied scenarios is providing easier and accurate methods to support describing requirements using such techniques as message sequence charts. Lastly, we also believe we can extend the mechanism to resolve issues highlighted in the results from verification and validation, by for example, tracing and highlighting parts of implementations that relate to violations in the models analysed. We currently support presentation in the form that the designer builds scenarios in a MSC editor, yet equally, the BPEL4WS engineer should also have an accessible view as to which part of the composition it relates to (by for example, MSC representations or syntax highlighting).

## 8.5 Closing Remarks

The need for pre-development and pre-deployment reasoning about the system behaviour has been addressed by software engineers and software engineering researchers for many years. To this end, a significant effort has been made in developing modelling notations, automated analysis techniques and tool support. However, providing an intuitive interface to building design models of service compositions and verifying these against implementations has been largely neglected. Model construction and elaboration are engineering activities in their own right, and developing support for these activities is a key challenge. Support which complements existing behaviour modelling notations, analysis techniques and tools should provide sound model-based engineering methods for software development.

# Bibliography


Aalst, W. M. P. v. d. (2004). Pi calculus versus Petri nets: Let us eat "humble pie" rather than further inflate the "Pi hype".

Aalst, W. M. P. v. d., M. Dumas, et al. (2003). Web Service Composition Languages: Old Wine in New Bottles? Proceeding of the 29th EUROMICRO Conference: New Waves in System Architecture, Los Alamitos, CA, IEEE Computer Society.

Abadi, M. and L. Lamport (1993). "Composing specifications." ACM Transactions on Programming Languages and Systems **15**(1): 73-132.

Abadi, M. and L. Lamport (1995). "Comjoining specifications." ACM Transactions on Programming Languages and Systems **17**(3): 507-534.

ACPO (2002). Association of Chief Police Officers (ACPO), ACPO Information Systems Strategy - Version 2.0. United Kingdom.

ActiveBPEL (2005). ActiveBPEL - The Open Source BPEL Engine. Available from: http://www.activebpel.org/, ActiveBPEL LLC.

Anderson, A., A. Nadalin, et al. (2004). "eXtensible Access Control Markup Language (XACML) - Committee draft 04, 6 Dec." from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

Ankolekar, A., M. Burstein, et al. (2002). DAML-S: Web Service Description for the Semantic Web. 1st International Semantic Web Conference (ISWC), Sardinia, Italy.

Arkin, A. (2002). "Business Process Markup Language (BPML) Specification Version 1.0." from http://www.bpmi.org/_vti_bin/shtml.exe/bpml-spec.htm.

Arkin, A., S. Askary, et al. (2002). Web Service Choreography Interface (WSCI) 1.0 - W3C Note 8 August 2002, W3C - Web Services Choreography Working Group.

Austin, M. (2004). "ENSE 622: Systems Engineering Requirements, Design, and Trade-Off Analysis." from http://www.isr.umd.edu/~austin/ense622.html.

Bajaj, S., D. Box, et al. (2004, September 2004). "Web Services Policy Framework (WS-Policy)." s. 2004, from http://www-106.ibm.com/developerworks/library/specification/ws-polfram/.

Balci, O. (1994). Validation, Verification, and Testing Techniques Throughout the Life Cycle of a Simulation Study. 26th conference on Winter simulation, Orlando, Florida, USA.

Banerji, A., C. Bartolini, et al. (2002). "Web Services Conversation Language (WSCL) v1.0." from http://www.w3.org/TR/wscl10/.

Basten, T. (1998). In Terms Of Nets: Systems Design with Petri Nets and Process Algebra. Eindhoven, Endhoven University of Technology. **PhD Thesis:** 237.

Bergstra, J. A., A. Ponse, et al. (2001). Handbook of Process Algebra, ELSEVIER.

Berners-Lee, T. (2000). Weaving the Web. San Francisco, HarperBusiness.

Berners-Lee, T., R. Fielding, et al. (1998). Uniform Resource Identifiers (URI): Generic Syntax, Internet Engineering Task Force.

Bharadwaj, R. and C. Heitmeyer (1999). "Model Checking Complete Requirements Specifications Using Abstraction." Automated Software Engineering **6**(1): 37-68.

Boehm, B. (1988). "A Spiral Model of Software Development and Enhancement." IEEE Computer **21**(5): 61-72.

Bolcer, G. A. and R. N. Taylor (1998). "Advanced workflow management technologies." Software Process - Improvement and Practice **4**(3): 125-171.

Bolognesi, T. and E. Brinksma (1987). Introduction to the ISO Specification Language LOTOS. Computer Networks and ISDN Systems.

Bonett, M. (2001). Personalization of Web Services: Opportunities and Challenges. Ariadne. **June**.

Booth, D., H. Haas, et al. (2004). "Web Services Architecture - W3C Working Group Note 11 February 2004." Retrieved 7th January, 2004, from http://www.w3.org/2002/ws/arch/.

Booth, D., H. Haas, et al. (2004, 28 October 2002). "Web Services Architecture (WS-A) - W3C Working Group Note 11 February 2004." Retrieved 7th January, 2004, from http://www.w3.org/TR/ws-arch/.

Box, D., D. Ehnebuske, et al. (2000). Simple Object Access Protocol (SOAP) 1.1. See: http://www.w3.org/TR/SOAP/, W3C SOAP WORKING GROUP.

Brogi, A., C. Canal, et al. (2004). Formalizing Web Services Choreographies. 1st International Workshop on Web Services and Formal Methods (WS-FM 2004), Pisa, Italy.

Brooks, F. P. (1987). "No Silver Bullets - Essence and Accidents of Software Engineering." IEEE Computer: 10-19.

Bukhres, O. and C. J. Crawley (1996). Failure Handling in Transactional Workflows Utilizing CORBA 2.0. 10th ERCIM Database Research Group Workshop on Heterogeneous Information Management, Prague.

Cabrera, F., G. Copeland, et al. (2002). Web Services Coordination (WS-Coordination), BEA Systems, IBM, Microsoft Corporation.

Carbone, M., K. Honda, et al. (2005). "Programming interaction with Types." from http://lists.w3.org/Archives/Public/public-ws-chor/2005Dec/0002.html.

Carbone, M., K. Honda, et al. (2005). A Theoretical Basis of Communication-Centred Concurrent Programming, Available from: http://lists.w3.org/Archives/Public/public-ws-chor/2005Nov/att-0015/part1_Nov25.pdf.

Castilho, M., L. A. Kunzle, et al. (2004). A Petri Net Based Representation for Planning Problems. 5th International Conference on Knowledge Based Computer Systems, Hyderabad, India.

Chandy, K. M. and A. Rifkin (1996). "Systematic Composition of Objects in Distributed Internet Applications: Processes and Sessions." Computer Journal **40**(8).

Chatley, R., J. Kramer, et al. (2003). Model-based Simulation of Web Applications for Usability Assessment. The workshop on Bridging the Gaps between Software Engineering and Human-Computer Interaction.

Checkland, P. (1982). Systems Thinking, Systems Practice. Chichester, UK, John Wiley and Sons.

Checkland, P. (1990). Soft Systems Methodology in Action. Chichester, UK, John Wiley and Sons.

Christensen, E., F. Curbera, et al. (2001). Web Services Description Language (WSDL) 1.1 - W3C Note 15 March 2001. Internet, W3C XML Activity on XML Protocols.

Christensen, E., F. Curbera, et al. (2003). Web Services Description Language (WSDL) 1.2, W3C.

Clarke, E. M., O. Grumberg, et al. (1994a). "Model Checking and modular verification." ACM Transactions on Programming Languages and Systems **16**(3): 843-871.

Clarke, E. M., O. Grumberg, et al. (1994b). "Model Checking and Abstraction." ACM Transactions on Programming Languages and Systems **16**(5): 1512-1542.

Clarke, E. M. and J. M. Wing (1996). "Formal Methods: State of the Art and Future Directions." ACM Computing Surveys **28**(4): pp626-643.

Cleaveland, R., J. Parrow, et al. (1993). "The concurrency workbench:A semantics-based tool for the verification of concurrent systems." ACM Transactions on Programming Languages and Systems **15**(1): 36 –72.

Cleaveland, R. and S. A. Smolka (1996). "Strategic Directions in Concurrency Research." ACM Computing Surveys **28**(4): 607-625.

Curbera, F., M. J. Duftler, et al. (2002). "BPWS4J: A platform for creating and executing BPEL4WS processes."

Curbera, F., M. J. Duftler, et al. (2004). "The IBM Business Process Execution Language for Web Services JavaTM Run Time (BPWS4J) - V2.1 - April 13, 2004." 2004, from http://www.alphaworks.ibm.com/tech/bpws4j.

Curbera, F., Y. Goland, et al. (2002). Business Process Execution Language For Web Services, Version 1.0.

Damianou, N., N. Dulay, et al. (2001). The Ponder Specification Language. Workshop on Policies for Distributed Systems and Networks (Policy2001), HP Labs, Bristol, UK.

De-Leon, H. and E. Grumberg (1993). "Modular Abstractions for Verifying Real-Time Distributed Systems." Formal Methods in System Design **2**(1): 7-43.

DSDM. (1995). "The Dynamic Systems Development Methodology - Version 2.0."

Duan, Z., A. Bernstein, et al. (2004). Semantics Based Verification and Synthesis of BPEL4WS Abstract Processes. 3rd IEEE International Conference on Web Services, San Diego, CA.

Duftler, M. J., N. K. Mukhi, et al. (2001). "Web Services Invocation Framework (WSIF)." 2004, from http://www.research.ibm.com/people/b/bth/OOWS2001/duftler.pdf.

Edelstein, H. (1994). "Unraveling Client/Server Architecture." DBMS **7**(5): 34.

Engels, G., J. M. Kuster, et al. (2003). "Model-Based Verification and Validation of Properties." Electronic Notes in Theoretical Computer Science **82**(7).

Erdogmus, H. (1997). "Architecture-Driven Verfication of Concurrent Systems." Nordic Journal of Computing **4**(NRC 41549): 380-413.

Ferrara, A. (2004). Web Services: A Process Algebra Approach. The 2nd International Conference on Service Oriented Computing (ICSOC'04), New York City, NY, USA, ACM Press.

Foster, H. (2003b). "LTSA-BPEL4WS Tool." from http://www.doc.ic.ac.uk/ltsa/bpel4ws.

Foster, H. (2004b). "BPEL Code Samples." from http://www.bpelsource.com/resources/code.html.

Foster, H., S. Uchitel, et al. (2003a). Model-based Verification of Web Service Compositions. Eighteenth IEEE International Conference on Automated Software Engineering (ASE), Montreal, Canada, IEEE.

Foster, H., S. Uchitel, et al. (2004a). Compatibility for Web Service Choreography. 3rd IEEE International Conference on Web Services (ICWS), San Diego, CA, IEEE.

Foster, H., S. Uchitel, et al. (2005). Tool Support for Model-Based Engineering of Web Service Compositions. 3rd IEEE International Conference on Web Services (ICWS2005), Orlando, FL, IEEE.

Foster, H., S. Uchitel, et al. (2005). Using a Rigorous Approach for Engineering Web Service Compositions: A Case Study. 2nd IEEE International Conference on Services Computing (SCC2005), Orlando, FL, IEEE.

Fowler, M. (2003). "Components and the World Of Chaos." IEEE Software **3**(3): 83-85.

Frantz, F. K. (1995). A Taxonomy of Model Abstraction Techniques. the Winter Simulation Conference, New York, NY, Association for Computing Machinery.

Fu, X. (2004d). Formal Specification and Verification of Asynchronously Communicating Web Services, Phd. Thesis. Santa Barbara, CA, USA, University of California.

Fu, X., T. Bultan, et al. (2004). "Conversation Protocols: A Formalism Specification and Verification of Reactive Electronic Services."

Fu, X., T. Bultan, et al. (2004). WSAT: A tool for Formal Analysis of Web Services. 16th International Conference on Computer Aided Verification (CAV), Boston, MA.

Fu, X., T. Bultan, et al. (2004b). Analysis of Interacting BPEL Web Services. 3rd IEEE International Conference on Web Services (ICWS), San Diego, CA.

Gardner, T. (2003). UML Modelling of Automated Business Process with Mapping to BPEL4WS. European Workshop on Object Orientation and Web Services, Darmstadt, Germany.

Gardner, T. (2003). UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. First European Workshop onWeb Services and Object Orientation (ECOOP 2003), Darmstadt, Germany.

Garg, P. K. and W. Scacchi (1989). "ISHYS: Design of an Intelligent Software Hypertext Environment." IEEE Expert **4**(3): 52-63.

Gluch, D. P., S. Cormella-Dorda, et al. (2001). Model-Based Verification: Abstraction Guidelines. Pitssburgh, PA, Software Engineering Institute.

Graubmann, P. (2003). "Describing interactions between MSC components: the MSC connectors." The International Journal of Computer and Telecommunications Networking **42**(3): 323-342.

Gudgin, M. and M. Hadley (2003). Web Services Description Language (WSDL Binding) 1.2 - W3C Working Draft 8 December 2004. Internet, W3C Web Services Activity.

Gudgin, M. and M. Hadley (2004). Web Services Description Language (WSDL Binding) 1.2 - W3C Working Draft 8 December 2004. Internet, W3C Web Services Activity.

Gudgin, M., A. Lewis, et al. (2004). "Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns - W3C Working Draft 26 March 2004." from http://www.w3.org/TR/2004/WD-wsdl20-patterns-20040326/.

Haas, H. (2002). Web Services Activity - W3C Web Services Activity Group.

Hailpern, B. and P.Santhanarn (2002). "Software debugging, testing and verification." IBM Systems Journal **41**(1): 4-12.

Hall, R. J. (2003). Open Modeling in Multi-stakeholder Distributed Systems: Model-based Requirements Engineering for the 21st Century. Proc. First Workshop on the State of the Art in Automated Software Engineering, U.C. Irvine Institute for Software Research.

Hamadi, R. and B. Benatallah (2004). A Petri Net-based Model for Web Services Composition. 3rd IEEE International Conference On Web Services (ICWS), San Diego, CA.

Heitmeyer, C., J. Kirby, et al. (1998). "Using Abstraction and Model-Checking to Detect Safety Violations in Requirements Specifications." IEEE Transactions on Software Engineering **24**(11): 932-941.

Hoare, C. A. R. (1985). Communicating Sequential Processes, Pentice-Hall.

Hogg, T. and B. A. Huberman (1991). "Controlling chaos in Distributed Systems." IEEE Transactions on Systems Management and Cybernetics **21**: 1325-1332.

Holzmann, G. J. (1997). "The Model Checker SPIN." IEEE Transactions on Software Engineering **23**(5): 1-17.

Holzmann, G. J. (1997). "The Model Checker Spin." IEEE Transactions on Software Engineering **23**(5): pp. 279-295.

Holzmann, G. J. (2003). The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley Professional.

Hruby, P. (1998). Specification of Workflow Management Systems with UML. OOPSLA Workshop on Implementation and Application of Object-oriented Workflow Management Systems, Vancouver, BC.

Hu, M. (2003). Web Services Composition, Partition, and Quality of Service in Distributed System Integration and Re-engineering. XML Conference 2003, Philadelphia, PA, IDEAlliance.

Hu, M. (2004). Quality of Service Composition and Factoring In Composite Web Services Based Business Process. XML Conference 2004, Washington D.C., USA, IDEAlliance.

Huff, K. E. and V. R. Lesser (1989). A Plan-Based Intelligent Assistant that Supports the Software Development Process. Third Software Engineering Symposium on Practical Software Development Environments.

IBM. (2004). "IBM Eclipse Innovation Awardees." from http://www-306.ibm.com/software/info/university/products/eclipse/eig-2004.html.

IBM. (2005). "IBM Eclipse Innovation Awardees." from http://www-306.ibm.com/software/info/university/products/eclipse/eig-2004.html.

ISO (1995). Open Distributed Processing - Reference Model - Part2: Foundations, International Standard 10746-2 / ITU-Recommendation X.902.

ITU (1996). Message Sequence Charts, Recommendation Z.120, International Telecommunications Union. Telecommunication Standardisation Sector.

Iyengar, S. (2003). Business Process Integration Using UML and BPEL4WS. XML Conference & Exposition 2003, Philadelphia, PA, IDE Alliance.

Iyengar, S. (2003). Business Process Integration Using UML and BPEL4WS. XML Conference and Exposition 2003, Philadelphia, PA.

Jacobson, I., J. Rumbaugh, et al. (1999). The Unified Software Development Process, Addison-Wesley, Harlow, UK.

Jiang, P., Q. Mair, et al. (2003). Using UML to design distributed collaborative workflows: from UML to XPDL. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Linz, Austria.

Johnson, J., T. L. Roberts, et al. (1989). "The Xerox "Star": A Retrospective." IEEE Computer **22**(9): pp 11-29.

Karamanolis, C., D.Giannakopoulou, et al. (1999). Modelling and Analysis of Workflow Processes. London, Imperial College of Science, Technology and Medicine.

Kavantzas, N., D. Burdett, et al. (2004). Web Service Choreography Description Language (WS-CDL) - W3C Working Draft 17 December 2004, W3C - Web Services Choreography Working Group.

Kavantzas, N., D. Burdett, et al. (2004). "Web Services Choreography Description Language Version 1.0 - W3C Working Draft 17 December 2004." from http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/.

Khalaf, R., N. Mukhi, et al. (2003). Service-Oriented Composition in BPEL4WS. The Twelfth International World Wide Web Conference, Budapest, HUNGARY, WWW2003.

Kling, R. and W. Scacchi (1982). "The Web of Computing: Computer Technology as Social Organization." Advances in Computers **21**: 1-90.

Koshkina, M. (2003). Verification of Business Processes for Web Services. Department of Computer Science. Toronto, Ontario, York University.

Lamsweerde, A. v. (2001). Goal-Oriented Requirements Engineering: A Guided Tour. 5th IEEE Intl. Sym. on Requirements Engineering (RE'01), Toronto, Canada.

Larrson, M. and I. Crnkovic (1999). New Challenges for Configuration Management. 9th Software Configuration Management Workshop, Toulouse, France.

Larrson, M. and I. Crnkovic (1999). New Challenges for Configuration Management. the SCM-9 workshop, Toulouse, France, Springer-Verlag.

Leiner, B. M., V. G. Cerf, et al. (2002). "A Brief History of the Internet." version 3.32. from http://www.isoc.org/internet/history/brief.shtml.

Levi, K. and A. Arsanjani (2002). "A goal-driven approach to enterprise component identification and specification." Communications of the ACM **45**(10): pp 45-52.

Leymann, F. (2001). Web Services Flow Language (WSFL 1.0), IBM Academy Of Technology.

Lynch, A. N. and M. R. Tuttle (1987). Hierarchical Correctness Proofs for Distributed Algorithms. 6th Annual Symp. on Principles of Distributed Computing, Vancouver, Canada.

Magee, J. and J. Kramer (1997). Exposing the Skeleton in the Coordination Closet. 2nd International Conference COORDINATION '97, Berlin, Germany.

Magee, J. and J. Kramer (1999). Concurrency - State Models and Java Programs, John Wiley.

Magee, J., J. Kramer, et al. (1997). Analysing the Behaviour of Distributed SoftwareArchitectures: a Case Study. 5th IEEE Workshop onFuture Trends of Distributed Computing Systems, Tunisia.

Magee, J., J. Kramer, et al. (1999). Behaviour analysis of Software Architectures. 1st Working IFIP Conference On Software Architecture (WICSA1), San Antonio, TX, USA.

Maghrabi, T. H. (2004). "ICS 411 "Senior Project" (032) - Research-Based Projects." 2004, from http://faculty.kfupm.edu.sa/ics/maghrabi/ics411-032/ics-topics.doc.

Mantell, K. (2003). From UML to BPEL. Available from: www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel, IBM.

Microsoft (2001).NET: Driving Business Value with the Microsoft Platform, Microsoft Corporation.

Miller, R. and M. Shanaham (1999). "The Event Calculus in Classical Logic - Alternative Axiomatisations." Linkoping Electronic Articles in Computer and Information Science **4**(16): p 1-27.

Milner, R. (1980). A Calculus of Communicating Systems.

Milner, R. (1989). Communication and Concurrency. Upper Saddle River, NJ, USA, Prentice-Hall, Inc.

Milner, R., J. Parrow, et al. (1992). "A Calculus of Mobile Processes." Information and Computation **100**(1): 1-40.

Murata, T. (1989). "Petri Nets: Properties, Analysis and Applications." Proceedings of the IEEE **77**(4): 541-580.

Nakajima, S. (2002). Model-Checking Verification for Reliable Web Service. OOPSLA 2002 Workshop on Object-Oriented Web Services, Seattle, Washington.

Nakajima, S. (2002). On Verifying Web Service Flows. SAINT 2002 Workshop - WebSE 2002.

Narayanan, S. and S. A. Mcllraith (2002). Simulation, Verification and Automated Composition of Web Services. Eleventh International World Wide Web Conference (WWW-11), Honolulu, Hawaii.

Nuseibeh, B. and S. Easterbrook (2000). Requirements engineering: A roadmap. International Conference on Software Engineering (ICSE'00), Limerick.

Nüttgens, M. (2003). "Business Process Modeling with EPC and UML Transformation or Integration?"

OASIS (1993). Organization for the Advancement of Structured Information Standards (http://www.oasis-open.org).

OMG (2002). Unified Modelling Language, Available at: http://www.omg.org.

ORACLE (2005). ORACLE BPEL Process Manager. Available from: http://www.oracle.com/technology/products/ias/bpel/index.html.

Osterweil, L. (1987). Software processes are software too. the 9th International Conference on Software Engineering, Monterey, CA USA.

Paananen, J. (1995). Introduction to and comparison of formalisms. Tik-110.501 Seminar on Network Security. Available at: http://www.tml.tkk.fi/Opinnot/Tik-110.501/1995/intfo.html. Helsinki University of Technology.

Papazoglou, M. and J. Yang (2002). "Design Methodology for Web Services and Business Pro- cesses." Lecture Notes in Computer Science **2444**.

Pavlovic, D. and D. R. Smith (2002). Guarded Transitions in Evolving Specifications. 9th International Conference on Algebraic Methodology And Software Technology (AMAST 2002), St. Gilles les Bains, Reunion Island, France, Springer-Verlag LNCS.

Petri, C. A. (1966). Technical Report RADC-TR-65-377. New York, Griffiss Air Force Base**:** Vol 1. Suppl 1.

Pistore, M., M. Roveri, et al. (2004). Requirements-driven Verification of Web Services. 1st International Workshop on Web Services and Formal Methods (WS-FM 2004), Pisa, Italy.

RATIONALSOFTWARE. (1997). "UML Extension for Business Modeling version 1.1." Unified Modeling Language version 1.1, from http://www.rational.com/uml/documentation.html.

Roberts, L. G. and B. D. Wessler (1970). Computer Network Development to Achieve Resource Sharing. Spring Joint Computer Conference, AFIPS Proceedings.

Ross-Talbot, S. (2004). "Web Services Choreography and Process Algebra." 2004.

Salaun, G., A. Ferrara, et al. (2004). Negotiation Among Web Services Using LOTOS/CADP. European Conference on Web Services (ECWS2004), Erfurt, Germany.

Scacchi, W. (2000). "Understanding software process redesign using modeling, analysis and simulation." Software Process–Improvement and Practice.

Schlimmer, J. C. (2002, 28 October 2002). "Web Services Description Requirements." Retrieved 7th January, 2002, from http://www.w3.org/TR/ws-desc-reqs/.

Schumaker, K. (1999). "A Taxonomy of Simulation Software." from http://antioch.rice.edu/etrac/lester/thesaurus_br.html.

Seeley, R. (2003). "Berners-Lee: Integrate Web services and Semantic Web. Quote from Gartner Web Services and Application Integration conference." from http://www.adtmag.com/article.asp?id=7662.

Sherman, D., D. Shaffer, et al. (2002). Orchestrating Asynchronous Web Services, Collaxa.

Siegel, J. (2003). Using OMG's Model Driven Architecture (MDA) to Integrate Web Services, Object Management Group.

Soley, R. (2003). White paper: Model-Driven Architecture, Object Management Group (OMG).

Srivastava, B. and J. Koehler (2003). Web Service Composition - Current Solutions and Open Problems. The 13th International Conference on Automated Planning & Scheduling (ICAPS), Trento, Italy.

Stevens, P. (1999). Tools and Algorithms for the Construction and Analysis of Systems. 5th International Conference, TACAS'99, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands.

Subramanian, S. (1993). A Mechanized Framework for Specifying Problem Domains and Verifying Plans. Department Of Computer Science. Austin, Texas, University of Texas.

Sun (2001). Implementing Services on Demand With the Sun Open Net Environment - Sun ONE. Sun Professional Services White Paper. I. Sun Microsystems. Palo Alto, CA.

Thatte, S. (2001). XLANG - Web Services For Business Process Design, Microsoft Corporation.

Uchitel S., R.Chately, et al. (2003). LTSA-MSC: Tool Support for Behaviour Model Elaboration Using Implied Scenarios. Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Warsaw, Poland.

Uchitel, S. (2003). Incremental Elaboration of Scenario-Based Specifications and Behaviour Models Using Implied Scenarios. Distributed Software Engineering. London, Imperial College London.

Uchitel, S., R. Chatley, et al. (2004). Fluent-Based Animation: Exploiting the Relation between Goals and Scenarios for Requirements Validation. Requirements Engineering (RE'04).

Uchitel, S., J.Magee, et al. (2001). Detecting Implied Scenarios in Message Sequence Chart Specifications. 9th European Software Engineering Conferece and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE'01), Vienna, Austria.

Uchitel, S. and J. Kramer (2001). A Workbench for Synthesising Behaviour Models from Scenarios. the 23rd IEEE International Conference on Software Engineering (ICSE'01), Toronto, Canada.

W3C-Org (1994). W3C - The World Wide Web Consortium (http://www.w3c.org).

Winograd, T. and F. Flores (1986). Understanding Computers and Cognition: A New Foundation for Design. Lexington, MA, Ablex Publishers.

Wohed, P., W. M. P. v. d. Aalst, et al. (2002). Pattern Based Analysis of BPEL4WS. Brisbane, Queensland University of Technology.

Woodman, S., D. Palmer, et al. (2004). Notations for the Specification and Verification of Composite Web Services. 8th IEEE International Enterprise Distributed Object Computing (EDOC) Conference, Monterey, California.

Yang, J. and M. P. Papazoglou (2003). "Service Components for Managing the Life-Cycle of Service Compositions." Information Systems.

Yi, X. and K. J.Kochut (2004). Towards Efficient Integration of Complex Web Services Using a Unified Model for Protocol and Process. 5th International Conference on Internet Computing (IC 2004), Las Vegas, Nevada, USA.

Yi, X. and Krys.J.Kochut (2004). Process Composition of Web Services with Complex Conversation Protocols: a Colored Petri Nets Based Approach. Design, Analysis, and Simulation of Distributed Systems Symposium, Washington DC, USA.

Yu, E. (1997). Towards Modeling and Reasoning Support For Early Requirements Engineering. 3rd International Symposium on Requirements Engineering (RE '97), Annapolis, MD.

# Appendix A – WS-*

## A.1 Web Service Standards

| Layer | Standards | Description |
|---|---|---|
| Profile | WS-I Basic Profile | Standard to ensure that vendors and web service providers meet an agreed usage of standards below. |
| Composition and Choreography | WS-BPEL/BPEL4WS (Business Process Execution Language) | The BPEL4WS specification defines a notation for specifying business process behaviour based on Web Services. BPEL4WS focuses on orchestration rather than choreography (i.e. by using a single controller) |
| | WS-CDL (Choreography Description Language) | Transactions among Web services and their clients must clearly be well defined at the time of their execution, and may consist of multiple separate interactions whose composition constitutes a complete transaction. This composition, its message protocols, interfaces, sequencing, and associated logic, is considered to be choreography. |
| Transaction | WS-Transaction | This specification describes coordination types that are used with the extensible coordination framework described in the WS-Coordination specification. It defines two coordination types: Atomic Transaction (AT) and Business Activity (BA). |
| | WS-Coordination | This specification (WS-Coordination) describes an extensible framework for providing protocols that coordinate the actions of distributed applications. Such coordination protocols are used to support a number of applications, including those that need to reach consistent agreement on the outcome of distributed activities. |
| | OASIS BTP | At a most simple level BTP allows a set of remote calls to be grouped together and the outcomes tied together. It allows for; all or nothing outcome; mixed outcome; service alternative recognition and selection; time qualification and exception reporting. |
| Reliability | WS-Reliability | WS-Reliability is a specification for open, reliable Web services messaging including guaranteed delivery, duplicate message elimination and message ordering, enabling reliable communication between Web services. |
| | WS-ReliableMessaging | This specification (WS-ReliableMessaging) describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures. |
| Security | WS-Security | A mechanism for incorporating security information into XML messages |
| | XML Encryption | Standard for encrypting and decrypting parts of xml documents |
| | SAML | Security Assertions Markup Language for role-based permissions on resources used by a web service |
| Discovery | WSDL | The Web Service Description Language is an XML format for describing network services as a set of endpoints |
| | UDDI (Universal Description, Discovery and Integration). | UDDI stands for Universal Description, Discovery and Integration. The UDDI specification enables businesses to quickly, easily, and dynamically find and transact with one another. UDDI enables a business to (i) describe its business and its services, (ii) discover other businesses that offer desired services, and (iii) integrate with these other businesses. |
| Message Format | SOAP | a lightweight xml protocol intended for exchanging structured information in a decentralized, distributed environment. |
| | UBL | a standard library of XML business documents (purchase orders, invoices, etc.) by modifying an already existing library of XML schemas to incorporate the best features of other existing XML business libraries |
| | ebXML | ebXML intends to develop a technical framework that will enable XML to be utilized in a consistent manner for the exchange of all electronic business data |
| Transport Mechanism | HTTP - HyperText Transfer Protocol | The protocol for moving hypertext (or other) files across the Internet. |
| Policies | WS-Policy | Policy frameworks and specifications for access control and obligations |
| | WS-SecurityPolicy | |
| | XACML | |

# Appendix B

# FSP SEMANTICS

## B.1 FSP Process Syntax

A process is defined by one or more local processes separated by commas. The definition is terminated by a full stop. STOP and ERROR are primitive local processes.
Examples: 1. Process = (a -> Local), 2. Local = (b -> STOP).

**Table B-1 FSP Process Operators**

| Operator | Description |
|---|---|
| **Action prefix**<br>`->` | `(x->P)` describes a process that initially engages in the action x and then behaves as described by the auxiliary process P |
| **Choice** "\|" | `(x->P \| y->Q)` describes a process which initially engages in either x or y, and whose subsequent behaviour is described by auxiliary processes P or Q, respectively |
| **Recursion** | the behaviour of a process may be defined in terms of itself, in order to express repetition |
| **End state**<br>`END` | describes a process that has terminated successfully and cannot perform any more actions |
| **Alphabet Extension**<br>+ | The alphabet of a process is the set of actions in which it can engage. P + S extends the alphabet of the process P with the actions in the set S. |

## B.2 Composite Processes

A composite process is the sequential or parallel composition of one or more processes. The definition of a sequential composite process is proceeded by ; (semi-colon) whereas a parallel composite process is proceeded by ||.
Example: 1. Sequence = P ; Q; END. 2. ||Composite = (P || Q).

**Table B-2 FSP Composite Process Operators**

| Operator | Description |
|---|---|
| **Sequential composition**<br>; | (P;Q) where P is a process with an END state, describes a process that behaves as P and when it reaches the END state of P starts behaving as the auxiliary process Q |
| **Parallel composition** \|\| | (P\|\|Q) describes the parallel composition of processes P and Q |
| **Replicator**<br>forall | Forall [i:1..N] P(i) is the parallel composition (P(1) \|\| ... \|\| P(N)) |
| **Process Labelling**<br>: | a:P prefixes each label in the alphabet of P with a. |
| **Process Sharing**<br>:: | {action1..actionx}::P replaces every label n in the alphabet of P with the labels action1.n , actionx.n. Further, every transition (n->Q) in the definition of P is replaced with the transitions ({action1.n,..,actionx.n}->Q). |

## B.3 Common Operators

The operators listed in table B-1 are common to both processes and composite processes.

**Table B-3 FSP Composite Process Operators**

| Operator | Description |
|---|---|
| **Condiitional**<br>`if then else` | The process (if B then P else Q) behaves as the process P if the condition B is true otherwise it behaves as Q. If the else Q is omitted and B is false then the process behaves as STOP. |
| **Relabelling**<br>/ | Re-labelling is applied to a process to change the names of action labels. The general form of re-labelling is / {newlabel/oldlabel}. |
| **Hiding**<br>\ | When applied to a process P, the hiding operator \ {action1, actionx} removes the action names from the alphabet of P and makes these concealed actions "silent". These silent actions are labelled tau. Silent actions in different processes are not shared. |
| **Interface**<br>@ | When applied to a process P, the interface operator @{action1, actionx….} hides all actions in the alphabet of P not labelled in the set action1…actionx. |

## B.4 Properties

**Table B-4 FSP Process Properties**

| Operator | Description |
|---|---|
| **Trace equivalence minimisation**<br>`deterministic` | `deterministic P` describes the minimal trace equivalent process to P. If no terminating traces are proper prefixes of other traces, then it also preserves END states |
| **Strong semantic equivalence minimisation**<br>`minimal` | Minimal P describes the minimal strong semantic equivalent process to P |
| **Safety**<br>`property` | A safety property P defines a deterministic process that asserts that any trace including actions in the alphabet of P, is accepted by P. |
| **Progress**<br>`progress` | progress P = {action1, …actionx} defines a progress property P which asserts that in an infinite execution of a target system, at least one of the actions action1…actionx will be executed infinitely often. |

# Appendix C

# BPEL4WS To FSP

## C.1 Primitive Activities to FSP

| BPEL4WS Construct | FSP Process Example | Comments |
|---|---|---|
| **(primitive activities)** | | |
| **Invoke**<br>**Receive**<br>**Reply** | `INVOKE = (invoke_p1_o1 -> END).`<br>`RECEIVE = (receive_p1_o1 -> END).`<br>`REPLY = (reply_p1_o1 -> END).` | Where p1 is a named partner, and o1 is a named operation. |
| **Terminate** | `INVOKE1 = (invoke_p1_o1 -> END).`<br>`INVOKE2 = (invoke_p2_o2 -> END).`<br>`Set ACTSET = {invoke_p1_o1,invoke_p2_o2}`<br>`TERMS = (ACTSET->TERMS \| terminate->END).`<br>`\|\|TERMINATE = (INVOKE1 \|\| INVOKE2 \|\| TERMS).` | Where p1 and p2 are named partners, and o1 and o2 are named operations. |

## C.2 Structured Activities to FSP

| BPEL4WS Construct | FSP Process Example | Comments |
|---|---|---|
| **Sequence** | `INVOKE = (invoke_p1_o1 -> END).`<br>`RECEIVE = (receive_p2_o2 -> END).`<br>`SEQUENCE = INVOKE; RECEIVE; END.` | Where p1 and p2 are named partners, and o1 and o2 are named operations. |
| **Flow** | `INVOKE = (invoke_p1_o1 -> END).`<br>`RECEIVE = (receive_p2_o2 -> END).`<br>`FLOW = (INVOKE \|\| RECEIVE).` | Where p1 and p2 are named partners, and o1 and o2 are named operations. |
| **Links** | `TLINK1 = (target_link -> END).`<br>`\|\|TARGETLINKS = (TLINK1).`<br>`SLINK1 = (source_link -> END).`<br>`\|\|SOURCELINKS = (SLINK1).`<br>`INVOKE = (invoke_p1_o1 -> END).`<br>`SEQUENCE =`<br>`TARGETLINKS; INVOKE; SOURCELINKS; END.`<br>`\|\|LINKPROCESS = (SEQUENCE).` | a) Where target_link is a named target operation link and source_link is a named source operation link.<br>b) Where p1 is a named partner, and o1 is a named operation.<br><br>Links are pre and post guarded transitions for an operation or a scope of operations. To model the synchronisation of linked transitions, the link process is composed with the source and target processes. |

## C.3 Guarded Activities to FSP

| BPEL4WS Construct | FSP Process Example | Comments |
|---|---|---|
| **(variable constructs – read/write operator process)**<br>`range VR = 0..n (where n is a possible range of values for evaluation)`<br>`VARIABLE(A=0) = VARIABLE[A},`<br>`VARIABLE[i:VR] = (write[j:VR]->VARIABLE[j] \| read[i] ->VARIABLE[i]),`<br>`VARIABLE['null] = (write[j:VR]->VARIABLE[j] \| read[i] -> VARIABLE['null]).` | | |
| **Assign** | `ASSIGN1 = (`*`assign_variable[x]`*` -> END.` | Where assign_variable is a process variable used in the composition, and x is an enumerated value to read or store. |
| BPEL4WS Construct | FSP Process Example | Comments |
| **While** | `WHILEAB = exp:WHILE_variable.`<br>`set WHILEEXP_alphabet =`<br>`{exp.{read,write}.[Range]}`<br>`WHILEEVAL = (exp.read[i:Range]-`<br>`>WHILEEVAL[i]),`<br>`WHILEEVAL[i:Range] = if (i==0)`<br>`then SEQ1; WHILEEVAL else END.`<br>`RECEIVE1 = (`*`receive_p1_o1`*` -> END).`<br>`SEQ1 = RECEIVE; END.`<br>`WHILESEQ = WHILEEVAL; END +`<br>`{WHILEAB_alphabet}.`<br>`\|\|WHILE = (WHILESEQ).` | a) where exp is of type WHILE_variable which is a declared read/write operator process.<br>b) where p1 is a named partner, and o1 is a named operation. |
| **Switch** | `CASE1EVALC = (exp.read[i:Range]`<br>`->CASE1EVALC[i]),`<br>`CASE1EVALC[i: Range] = if (i==0) then`<br>`CASE1; END else OTHERWISE; END.`<br>`\|\|CASE1EVAL = (CASE1EVALC).`<br>*`case1process`*` = (exp.write[0] ->END).`<br>`\|\|CASE1 = (`*`case1process`*`).`<br>*`case2process`*` = (exp.write[1] -> END).`<br>`\|\|OTHERWISE = (`*`case2process`*`).`<br>`MARKETPLACESWITCH = CASE1EVALC; END.` | a) where exp is of type WHILE_variable which is a declared read/write operator process. |
| **Pick** | `ATM_ONMESSAGE_DISCONNECT =`<br>`(disconnect->END).`<br>`DISCONNECT =`<br>`(connected.value.write[1] ->END).`<br>`ATM_ONMESSAGE_DIS_SEQ =`<br>`ATM_ONMESSAGE_DISCONNECT; DISCONNECT; END.`<br>`ATM_ONMESSAGE_LOGON = (logon->END).`<br>`LOGON = (loggedon.value.write[0] -> END).`<br>`ATM_ONMESSAGE_LOGON_SEQ`<br>`ATM_ONMESSAGE_LOGON; LOGON; END.`<br>`PICK = (ATM_ONMESSAGE_DIS_SEQ \|\|`<br>`ATM_ONMESSAGE_LOGON_SEQ).` | Each onMessage activity is translated to a process and composed as an alternative message path in the parallel composition process. |

## C.4 Fault Handling Activities to FSP

| BPEL4WS Construct | FSP Process Example | Comments |
|---|---|---|
| **Fault Handlers (inline)** | FAULTHAND1 = (fault.read[i:Compensate_IntRange]->FAULT1[i]), FAULTHAND1[i:Compensate_IntRange] = if (i==0) then DOFAULT1; END else if (i==1) then DOFAULT2; END. DOFAULT1 = (fault1raised->END). DOFAULT2 = (fault2raised->END). INVOKE1 =(invoke_seller_SyncPurchase->END). INVOKE2 =(invoke_shipper_OrderShipment->END). ACTSEQ = INVOKE1 ; INVOKE2 ; END. \|\|ACTIVITIES = (INVOKE1 \|\| INVOKE2). set ACTSET = {invoke_seller_SyncPurchase, invoke_shipper_OrderShipment} TERMS = (ACTSET -> TERMS \| fault1raised -> END \| fault2raised -> END). \|\|FAULTMON =(ACTSEQ \|\| TERMS \|\| FAULTHAND1). | Each faultHandler type of fault is modelled as an alternative process path in the composition.<br><br>As a fault is raised, all remaining activities in the scoped faultHandler are terminated. The process therefore is a choice of paths (e.g. TERMS). |
| **Compensation Handlers (inline)** | COMPENSATE = (compensate.read[i:TRUEFALSE_variable] ->COMPENSATE[i]), COMPENSATE[i: TRUEFALSE_variable] = if (i=='true') then COMPENSATE_INVOKE; END else END. COMPENSATE_INVOKE = (invoke_seller_CancelPurchase -> END). INVOKE = (invoke_seller_SyncPurchase -> END). INVOKE_SEQ = INVOKE; COMPENSATE; END. \|\|COMPENSATEEXAMPLE = (INVOKE_SEQ). | A compensation Handler (inline) is represented as an alternative process path in the composition of a single activity. If an evaluation at the end of the normal process execution results to TRUE, then the compensation process if followed. |
| BPEL4WS Construct | FSP Process Example | Comments |
| **(scoped)** | COMPENSATE = (compensate.read[i:TRUEFALSE_variable]->COMPENSATE[i]), COMPENSATE[i: TRUEFALSE_variable] = if (i=='true') then COMPENSATE_INVOKE; END else END. INVOKE3 = (invoke_seller_CancelPurchase->END). INVOKE4 = (invoke_supplier_CancelSupplier->END). COMPENSATE_INVOKE = INVOKE3; INVOKE4; END. INVOKE1 = (invoke_seller_CancelP->END). INVOKE2 = (invoke_supplier_CancelS->END). INVOKE_SEQ = INVOKE1; INVOKE2; COMPENSATE; END. \|\|COMPENSATESCOPE = (INVOKE_SEQ). | A compensation Handler (scoped) is represented as an alternative process path in the composition of a process of one or many activities. If an evaluation at the end of the normal process execution results to TRUE, then the compensation process if followed. |