Name: Chiyoko Kawano
Degree: Master of Science
Title of Project: Extension of an Executable Formal Model of BPEL to Permit Comprehensive Validation

Examining Committee:
Chair: Dr. Dirk Beyer
Assistant Professor, School of Computing Science

Dr. Uwe Glässer
Senior Supervisor
Associate Professor, School of Computing Science

Dr. Martin Ester
Supervisor
Associate Professor, School of Computing Science

Dr. Oliver Schulte
Internal Examiner
Associate Professor, School of Computing Science

Date Defended/Approved: Nov. 9, 2006
DECLARATION OF
PARTIAL COPYRIGHT LICENCE

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website <www.lib.sfu.ca> at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada
ABSTRACT

The Business Process Execution Language for Web Services (BPEL) is a specification language designed for automating business processes and transactions through Web services between distributed computers. This project presents an executable model for validation of version 1.1 of the BPEL specification by extending a formal model written in Abstract State Machine Language (AsmL). AsmL is a high-level executable formal specification language based on the theory of Abstract State Machines (ASMs). We integrate this model with a particular Abstract Communication Model (ACM) in order to simulate the behaviour of multiple BPEL communication endpoints representing a credit card application process executing on asynchronous communication architectures. These extensions allow further experimental validation with a more comprehensive model in the design phase of system development.

Keywords: Web Services; BPEL; Abstract State Machines; AsmL

Subject Terms: Software Engineering; Web Services – Design; Computer software – Specifications; Computer software – Testing; Computer software – Validation
DEDICATION

To my family
First of all, I would like to express my deepest gratitude to my senior supervisor, Dr. Uwe Glässer, for his patient supervision and enthusiasm, which has guided me to accomplish my work. Without his unfailing guidance, I could not have finished this project.

Secondly, I cannot thank enough Michael Letourneau, who has acted as a “supervisor” despite of his busy life as a president of the CSGSA and as a Ph. D. student. He has given me brilliant advice on my project with his profound insight and has kindly proofread this report, using his natural talent with the English language to give me many helpful suggestions.

I also would like to thank Sonu Sharma, Dr. Yi Sun, Chris Boutilier and Luis Solis for their great support and encouragement, and all the members of the Software Technology Lab for sparing their precious time on coming to both my practice and real defence and giving me useful feedback. I gratefully acknowledge Mona Vajihollahi with deep appreciation for providing me with the source code of the original BPEL base model and her excellent suggestions, and Charu Jain for adapting her project to be suitable for integration. I am also grateful to the faculty and staff of the School of Computing Science for providing a great environment for research, and to Dr. Charles Brown, Mrs. Desanka Polajnar, Dr. Jernej Polajnar, Dr. Jennifer Hyndman and Dr. David Casperson for inspiring me to study computer science.

Finally, I thank my parents for letting me pursue my studies and for their unconditional love.
TABLE OF CONTENTS

Approval ..............................................................................................................ii
Abstract ..............................................................................................................iii
Dedication ...........................................................................................................iv
Acknowledgements ..........................................................................................v
Table of Contents ............................................................................................vi
List of Figures ....................................................................................................viii
Glossary ................................................................................................................ix

Chapter 1  Introduction .................................................................................1
  1.1  Motivation ..............................................................................................4
  1.1.1  Comprehensive Model with Multiple Endpoints .......................4
  1.1.2  Executable Specification ...............................................................5
  1.1.3  Problems and Challenges ...............................................................7
  1.2  Objectives ..............................................................................................8

Chapter 2  Approaches .................................................................................9
  2.1  Abstract State Machines .......................................................................9
  2.1.1  Basic Abstract State Machine ......................................................10
  2.1.2  Distributed Abstract State Machine ...........................................15
  2.2  Abstract State Machine Language ...................................................19

Chapter 3  BPEL .........................................................................................21
  3.1  Example ...............................................................................................22
  3.2  Service Description ...............................................................................28
  3.2.1  Message .........................................................................................29
  3.2.2  Port Type .........................................................................................29
  3.2.3  Partner Link Type ...........................................................................29
  3.2.4  Property ..........................................................................................30
  3.2.5  Property Alias ...............................................................................30
  3.3  Process Definition ................................................................................31
  3.3.1  Partner Links ..................................................................................33
  3.3.2  Partners ..........................................................................................33
  3.3.3  Variables ........................................................................................34
  3.3.4  Correlation Sets .............................................................................34
  3.3.5  Compensation Handler .................................................................35
  3.3.6  Fault Handlers .................................................................................35
  3.3.7  Event Handlers ...............................................................................36
  3.3.8  Activity ...........................................................................................37
<table>
<thead>
<tr>
<th>Chapter 4</th>
<th>Implementation</th>
<th>46</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>BPEL Service Abstract Machine Model</td>
<td>47</td>
</tr>
<tr>
<td>4.1.1</td>
<td>The Original Model</td>
<td>47</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Transformation and Extension of the Model</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>Abstract Communication Model</td>
<td>62</td>
</tr>
<tr>
<td>4.3</td>
<td>Network Integration</td>
<td>63</td>
</tr>
<tr>
<td>4.4</td>
<td>Simulation</td>
<td>65</td>
</tr>
</tbody>
</table>

| Chapter 5 | Conclusions | 71 |

Appendix A. The Extended BPEL Service Abstract Machine Model Extract | 73 |

Appendix B. The Network Interface Model | 104 |

Appendix C. The Service Description for the Credit Card Evaluation Division | 111 |

Appendix D. The Process Definition for the Credit Card Evaluation Division | 114 |

Appendix E. The Service Description for the Credit Card Application Handling Division | 118 |

Appendix F. The Process Definition for the Credit Card Application Handling Division | 120 |

Appendix G. The Service Description for the Credit Reporting Bureau | 122 |

Appendix H. The Process Definition for the Credit Reporting Bureau | 124 |

Appendix I. The Service Description for the Credit Card Issuing Division | 126 |

Appendix J. The Process Definition for the Credit Card Issuing Division | 128 |

Bibliography | 130 |
LIST OF FIGURES

Figure 1: Originally the Network Abstract Model (NAM) and the BPEL Service Abstract Machine Model were not connected......................3
Figure 2: The integrated model ..................................................................................................................3
Figure 3: Software development process. The correctness of formal specifications against informal requirements cannot be verified. ........6
Figure 4: Software development process with the ASM method. The correctness of an ASM Model against informal requirements can be validated by observation....................................................7
Figure 5: State transitions of the all-possible runs of the example .................18
Figure 6: The workflow of the credit card evaluation division .........................23
Figure 7: The workflow of the credit card application handling division ..........24
Figure 8: The initial state of the BPEL Service Abstract State Machine ........48
Figure 9: A dummy process becomes a real process when a message is assigned and a new dummy process is created......................49
Figure 10: The data structure of the Message class and its subordinate classes ........................................................................................................61
Figure 11: Abstract Communication Model [14].................................................63
Figure 12: Network Interface Model ........................................................................64
Figure 13: Simulation .................................................................................................................................66
Figure 14: An instance of the extended BPEL Service Abstract Machine representing the credit card application division..............................67
Figure 15: The credit card application form ..........................................................68
Figure 16: Network Abstract Model [20]. Each of Pr2, Pr4, Pr6 and Pr8 is connected to a corresponding instance of the Network Interface Model. ..................................................................................69
Figure 17: The control panel .....................................................................................................................70
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACM</td>
<td>Abstract Communication Model</td>
</tr>
<tr>
<td>ASM</td>
<td>Abstract State Machine</td>
</tr>
<tr>
<td>AsmL</td>
<td>Abstract State Machine Language</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language for Web Services</td>
</tr>
<tr>
<td>DASM</td>
<td>Distributed Abstract State Machine</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
</tr>
<tr>
<td>NAM</td>
<td>Network Abstract Model</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>SDL</td>
<td>Specification and Description Language</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>UPnP</td>
<td>Universal Plug and Play</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC (Very High Speed Integrated Circuits) Hardware Description Language</td>
</tr>
<tr>
<td>WSFL</td>
<td>Web Services Flow Language</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XPath</td>
<td>XML Path Language</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
CHAPTER 1 INTRODUCTION

In this project, we simulate the behaviour of multiple communication endpoints for BPEL, the Business Process Execution Language for Web Services [1], on asynchronous communication architectures. This is done by extending the Service Abstract Machine Model of BPEL [32] and integrating the model with the Network Abstract Model (NAM) [20] based on the Abstract Communication Model (ACM) [13].

BPEL is a specification language designed for automating business processes and transactions through Web services between distributed computers. The Organization for the Advancement of Structured Information Standards (OASIS) [27] has formed the Technical Committee for BPEL, which includes BEA, IBM, Microsoft, SAP and Siebel among others. The committee has issued version 1.1 of the BPEL specification (referred to here as the BPEL specification). BPEL has been proposed in order to supersede the IBM Web Service Flow Language (WSFL) [21] and Microsoft XLANG [30] specification with a language having additional features and more flexibility.

BPEL enables computers to automatically process complex tasks currently done manually. For example, suppose that a travel agency is arranging a trip that requires booking flights, hotel rooms and a car rental for a customer. In order to make all the reservations for the trip, the agency must obtain confirmations from all the parties involved in each booking. If one reservation
fails, the effect could propagate through the entire process and the compensation process would be intricate. Unfortunately, compensations cannot be processed by the traditional Web service technologies such as the Web Service Definition Language (WSDL) [8] because the processes of technologies like WSDL are stateless. In contrast, since BPEL processes keep track of the state of the related message exchanges, BPEL preserves long-lasting sophisticated interactions between multiple partners such as in the above example [28].

The underlying communication network model used to connect BPEL communication endpoints for this project is referred to as the Abstract Communication Model (ACM). The ACM is a generic model of communication networks, which was originally designed as an abstract model of the Universal Plug and Play (UPnP) [15] architecture by the Foundations of Software Engineering Group of Microsoft [22]. In this project, we use the Network Abstract Model (NAM) [20] based on the ACM. Although the ACM was derived from TCP/IP networks for UPnP, the ACM is generic and not restricted to TCP/IP networks.

The BPEL Service Abstract Machine models a BPEL communication endpoint. It is an executable formal model of BPEL focused on endpoint services and does not include a communication network. The process of sending and receiving messages was replaced by user input and output. Therefore, integration of the BPEL Service Abstract Machine Model and the ACM is needed in order to experimentally validate asynchronous communication behaviours of high-level models of distributed architectures.
Figure 1 shows the original models where the BPEL Service Abstract Machine Model and the NAM are not connected and stand alone.

Figure 1: Originally the Network Abstract Model (NAM) and the BPEL Service Abstract Machine Model were not connected.

In Figure 2, each instance of the BPEL Service Abstract Machine Model is connected to the NAM via its enclosing instance of the Network Interface Model.

Figure 2: The integrated model
We have implemented the model in Abstract State Machine Language (AsmL) [23], a high-level executable formal specification language based on the theory of Abstract State Machines (ASMs) [17]. AsmL has been developed by the Foundations of Software Engineering Group of Microsoft Research [22], and is interoperable with Microsoft .NET. The fact that AsmL is executable helps to capture errors and inconsistencies in specifications at an early system development stage. It allows users and designers to observe the behaviours of a system during the design phase of development. This is cost- and time-effective when compared with finding errors or inconsistencies in the testing phase.

In Chapter 2, we discuss the ASM method in more detail and briefly introduce AsmL. Chapter 3 shows an example and the structure of BPEL. In Chapter 4, we present the implementation details of this project, an overview of the ACM, and describe the simulation we constructed. We present our conclusions in Chapter 5.

1.1 Motivation

1.1.1 Comprehensive Model with Multiple Endpoints

The Service Abstract Machine Model of BPEL [32] consists of only a single BPEL endpoint and does not contain any other endpoints or networks. Therefore, communication between BPEL endpoints cannot be tested. In order to simulate asynchronous message exchanges between autonomous agents and to perform experimental validation of network behaviours, we need to modify the original model of BPEL and combine it with the ACM so that it consists of multiple
endpoints and allows the endpoints to communicate with one another through the communication network.

1.1.2 Executable Specification

"How can we verify that a system works as intended?" This is one of the most essential and challenging issues in Systems Engineering. Formal methods allow us to specify systems unambiguously and to verify an implementation's conformance to the specification. However, no methods can formally verify that specifications precisely capture and satisfy the requirements for a system. This is because there is no way to formally translate intentions into formal specifications. In the first phase of software development, we typically elicit the requirements and document them in natural language. Then, we produce formal specifications based on these informal requirements specifications. However, since the transformation process is informal, no methods can formally prove the correctness of these formal specifications against the underlying informal specifications.

Figure 3 shows a typical software development process.
The ASM method proposes an approach to answer the question, "How can we verify that a system works as intended?" We model a system by ASMs, execute the model, and validate the behaviour of that model by observation. This is possible because ASMs allow us to model a system precisely step for step. Its correctness is founded on the mathematical proofs of the ASM theses [18], [17]. Because of their preciseness and executability, ASMs have been employed to specify and validate various kinds of computer systems for both hardware and software. C [16], Prolog [3], [4], VHDL [5], [6], and Java [29] have all been modelled with ASMs. The International Telecommunication Union also uses ASMs for a formal definition of SDL [11].

Figure 4 shows a software development process with the ASM method.
1.1.3 Problems and Challenges

One of the difficulties faced by this project is the ambiguousness in BPEL caused by obscure descriptions in the informal specification; the BPEL specification has 136 pages and is written in plain English. As [32] notes, the definition of one subject is not always stated in one place. One needs to understand the whole specification in order not to miss any details or constraints.

In addition to the complexity of the informal specification, understanding the Service Abstract Machine Model of BPEL was not trivial. To modify and extend the model, one needs to have a thorough understanding of both the implementation details of the model in the executable language of AsmL and the underlying ASM formalism and abstraction principles, which comprises 150 pages of thesis [32]. In addition, a tutorial [25] and a reference manual of AsmL [24] are fairly intuitive, but very long, and have several errors and ambiguities.
Another challenging aspect is that the project is based on various kinds of technologies, methods and tools. Knowledge of BPEL is indispensable, and one must be familiar with the underlying technologies such as WSDL, XML Schema [31] and XPath1.0 [9].

1.2 Objectives

The main objective of the project is to provide an enhanced executable model to validate the BPEL specification. We have extended a formal model of BPEL by combining it with the NAM and constructed a simulation to execute the integrated model. The model can be used as a specification to complement informal specifications. By performing experiments on the model and observing its behaviours, we can detect discrepancies and flaws in the specification in early stages, which is far less expensive than if they were found in later stages of development. In our approach, we apply the notion of Abstract State Machines (ASMs). Although the method is based on mathematical theories, the model provides readers an intuitive understanding with its pseudo-code like syntax.
CHAPTER 2 APPROACHES

2.1 Abstract State Machines

Traditional natural language specifications are intuitive yet ambiguous, often incomplete and sometimes just too long to be practical. Although natural language specifications can be complemented with structural and graphical tools such as UML, those tools are often informal. Therefore, ambiguity is still a major concern. To avoid these problems in our model, we use a mathematical modelling paradigm, the Abstract State Machine (ASM) method. Unlike traditional programming languages, the ASM method, its flexibility in choosing the level of abstraction, allows us to document computer systems at any stage of the development process. An ASM may be viewed as a kind of pseudo-code, though it differs from pseudo-code in several ways. First, an ASM is based on a formal theory [18], [17]. Its correctness is founded on the mathematical proofs of the ASM theses. Secondly, an ASM program can be executable. By using a tool such as AsmL, ASM specifications can be executed for testing to detect inconsistencies and errors at an early stage of the development cycle. Therefore, it is more practical than traditional formal methods [7].

The ASM method defines an abstract machine with its state transitions. Unlike other operational models such as Turing Machines, ASMs are more realistic than other methods since we can define high-level models with them as
opposed to the low-level operations in Turing Machines. ASMs can simulate systems in the step-for-step manner [18].

In this chapter, we first introduce basic ASMs [18]. Then, we explain Distributed ASMs (DASM), which are intended for distributed computations.

2.1.1 Basic Abstract State Machine

Our definition of a basic ASM comes from [18], [17]. An ASM is composed of a set of states, which can be deemed as first-order structures, and a finite number of transition rules, namely a program. The initial states transition according to the transition rules when executed [18].

2.1.1.1 Vocabulary

The vocabulary of an ASM is a finite set of function names, each of which has a fixed non-negative arity. Constants are static nullary function names. Variables are dynamic function names. The static nullary names true, false, and undef are always elements of any vocabulary as well as the equality sign (=), and the names of the basic Boolean operators of and, or, and not [18].

2.1.1.2 Terms

We define terms by induction. A nullary function name is a term. If $f$ is a function name of positive arity $j$ and if $t_1$, $t_2$, ..., $t_j$ are terms, then $f(t_1, t_2, ..., t_j)$ is a term. If the outermost function name is relational, then the term is Boolean [18].
2.1.1.3 States

A state $S$ of vocabulary $\Gamma$ is a nonempty set $X$ (the base set of $S$) together with interpretations of the function names in $\Gamma$ on $X$ [17]. An $r$-ary function name is interpreted as a function from $X'$ to $X$. We call this function a basic function of $S$. The interpretation of an $r$-ary relation name is a function from $X'$ to \{true; false\}, a basic relation of $S$ [17], [18].

2.1.1.4 ASM Rules

In this section, we explain ASM rules [17], [19], which are comparable to statements in traditional programming languages.

Update Rule

The update rule has the following form:

$$f(t_1, t_2, \ldots, t_n) := t_0$$

$f$ is a dynamic function name with arity $n$ and $t_0$, ..., $t_n$ are terms. When the rule is executed, the value of the function $f$ with the specified arguments is updated to $t_0$. A state transition takes place by simultaneously executing all the applicable update rules; i.e. unlike traditional programming languages, all update rules in a state are executed simultaneously at the state transition.

Conditional Rule

The conditional rule has the following form:

$\text{If condition}_1 \text{ then rule}_1$

\[\text{elseif condition}_2 \text{ then rule}_2\]

...
[elseif \(condition_1\) then \(rule_1\)]

[else \(rule_0\)]

endif

If the value of \(condition_1\) is true, \(rule_1\) is executed. If the value of \(condition_1\) is false and the value of \(condition_2\) is true, \(rule_2\) is executed. If the values of \(condition_1, \ldots, condition_{n-1}\) are false and the value of \(condition_n\) is true, \(rule_n\) is executed. Otherwise, \(rule_0\) is executed.

Do-in-parallel Rule

The do-in-parallel rule has the following form:

\[
doi\text{-parallel} \rule_1 \rule_2 \ldots \rule_n enddo
\]

All the rules inside do-in-parallel block are executed simultaneously.

This parallelism is useful when we want to clearly specify two statements whose order does not matter. For example, when we want to swap the values of two variables, we typically express this in traditional programming languages as follows:

\[
tmp := v_1
\]
\[
v_1 := v_2
\]
\[ v_2 := \text{tmp} \]

The original value of \( v_1 \) is assigned to \( v_2 \) and the original value of \( v_2 \) is assigned to \( v_1 \).

Here, we have the same results if we change the order as follows:

\[ \text{tmp} := v_2 \]
\[ v_2 := v_1 \]
\[ v_1 := \text{tmp} \]

In this example, it is obvious that the result is the same in both cases. However, it might not be so clear whether the order matters in more complex cases.

Since ASMs allow parallel executions, the same result can be obtained by the following construct:

\[
\text{do-in-parallel}
\begin{align*}
v_1 &:= v_2 \\
v_2 &:= v_1
\end{align*}
\text{enddo}
\]

Note that \textit{do-in-parallel} and \textit{enddo} keywords are often omitted in ASMs. Therefore, all the rules in a block are executed simultaneously by default even without the \textit{do-in-parallel} keywords. When a sequential execution order is required, we write the following:

\[
\text{if not Executed then}
\begin{align*}
\text{First}
\end{align*}
\]
Executed := true
c
else
  Next
cendif
c
In the above example, we initially set the value of Executed to false. Then, Next is executed only after First is executed for the first step [19].

Choose Rule
c
In order to abstract from details, nondeterminism is provided as an ASM rule. Nondeterminism is also useful for testing and simulations. The choose rule has the following form:

\[
\text{choose } v : S: P_v \\
\text{rule}_v \\
\text{ifnone} \\
\text{rule}_0 \\
\text{endchoose}
\]
c
where \( v \) is a variable, \( S \) is an arbitrary set, and \( P_v \) is a Boolean term. We nondeterministically choose a value \( v \) from the set \( S \) such that \( P_v \) is true, and execute \( \text{rule}_v \) with the chosen value of \( v \). \( \text{rule}_0 \) is executed if such \( v \) cannot be found.

Forall Rule
c
The forall rule has the similar form as the choose rule.

\[
\text{do-forall } v : S: P_v
\]
where \( v \) is a variable, \( S \) is an arbitrary set, and \( P_v \) is a Boolean term. For all \( v \) from \( S \) where \( P_v \) is true, execute \( \text{rule}_v \). \( \text{rule}_v \) is executed simultaneously for all \( v \) satisfying the condition. Like the do-in-parallel rule, this rule is powerful when the order of the executions is insignificant.

2.1.2 Distributed Abstract State Machine

Our definition of a Distributed Abstract State Machine (DASM) comes from [17]. A DASM consists of a dynamic, finite set of agents, each of which is a sequential ASM operating on a common global state. We define a single step of the program of an agent as a \textit{move}. A \textit{run} of a DASM is a partially ordered set of moves of multiple agents, whereas a run of a sequential ASM consists of sequentially ordered steps of the program of a single agent. Partially ordered moves are the moves that satisfy the following conditions:

1. Each move has a finite set of predecessors.
2. The moves of a single agent are linearly ordered.
3. \textit{The coherence condition}

The coherence condition guarantees the consistency of the result of program execution. We define a finite initial segment \( X \) as a subset of a partially ordered set of moves \( P \) such that if \( x \) is the maximal element of \( X \); i.e., \( x \) is the last element, \( X \) contains all the elements of the partially ordered set \( P \) that are less than \( x \). The coherence condition holds that if \( x \) is the maximal element of a
finite initial segment $X$ and $\sigma(X)$ is the state obtained by executing all the moves in $X$, $\sigma(X)$ can be reached from the state $\sigma(X - \{x\})$ by executing $x$ [2], [32].

2.1.2.1 Example

In this section, we explain Distributed ASMs in an intuitive manner. This section is inspired by [32], where the concept of a partially ordered run is well explained. We show an example of a modified version of the classical dinning philosophers' problem. In the original problem, there are several philosophers at a round dining table and a chopstick is located between each of the philosophers. A philosopher can eat only if he or she has taken both chopsticks that are located beside him or her. In order to simplify the problem, we have only two philosophers and we use two forks instead of chopsticks. Two philosophers, $p_1$ and $p_2$, are sitting at a dinning table and there are two forks, $f_1$ and $f_2$. A philosopher can pick up either of the forks if that fork has not been taken yet and the philosopher has not picked up a fork yet. A philosopher can eat if he or she picks up a fork. Therefore, in this problem, both philosophers could eat if each picks up a different fork. We define symbols as follows:

$f_i = \text{true}$: $f_i$ has been taken

$f_i = \text{false}$: $f_i$ has not been taken

$p_j = \text{hasFork}_i$: $p_j$ has $f_i$

$p_j = \text{undef}$: $p_j$ has not picked up a fork

There are four possible moves:
Move $m_1$: $p_1$ picks up $f_1$ if $p_1$ has not picked up a fork and $f_1$ has not been taken.

Move $m_2$: $p_1$ picks up $f_2$ if $p_1$ has not picked up a fork and $f_2$ has not been taken.

Move $m_3$: $p_2$ picks up $f_1$ if $p_2$ has not picked up a fork and $f_1$ has not been taken.

Move $m_4$: $p_2$ picks up $f_2$ if $p_2$ has not picked up a fork and $f_2$ has not been taken.

Our initial state is $p_1 = \text{undef}$, $p_2 = \text{undef}$, $f_1 = \text{false}$ and $f_2 = \text{false}$. The possible runs are shown as follows:

1. $\{m_1, m_4\}$ with $\text{Time}(m_1) < \text{Time}(m_4)$
2. $\{m_1, m_4\}$ with $\text{Time}(m_1) > \text{Time}(m_4)$
3. $\{m_1, m_4\}$ with $\text{Time}(m_1) = \text{Time}(m_4)$
4. $\{m_2, m_3\}$ with $\text{Time}(m_2) < \text{Time}(m_3)$
5. $\{m_2, m_3\}$ with $\text{Time}(m_2) > \text{Time}(m_3)$
6. $\{m_2, m_3\}$ with $\text{Time}(m_2) = \text{Time}(m_3)$

Figure 5 illustrates the state transitions.
$m_1$ and $m_2$ cannot be in the same run since a philosopher can pick up only one fork. Similarly, $m_3$ and $m_4$ cannot be in the same run. $m_1$ and $m_3$ cannot be in the same run since a fork cannot be picked up by more than one philosopher. $m_2$ and $m_4$ cannot be in the same run for the same reason.
All the runs that consist of \( m_1 \) and \( m_4 \) result in the same state, State 3, and the all runs that consist of \( m_2 \) and \( m_3 \) result in the same state, State 6. The coherence condition does not allow any runs of the same set to yield different states.

The following is a possible ASM program of the philosopher problem.

\[
\text{PhilosopherProgram} \equiv \\
\text{if } p_i = \text{undef } \text{then} \\
\quad \text{choose } j \text{ from } \{1, 2\} \text{ where } f_j = \text{false} \\
\quad p_i := \text{hasFork}_j \\
\text{endchoose} \\
\text{endif}
\]

where \( i \) defines a person with \( i = \{1, 2\} \) and \( j \) defines a fork with \( j = \{1, 2\} \).

### 2.2 Abstract State Machine Language

AsmL compiles and runs by itself, or can be integrated with Microsoft Visual Studio C#. Since AsmL specifications are executable, simulations can be easily performed by running the specification in order to observe the behaviour of a system or to validate if the behaviour is consistent to the intention. Although traditional programming languages might seem to play a similar role, AsmL is unique because of its conciseness; it can specify precisely what is required without any redundancies or omissions at the chosen level of abstraction. For example, in AsmL, all operations in the same step are executed simultaneously, whereas traditional programming languages execute operations sequentially.
Thus, even though the order of the executions may be irrelevant, they are executed in the order in which instructions are written in the program. It is not always the case that the order of execution matters. Other characteristics are that AsmL supports mathematical set operations, which allow the reader to capture the high-level view easily. See [24] and Section 2.1.1.4.
CHAPTER 3  BPEL

BPEL is a specification language devised for standardising and facilitating message exchanges through Web services. The comparative advantage of BPEL over other Web services such as SOAP [26], WSDL, and Universal Description Discovery and Integration (UDDI) [10] is that BPEL supports more complicated interactions between multiple parties [1]. Other Web services are stateless; that is, they support only synchronous interactions or uncorrelated asynchronous interactions. In contrast, BPEL processes are stateful; they keep track of the states of related asynchronous message exchanges, which in turn realise long-lasting sophisticated interactions between multiple partners.

To specify Web services, BPEL requires two documents for each endpoint: a service description and a process definition. A service description specifies message types and network endpoints. A process definition describes the behaviour of the process; that is, how the process interacts with its partners processes. BPEL documents are written in WSDL, which uses "an XML format for describing network services as a set of endpoints operating on messages [8]." WSDL allows us to abstractly describe endpoints and messages, which can then be mapped to concrete contents by bindings.

In this chapter, we first show an example and then describe the structure of the two BPEL documents. The definitions and the syntax forms of BPEL in this chapter come from [1], [8].
3.1 Example

In order to illustrate what BPEL is, we show an example of an on-line credit card application system, which we have constructed as a simulation. A credit card company, ABC card, has an on-line application system on its Web pages. Applicants submit financial information required for the process of the approval for a credit card when they apply. When the credit card application handling division receives an application, it will send the financial information to the credit card evaluation division and will wait for a reply. The credit card evaluation division inquires for the credit status of the applicant with a credit reporting bureau. The credit reporting bureau sends a report on the applicant to the credit card evaluation division. According to the report from the bureau, the credit card evaluation division decides whether it will approve the application or not.

If the credit card evaluation division receives a report of a good credit status, it will decide the maximum credit limit and send the credit card issuing division a request to issue a credit card. When the credit card issuing division receives the request, it will send a confirmation to the credit card evaluation division. When the credit card evaluation division receives the confirmation, it will send a message of approval to the credit card application handling division.

If the credit reporting bureau reports a bad credit status, the credit card evaluation division will send a message of rejection to the credit card application handling division.

Figure 6 shows the workflow of this credit card approval procedure.
Each of the credit card application handling division, the credit card evaluation division, the credit reporting bureau and the credit card issuing division is a BPEL endpoint. In BPEL, we define a process in terms of activities. The process of the credit card evaluation division consists of a sequence of \textit{receive}, \textit{invoke} and \textit{switch} activities where the \textit{switch} activity defines two conditional branches; good credit status and otherwise. The good credit status branch consists of a sequence of \textit{invoke}, \textit{assign} and \textit{reply} activities whereas
the otherwise branch consists of a sequence of assign and reply activities. The activities such as sequence and switch are called structured activities since they control the order of the nested activities in them. The activities such as receive, invoke, reply and assign are called basic activities. These activities cannot nest other activities in them. We define these structures of activities in a process definition.

Although the process definition for the credit card evaluation process is the most interesting one, it is too long to embed in the text. Instead, we explain the process of the credit card application handling division.

Figure 7 shows the workflow of the credit card application handling division.

![Diagram of the credit card application handling division workflow](image)

Figure 7: The workflow of the credit card application handling division

The process of the credit card application handling division is fairly simple. When the credit card application handling division receives an application for a credit card from a customer, it will request an evaluation from the credit card evaluation division and wait for a reply.
The service description of the credit card application handling division is below.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<definitions
    targetNamespace="http://credit.org/wsd1/card-application"
    xmlns="http://schemas.xmlsoap.org/wsd1/
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:pink="http://schemas.xmlsoap.org/ws/2003/05/partner-link/
    xmlns:cns="http://credit.org/wsd1/card-application">
  <message name="applicationInfoMessage">
    <part name="applicationNo" type="xsd:string"/>
    <part name="firstName" type="xsd:string"/>
    <part name="lastName" type="xsd:string"/>
    <part name="gender" type="xsd:string"/>
    <part name="birthDate" type="xsd:date"/>
    <part name="phoneNumber" type="xsd:string"/>
    <part name="address" type="xsd:string"/>
    <part name="income" type="xsd:int"/>
  </message>

  <message name="resultMessage">
    <part name="applicationNo" type="xsd:string"/>
    <part name="result" type="xsd:string"/>
    <part name="amountLimit" type="xsd:int"/>
  </message>

  <portType name="cardApplicationPT">
    <operation name="apply">
      <input message="cns:applicationInfoMessage"/>
    </operation>
  </portType>

  <portType name="cardServicePT">
    <operation name="request">
      <input message="cns:applicationInfoMessage"/>
      <output message="cns:resultMessage"/>
      <fault name="unableToHandleRequest"
            message="cns:errorMessage"/>
    </operation>
  </portType>

  <bpws:property name="applicationNo"
      type="xsd:string"/>

  <bpws:propertyAlias propertyName="cns:applicationNo"
             messageType="cns:applicationInfoMessage"
             part="applicationNo"/>
</definitions>
```
There are two message types defined in the service description, applicationInfoMessage and resultMessage. applicationInfoMessage contains an application number and the applicant's personal information. resultMessage holds an application number, the result of the credit card evaluation and the amount limit for the credit card.

The property definition and the property alias definitions of applicationNo are necessary in order to use applicationNo as a correlation set. Since multiple process instances can be running in BPEL, correlation sets are used as a kind of ID token to deliver a message to the right process instance. The property alias definition specifies the location of the property in a message. In this example, the property (applicationNo) is mapped to the part (applicationNo) in applicationInfoMessage and to the part (applicationNo) in resultMessage.

The process definition of the credit card application handling division is shown below.
<?xml version="1.0" encoding="utf-8"?>

<process name="creditCardApplication"

targetNamespace="http://acme.com/creditCardApplication"

xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/

xmlns:cn="http://credit.org/wsdl/card-application">

<partnerLinks>
  <partnerLink name="cardApplication"
    partnerLinkType="cn:cardApplicationLinkType"
    myRole="applicationService"/>
  <partnerLink name="cardService"
    partnerLinkType="cn:cardServiceLinkType"
    partnerRole="cardService"/>
</partnerLinks>

<variables>
  <variable name="apply"
    messageType="cn:applicationInfoMessage"/>
  <variable name="result"
    messageType="cn:resultMessage"/>
  <variable name="error"
    messageType="cn:errorMessage"/>
</variables>

<correlationSets>
  <correlationSet name="applicationId"
    properties="cn:applicationNo"/>
</correlationSets>

<sequence>
  <receive partnerLink="cardApplication"
    portType="cn:cardApplicationPT"
    operation="apply"
    variable="apply" createInstance="yes">
    <correlations>
      <correlation set="applicationId"
        initiate="yes"/>
    </correlations>
  </receive>
  
  <invoke partnerLink="cardService"
    portType="cn:cardServicePT"
    operation="request"
    inputVariable="apply"
    outputVariable="result" createInstance="no">
    <correlations>
      <correlation set="applicationId"
        initiate="no" pattern="in-out"/>
    </correlations>
  </invoke>
</sequence>
</process>
The correlation set definition of applicationId specifies the property applicationNo as a correlation set. Since the location of this property is defined in the property alias definition, the location of this correlation set can be found by matching the property name in the property alias definition.

In the receive activity, the initiate attribute is set to "yes". This is necessary to keep track of message exchanges. By setting the initiate attribute to "yes", when an instance of the process receives a message, it initiates the values of the correlation sets and it uses the same values until it finishes executing.

The service descriptions and the process definitions of all BPEL endpoints are provided in Appendix C-J.

3.2 Service Description

A service description defines messages, port types, properties, property aliases and partner link types. The definitions of these elements should be defined in the block of a service description. The syntax for a service description is as follows:

```xml
<definitions name="ncname" targetNamespace="uri"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    ....>

<!-- The definitions of the elements should be here --> ...

</definitions>
```
3.2.1 Message

A message consists of a name and parts. For each part, a name and a type must be specified. The message definition uses the following format:

\[
\text{<message name="ncname"}>
\text{<part name="ncname" type="qname" >+}
\text{</message>}
\]

3.2.2 Port Type

A port type is a set of abstract operations. This abstraction allows us to reuse the abstract definitions and to dynamically bind port types and addresses. Each operation refers to an input message and/or an output message and optional fault elements for any error messages [8]. The syntax for a port type is as follows:

\[
\text{<portType name="ncname"}>
\text{<operation name="ncname">+}
\text{<input message="qname"}>
\text{<output message="qname"}>
\text{<fault name="ncname" message="qname"}>*
\text{</operation>}
\text{</portType>}
\]

3.2.3 Partner Link Type

A partner link type defines a type of relationship between two services. A partner link type definition contains up to two roles played by services. Each role
must be associated with a port type. If only one role is specified in a partner link type definition, it means that the service that plays the role can link with any other service.

```xml
<plnk:partnerLinkType name="ncname">
  <plnk:role name="ncname">
    <plnk:portType name="qname"/>
  </plnk:role>
  <plnk:role name="ncname">
    <plnk:portType name="qname"/>
  </plnk:role>
</plnk:partnerLinkType>
```

3.2.4 Property

In BPEL, properties can be defined to create globally unique names and to associate XML Schema simple types with them. Properties have greater significance than types. A property is usually used to create a new name, often for a specific purpose. For example, we could define properties such as invoice number and phone number rather than just having a type, integer. A property definition uses the following format.

```xml
<bpws:property name="ncname" type="qname"/>
```

3.2.5 Property Alias

A property alias defines a property to be an alias for a location in a message. The syntax for the property alias definition is:
3.3 Process Definition

A process definition describes the behaviours of an endpoint. It can be viewed as a program that defines partner links, partners, variables, correlation sets, fault handlers, a compensation handler, event handlers and activities. The definitions of these elements should be given in the block of a process definition.

In this section, we use the symbol activity. This symbol means any of the activities except the terminate activity. The terminate activity can be used only in executable processes.

The syntax for a process definition is as follows:

```xml
<process name="ncname" targetNamespace="uri"
    queryLanguage="anyURI"?
    expressionLanguage="anyURI"?
    suppressJoinFailure="yes|no"?
    enableInstanceCompensation="yes|no"?
    abstractProcess="yes|no"
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
    <!-- The definitions of elements should be here --> ...
</process>
```
*queryLanguage* specifies the XML query language used to determine a single value within a message part. The default language is XPath 1.0.

*expressionLanguage* specifies the expression language to evaluate expressions. The default language is XPath 1.0. Version 1.1 of BPEL must be XPath 1.0 compliant.

*suppressJoinFailure* specifies whether the join failure fault should be suppressed for all activities. This attribute is used when synchronising activities nested in a *flow* activity. Since we do not implement synchronisation in this project, we do not discuss the details of this flag. See [1] for more details.

If *enableInstanceCompensation* is set to “yes” for a whole process, platform-specific compensation for the process instance is possible [1]. The default value is “no”.

*abstractProcess* specifies whether the process is an abstract process or an executable process. The default value is “no”. Executable processes are for specifying behaviours of a participant, whereas abstract processes define public message exchange protocols between multiple parties. Abstract processes should be used when we wish to restrict external parties from observing the internal behaviour. Therefore, abstract processes should be used for externally visible protocols, whereas the internal logic should be described by executable processes.
3.3.1 Partner Links

Partner links define all services that the business process interacts with. Each partner link specifies both a partner link type that is defined in the service description, and the roles of that partner link type. The attribute \textit{myRole} defines the role of the business process and the attribute \textit{partnerRole} defines the role of the partner. If the partner link type has only one role, the other role can be omitted. More than one partner link can be associated with the same partner link type. The syntax for partner links is as follows:

\[
\text{<partnerLinks>}
\]
\[
\text{<partnerLink name="ncname" partnerLinkType="qname" myRole="ncname"? partnerRole="ncname"?>}
\]
\[
\text{</partnerLink>}
\]
\[
\text{</partnerLinks>}
\]

3.3.2 Partners

A partner specifies a subset of partner links. All the partner links specified in the same block of a partner must be provided by the same partner. Therefore, a partner link can appear under only one partner. Partner definitions are optional. The syntax for partner definitions is as follows:

\[
\text{<partners>}
\]
\[
\text{<partner name="ncname">}
\]
\[
\text{<partnerLink name="ncname"/>}
\]
\[
\text{</partner>}
\]
\[
\text{</partners>}
\]
3.3.3 Variables

Variables define the data variables used by the process. The type of a variable must be exactly one of a WSDL message type, an XML Schema simple type or an XML Schema element. Variables are used for holding both data and the state of the process. The syntax of the variables definition is:

```xml
<variables>
  <variable name="ncname" messageType="qname" type="qname" element='qname'/>
</variables>
```

`messageType`, `type` and `element` are mutually exclusive attributes. `messageType` must be chosen for containing message data.

3.3.4 Correlation Sets

BPEL uses correlation sets to group related operations. Correlation sets are a kind of identification token used to determine which process instance to route messages to. Since there could be multiple instances of a process, correlation sets are necessary to successfully deliver messages to the correct process instance. The syntax of correlation sets is as follows:

```xml
<correlationSets>
  <correlationSet name="ncname" properties="qname-list"/>
</correlationSets>
```
The property names listed at *properties* indirectly determine the locations of the correlation sets by referring the corresponding property alias definitions in the service description; see Section 3.2.5.

3.3.5 Compensation Handler

When a task, or a group of activities, needs to be cancelled, compensation handlers provide a mechanism to undo the effects of the previous activities that have been performed as part of the task. The syntax of the compensation handler is:

```xml
<compensationHandler>?
    activity
</compensationHandler>
```

3.3.6 Fault Handlers

Unsuccessful work can be reversed by specifying an activity in the `catch` clause of the fault handler of the scope where a fault occurs. The optional `catchall` clause catches all faults that have not been caught by the `catch` clause.

```xml
<faultHandlers>?
    <catch faultName="qname"? faultVariable="ncname"?><* 
        activity
    </catch>
    <catchAll>?
        activity
    </catchAll>
</faultHandlers>
```
3.3.7 Event Handlers

Event handlers control activities that take place only when the specified event happens. There are two types of events in BPEL, the `onMessage` event and the `onAlarm` event. The `onMessage` event occurs upon receipt of a message and is triggered every time a specified message arrives. The `onAlarm` event can be set in two ways. If the `for` attribute is chosen, it waits for the specified period of time and then the specified activity is executed. If `until` attribute is defined, the activity is performed when the specified time has come. There must be at least one `onMessage` or `onAlarm` handler. The syntax for the event handler is as follows:

```xml
<eventHandlers>
  <onMessage partnerLink="ncname" portType="qname"
    operation="ncname" variable="ncname"/>
  <correlations>
    <correlation set="ncname" initiate="yes/no">+
    <correlation set="ncname" initiate="yes/no">+
    <correlations>
  activity
</onMessage>

<onAlarm for="duration-expr" until="deadline-expr">*
  activity
</onAlarm>
</eventHandlers>
```
deadline-expr must be either dateTime, date or a constant literal, and
duration-expr must be a duration.

3.3.8 Activity

A process executes the activities specified in the process document.

There are two types of activities in BPEL, basic activities and structured activities. Basic activities are essential operations of Web services. Structured activities control the execution order of the nested activities in them. Both basic activities and structured activities can be nested in a structured activity. This nesting structure allows a process to realise complex tasks.

3.3.8.1 Standard Attributes and Standard Elements

Each activity has optional standard-attributes and optional nested standard-elements. standard-attributes and standard-elements specify synchronisation relationships between activities that are nested under a flow activity. Although they appear in the syntax forms, we do not discuss the details in this report since our model does not have the feature to impose synchronisation; we can simply ignore them. See [1] for the details.

3.3.8.2 Basic Activities

Basic activities are receive, reply, invoke, wait, terminate, empty, assign and throw. Basic activities cannot nest activities.
3.3.8.2.1 Receive

The *receive* activity waits for a message from the specified partner. If the *createInstance* property is set to "yes", it creates a new process upon message arrival. It also specifies the names of the correlation sets it uses (if any). If the *initiate* attribute is set to "yes", it initiates the value of the correlation set and all the messages in the subsequent operations with the correlation set must carry the same value. The syntax for the *receive* activity is:

```xml
<receive partnerLink="ncname" portType="qname" operation="ncname"
    variable="ncname"? createInstance="yes|no"?
    standard-attributes>
standard-elements
<correlations>?
    <correlation set="ncname" initiate="yes|no"?>+
</correlations>
</receive>
```

3.3.8.2.2 Reply

The *reply* activity replies to the partner from which a message has been received. Therefore, there should be a *receive* activity before a *reply* activity. The syntax for the *reply* activity is:

```xml
<reply partnerLink="ncname" portType="qname" operation="ncname"
    variable="ncname"? faultName="qname"?
    standard-attributes>
standard-elements
```
3.3.8.2.3 Invoke

The **invoke** activity sends a message to a partner and waits for either a synchronous reply or asynchronous reply from the partner.

```xml
<invoke partnerLink="ncname" portType="qname" operation="ncname"
   inputVariable="ncname"? outputVariable="ncname"?
   standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="ncname" initiate="yes|no"?
         pattern="in|out|out-in"/>+
   </correlations>
   <catch faultName="qname" faultVariable="ncname"?>*
      activity
   </catch>
   <catchAll>?
      activity
   </catchAll>
   <compensationHandler>?
      activity
   </compensationHandler>
```
3.3.8.2.4 Wait

The wait activity delays a process for a certain period of time or until a deadline.

<wait (for="duration-expr" | until="deadline-expr") standard-attributes>

standard-elements
</wait>

3.3.8.2.5 Terminate

The terminate activity finishes all the running activities and the process.

<terminate standard-attributes>

standard-elements
</terminate>

The terminate activity can be used only in executable processes.

3.3.8.2.6 Empty

The empty activity does nothing. A typical use of this activity would be an operation for an error handling when we only need to catch an error but do not need to do anything.

<empty standard-attributes>

standard-elements
</empty>
3.3.8.2.7 Assign

The **assign** activity is for copying data from one variable to another, constructing and inserting new data using expressions, and copying endpoint references, which enable users to dynamically choose a service provider.

```xml
<assign standard-attributes>
    standard-elements
    <copy>+
        from-spec
        to-spec
    </copy>
</assign>
```

The **from-spec** must be one of the following forms:

```xml
<from variable="ncname" part="ncname"/>
<from variable="ncname" part="ncname"? query="queryString"/>
<from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
<from variable="ncname" property="qname"/>
<from expression="general-expr"/>
<from> ... literal value ... </from>
<from opaque="yes"/>
```

The **to-spec** must be one of the following forms:

```xml
<to variable="ncname" part="ncname"/>
<to variable="ncname" part="ncname"? query="queryString"/>
<to partnerLink="ncname"/>
```
3.3.8.2.8 Throw

The *throw* activity throws an error. A globally unique name has to be provided for each fault with an optional variable to hold further information.

```xml
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
  standard-elements
</throw>
```

3.3.8.3 Structured Activities

Structured activities define the order in which the nested activities in them occur. Structured activities can be categorised into sequential control (*sequence*, *switch*, and *while*), concurrency control (*flow*), and nondeterministic choice (*pick*).

3.3.8.3.1 Sequence

The *sequence* activity specifies the sequential order of execution of the activities contained in it. The activities take place sequentially in the order in which they are presented in the *sequence* activity.

```xml
<sequence standard-attributes>
  standard-elements
  activity+
</sequence>
```
3.3.8.3.2 Switch

The `switch` activity is composed of one or more case branches and an optional otherwise branch; the otherwise branch, if any, should be listed last. The first activity in the case branches whose condition returns true is executed. The conditions are tested in the order presented in the nesting `switch` activity. If none of the cases matches, the otherwise branch, if present, is executed.

```xml
<switch standard-attributes>
  standard-elements
  <case condition="bool-expr">+
    activity
  </case>
  <otherwise>
    activity
  </otherwise>
</switch>
```

`bool-expr` is a Boolean expression.

3.3.8.3.3 While

The `while` activity defines a condition and an iterative activity. The iterative activity is repeatedly executed while the condition holds true.

```xml
<while condition="bool-expr" standard-attributes>
  standard-elements
  activity
</while>
```
bool-expr is a Boolean expression

3.3.8.3.4 Pick

The pick activity contains at least one onMessage event and an optional set of onAlarm events. When one of the onMessage or onAlarm events occurs, the pick activity executes the activity associated with the first event that occurred. If two or more events occur simultaneously, only one event, selected at random, is executed. Once an event occurs, it waits for no more events. An onMessage event is triggered on arrival of a message. An onAlarm event is triggered either when the specified duration has passed (if for attribute is specified) or when the specified time has come (if until attribute is specified).

<pick createInstance="yes/no"? standard-attributes>
  standard-elements
  <onMessage partnerLink="ncname" portType="qname"
    operation="ncname" variable="ncname"?>+
    <correlations>?
      <correlation set="ncname" initiate="yes/no"?>+
        </correlations>
        activity
        </onMessage>
      <onAlarm (for="duration-expr" / until="deadline-expr")>*
      activity
      </onAlarm>
    </pick>
3.3.8.4 Flow

The flow activity supports concurrency for the activities nested in it. Links are used to control the synchronisation dependencies of the activities inside a flow activity. Links associate a source activity and a target activity. Since we do not address synchronisation, we will not discuss the details of the link semantics; see [1] for more details. Each activity specifies itself as a source activity or a target activity. If there are no links, all activities inside a flow activity are executed concurrently. If an activity is a target activity, it does not start until all its source activities terminate.

<flow standard-attributes>
    standard-elements
    <links>?
        <link name="ncname">+
        </link>
    </links>
    activity+
</flow>

3.3.8.5 Scope and Compensate

BPEL provides a mechanism to define a scope and compensate on a scope. Since we do not address these activities in this project, we omit the details. See [1], [12] for more details.
CHAPTER 4 IMPLEMENTATION

The BPEL architecture can be decomposed into two parts. One is endpoint services and the other is the communication network. In this project, we have extended the BPEL Service Abstract Machine Model [32], which focuses on endpoint services, and combined it with the NAM [20] based on the Abstract Communication Model (ACM), which models a communication network. To integrate the two, we have developed a Network Interface Model enclosing a BPEL Service Abstract Machine Model as a class in a Dynamic Link Library (DLL). We have also changed the BPEL Service Abstract Machine Model into a class in a DLL so that multiple instances of the model can be created and they can communicate to another through the NAM via an instance of the Network Interface Model.

As a result, this project consists of three DLL components for Microsoft .NET; the BPEL Service Abstract Machine, the Network Interface Model and the NAM, and a simulation to perform a credit card application process that uses the three DLL components. All components have been implemented in AsmL and in Microsoft Visual C#.

We have also made other modifications to the original BPEL Service Abstract Machine Model by enhancing the message data structure, evaluating expressions for the switch activity, adding a BPEL documents loader, and implementing more activities.
4.1 BPEL Service Abstract Machine Model

In this section, we describe the internal structure and the functions of the original BPEL Service Abstract Machine Model. We explain what features have been added to the original model and how the modified model differs from the original model. For more details of the original model, see [32].

4.1.1 The Original Model

The BPEL Service Abstract Machine Model, which represents a BPEL endpoint, consists of a set of processes, an inbox manager and an outbox manager. Each of these is an agent of a DASM. Each process executes the activities specified in the process document. (In the original model, functions to read the process document were not implemented; instead, the program was hard-coded). All incoming messages arrive in the inbox. The inbox manager delivers messages from the inbox to processes. Since the original model was not connected to a network, all incoming messages were created by user inputs. The outbox manager is responsible for transferring messages in the outbox to the network. This part was not implemented since the original model was not connected to a network.

Initially, a BPEL endpoint creates a dummy process, which waits for the first event or the first message specified in the process document. The inbox manager periodically checks the inbox to see if there are any messages. If the inbox manager finds messages, it assigns each message in the inbox to the right process waiting for the message.
Figure 8 shows the initial state of the BPEL Service Abstract Machine, which contains DASM agents of a set of processes, an inbox manager and an outbox manager.

Figure 8: The initial state of the BPEL Service Abstract State Machine

If the process that has received a message is the dummy process, it becomes a real process and a new dummy process is created for a new instance. The real process starts executing the activities. For each structured activity, an activity agent is created to execute the activity. Figure 9 shows that a dummy process becomes a real process when the inbox manager assigns a message to it.
In [32], the receive, reply, invoke, wait, terminate, empty, sequence, switch (except for evaluation of the switch/case conditions), while, flow and pick activities were formally defined and of these activities, receive, reply, sequence and flow were implemented.

4.1.2 Transformation and Extension of the Model

In order to see that the integrated executable model actually works, we need to be able to create multiple endpoints communicating to one another through the network. For this reason, and the restriction of Microsoft .NET that only one AsmL document can exist in a project (otherwise, we would need to integrate all four models; the BPEL Service Abstract Machine, the Network Interface Model, the Network Abstract Model and the simulation, in a single Asml document), we have transformed the BPEL Service Abstract Machine Model into a class in a Dynamic Link Library (DLL) so that multiple instances of the BPEL
Service Abstract Machine can be created. For each endpoint, an instance of the Network Interface Model must be created to be connected to the NAM.

4.1.2.1 Implementation of Activities

The *pick*, *invoke*, *wait*, *terminate*, *empty*, *switch* and *while* activities have been implemented as specified in [32]. As for the *switch* activity, the evaluation of the switch/case condition was not specified in [32]. We implemented the switch/case condition, which we explain in the next section.

In addition to implementing these activities, we have also added the *assign* activity so that message data can be created or modified. The *assign* activity assigns and copies data from *from-spec* to *to-spec*; see Section 3.3.8.2.7. Since there are seven forms for *from-spec* and four forms for *to-spec*, we must consider 28 cases. Although we do not support the assignment of partner links or expressions, there still remains 18 cases. For each case, type compatibility must be tested before assigning.

In this section, we focus on the implementation of the *pick* activity. See Appendix A for the implementation of the *invoke*, *wait*, *terminate*, *empty*, *switch* and *while* activities.

The *pick* activity waits for one of the *onMessage* events or the *onAlarm* events to occur and executes the associated activity. Once an event occurs, the *pick* activity does not wait for other events; i.e. only the activity associated with the first event is executed. For each structured activity, an activity agent is created to execute the activity. The *pick* agent is the activity agent that executes
the pick activity. Since the pick activity waits for two kinds of events, the pick agent uses two other agents, the pick message agent and the pick alarm agent.

The PICK_AGENT class is shown below:

```java
public class PICK_AGENT extends ACTIVITYAGENT {
    var triggeredEvents as Set of (BPEL_EVENT, Time) = {}
    var chosenActivity as ACTIVITY? = undef

    override Program()
    if suspended = false then
        if chosenActivity = undef then
            step
            var pAlarm as PICK_ALARM_AGENT = new PICK_ALARM_AGENT(id + ":pa" + idCounter, baseActivity)
            var pMessage as PICK_MESSAGE_AGENT = new PICK_MESSAGE_AGENT(id + ":pm" + idCounter, baseActivity)
            step
            addNewActivityAgent(pAlarm)
            add pAlarm to me.rootProcess.model.agents
            addNewActivityAgent(pMessage)
            add pMessage to me.rootProcess.model.agents
            suspended := true
        else
            parentAgent.suspended := false
            finished(me)
        end
        if suspended = true then
            if chosenActivity = undef then
                choose (evt, time) in triggeredEvents where forall (e, t) in triggeredEvents holds time <= t
                chosenActivity := evt.oneEventActivity
                WriteLine("Activity chosen")
            else
                WriteLine("chosen activity set: "+ chosenActivity.ToString())
                me.rootProcess.model.execute_Activity(me, chosenActivity)
        end
    else
        override addTriggeredEvents(evt as BPEL_EVENT, time as Time or Null)
        add (evt, time) to triggeredEvents
    end
}
```

suspended is a flag that indicates if the agent is busy. The agent is busy while it executes one of its activities. chosenActivity holds the activity chosen to execute.

If the pick agent is not busy and there is no activity chosen to execute, it creates a pick alarm agent and a pick message agent and adds these agents
to the agent set of the model. In this way, the model can keep track of the active agents and execute the programs of all the agents in the agent set when its program is executed. *addNewActivityAgent* constructs a parent-child relationship with the specified agent and also adds a new agent to the *dependentAgentSet* of the root process. In this case, the *pick* agent is the parent agent of the *pick alarm* agent and the *pick message* agent. It also sets *suspended* to *true*; i.e. the *pick* agent becomes busy.

If the *pick* agent is not busy and *chosenActivity* is set, it means that *chosenActivity* has finished its execution. The *pick* agent removes itself from the active agent set of the model and dissolves the parent-child relationship with its dependent agents.

If the *pick* agent is busy and *chosenActivity* is not set, it chooses the first event that has occurred. *triggeredEvents* is a set of tuples where the tuples consist of an event and the time when the event occurred. The *pick alarm* agent and the *pick message* agent add such a tuple when a corresponding event occurs.

If the *pick* agent is busy and *chosenActivity* is set, it executes the chosen activity.

The *PICK_ALARM_AGENT* class is shown below:

```java
public class PICK_ALARM_AGENT extends ACTIVITYAGENT

    override Program()
        if suspended = false then
            startTime := DateTime.Now.Ticks
            suspended := true
        else

```
forall evt in baseActivity.getOnAlarmSet() where 
evt.triggerTime(startTime) <= DateTime.Now.Ticks 
    parentAgent.addTriggeredEvents(evt, evt.triggerTime(startTime)) 
    finished(me)

The **pick alarm** agent must record the start time in order to calculate the 
time when an alarm event occurs. If the **pick alarm** agent is not busy, it sets 
**startTime** to the current time and **suspended** to **true**.

If it is busy, it selects all the events that have been triggered and their 
triggered time, adds them to **triggeredEvents**, and finishes.

The **ON_ALARM_EVENT** class, which defines **triggerTime function**, is 
shown below:

```java
public class ON_ALARM_EVENT extends BPEL_EVENT
    var typeFor as Boolean // for or until
    var timeValue as Time

    triggerTime(startTime as Time or Null) as Time
    match startTime
        sTime as Null: 
            return timeValue
        sTime as Time:
            if typeFor then 
                return sTime + timeValue
            else
                return timeValue
```

**typeFor** is a flag to determine whether **timeValue** is a period of time or a 
deadline. If the alarm is set by specifying a deadline with "until", the **typeFor** flag 
is set to **false**. If the alarm is set by specifying a period of time with "for", the 
**typeFor** flag is **true**.
**triggerTime** calculates the time when the alarm is to be triggered by adding the start time to the time value if a period of time is specified. If a deadline is used, it returns the time value.

The **PICK_MESSAGE_AGENT** class is shown below:

```java
class PICK_MESSAGE_AGENT extends ACTIVITY_AGENT

override Program()
if suspended = false then
    for all evt in baseActivity.getOnMessageSet()
        add (me, evt) to rootProcess.waitingForMessage
        writeLine("Pick Message Agent: added waitingForMessage")
        suspended := true
else
    choose (agent, op, time) in rootProcess.completedInOperations
        where op in baseActivity.getOnMessageSet() and agent = me
        match op
            evt as ON_MESSAGEEVENT:
                writeLine("event: " + evt.ToString() + " time: " + DateTime(time))
                parentAgent.addTriggeredEvents(evt, time)
                finished(me)
```

If the **pick message** agent is not busy, it adds all events to the **waitingForMessage** set of the root process and sets **suspended** to **true**. By adding events to the **waitingForMessage** set, the inbox manager can look for a matching activity or event in this set when a message arrives.

If the **pick message** agent is busy, it looks for an **onMessage** event whose running agent is itself from the **completedInOperations** set of the root process. **completedInOperations** contains a set of tuples where the tuples consist of a running agent, an input operation that has completed, and the time when the message arrived. An input operation is an **onMessage** event, or a **receive** or **invoke** activity.

54
When a first event occurs and its associated activity is chosen, no other event should be accepted. Since the *pick message* agent adds all events to the *waitingForMessage* set of the root process when started, all these events except the accepted event must be discarded. This is done by *PickActivityClearance*.

```java
public class INBOX_MANAGER
    PickActivityClearance(p as PROCESS, ra as RUNNING_AGENT, inActivity as INPUT_OUTPUT_ACTIVITY or ON_MESSAGE_EVENT)
    match ra
        pma as PICK_MESSAGE_AGENT:
            forall (a, o) in p.waitingForMessage where a = ra and o <> inActivity
            remove (a, o) from p.waitingForMessage
            WriteLine("removing PickActivity: " + a.ToString() + o.ToString())
```

### 4.1.2.2 Evaluation of Expressions

Although the default expression language for BPEL is XPath 1.0 [9], you can specify the expression language in the process definition. In this project, we have implemented a simplified version of expression evaluation based on the definition of expressions in XPath 1.0 [9] for the purpose of testing the *switch* activity.

In order to evaluate an expression, a lexical analyser first analyses the expression and partitions it into a sequence of units by classifying each unit as Boolean, parenthesis, number, operator, literal, comma or name. After that, the sequence of units is parsed to observe if it is a valid expression. To read switch/case conditions, we define a lexical analyser called *readCondition*. The implementation can be found in Appendix A.

#### 4.1.2.2.1 Expression

In this project, we define Expression as follows:
Expr ::= OrExpr

OrExpr ::= AndExpr

    | OrExpr 'or' AndExpr

AndExpr ::= EqualityExpr

    | AndExpr 'and' EqualityExpr

EqualityExpr ::= RelationalExpr

    | EqualityExpr '=' RelationalExpr
    | EqualityExpr '!=' RelationalExpr

RelationalExpr ::= AdditiveExpr

    | RelationalExpr '<' AdditiveExpr
    | RelationalExpr '>' AdditiveExpr
    | RelationalExpr '<=' AdditiveExpr
    | RelationalExpr '>= AdditiveExpr

AdditiveExpr ::= MultiplicativeExpr

    | AdditiveExpr '+' MultiplicativeExpr
    | AdditiveExpr '-' MultiplicativeExpr

MultiplicativeExpr ::= UnaryExpr

    | MultiplicativeExpr '*' UnaryExpr
    | MultiplicativeExpr 'div' UnaryExpr
    | MultiplicativeExpr 'mod' UnaryExpr

UnaryExpr ::= PrimaryExpr

    | '-' UnaryExpr
PrimaryExpr ::= Boolean
  | Literal
  | Number
  | FunctionCall
  | '('Expr')'

In this implementation, we assume that there are no parentheses used in the arguments of a function call and there is only one valid function, bpws:getVariableData. bpws:getVariableData extracts a value from a variable.

We present the implementations of isExpr and isOrExpr below. The implementations of other expressions are similar to isOrExpr.

```plaintext
isExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
  if not hasError and isOrExpr(p, expressions, data, hasError) then
    return true
  else
    writeLine(expressions + " is not an expression")
    return false
```

```plaintext
isOrExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
  primitive var d1 as DATA = null
  primitive var d2 as DATA = null
  primitive var expressions1 as Map of Integer to DATA = {->}
  primitive var expressions2 as Map of Integer to DATA = {->}
  var boolval as Boolean
  if not hasError and splitIfIncludesReverse("or", expressions, expressions1, expressions2, hasError) then
    if not isOrExpr(p, expressions1, d1, hasError) then
      writeLine("isOrExpr: isOrExpr(expression1=" + expressions1 + ") is false")
      return false
    elseif isAndExpr(p, expressions2, d2, hasError) then
      if (d1.dataType = "boolean" or d1.dataType = "int") and (d2.dataType = "boolean" or d2.dataType = "int") then
```

57
step
step
  data := new DATA(boolVal.ToString())
step
  datadataType := "boolean"
  data.name := "boolean"
step
  WriteLine("isOrExpr: expression is or expression!=" + expressions)
  WriteLine("isOrExpr: d1=" + d1 + ", d2=" + d2)
  WriteLine("isOrExpr: data=" + data)
  return true
else
  WriteLine("isOrExpr: dataTypes are not boolean or int, d1.dataType="
           + d1.dataType + ", d2.dataType=" + d2.dataType)
  return false
else
  step
    WriteLine("isOrExpr: isAndExpr(expression2=" + expressions2 + ") is false")
    return false
elseif not hasError and isAndExpr(p, expressions, data, hasError) then
  return true
else
  return false

isOrExpr function searches for the 'or' operator in the sequence of expressions from the end of the sequence.

If it finds one, it splits the expressions into two sequences of expressions at the position of 'or'. Then it evaluates if the first sequence as an OrExpr. If the first sequence is not an OrExpr, it returns false. If the first sequence is an OrExpr, it evaluates the second sequence as an AndExpr. If the second sequence is an AndExpr, it performs an 'or' operation on the values of the first sequence and the second sequence, sets the data to the result value, and returns true.

If it does not find 'or', it evaluates the sequence as an AndExpr.
4.1.2.3 BPEL Documents Loader

In [32], there are no functions to read BPEL documents, which describe the behaviour of a process. The process definitions are hard-coded in the program of the model. Therefore, changing process definitions always requires changing the program. This is not practical since every time we change process definitions, we need to recompile and rebuild the program. In addition, with hard-coded process definitions, interactions between endpoints cannot occur since all endpoints will be created as instances of the same model with the same process definition. Therefore, we cannot define a different process for each endpoint. With this integrated model, the business process definitions can be provided in the form of XML documents. Thus, each endpoint can load its own process, which allows it to interact with other processes.

Since Microsoft .NET provides an XML library, we have created XMLReader class in C#. XMLReader provides two public methods to read documents, readDefinition and readProcessDoc. readDefinition opens the service description document and reads message, property, propertyAlias and portType into array lists. readProcessDoc opens the process definition document and reads variables, correlationSets and activities into array lists. The values of these array lists can be obtained by invoking the corresponding get methods.

4.1.2.4 Message Data Extension

The message data structure has been enhanced to accommodate complex data. The new structure conforms to the message definition in the
service description and consists of a message type name, parts and a port type name. *parts* maps a part name to *PART*, which corresponds to *part* of the message definition. Although a port type name is not in the message definition, it is needed for the Network Interface Model to transfer messages. The *MESSAGE* class data members are shown below:

```java
public class MESSAGE
    public var parts as Map of String to PART = {->} // name to PART
    var msgTypeName as String
    var portTypeName as String = ""
```

The data members of *PART* class are shown below:

```java
public class PART
    var name as String
    var partType as String = ""
    var dataset as Set of DATA = {}
```

Since a part may consist of multiple fields, each part contains a set of data as well as the part name and the part type.

The data members of *DATA* class are shown below:

```java
public class DATA
    public var value as String
    var name as String = ""
    var dataType as String = ""
```

*DATA* consists of *value*, *name* and *dataType*. Since any data is held in the form of *String*, *dataType* is needed to record the original type.

Figure 10 shows the data structure of the *Message* Class and its subordinate classes.
Variables are used to hold data and the state of a process. A variable consists of a name, an attribute, a type name, and a set of data or a message. The data members of the VARIABLE class are shown below:

```java
public class VARIABLE extends Object {
    var name as String
    var attribute as String // messageType, type or element
    var typeName as String // messageTypeName or type name or element name
    var dataset as set of DATA = {}
    var message as MESSAGE = null
}
```

A variable can contain one of the three types of data, WSDL message type, XML Schema type and XML Schema global element; **attribute** denotes which type: "messageType" for WSDL message type, "type" for XML Schema type and "element" for XML Schema global element. **typeName** is for specifying
the name of the type selected for attribute such as int, string, etc. dataset should be used only if the attribute is "type" or "element". When "messageType" is chosen for the attribute, the data should be contained in message.

This variable extension allows the process to meaningfully perform the assign activity, which assigns and copies data.

4.2 Abstract Communication Model

The Abstract Communication Model (ACM) [13], [14] is a high-level communication model designed by the Foundations of Software Engineering Group of Microsoft. They devised it after they noted the importance of distinguishing network behaviour from the behaviour of an application running on a network. Although the model was originally derived from TCP/IP networks for the UPnP architecture, it is not restricted to represent particular networks. It has been used as a generic model of a network infrastructure in designing different types of applications using network services [14].

In this project, we have applied a model implemented in AsmL, the Network Abstract Machine (NAM) [20]. The model has a versatile user interface written in C#. Since we have used the NAM model provided as a Dynamic Link Library (DLL), we do not discuss the implementation details of this model. Readers who are interested in the implementation details should refer to [20]. Instead, we summarise the overview structure of the ACM.

An ACM consists of a set of subnetworks, each of which has a communicator that is responsible for routing messages between applications.
running on computers connected to a network. A communicator controls all the routers of the subnetwork that connect to other networks. Each communicator has a routing table that maps from address to either communicator or application, and a mailbox to receive messages.

Figure 11 illustrates an instance of the abstract communication model in which Communicator 2 abstractly models the three routers [14].

![Abstract Communication Model](image)

**Figure 11: Abstract Communication Model [14]**

© 2004 IEEE, by permission

### 4.3 Network Integration

For each instance of the BPEL Service Abstract Machine, an instance of the Network Interface Model is required to transfer and to receive messages through the NAM. The Network Interface Model contains an address-mapping table, an outgoing message manager and an incoming message manager. The address-mapping table maps from port type to endpoint address and vice versa.
The outgoing message manager keeps checking the outbox of the BPEL Service Abstract Machine to see if there are any messages to send. If there are any messages, it picks a message, resolves the endpoint address of the receiver by using the address-mapping table, formats the message and sends it through the NAM. The outgoing message manager repeats this procedure until the outbox is empty. The incoming message manager periodically checks the mailbox of the associated peripheral of the NAM. If there are messages, it retrieves all the messages, formats them, converts the addresses into port types, and delivers the messages to the inbox of the BPEL Service Abstract Machine. Figure 12 shows the internal structure of the Network Interface Model. The code for the Network Interface Model is provided in Appendix B.

![Network Interface Model](image)

*Figure 12: Network Interface Model*
4.4 Simulation

We have constructed a simulation of the on-line credit card application system described in the example in Section 3.1.

Figure 13 shows the simulation composed of four instances of the BPEL Service Abstract Machine Model representing the credit card application handling division, the credit card evaluation process, the credit reporting bureau and the credit card issuing division. Each of them is connected to the NAM through its enclosing instance of the Network Interface Model. The line denoting message flow from a customer to the credit card application handling division is dashed since the initial message arrival from a customer is replaced by a user input in this simulation.
Figure 14 shows a screenshot of the user interface for the BPEL Service Abstract Machine Model representing the credit card application division. The top right corner area shows the loaded program from the process definition. The top middle area shows the list of processes. For each process, its currently running agent and the current activity are displayed. The top left corner area shows pending messages in the inbox. The bottom left corner area shows the history of the messages received. The bottom right corner area shows the history of the messages sent.
Figure 14: An instance of the extended BPEL Service Abstract Machine representing the credit card application division.

Figure 15 shows a screenshot of the credit card application form. This window shows up by pressing the **Apply** button from the credit card application window. Pressing the **Submit** button will close the credit card application window, create a message and simulate the initial message arrival at the credit card application division from a customer.
Figure 15: The credit card application form
Figure 16 shows a screenshot of the NAM. The credit card application handling division, the credit card evaluation process, the credit reporting bureau and the credit card issuing division are associated with Port 2 (Pr2), Port 4 (Pr4), Port 6 (Pr6) and Port 8 (Pr8) in the NAM respectively.

![Network Abstract Model](image)

Figure 16: Network Abstract Model [20]. Each of Pr2, Pr4, Pr6 and Pr8 is connected to a corresponding instance of the Network Interface Model.

Figure 17 shows a screenshot of the control panel of the simulation. The **Step** button and the **10 Steps** button will execute a single step and ten steps of each agent respectively. The **Auto Run** button periodically executes a step. The **Stop** button will stop periodical execution by the **Auto Run** button.
In order to run this simulation, a service description and a process definition should be provided to each BPEL endpoint as BPEL documents. The service descriptions and the process definitions of all BPEL endpoints are provided in Appendix C-J. After loading these documents, each endpoint creates a process to execute the activities as specified in the document upon receipt of an initial message.
CHAPTER 5 CONCLUSIONS

In this project, we have extended the executable formal model of BPEL [32] written in AsmL and integrated it with the NAM model based on the ACM in order to simulate multiple BPEL endpoints on asynchronous communication architectures. This integration yields a more realistic model that is suitable for extensive testing and validation.

We encountered some difficulties in implementing this project. Since BPEL uses numerous languages and technologies such as WSDL, XML Schema and XPath1.0, understanding the underlying technologies was an indispensable prerequisite. Secondly, the BPEL specification was not well structured. Since the specification discusses one subject in many places, one must pay attention when reading the specification to avoid omissions. In addition to these difficulties, debugging was not easy in AsmL. Since the debugger for .NET shows the values of variables in AsmL only in the assembly level, we needed to print out the values of variables for debugging. Another problem is that the AsmL program for the extended BPEL Service Abstract Machine Model, which is approximately 4000 lines of codes long, cannot be divided into several files. This is because of the restriction that a project of .NET does not allow having more than one AsmL file, which makes it harder to organise the structure of AsmL code.

With the foundation of the ASM method, the constructed model serves as an executable specification that specifies the system not only precisely, but also
intuitively, due to its pseudo-code-like syntax. The model provides
comprehensive views with abstract and nondeterministic descriptions, which help
readers to catch its drift without being distracted by details. This is useful
especially at early stages of the design process when the details of a system are
not known yet or are not important. In fact, since the level of abstraction can be
freely chosen in AsmL, the model can be easily adjusted as the development
evolves. Unlike ordinary prototyping, no commitments are needed for
implementation languages and tools, which makes the model more flexible and
independent of the implementation. The executability of the integrated model
greatly assists requirements capture, model validation, behaviour observation
and error detection before coding.
APPENDIX A. THE EXTENDED BPEL SERVICE ABSTRACT MACHINE MODEL EXTRACT

This extended BPEL Service Abstract Machine Model is based on the BPEL Service Abstract Machine Model [32]. The code of the original base model was created by Mona Vajihollahi. It has been extended and adapted by permission. We extract some code implemented in this project and also include the related parts that have already been implemented in [32].

Namespace

```csharp
namespace ModelGUI
{ import System.Collections
import System.Windows.Forms
import System.DateTime
import System.TimeSpan
}
```

Global Definitions

```csharp
type Time = Long
type Priority = Integer

[EntryPoint]
public enum EnumAssignType
    EnumAssignTypeVariable
    EnumAssignTypeVariablePart
    EnumAssignTypeVariablePartQuery
    EnumAssignTypeVariableProperty
    EnumAssignTypePartnerLink
    EnumAssignTypeExpression
    EnumAssignTypeLiteral
    EnumAssignTypeOpaque
    EnumAssignTypeElse

BASIC TYPES

Agent

```csharp
public class AGENT
    virtual Program()
```

Running Agent

```java
public class RUNNING_AGENT extends AGENT {
    var suspended as Boolean = false
    var rootProcess as PROCESS? = undef
    var parentAgent as RUNNING_AGENT? = undef
    var receiveMode as Boolean = false
    var id as String

    virtual finished()
    virtual addTriggeredEvents(evt as BPEL_EVENT, time as Time or Null)

    addNewActivityAgent(agent as ACTIVITYAGENT)
    agent.parentAgent := me
    agent.rootProcess := me.rootProcess
    add agent to me.rootProcess.dependentAgentSet
}
```

Event/Activity

```java
public class EVENT_ACTIVITY {
    var refNumber as Integer
}
```

Events

```java
public class BPEL_EVENT extends EVENT_ACTIVITY {
    var onEventActivity as ACTIVITY
}
```

```java
public class ON_MESSAGE_EVENT extends BPEL_EVENT {
    var typeFor as Boolean // for or until
    var timeValue as Time

    triggerTime(startTime as Time or Null) as Time
    match startTime
    sTime as Null:
        return timeValue
    sTime as Time:
        if typeFor then
            return sTime + timeValue
        else
            return timeValue
}
```

Activities

```java
[EntryPoint]
public class ACTIVITY extends EVENT_ACTIVITY {
    virtual getInfo() as String
    virtual getOnMessageSet() as Set of ON_MESSAGE_EVENT
    virtual getOnAlarmSet() as Set of ON_ALARM_EVENT
    virtual getSwCaseSet() as Set of SWCASE
}
```
virtual waCondition() as Boolean
virtual getInnerActivity() as ACTIVITY

public class STRUCTURED_ACTIVITY extends ACTIVITY
var assignedAgent as ACTIVITYAGENT? = undef

public class PICK extends STRUCTURED_ACTIVITY
var onMessageSet as Set of ON_MESSAGE_EVENT
var onAlarmSet as Set of ON_ALARM_EVENT

override getOnMessageSet() as Set of ON_MESSAGE_EVENT
return onMessageSet

override getOnAlarmSet() as Set of ON_ALARM_EVENT
return onAlarmSet

public class WHILE extends STRUCTURED_ACTIVITY
var condition as String
var innerActivity as ACTIVITY

override getInnerActivity() as ACTIVITY
return innerActivity

override waCondition() as Boolean
let result = random()
WriteLine("waCondition: " + result)
return result

public class SWITCH extends STRUCTURED_ACTIVITY
var swCaseSet as Set of SWCASE

override getSwCaseSet() as Set of SWCASE
return swCaseSet

public class WAIT extends ACTIVITY
var startTime as Time or Null = undef
var typeFor as Boolean //for or until
var timeValue as Time

completionTime() as Time
match startTime
stime as Null:
    return timeValue
stime as Time:
    if typeFor then
        return stime + timeValue
    else
        return timeValue

public class ASSIGN extends ACTIVITY
var copyMap as Map of Integer to COPY
Activity Agents

public class ACTIVITYAGENT extends RUNNING_AGENT
    var baseActivity as ACTIVITY

    override finished()
        remove me from me.rootProcess.model.agents
        remove me from me.rootProcess.dependentAgentSet

Pick Agent

public class PICK_MESSAGE_AGENT extends ACTIVITYAGENT

    override Program()
        if suspended = false then
            forall evt in baseActivity.getOnMessageSet()
                add (me, evt) to rootProcess.waitingForMessage
                writeln("Pick Message Agent: added waitingForMessage")
                suspended := true
        else
            choose (agent, op, time) in rootProcess.completedOperations where op in
            baseActivity.getOnMessageSet() and agent = me
            match op
                evt as ON_MESSAGE_EVENT:
                    writeln("event: " + evt.ToString() + " time: " + DateTime(time))
                    parentAgent.addTriggeredEvents(evt, time)
                    finished(me)

public class PICK_ALARM_AGENT extends ACTIVITYAGENT

    override Program()
        if suspended = false then
            startTime := DateTime.Now.Ticks
            suspended := true
        else
            forall evt in baseActivity.getOnAlarmSet() where
            evt.triggerTime(startTime) <= DateTime.Now.Ticks
                parentAgent.addTriggeredEvents(evt, evt.triggerTime(startTime))
                finished(me)

public class PICK_AGENT extends ACTIVITYAGENT
    var triggeredEvents as Set of (BPEL_EVENT, Time) = {}
    var chosenActivity as ACTIVITY? = undef

    override Program()
        if suspended = false then
            if chosenActivity = undef then
                step
                    var pAlarm as PICK_ALARM_AGENT = new PICK_ALARM_AGENT(id + ":pa" + idCounter, baseActivity)
                    var pMessage as PICK_MESSAGE_AGENT = new PICK_MESSAGE_AGENT(id + ":pm" + idCounter, baseActivity)
step
  addNewActivityAgent(pAlarm)
  add pAlarm to me.rootProcess.model.agents
  addNewActivityAgent(pMessage)
  add pMessage to me.rootProcess.model.agents
  suspended := true
else
  parentAgent.suspended := false
  finished(me)
if suspended = true then
  if chosenActivity = undef then
    choose (evt, time) in triggeredEvents where forall (e, t) in
    triggeredEvents holds time <= t
    chosenActivity := evt.onEventActivity
    writeln("Activity chosen")
  else
    writeln("chosen Activity set: " + chosenActivity.ToString())
    me.rootProcess.model.executeActivity(me, chosenActivity)
  override addTriggeredEvents(evt as BPEL_EVENT, time as Time or Null)
  add (evt, time) to triggeredEvents

While Agent

public class WHILE_AGENT extends ACTIVITYAGENT

  override Program()
  if suspended = true then
    me.rootProcess.model.executeActivity(me, baseActivity.getInnerActivity())
  else
    if baseActivity.waCondition() then
      suspended := true
    else
      parentAgent.suspended := false
      finished(me)

Switch Agent

public class SWITCH_AGENT extends ACTIVITYAGENT

  var foundBranch as ACTIVITY = undef

  override Program()
  if suspended = false then
    if foundBranch = undef then
      let caseSet = baseActivity.getSwCaseSet()
      choose c in caseSet where c.swCaseCondition(me.rootProcess) = true and
      (forall x in caseSet where x.swCaseCondition(me.rootProcess) = true holds
      c.swPriority >= x.swPriority)
      writeln("chosen case: " + c.ToString() + " result: " +
      c.swCaseCondition(me.rootProcess))
foundBranch := c.swCaseActivity
suspended := true
else
    parentAgent.suspended := false
    finished(me)
if suspended = true then
    me.rootProcess.model.execute_activity(me, foundBranch)

---

public class MODEL
execute_Invoke(self as RUNNING_AGENT, activity as INVOKE)
    writeLine("In Execute Invoke")
    let ioDescriptor = (self, activity)
    if not self.receiveMode then
        add ioDescriptor to self.rootProcess.outboxManager.outboxSpace
    if not activity.synchronous then
        self.suspended := false
    if activity.synchronous then
        self.receiveMode := true
        add ioDescriptor to self.rootProcess.waitingForMessage
    if self.receiveMode and not (ioDescriptor in self.rootProcess.waitingForMessage) then
        self.receiveMode := false
    self.suspended := false
    writeLine("The Invoke Activity is completed!")

---

public class MODEL
execute_Terminate(self as RUNNING_AGENT, activity as ACTIVITY)
    writeLine("In Execute Terminate")
    forall a in self.rootProcess.dependentAgentSet
        finished(a)
    finished(self.rootProcess)
    writeLine("The Terminate Activity is completed!")

---

public class MODEL
execute_Wait(self as RUNNING_AGENT, activity as WAIT)
    writeLine("In Execute Terminate")
    if activity.startTime = undef then
        activity.startTime := DateTime.Now.Ticks
    else
        if completionTime(activity) <= DateTime.Now.Ticks then
            self.suspended := false
            writeLine("The Wait Activity is completed!")
Execute Empty

```java
public class MODEL
    execute_Empty(self as RUNNING_AGENT, activity as ACTIVITY)
    writeln("In Execute Empty")
    self.suspended := false
    writeln("The Empty Activity is completed!")
```

Execute Assign

```java
public class MODEL
    execute_Assign(self as RUNNING_AGENT, act as ACTIVITY)
    var data as DATA
    var activity as ASSIGN

    step
    activity := (act as ASSIGN)
    step for i = 0 to size(activity.copyMap) - 1
        writeln("executeCopy(" + i + ")")
        (activity.copyMap(i) as COPY).executeCopy(self)
    step
    self.suspended := false
    writeln("The Assign Activity is completed!")
```

Execute Copy

We show only `executeCopy` since the subordinate functions are too many and too long.

Copy

```java
class COPY
    var fromType as EnumAssignType
    var fromString1 as String
    var fromString2 as String
    var fromString3 as String
    var toType as EnumAssignType
    var toString1 as String
    var toString2 as String
    var toString3 as String

    executeCopy(self as RUNNING_AGENT)
    var data as DATA
    var activity as ASSIGN
    var toData as DATA

    writeln("In executeCopy")

    p = self.rootProcess
```

79
if toType <= EnumAssignTypeVariableProperty then
    toV = p.getVariable(toString1)
    if toV = null then
        WriteLine("Cannot find variable name: " + toString1)
    elseif toType = EnumAssignTypeVariable then
        if fromType <= EnumAssignTypeVariableProperty then   //if variable
            fromV = p.getVariable(fromString1)
            if fromV = null then
                WriteLine("Cannot find variable name: " + fromString1)
            elseif fromType = EnumAssignTypeVariable then
                assignVariableToVariable(fromV, toV)
            elseif fromType = EnumAssignTypeVariablePart then
                assignVariablePartToVariable(fromV, toV, me)
            elseif fromType = EnumAssignTypeVariablePartQuery then
                assignVariablePartQueryToVariable(fromV, toV, me)
            elseif fromType = EnumAssignTypeVariableProperty then
                assignVariablePropertyToVariable(p, fromV, toV, me)
            else
                WriteLine("copy variable Error: Wrong attribute")
            elseif fromType = EnumAssignTypeExpression then
                //currently expression is not supported
                assignExpressionToVariable(fromString1, toV)
            elseif fromType = EnumAssignTypeLiteral then
                assignLiteralToVariable(fromString1, toV)
            elseif fromType = EnumAssignTypeOpaque then
                assignOpaqueToVariable(toV)
            elseif fromType = EnumAssignTypePartnerLink then
                WriteLine("Currently PartnerLink is not supported")
            else
                WriteLine(”Undefined fromType Error”)
            elseif toType = EnumAssignTypeVariablePart then
                toPart = toV.getPart(toString2, "", false)
                if toPart = null then
                    WriteLine("cannot find variable:" + toString1 + " part:" +
                                toString2)
                else
                    assignToPart(p, toV, toPart)
                elseif toType = EnumAssignTypeVariablePartQuery then
                    if toV.getAttribute <> "messageType" or toV.message = null then
                        WriteLine("to variable:" + toV.name + " type is not messageType")
                    else
                        step   //this must be done after knowing the type of the source data
                        if toV.getData(toString2, "", false, toString3,
                                        p.model.intEnv.getTypeByMessagePartQuery(toV.message.msgTypeName, toString2,
                                                                                         toString3), true) = null then
                            WriteLine("cannot find part(" + toString2 + ") query(" +
                                    toString3 + ")")
                        else
                            step
                        end
                    end
                end
            end
        end
    end
end
toData := toV.getData(toString2, "", false, toString3, 
  p.model.intEnv.getTypeByMessagePartQuery(toV.message=msgTypeName, toString2, 
  toString3), true) 
  step 
  assignToData(p, toV, toData) 
  elseif toType = EnumAssignTypeVariableProperty then 
    pAlias = 
    p.model.intEnv.getPropertyAliasByPropertyAndMessageType(toString2, 
    toV.typeName) 
    if pAlias = null then 
      WriteLine("Cannot find Variable:" + toString1 + " property:" + 
    toString2) 
    elseif pAlias.query = "" then //No query means only one data per 
      part 
        toPart = toV.getPart(pAlias.partName, 
        p.model.intEnv.getTypeByProperty(pAlias.name), false) //partType should be set 
        when initialized 
        //Same as toType = EnumAssignTypeVariablePart 
        assignToPart(p, toV, toPart) 
    else 
      step 
      toData := toV.getData(pAlias.partName, "", false, pAlias.query, 
      p.model.intEnv.getTypeByProperty(pAlias.name), true) 
      step 
      assignToData(p, toV, toData) 
      elseif toType = EnumAssignTypePartnerLink then 
        WriteLine("PartnerLink is not currently supported") 
    else 
      WriteLine("Undefined fromType Error") 
    end 
    WriteLine("Copy(toType=" + toType + ", fromType=" + fromType + " from1=" + 
      fromString1 + ", from2=" + fromString2 + ", from3=" + fromString3 + ", to1=" + 
      toString1 + ", to2=" + toString2 + ", to3=" + toString3 + ") is completed!")

Pick Activity Clearance

public class INBOX_MANAGER

  PickActivityClearance(p as PROCESS, ra as RUNNING_AGENT, inActivity as 
  INPUT_OUTPUT_ACTIVITY or ON_MESSAGE_EVENT) 
  match ra 
    pm a as PICK_MESSAGE_AGENT: 
      forall (a, o) in p.waitingForMessage where a = ra and o <> inActivity 
      remove (a, o) from p.waitingForMessage 
      WriteLine("removing pickActivity: " + a.ToString() + o.ToString())

Switch Case

public class SWCASE

  var condition as String 
  var swCaseActivity as ACTIVITY
var swPriority as Priority

swCaseCondition(p as PROCESS) as Boolean
primitive var expressions as Map of Integer to DATA = { -> }
var condStr as String = condition
primitive var hasError as Boolean = false
primitive var data as DATA = null
var noError as Boolean = false

writeLine("swCaseCondition: condStr:" + condStr)
if condStr = "" then //must be Otherwise
    return true
else
    step
    noError := readCondition(p, condStr, expressions)
    step
    if noError then
        step
        writeLine("swCaseCondition: expressions=" + expressions)
        step
        if expressions = { -> } then
            writeLine("expressions is empty")
            return true
        elseif isExpr(p, expressions, data, hasError) then
            return System.Boolean.Parse(data.value)
        else
            writeLine("Switch Case condition=" + condition + " is not an expression")
            return false
    else
        writeLine("swCaseCondition: Syntax Error")
        return false

isExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
if not hasError and isOrExpr(p, expressions, data, hasError) then
    return true
else
    writeLine(expressions + " is not an expression")
    return false

isPrimaryExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
primitive var stripped as Map of Integer to DATA = { -> }
if expressions = { -> } then
    writeLine("Expression is empty")
    return false
elseif not hasError and isBoolean(expressions, data, hasError) then
    return true
elseif not hasError and isLiteral(expressions, data, hasError) then
    return true

82
elseif not hasError and isNumber(expressions, data, hasError) then
  return true
elseif not hasError and isFunctionCall(p, expressions, data, hasError) then
  return true
elseif not hasError and inParenthesis(expressions, stripped, hasError) then
  return isExpr(p, stripped, data, hasError)
else
  writeLine(expressions + " is not a primary expression")
  return false

isBoolean(expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
if not hasError and expressions <> {} and expressions(0).dataType = "boolean" and expressions(0).value = "true" or expressions(0).value = "false" and Size(expressions) = 1 then
  step
  writeLine("isBoolean: Boolean expression!")
  data := expressions(0).copy()
  step
  return true
else
  writeLine("isBoolean: Not Boolean expression!")
  return false

isLiteral(expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
if expressions(0).dataType = "String" and expressions(0).name = "Literal" and Size(expressions) = 1 then
  step
  writeLine("isLiteral: expression is Literal! " + expressions)
  data := expressions(0).copy()
  step
  return true
else
  writeLine("isLiteral: expression is Not Literal! " + expressions)
  return false

isNumber(expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
if expressions(0).name = "Number" and Size(expressions) = 1 then
  step
  writeLine("isNumber: expression is Numbers! " + expressions)
  data := expressions(0).copy()
  step
  return true
else
  writeLine("isNumber: expression is not Numbers! " + expressions)
  return false

inParenthesis(expressions as Map of Integer to DATA, primitive out stripped as Map of Integer to DATA, primitive ref hasError as Boolean) as Boolean
primitive var expressions1 as Map of Integer to DATA = {->}
primitive var expressions2 as Map of Integer to DATA = {->}
step
  stripped := {->}
step
  if expressions(0).value = "(" and expressions(Size(expressions)-1).value = ")"
    then
      splitExpressions(expressions, expressions1, expressions2, 1)
      cutExpressions(expressions2, stripped, Size(expressions2) - 1)
      writeLine("inParenthesis: stripped=" + stripped)
      return true
    else
      writeLine("inParenthesis: expression does not include parenthesis! " + expressions)
      return false

includes(expressions as Map of Integer to DATA, str as String, start as Integer, end as Integer, primitive out pos as Integer) as Boolean
  var found as Boolean = false
  var i as Integer = start
step
  pos := -1
step
  step while (not found and i <= end)
  step
    if expressions(i).value = str then
      pos := i
      found := true
    step
  i := i + 1
step
  return found

includesInSet(expressions as Map of Integer to DATA, strs as Set of String, start as Integer, end as Integer, primitive out pos as Integer, primitive out sign as String) as Boolean
  var found as Boolean = false
  var i as Integer = start
step
  pos := -1
  sign := ""
step
  step while (not found and i <= end)
  step
    choose str in strs where expressions(i).value = str
    pos := i
    found := true
    sign := str
  step
  i := i + 1
step
return found

includesReverse(expressions as Map of Integer to DATA, str as String, start as Integer, end as Integer, primitive out pos as Integer) as Boolean
var found as Boolean = false
var i as Integer = end

step
  pos := -1
step
  step while (not found and i >= start)
    step
      if expressions(i).value = str then
        pos := i
        found := true
    step
    i := i - 1
step
  return found

includesInSetReverse(expressions as Map of Integer to DATA, set of String, start as Integer, end as Integer, primitive out pos as Integer, primitive out sign as String) as Boolean
var found as Boolean = false
var i as Integer = end

step
  pos := -1
  sign := ""
step
  step while (not found and i >= start)
    step
      choose str in set of Str where expressions(i).value = str
      pos := i
      found := true
      sign := str
    step
    i := i - 1
step
  return found

endsWith(expressions as Map of Integer to DATA, str as String) as Boolean
var found as Boolean = false

step
  if expressions(Size(expressions) - 1).value = str then
    found := true
step
  return found
splitExpressions(expressions as Map of Integer to DATA, primitive out expressions1 as Map of Integer to DATA, primitive out expressions2 as Map of Integer to DATA, exp2startPos as Integer)

step
  expressions1 := {->}
  expressions2 := {->}

step
  forall i in Indices(expressions) where i < exp2StartPos
  expressions1(i) := expressions(i)
  forall i in Indices(expressions) where i >= exp2StartPos
  expressions2(i - exp2StartPos) := expressions(i)

cutExpressions(primitive ref expressions as Map of Integer to DATA, primitive out expressions2 as Map of Integer to DATA, exp2StartPos as Integer)

  var expressions1 as Map of Integer to DATA = {->}

  step
    expressions2 := {->}

  step
    forall i in Indices(expressions) where i < exp2StartPos
    expressions1(i) := expressions(i)
    forall i in Indices(expressions) where i >= exp2StartPos
    expressions2(i - exp2StartPos) := expressions(i)

  step
    forall i in Indices(expressions1) where i < exp2StartPos
    expressions(i) := expressions1(i)

isFunctionCall(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean

if not hasError and expressions(O).dataType = "functionName" then
  if (expressions(O).value = "bpws:getvariableData" or expressions(O).value = "getvariableData") and expressions(1).value = "(" and endswith(expressions, ")") then
    return isGetVariableData(p, expressions, data, hasError)
  else //currently only GetVariableData is supported
    writeln("isFunctionCall: Not a correct function expressions(O).value =" + expressions(O).value)
    return false
  else
    writeln("isFunctionCall: datatype is not functionName
    expressions(O).dataType =" + expressions(O).dataType)
    return false

isGetVariableData(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean

  primitive var pos as Integer
  var variable as VARIABLE
  var part as PART
  primitive var expressions0 as Map of Integer to DATA = {->}
  primitive var expressions1 as Map of Integer to DATA = {->}
  primitive var expressions2 as Map of Integer to DATA = {->}

  86
primitive var expressions3 as Map of Integer to DATA = {
}
primitive var expressions4 as Map of Integer to DATA = {
}
primitive var expressions5 as Map of Integer to DATA = {

if not hasError and expressions(0).dataType = "functionName" then
  if (expressions(0).value = "bpws:getVariableData" or expressions(0).value = "getVariableData") and expressions(1).value = "(" and endswith(expressions, ")") then
    if expressions(2).value <> "(" and expressions(2).value <> "(" then
      step
      WriteLine("isGetVariableData: expressions(2) is not (" + expressions(2).value)
      splitExpressions(expressions, expressions0, expressions1, 2)
      step
      WriteLine("isGetVariableData: after split expressions1=" + expressions1)
    end
    if splitIfIncludes("", expressions1, expressions2, expressions3, hasError) then
      WriteLine("isGetVariableData: expressions1 includes "," + expressions1)
      WriteLine("isGetVariableData: expressions2 =" + expressions2)
      WriteLine("isGetVariableData: expressions3 =" + expressions3)
  end
  variable := p.getVariable(expressions2(0).value)
  step
  if expressions3(0).value <> "(" then
    if splitIfIncludes("", expressions3, expressions4, expressions5, hasError) then
      step
      part := variable.getPart(expressions4(0).value, "", false)
      step
      choose d in part.dataset where d.name = expressions5(0).value
      step
      data := d.copy()
      step
      return true
    else //expression3(0) is partName. variable Name and part Name only
      step
      part := variable.getPart(expressions3(0).value, "", false)
      step
      choose d in part.dataset
      step
      data := d.copy()
      step
      WriteLine("isGetVariableData true! data=" + data)
      return true
    else //expression3(0).value is "("
//Currently this case is not supported
hasError := true
return false
else //variable Name only -> It must be XML Schema simple type or
element
step
variable := p.getVariable(expressions1(0).value)
step
choose d in variable.dataset
step
data := d.copy()
step
WriteLine("isGetVariableData true! data=" + data)
return true
else //expressions(2).value = " or expression(2).value = " then
step
hasError := true
step
return false
else //not getVariableData
//This case is currently not supported
step
hasError := true
return false
else // Not a function Name
return false

isOrExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref
data as DATA, primitive ref hasError as Boolean) as Boolean
primitive var d1 as DATA = null
primitive var d2 as DATA = null
primitive var expressions1 as Map of Integer to DATA = {->}
primitive var expressions2 as Map of Integer to DATA = {->}
var boolVal as Boolean
if not hasError and splitIfIncludesReverse("or", expressions, expressions1,
expressions2, hasError) then
   if not isOrExpr(p, expressions1, d1, hasError) then
      WriteLine("isOrExpr: isOrExpr(expressions1=" + expressions1 + ") is
false")
   return false
else if isAndExpr(p, expressions2, d2, hasError) then
   if (d1.dataType = "boolean" or d1.dataType = "int") and (d2.dataType = 
"boolean" or d2.dataType = "int") then
      boolVal := System.Boolean.Parse(d1.value) or 
      System.Boolean.Parse(d2.value)
   step
data := new DATA(boolVal.ToString())
   step
```java
data.dataType := "boolean"
data.name := "boolean"

step
    WriteLine("isOrExpr: expression is or expression!=" + expressions)
    WriteLine("isOrExpr: dl=" + d1 + "d2=" + d2)
    WriteLine("isOrExpr: data=" + data)
    return true
else
    WriteLine("isOrExpr: DataTypes are not boolean or int, dl.dataType=" + d1.dataType + "d2.dataType=" + d2.dataType)
    return false
else
    step
        WriteLine("isOrExpr: isAndExpr(expression2=" + expressions2 + ") is false")
        return false
elseif not hasError and isAndExpr(p, expressions, data, hasError) then
    return true
else
    return false

splitIfIncludesReverse(str as String, expressions as Map of Integer to DATA, primitive out expressions1 as Map of Integer to DATA, primitive out expressions2 as Map of Integer to DATA, primitive ref hasError as Boolean) as Boolean
    primitive var expressions0 as Map of Integer to DATA
    primitive var pos as Integer
    step
        expressions1 := {}
        expressions2 := {}
        retval = includesReverse(expressions, str, 1, Size(expressions) - 1, pos)
        step
            retval
            if not hasError and includesReverse(expressions, str, 1, Size(expressions) - 1, pos) then
                splitExpressions(expressions, expressions1, expressions0, pos)
                cutExpressions(expressions0, expressions2, 1)
                return true
            else
                return false

splitIfIncludesSetReverse(strs as Set of String, expressions as Map of Integer to DATA, primitive out expressions1 as Map of Integer to DATA, primitive out expressions2 as Map of Integer to DATA, primitive ref hasError as Boolean, primitive out sign as String) as Boolean
    primitive var expressions0 as Map of Integer to DATA
    primitive var pos as Integer
    step
        expressions1 := {}
        expressions2 := {}
        step
```
if not hasError and includesInSetReverse(expressions, strs, 1, Size(expressions) - 1, pos, sign) then
    splitExpressions(expressions, expressions1, expressions0, pos)
cutExpressions(expressions0, expressions2, 1)
return true
else
return false

splitIfIncludes(str as String, expressions as Map of Integer to DATA, primitive ref expressions1 as Map of Integer to DATA, primitive ref expressions2 as Map of Integer to DATA, primitive ref hasError as Boolean) as Boolean
    primitive var expressions0 as Map of Integer to DATA
    primitive var pos as Integer
    if not hasError and includes(expressions, str, 1, Size(expressions) - 1, pos) then
        splitExpressions(expressions, expressions1, expressions0, pos)
cutExpressions(expressions0, expressions2, 1)
return true
else
return false

splitIfIncludesSet(strs as Set of String, expressions as Map of Integer to DATA, primitive ref expressions1 as Map of Integer to DATA, primitive ref expressions2 as Map of Integer to DATA, primitive ref hasError as Boolean, primitive out sign as String) as Boolean
    primitive var expressions0 as Map of Integer to DATA = {}->
    primitive var pos as Integer
    if not hasError and includesInSet(expressions, strs, 1, Size(expressions) - 1, pos, sign) then
        splitExpressions(expressions, expressions1, expressions0, pos)
cutExpressions(expressions0, expressions2, 1)
return true
else
return false

isAndExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
    primitive var d1 as DATA = null
    primitive var d2 as DATA = null
    primitive var expressions1 as Map of Integer to DATA = {}->
    primitive var expressions2 as Map of Integer to DATA = {}->
    var boolval as Boolean = false
    var ret1 as Boolean
    var ret2 as Boolean
    if not hasError and splitIfIncludesReverse("and", expressions, expressions1, expressions2, hasError) then
        step
        ret1 := isAndExpr(p, expressions1, d1, hasError)
        ret2 := isEqualityExpr(p, expressions2, d2, hasError)
step
  if not ret1 then
    writeln("isAndExpr: isAndExpr(expression1=" + expressions1 + ") is false")
    return false
  elseif ret2 then
    if (dl.dataType = "boolean" or dl.dataType = "int") and (d2.dataType = "boolean" or d2.dataType = "int") then
      step
        boolVal := System.Boolean.Parse(dl.value) and
          System.Boolean.Parse(d2.value)
      step
        data := new DATA(boolVal.ToString())
      step
        data.datatype := "boolean"
        data.name := "boolean"
      step
        writeln("isAndExpr: expression is and expression!=" + expressions1 + ")
        writeln("isAndExpr: d1=" + d1 + "+ d2=" + d2)
        writeln("isAndExpr: data=" + data)
        return true
    else
      step
        writeln("isAndExpr: dataTypes are not boolean or int, d1.dataType=" + dl.dataType + " d2.dataType=" + d2.dataType)
        hasError := true
        return false
    else
      step
        writeln("isAndExpr: isEqualityExpr(expression2=" + expressions2 + ") is false")
        return false
      elseif not hasError and isEqualityExpr(p, expressions, data, hasError) then
        return true
      else
        return false
    end if
end if

isEqualityExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
  primitive var dl as DATA = null
  primitive var d2 as DATA = null
  primitive var expressions1 as Map of Integer to DATA = {}->
  primitive var expressions2 as Map of Integer to DATA = {}->
  primitive var sign as String = ""
  var ret1 as Boolean
  var ret2 as Boolean

  if not hasError and splitIfIncludesSetReverse("=", "/="), expressions, expressions1, expressions2, hasError, sign) then
    if not isEqualityExpr(p, expressions1, d1, hasError) then

  end if

91
writeLine("isEqualityExpr: isEqualityExpr(expession1=" + expressions1 + ") is false")
  return false
else if isRelationalExpr(p, expressions2, d2, hasError) then
  if d1.dataType = d2.dataType then
    step
    if sign = "=" then
      data := new DATA(BooleanToString(d1.value = d2.value))
    elseif sign = "!=" then
      data := new DATA(BooleanToString(d1.value <> d2.value))
    step
    data.dataType := "boolean"
    data.name := "boolean"
    step
    writeLine("isEqualityExpr: expression is equality expression!=" + expressions)
    writeLine("isEqualityExpr: d1=" + d1 + "d2=" + d2)
    writeLine("isEqualityExpr: data=" + data)
    return true
  else
    step
    hasError := true
    writeLine("isEqualityExpr: dataTypes are not boolean or int, d1=" + d1 + "d2=" + d2)
    return false
  else
    writeLine("isEqualityExpr: isRelationalExpr(experssion2=" + expressions2 + ") is false")
    return false
  elseif isRelationalExpr(p, expressions, data, hasError) then
    return true
  else
    return false

BooleanToString(cond as Boolean) as String
  if cond then
    return System.Boolean.TrueString
  else
    return System.Boolean.FalseString

isRelationalExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
  primitive var d1 as DATA = null
  primitive var d2 as DATA = null
  primitive var expressions1 as Map of Integer to DATA = {->}
  primitive var expressions2 as Map of Integer to DATA = {->}
  primitive var sign as String
  var boolVal as Boolean = false
  var ret1 as Boolean
  var ret2 as Boolean
if not hasError and splitIfIncluudesSetReverse("<", ">", "<=", ">=")}, expressions, expressions1, expressions2, hasError, sign)
step
ret1 := isRelationalExpr(p, expressions1, d1, hasError)
ret2 := isAdditiveExpr(p, expressions2, d2, hasError)
step
if not ret1 then
writeLine("isRelationalExpr: isRelationalExpr(expression1=" + expressions1 + ") is false")
return false
elseif ret2 then
if d1.dataType = d2.dataType and d1.dataType = "int" then
step
if sign = "<" then
elseif sign = ">" then
elseif sign = "<=" then
elseif sign = ">=" then
step
data := new DATA(boolVal.ToString())
step
data.dataType := "boolean"
data.name := "boolean"
step
writeLine("isRelationalExpr: expression is expression1=" + expressions)
writeLine("isRelationalExpr: d1=" + d1 + "d2=" + d2)
writeLine("isRelationalExpr: data=" + data)
return true
else
step
hasError := true
writeLine("isRelationalExpr: dataTypes are not int, d1.dataType=" + d1.dataType + "d2.dataType=" + d2.dataType)
return false
else
writeLine("isRelationalExpr: isAdditiveExpr(expression2=" + expressions2 + ") is false")
return false
elseif not hasError and isAdditiveExpr(p, expressions, data, hasError) then
return true
else
return false
isAdditiveExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive
ref data as DATA, primitive ref hasError as Boolean) as Boolean
primitive var d1 as DATA = null
primitive var d2 as DATA = null
primitive var expressions1 as Map of Integer to DATA = { -> }
primitive var expressions2 as Map of Integer to DATA = { -> }
primitive var sign as String
var intval as Integer = 0
if not hasError and splitIfIncludesSetReverse("+", ", ", expressions, expressions1, expressions2, hasError, sign) then
  if not isAdditiveExpr(p, expressions1, d1, hasError) then
    writeLine("isAdditiveExpr: isAdditiveExpr(expression1=" + expressions1 + ") is false")
    return false
  elseif isMultiplicativeExpr(p, expressions2, d2, hasError) then
    if d1.dataType = d2.dataType and d1.dataType = "int" then
      step
      if sign = "+" then
      elseif sign = "-" then
        intval := System.Int32.Parse(d1.value) - System.Int32.Parse(d2.value)
      step
      data := new DATA(intVal.ToString())
      step
data.dataType := d1.dataType
data.name := d1.dataType
      step
      writeLine("isAdditiveExpr: expression is additive expression!=" + expressions)
      writeLine("isAdditiveExpr: d1=" + d1 + "d2=" + d2)
      writeLine("isAdditiveExpr: data=" + data)
      return true
    else
      hasError := true
      writeLine("isAdditiveExpr: dataTypes are not boolean or int, d1.dataType=" + d1.dataType + "d2.dataType=" + d2.dataType)
      return false
    else
      writeLine("isAdditiveExpr: isMultiplicativeExpr(expression2=" + expressions2 + ") is false")
      return false
    elseif not hasError and isMultiplicativeExpr(p, expressions, data, hasError) then
      return true
    else
      return true
isMultiplicativeExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean

primitive var expressions1 as Map of Integer to DATA = {->}
primitive var expressions2 as Map of Integer to DATA = {->}
primitive var sign as String
var intval as Integer = 0
primitive var d1 as DATA = null
primitive var d2 as DATA = null

if not hasError and splitIfIncludesSetReverse("\", "mod"), expressions, expressions1, expressions2, hasError, sign) then
  if not isMultiplicativeExpr(p, expressions1, d1, hasError) then
    WriteLine("isMultiplicativeExpr: isMultiplicativeExpr (expression1=" + expressions1 + ") is false")
    return false
  elseif isUnaryExpr(p, expressions2, d2, hasError) then
    if d1.dataType = d2.dataType and d1.dataType = "int" then
      step
        if sign = "\" then
          intval := System.Int32.Parse(d1.value) *
          System.Int32.Parse(d2.value)
        elseif sign = "mod" then
          intval := System.Int32.Parse(d1.value) mod
          System.Int32.Parse(d2.value)
        else
          hasError := true
      step
      if hasError then
        return false
      else
        step
          data := new DATA(intval.ToString())
        step
          data.dataType := d1.dataType
          data.name := d1.dataType
        step
          WriteLine("isMultiplicativeExpr: expression is multiplicative" + expressions)
          WriteLine("isMultiplicativeExpr: d1=" + d1 + "d2=" + d2)
          WriteLine("isMultiplicativeExpr: data=" + data)
          return true
      else
        hasError := true
        WriteLine("isMultiplicativeExpr: dataTypes are not boolean or int, d1.dataType=" + d1.dataType + "d2.dataType=" + d2.dataType)
        return false
      else
        WriteLine("isMultiplicativeExpr: isUnaryExpr(expression2=" + expressions2 + ") is false")
        return false
    elseif not hasError and isUnaryExpr(p, expressions, data, hasError) then
return true
else
    return false

isUnaryExpr(p as PROCESS, expressions as Map of Integer to DATA, primitive ref data as DATA, primitive ref hasError as Boolean) as Boolean
primitiv var expressions1 as Map of Integer to DATA = {->}
    if not hasError and includeUnary(expressions, expressions1, hasError) then
        if isUnaryExpr(p, expressions1, data, hasError) then
            if data.dataType = "int" then
                step
data := new DATA("-" + data.value)
step
data.dataType := "int"
data.name := "int"
step
        writeLine("isUnaryExpr: expression is unary expression!=" + expressions)
        writeLine("isUnaryExpr: data=" + data)
        return true
    else
        hasError := true
        writeLine("isUnaryExpr: dataTypes are not int, data.dataType=" + data.dataType)
        return false
    else
        writeLine("isUnaryExpr: expression1=" + expressions1 + " is not an expression")
        return false
    elseif not hasError and isPrimaryExpr(p, expressions, data, hasError) then
        return true
    else
        return false

includeUnary(expressions as Map of Integer to DATA, primitive ref expressions1 as Map of Integer to DATA, primitive ref hasError as Boolean) as Boolean
primitiv var expressions0 as Map of Integer to DATA = {->}
primitiv var pos as Integer
    if not hasError and includes(expressions, "+", 0, 0, pos) then
        splitExpressions(expressions, expressions0, expressions1, pos + 1)
        return true
    else
        return false

readCondition(p as PROCESS, conditionStr as String, primitive ref expressions as Map of Integer to DATA) as Boolean
primitiv var condStr as String = ""
primitiv var data as DATA as new DATA(""
var i as Integer = 0
var noError as Boolean = true

/** Assuming
No parentheses
No function other than getVariableData
*/
step
condStr:= conditionStr.TrimStart(null)
expressions := {->}
step
if condStr = "" then //must be otherwise
    return true
else
    condStr:= condStr.TrimEnd(null)
step
while (condStr.Length > 0 and noError)
    step
    condStr:= condStr.TrimStart(null)
    step
    if readBoolean(condStr, data) then
        skip
    elseif readParenthesis(condStr, data) then
        skip
    elseif readNumber(condStr, data) then
        skip
    elseif readOperator(condStr, data) then
        skip
    elseif readComma(condStr, data) then
        skip
    elseif readComma(condStr, data) then
        skip
    elseif readName(p, condStr, data) then
        step
            WriteLine("read condition: readName data=" + data)
    else
        WriteLine("read condition error: " + condStr)
        noError := false
    step
    if noError then
        step
        expressions(i) := data.copy()
        step
            WriteLine("readCondition expressions(" + i + ")=" +
        expressions(i))
        i += 1
    step
    if noError then
        WriteLine("readCondition: " + expressions)
        return true
    else
        WriteLine("readCondition: Error" + expressions)
        return false
readParenthesis(primitive ref condStr as String, primitive ref data as DATA) as Boolean
if condStr.StartsWith("(") then
  step
  data.value := ")"
  step
  WriteLine("readParenthesis: Left Parenthesis =" + data.value)
  data.dataType := "Paren"
  data.name := "Left Paren"
  condStr := condStr.Substring(1).TrimStart(null)
  step
  return true
elseif condStr.StartsWith(")") then
  step
  data.value := ")")
  step
  WriteLine("readParenthesis: Right Parenthesis =" + data.value)
  data.dataType := "Paren"
  data.name := "Right Paren"
  condStr := condStr.Substring(1).TrimStart(null)
  step
  return true
else
  return false

readNumber(primitive ref condStr as String, primitive ref data as DATA) as Boolean
var i as Integer = 0
var isNumber as Boolean = true
step while i < condStr.Length and isNumber
  if System.Char.IsNumber(condStr, i) then
    step
    i += 1
  else
    step
    isNumber := false
  step
  if i = 0 then
    WriteLine("readNumber: Not a number")
    return false
  else
    step
    data.value := condStr.Substring(0, i)
    step
    WriteLine("readNumber: value =" + data.value)
    data.dataType := "int"
    data.name := "Number"
    condStr := condStr.Substring(i).TrimStart(null)
    return true
readBoolean(primitive ref condStr as String, primitive ref data as DATA) as Boolean

var endPos as Integer
var boolStr as String

step
    endPos := condStr.IndexOf(" ")

step
    if endPos <> -1 then
        boolStr := condStr.Substring(0, endPos)
    else
        // If no space, the whole string
        boolStr := condStr

step
    if boolStr = "true" or boolStr = "false" then
        step
            data.value := boolStr
            step
                WriteLine("readBoolean: value=" + data.value)
                data.getDataType := "boolean"
                data.setName := "boolean"
                WriteLine("readBoolean: condStr=" + condStr)
                if endPos = -1 then
                    condStr := ""
                else
                    condStr := condStr.Substring(endPos).TrimStart(null)
            step
                WriteLine("readBoolean: condStr=" + condStr)
                return true
    else
        WriteLine("readBoolean: Not Boolean")
        return false

readLiteral(primitive ref condStr as String, primitive ref data as DATA) as Boolean

var endPos as Integer

if condStr.StartsWith(" ") then
    step
        endPos := condStr.IndexOf(" ", 1)

step
    if endPos > 0 then
        step
            data.value := condStr.Substring(1, endPos - 1)
            step
                WriteLine("readLiteral: value=" + data.value)
                data.getDataType := "String"
                data.setName := "String"
                condStr := condStr.Substring(endPos + 1).TrimStart(null)
                return true
else
writeLine("readLiteral Error: String does not end with ")
return false

else
writeLine("readLiteral: Not Literal")
return false

readName(p as PROCESS, primitive ref condStr as String, primitive ref data as DATA) as Boolean
var i as Integer = 1
var isChar as Boolean = true

writeLine("readName")

if not System.Char.IsLetter(condStr, 0) and not condStr.StartsWith("_")
then
writeLine("readName: The First char is not a letter or _")
return false
else
step while i < Size(condStr) and isChar
    if System.Char.IsLetterOrDigit(condStr, i) then
        i := i + 1
    elseif i + 1 < Size(condStr) then
        if condStr.Substring(i, 1) = "_" or condStr.Substring(i, 1) = ":" or condStr.Substring(i, 1) = "." or condStr.Substring(i, 1) = ":" then
            i := i + 1
        else
            isChar := false
        end
    else
        isChar := false
    end
step
writeLine("readName: i=" + i + "data:" + condStr.Substring(0, i))
writeLine("readName: condStr=" + condStr.Substring(i).TrimStart(null))
data.value := condStr.Substring(0, i)
step
if condStr.Substring(0, i) = "bpws:getvariableData" or condStr.Substring(0, i) = "getvariableData" then
data.setDatName("functionName")
data.setDatType("functionName")
else
data.name := "name"
data.datType := "name"
condStr := condStr.Substring(i).TrimStart(null)
step
writeLine("readName: data=" + data)
return true

readComma(primitive ref condStr as String, primitive ref data as DATA) as Boolean
if condStr.StartsWith(",") then
step
condStr := condStr.Substring(Size(",").TrimStart(null)
data.value := "," 

step
   WriteLine("readComma: value=" + data.value)
   data.name := "comma"
   data.dataType := "comma"
   return true
else
   WriteLine("readComma: Not Comma")
   return false

// or and = != < > <= >= + - div mod *
readOperator(primitive ref operatorString as String, primitive ref data as DATA) as Boolean
   var tempString as String
   var lessThanString as String = "\"<\"
   var greaterThanString as String = "\">\\"
   if operatorString.TrimStart(null).StartsWith(lessThanString) then
      step
      tempString :=
      operatorString.Substring(lessThanString.Length).TrimStart(null)
      step
      if tempString.StartsWith("\"=\"") then
         step
         operatorString := tempString.Substring(1).TrimStart(null)
         data.value := "\"<=\"
         step
         WriteLine("readOperator: value=" + data.value)
         data.name := "operator"
         data.dataType := "operator"
         return true
   else
      step
      operatorString := tempString.TrimStart(null)
      data.value := "\"<\"
      step
      WriteLine("readOperator: value=" + data.value)
      data.name := "operator"
      data.dataType := "operator"
      return true
   elseif operatorString.StartsWith(greaterThanString) then
      step
      tempString :=
      operatorString.Substring(greaterThanString.Length).TrimStart(null)
      step
      if tempString.StartsWith("\"=\"") then
         step
         operatorString := tempString.Substring(1).TrimStart(null)
         data.value := "\">="
         step
         WriteLine("readOperator: value=" + data.value)
         data.name := "operator"
         data.dataType := "operator"
         return true
   else
      // or and = != < > <= >= + - div mod *
      return false
return true
else
step
operatorString := tempString.TrimStart(null)
data.value := ">
step
StreamWriter("readOperator: value=" + data.value)
data.name := "operator"
data.dataType := "operator"
return true
elseif isOperator(operatorString, "/", data) then
return true
elseif isOperator(operatorString, "+", data) then
return true
elseif isOperator(operatorString, "-", data) then
return true
elseif isOperator(operatorString, "\"", data) then
return true
elseif isOperator(operatorString, "mod", data) then
return true
else
writeLine("NO Operator String")
return false

// Add * + - div mod
isOperator(primitive ref operatorString as String, op as String, primitive
ref data as DATA) as Boolean
if operatorString.TrimStart(null).StartsWith(op) then
operatorString := operatorString.TrimStart(null).Substring(Size(op))
step
data.value := op
step
writeLine("isOperator: value=" + data.value)
data.name := "operator"
data.dataType := "operator"
return true
else
return false

override ToString() as String
return "condition: " + condition + " activity: " +
swCaseActivity.ToString() + " priority: " + swPriority

102
Copy

class COPY
    var fromType as EnumAssignType
    var fromString1 as String
    var fromString2 as String
    var fromString3 as String
    var toType as EnumAssignType
    var toString1 as String
    var toString2 as String
    var toString3 as String

    ToString() as String
        return "fromType: " + fromType + ", fromString1 : " + fromString1 + ", fromString2 : " + fromString2 + ", fromString3 : " + fromString3 + ", toType: " + toType + ", toString1 : " + toString1 + ", toString2 : " + toString2 + ", toString3 : " + toString3
APPENDIX B. THE NETWORK INTERFACE MODEL

Name Space

namespace NetIF
import ModelGUI
import NAM
import System.String

Global Definitions

const DELIMITER = ";"
const DATA_DELIMITER = "/"

External

public class MODEL
    public run()

Agent

public class AGENT
    public virtual program()

BPEL Network Interface

public class BPEL_NETWORK_INTERFACE extends NetIF.AGENT
    public var namForm as NamForm
    public var bpel as ModelGUI.MODEL
    var address as Integer = -1
    var addressTable as ADDRESS_TABLE = new ADDRESS_TABLE()
    var oManager as OUT_MANAGER = undef
    var inManager as IN_MANAGER = undef

    public BPEL_NETWORK_INTERFACE(theNetwork as NamForm, theBpel as ModelGUI.MODEL, theAddress as Integer)
        step
            namForm := theNetwork
            bpel := theBpel
            address := theAddress
        step
            oManager := new OUT_MANAGER(me, bpel)
            inManager := new IN_MANAGER(me)
public BPEL_NETWORK_INTERFACE(thenetwork as NamForm, theBpel as
ModelGUI.MODEL, theAddress as Integer, theTable as ADDRESS_TABLE)

initially bpel = theBpel

step
  namForm := thenetwork
  address := theAddress
  addressTable := theTable

step
  oManager := new OUT_MANAGER(me, bpel)
  inManager := new IN_MANAGER(me)

public getAddress() as Integer
  return address

public updateAddress(portType as String, address as Integer)
  addressTable.update(portType, address)

override public program()
  oManager.program()
  inManager.program()
  bpel.run()

[EntryPoint]

public run()
  program()
WriteLine("createStringMsg: message=") + m)
WriteLine("createStringMsg: msg=") + msg)

step foreach i in Indices (m.parts)
step
  msg := msg + DELIMITER + m.parts(i).getPartName() + DELIMITER + m.parts(i).getPartType() + DELIMITER
  WriteLine("createStringMsg: msg=") + msg)
step foreach (d as ModelGU1.DATA) in m.parts(i).getDataset()
  msg := msg + (d as ModelGU1.DATA).getDataName() + DATA_DELIMITER + d.getDataType() + DATA_DELIMITER + d.value + DATA_DELIMITER
step
  msg := msg + DELIMITER
step
  WriteLine("createStringMsg: ") + msg)
return msg

public send(m as MESSAGE) as Boolean
var address as Integer
var receivers as Set of Integer = {}
var msg as String

step
  msg := createStringMsg(m)
step
  address := parent.addressTable.getAddress(m.getPortTypeName())
step
  WriteLine("send: Receiver's portType=") + m.getPortTypeName())
  WriteLine("send: Receiver's address=") + address)
  WriteLine("send: M: ") + msg)
  add address to receivers
step
  return insertMsg(receivers, msg)

public send(portTypes as Set of String, m as MESSAGE) as Boolean

var address as Integer
var receivers as Set of Integer = {}
var msg as String

step
  msg := createStringMsg(m)
step foreach portType in portTypes
step
  address := parent.addressTable.getAddress(portType)
step
  add address to receivers

return insertMsg(receivers, msg)

private insertMsg(receivers as Set of Integer, msg as String) as Boolean
var m as NAM.Message

step
  m := new NAM.Message(parent.getAddress(), msg)
step
  writeLine("Sender: " + parent.getAddress())

step foreach receiver in receivers
  step
    m.addReceiver(receiver)

return true

Incoming Message Manager
Incoming Message Manager is responsible to deliver incoming messages from
the Network to a BPEL endpoint.

public class IN_MANAGER extends NetIF.AGENT
  var parent as BPEL_NETWORK_INTERFACE

override program()
  var msgs as Set of NAM.Message = {}
  var m as MESSAGE

  step
    writeLine("NETIF IN_MANAGER")
    writeLine("My address: " + parent.getAddress())
  step
    msgs := Model.getMsgs(parent.getAddress())
  step
    if msgs <> {} then
      step
        writeLine("After getMsg ")
      step foreach msg in msgs where msg.getSender() <> parent.getAddress()
        step
          writeLine("IN_MANAGER: before convertMessage")
          m := convertMessage(msg)
        step
          writeLine("IN_MANAGER: before addMessage")
        step
          parent.bpel.addMessage(m)
    else
      writeLine("IN_MANAGER:No Messages at address:" + parent.getAddress())

convertMessage(msg as NAM.Message) as MESSAGE
  var msgTypePos as Integer
  var stringMsg as String
  var stringSender as String
  var m as MESSAGE
  var partStartPos as Integer
  var partEndPos as Integer
  var dataNameStartPos as Integer
var dataValueStartPos as Integer
var dataTypeStartPos as Integer
var partTypeStartPos as Integer
var partName as String
var dataName as String
var dataType as String
var value as String
var data as ModelGUI.DATA
var partType as String

step
    stringMsg := msg.data
step
    writeln("convertMessage: stringMsg=" + stringMsg)
    msgTypePos := stringMsg.IndexOf(DELIMITER) + 1
step
    writeln("convertMessage: msgTypePos=" + msgTypePos)
    partStartPos := stringMsg.IndexOf(DELIMITER, msgTypePos) + 1
    stringSender := stringMsg.Substring(0, msgTypePos - 1)
step
    writeln("convertMessage: partStartPos=" + partStartPos)
    writeln("convertMessage: stringSender=" + stringSender)
    writeln("convertMessage: messageType=" + stringMsg.Substring(msgTypePos, partStartPos - msgTypePos - 1))
    m := new MESSAGE(stringMsg.Substring(msgTypePos, partStartPos - msgTypePos - 1))
step
    writeln("addressTable: " + parent.addressTable.ToString())

    m.setPortTypeName(parent.addressTable.getPortType(ToInteger(stringSender)))
    writeln("convertMessage: portType=" + parent.addressTable.getPortType(ToInteger(stringSender)))
    writeln("convertMessage: dataNameStartPos=" + stringMsg.IndexOf(DELIMITER, partStartPos))
step while(partStartPos < stringMsg.Length)
    step
        partTypeStartPos := stringMsg.IndexOf(DELIMITER, partStartPos) + 1
        //since \; or end of messageType for the first time
        step
            writeln("convertMessage: partStartPos=" + partStartPos)
            partName := stringMsg.Substring(partStartPos, partTypeStartPos - partStartPos - 1)
            dataNameStartPos := stringMsg.IndexOf(DELIMITER, partTypeStartPos) + 1
        step
            writeln("convertMessage: dataNameStartPos=" + dataNameStartPos)
            partType := stringMsg.Substring(partTypeStartPos, dataNameStartPos - partTypeStartPos - 1)
    step
        partEndPos := stringMsg.IndexOf(DELIMITER, dataNameStartPos)
        writeln("convertMessage: part=" + partName)
        m.parts(partName) := new ModelGUI.PART(partName)
step
    m.parts(partName).setPartType(partType)
step while (dataNameStartPos < partEndPos)
step
dataTypeStartPos := stringMsg.IndexOf(DATA_DELIMITER, dataNameStartPos) + 1
step
dataValueStartPos := stringMsg.IndexOf(DATA_DELIMITER, dataTypeStartPos) + 1
step
dataType := stringMsg.Substring(dataTypeStartPos, dataTypeStartPos - 1)
dataName := stringMsg.Substring(dataNameStartPos, dataTypeStartPos - 1)
dataNameStartPos := stringMsg.IndexOf(DATA_DELIMITER, dataValueStartPos) + 1
step
value := stringMsg.Substring(dataValueStartPos, dataNameStartPos - dataValueStartPos - 1)
step
data := new ModelGUI.DATA(value)
step
data.setDataType(dataType)
step
writeLine("convertMessage: dataName=" + data.getDataName())
writeLine("convertMessage: dataType=" + data.getDataType())
writeLine("convertMessage: dataValue=" + data.value)
m.parts(partName).addData(data)
step
partStartPos := partEndPos + 1
step
return m

---

**Address Table**

Address Table maps PortTypes to IP addresses

```
public class ADDRESS_TABLE

var portTypeMap as Map of String to Integer = {} //portType to address

ADDRESS_TABLE(thePortTypeMap as Map of String to Integer)
initially portTypeMap = thePortTypeMap

ADDRESS_TABLE()

public getAddress(portType as String) as Integer
return portTypeMap(portType)

public getPortType(address as Integer) as String
choose a in Indices(portTypeMap) where portTypeMap(a) = address
return a

public update(portType as String, address as Integer)
```
step
    portTypeMap(portType) := address

public delete(portType as String)
    remove portTypeMap(portType)

public clearAll()
    step
        portTypeMap := {->}

public ToString() as String
    var s as String = ""
    step foreach a in Indices(portTypeMap)
    step
        s := s + portTypeMap(a)
    return s
APPENDIX C. THE SERVICE DESCRIPTION FOR THE CREDIT CARD EVALUATION DIVISION

<?xml version="1.0" encoding="utf-8" ?>

<definitions
  targetNamespace="http://credit.org/wxsd/card-service"
  xmlns="http://schemas.xmlsoap.org/wsd1/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:cnr="http://credit.org/wsd1/card-service">

<message name="applicationInfoMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="firstName" type="xsd:string"/>
  <part name="lastName" type="xsd:string"/>
  <part name="gender" type="xsd:string"/>
  <part name="birthDate" type="xsd:date"/>
  <part name="phoneNumber" type="xsd:string"/>
  <part name="address" type="xsd:string"/>
  <part name="income" type="xsd:int"/>
</message>

<message name="creditReportMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="goodCreditStatus" type="xsd:boolean"/>
</message>

<message name="requestCardMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="firstName" type="xsd:string"/>
  <part name="lastName" type="xsd:string"/>
  <part name="address" type="xsd:string"/>
  <part name="amountLimit" type="xsd:int"/>
</message>

<message name="issueCardMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="issueDate" type="xsd:date"/>
</message>

<message name="resultMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="result" type="xsd:string"/>
  <part name="amountLimit" type="xsd:int"/>
</message>

</definitions>
<portType name="cardServicePT">
  <operation name="request">
    <input message="cns:applicationInfoMessage"/>
    <output message="cns:resultMessage"/>
    <fault name="unableToHandleRequest" message="cns:errorMessage"/>
  </operation>
</portType>

<portType name="creditReportPT">
  <operation name="report">
    <input message="cns:applicationInfoMessage"/>
    <output message="cns:creditReportMessage"/>
    <fault name="creditCardProcessFault" message="cns:errorMessage"/>
  </operation>
</portType>

<portType name="cardIssuingPT">
  <operation name="issue">
    <input message="cns:requestCardMessage"/>
    <output message="cns:issueCardMessage"/>
    <fault name="creditCardProcessFault" message="cns:errorMessage"/>
  </operation>
</portType>

<bpws:property name="applicationNo" type="xsd:string"/>

<bpws:propertyAlias propertyName="cns:applicationNo" messageType="cns:applicationInfoMessage" part="applicationNo"/>

<bpws:propertyAlias propertyName="cns:applicationNo" messageType="cns:reportMessage" part="applicationNo"/>

<bpws:propertyAlias propertyName="cns:applicationNo" messageType="cns:requestCardMessage" part="applicationNo"/>

<bpws:propertyAlias propertyName="cns:applicationNo" messageType="cns:issueCardMessage" part="applicationNo"/>

<bpws:propertyAlias propertyName="cns:applicationNo" messageType="cns:resultMessage" part="applicationNo"/>

<pnk:partnerLinkType name="cardServiceLinkType">
  <pnk:role name="cardService">
    <pnk:portType name="cns:cardServicePT"/>
  </pnk:role>
</pnk:partnerLinkType>
</plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="creditReportLinkType">
  <plnk:role name="assessor">
    <plnk:portType name="cns:creditReportPT"/>
  </plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="cardIssuingLinkType">
  <plnk:role name="issuer">
    <plnk:portType name="cns:cardIssuingPT"/>
  </plnk:role>
</plnk:partnerLinkType>

</definitions>
APPENDIX D. THE PROCESS DEFINITION FOR THE CREDIT CARD EVALUATION DIVISION

<app>

<partnerLinks>
  <partnerLink name="applicationAgent"
    partnerLinkType="cns:cardServiceLinkType"
    partnerRole="applicationAgent"/>
  <partnerLink name="assessor"
    partnerLinkType="cns:creditReportLinkType"
    partnerRole="assessor"/>
  <partnerLink name="issuer"
    partnerLinkType="cns:cardIssuingLinkType"
    partnerRole="issuer"/>
</partnerLinks>

<variables>
  <variable name="request"
    messageType="cns:applicationInfoMessage"/>
  <variable name="inquiry"
    messageType="cns:applicationInfoMessage"/>
  <variable name="report"
    messageType="cns:creditReportMessage"/>
  <variable name="requestCard"
    messageType="cns:requestCardMessage"/>
  <variable name="issueCard"
    messageType="cns:issueCardMessage"/>
  <variable name="result"
    messageType="cns:resultMessage"/>
  <variable name="error"
    messageType="cns:errorMessage"/>
</variables>

<correlationSets>
  <correlationSet name="applicationId"
    properties="cns:applicationNo"/>
</correlationSets>

<sequence>
  <receive partnerLink="applicationAgent"
    portType="cns:cardServicePt"
operation="request"
variable="request" createInstance="yes">
<correlations>
  <correlation set="applicationId"
    initiate="yes"/>
</correlations>
</receive>

<assign>
  <copy>
    <from variable="request"/>
    <to variable="inquiry"/>
  </copy>
</assign>

<invoke partnerLink="assessor"
portType="cns:creditReportPT"
operation="report"
inputVariable="inquiry"
outputVariable="report">
<correlations>
  <correlation set="applicationId"
    initiate="no" pattern="in-out"/>
</correlations>
</invoke>

<switch>
  <case condition="bpws:getVariableData('report', 'goodCreditStatus') = true and bpws:getVariableData('inquiry', 'income') &gt;= 10000">
    <sequence>
      <assign>
        <copy>
          <from variable="inquiry" part="firstName"/>
          <to variable="requestCard" part="firstName"/>
        </copy>
        <copy>
          <from variable="inquiry" part="lastName"/>
          <to variable="requestCard" part="lastName"/>
        </copy>
        <copy>
          <from variable="inquiry" part="applicationNo"/>
          <to variable="requestCard" part="applicationNo"/>
        </copy>
        <copy>
          <from variable="inquiry" part="address"/>
          <to variable="requestCard" part="address"/>
        </copy>
      </assign>
      <switch>
        <case condition="bpws:getVariableData('inquiry', 'income') &lt; 30000">
        <assign>
          <copy>
            <from expression="2000"/>
          </copy>
        </assign>
        </case>
      </switch>
    </sequence>
  </case>
</switch>
<to variable="requestCard" part="amountLimit"/>
</copy>
</assign>
</case>
<case condition="bpws:getPageData('income') &gt; 30000 and bpws:getPageData('income') &lt; 60000">
<assign>
<copy>
<from expression="4000"/>
<to variable="requestCard" part="amountLimit"/>
</copy>
</assign>
</case>
<case condition="bpws:getPageData('income') &gt; 60000">
<assign>
<copy>
<from expression="6000"/>
<to variable="requestCard" part="amountLimit"/>
</copy>
</assign>
</case>
</switch>
<invoke partnerLink="issuer"
portType="cns:cardIssuingPT"
operation="issue"
inputVariable="requestCard"
outputVariable="issueCard">
<correlations>
<correlation set="applicationId"
initiate="no" pattern="in-out"/>
</correlations>
</invoke>
<assign>
<copy>
<from variable="requestCard" part="applicationNo"/>
<to variable="result" part="applicationNo"/>
</copy>
</assign>
<copy>
<from variable="requestCard" part="amountLimit"/>
<to variable="result" part="amountLimit"/>
</copy>
<copy>
<from"approved"</from>
<to variable="result" part="result"/>
</copy>
</assign>
<reply partnerLink="applicationAgent"
portType="cns:cardServicePT"
operation="request"
variable="result">
<correlations>
<correlation set="applicationId"
initiate="no"/>
</correlations>
</reply>
</sequence>
</case>
<otherwise>
<sequence>
<assign>
<copy>
[from variable="inquiry" part="applicationNo"/>
[to variable="result" part="applicationNo"/>
</copy>
<copy>
[from="rejected"/></copy>
[to variable="result" part="result"/>
</copy>
</assign>
<reply partnerLink="applicationAgent"
portType="cns:cardServicePT"
operation="request"
variable="result">
<correlations>
<correlation set="applicationId"
initiate="no"/>
</correlations>
</reply>
</sequence>
</otherwise>
</switch>
</sequence>
</process>
APPENDIX E. THE SERVICE DESCRIPTION FOR THE CREDIT CARD APPLICATION HANDLING DIVISION

```xml
<?xml version="1.0" encoding="utf-8" ?>
<definitions

targetNamespace="http://credit.org/wsd1/card-application"
xmlns="http://schemas.xmlsoap.org/wsdl/
xmlns:xsd=http://www.w3.org/2001/XMLSchema"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/
xmlns:cns="http://credit.org/wsd1/card-application">

<message name="applicationInfoMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="firstName" type="xsd:string"/>
  <part name="lastName" type="xsd:string"/>
  <part name="gender" type="xsd:string"/>
  <part name="birthdate" type="xsd:date"/>
  <part name="phoneNumber" type="xsd:string"/>
  <part name="address" type="xsd:string"/>
  <part name="income" type="xsd:int"/>
</message>

<message name="resultMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="result" type="xsd:string"/>
  <part name="amountLimit" type="xsd:int"/>
</message>

<portType name="cardApplicationPT">
  <operation name="apply">
    <input message="cns:applicationInfoMessage"/>
  </operation>
</portType>

<portType name="cardServicePT">
  <operation name="request">
    <input message="cns:applicationInfoMessage"/>
    <output message="cns:resultMessage"/>
    <fault name="unableToHandleRequest" message="cns:errorMessage"/>
  </operation>
</portType>

<bpws:property name="applicationNo" type="xsd:string"/>
```

118
<bpws:propertyAlias propertyName="cns:applicationNo"
    messageType="cns:applicationInfoMessage"
    part="applicationNo"/>

<bpws:propertyAlias propertyName="cns:applicationNo"
    messageType="cns:resultMessage"
    part="applicationNo"/>

<plnk:partnerLinkType name="cardServiceLinkType">
    <plnk:role name="cardService">
        <plnk:portType name="cns:cardServicePT"/>
    </plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="cardApplicationLinkType">
    <plnk:role name="applicationservice">
        <plnk:portType name="cns:cardApplicationPT"/>
    </plnk:role>
</plnk:partnerLinkType>

</definitions>
APPENDIX F. THE PROCESS DEFINITION FOR THE CREDIT CARD APPLICATION HANDLING DIVISION

```xml
<?xml version="1.0" encoding="utf-8" ?>

<process name="creditCardApplication"
    targetNamespace="http://acme.com/creditCardApplication"
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/
    xmlns:ns="http://credit.org/wsd1/card-application">

    <partnerLinks>
        <partnerLink name="cardApplication"
            partnerLinkType="ns:cardApplicationLinkType"
            myRole="applicationService"/>
        <partnerLink name="cardService"
            partnerLinkType="ns:cardServiceLinkType"
            partnerRole="cardService"/>
    </partnerLinks>

    <variables>
        <variable name="apply"
            messageType="ns:applicationInfoMessage"/>
        <variable name="result"
            messageType="ns:resultMessage"/>
        <variable name="error"
            messageType="ns:errorMessage"/>
    </variables>

    <correlationSets>
        <correlationSet name="applicationId"
            properties="ns:applicationNO"/>
    </correlationSets>

    <sequence>
        <receive partnerLink="cardApplication"
            portType="ns:cardApplicationPT"
            operation="apply"
            variable="apply" createInstance="yes">
            <correlations>
                <correlation set="applicationId"
                    initiate="yes"/>
            </correlations>
        </receive>

        <invoke partnerLink="cardService"
            portType="ns:cardServicePT"
            operation="request"
```
<process>
  <invoke>
    <inputVariable>apply</inputVariable>
    <outputVariable>result</outputVariable>
    <createInstance>no</createInstance>
    <correlations>
      <correlation set="applicationId"
        initiate="no" pattern="in-out"/>
    </correlations>
  </invoke>
</sequence>
</process>
APPENDIX G. THE SERVICE DESCRIPTION FOR THE CREDIT REPORTING BUREAU

```xml
<definitions
  targetNamespace="http://credit.org/wsdl/credit-report"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:plink="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:cns="http://credit.org/wsdl/credit-report">
  <message name="applicationInfoMessage">
    <part name="applicationNo" type="xsd:string"/>
    <part name="firstName" type="xsd:string"/>
    <part name="lastName" type="xsd:string"/>
    <part name="gender" type="xsd:string"/>
    <part name="birthDate" type="xsd:date"/>
    <part name="phoneNumber" type="xsd:string"/>
    <part name="address" type="xsd:string"/>
    <part name="income" type="xsd:int"/>
  </message>
  <message name="creditReportMessage">
    <part name="applicationNo" type="xsd:string"/>
    <part name="goodCreditStatus" type="xsd:boolean"/>
  </message>
  <portType name="creditReportPT">
    <operation name="report">
      <input message="cns:applicationInfoMessage"/>
      <output message="cns:creditReportMessage"/>
      <fault name="creditCardProcessFault"
        message="cns:errorMessage"/>
    </operation>
  </portType>
  <bpws:property name="applicationNo" type="xsd:string"/>
  <bpws:propertyAlias propertyName="cns:applicationNo" message="cns:applicationInfoMessage" part="applicationNo"/>
  <bpws:propertyAlias propertyName="cns:applicationNo" message="cns:creditReportMessage"/>
</definitions>
```
<plnk:partnerLinkType name="creditReportLinkType">
  <plnk:role name="assessor">
    <plnk:portType name="cns:creditRepotPT"/>
  </plnk:role>
</plnk:partnerLinkType>

</definitions>
APPENDIX H. THE PROCESS DEFINITION FOR THE CREDIT REPORTING BUREAU

<?xml version="1.0" encoding="utf-8" ?>

<process name="creditCardReport"
  targetNamespace="http://acme.com/creditCardReport"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/
  xmlns:cns="http://credit.org/wsdl/credit-report">
  <partnerLinks>
    <partnerLink name="assessor"
      partnerLinkType="cns:creditReportLinkType"
      myRole="assessor"/>
  </partnerLinks>
  <variables>
    <variable name="inquiry" messageType="cns:applicationInfoMessage"/>
    <variable name="report" messageType="cns:creditReportMessage"/>
    <variable name="error" messageType="cns:errorMessage"/>
  </variables>
  <correlationSets>
    <correlationSet name="applicationId"
      properties="cns:applicationNo"/>
  </correlationSets>
  <sequence>
    <receive partnerLink="assessor"
      portType="cns:creditReportPT"
      operation="report"
      variable="inquiry" createInstance="yes">
      <correlations>
        <correlation set="applicationId"
          initiate="yes"/>
      </correlations>
    </receive>
    <!-- randomly assign the value of report -->
    <assign>
      <copy>
        <from variable="inquiry" part="applicationNo"/>
      </copy>
    </assign>
  </sequence>
</process>
<to variable="report" part="applicationNo"/>
</copy>
<copy>
  <from opaque="yes"/>
  <to variable="report" part="goodCreditStatus"/>
</copy>
</assign>

<reply partnerLink="assessor"
  portType="cns:creditReportPT"
  operation="report"
  variable="report">
  <correlations>
    <correlation set="applicationId"
      initiate="no"/>
  </correlations>
</reply>

</sequence>
</process>
APPENDIX I. THE SERVICE DESCRIPTION FOR THE CREDIT CARD ISSUING DIVISION

<?xml version="1.0" encoding="utf-8" ?>

<definitions

targetNamespace="http://credit.org/wsd1/card-issue"
xmlns="http://schemas.xmlsoap.org/wsd1/"
xmns:xsd="http://www.w3.org/2001/XMLSchema"
xmns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmns:cn="http://credit.org/wsd1/card-issue">

<message name="applicationInfoMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="firstName" type="xsd:string"/>
  <part name="lastName" type="xsd:string"/>
  <part name="gender" type="xsd:string"/>
  <part name="birthDate" type="xsd:date"/>
  <part name="phoneNumber" type="xsd:string"/>
  <part name="address" type="xsd:string"/>
  <part name="income" type="xsd:int"/>
</message>

<message name="requestCardMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="firstName" type="xsd:string"/>
  <part name="lastName" type="xsd:string"/>
  <part name="address" type="xsd:string"/>
  <part name="amountLimit" type="xsd:int"/>
</message>

<message name="issueCardMessage">
  <part name="applicationNo" type="xsd:string"/>
  <part name="issueDate" type="xsd:date"/>
</message>

<portType name="cardIssuingPT">
  <operation name="issue">
    <input message="cn:requestCardMessage"/>
    <output message="cn:issueCardMessage"/>
    <fault name="creditCardProcessFault" message="cn:errorMessage"/>
  </operation>
</portType>

<bpws:property name="applicationNo"
type="xsd:string"/>

<bpws:propertyAlias propertyName="cns:applicationNo"
   messageType="cns:requestCardMessage"
   part="applicationNo"/>

<bpws:propertyAlias propertyName="cns:applicationNo"
   messageType="cns:issueCardMessage"
   part="applicationNo"/>

<plnk:partnerLinkType name="cardIssuingLinkType">
   <plnk:role name="issuer">
      <plnk:portType name="cns:cardIssuingPT"/>
   </plnk:role>
</plnk:partnerLinkType>

</definitions>
APPENDIX J. THE PROCESS DEFINITION FOR THE CREDIT CARD ISSUING DIVISION

```xml
<?xml version="1.0" encoding="utf-8" ?>

<process name="creditCardIssue"
   targetNamespace="http://acme.com/creditCardIssue"
   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/
   xmlns:cnS="http://credit.org/wSDL/card-issue">

  <partnerLinks>
    <partnerLink name="issuer"
      partnerLinkType="cnS:cardIssuingLinkType"
      myRole="issuer"/>
  </partnerLinks>

  <variables>
    <variable name="requestCard"
      messageType="cnS:requestCardMessage"/>
    <variable name="issueCard"
      messageType="cnS:issueCardMessage"/>
    <variable name="error"
      messageType="cnS:errorMessage"/>
  </variables>

  <correlationSets>
    <correlationSet name="applicationId"
      properties="cnS:applicationNo"/>
  </correlationSets>

  <sequence>
    <receive partnerLink="issuer"
      portType="cnS:cardIssuingPt"
      operation="issue"
      variable="requestCard" createInstance="yes">
      <!-- randomly assign the value of report -->
      <assign>
        <copy>
```
<from variable="requestCard" part="applicationNo"/>
<to variable="issueCard" part="applicationNo"/>
</copy>
<copy>
    <from expression="today()"/>
    <to variable="issueCard" part="issueDate"/>
</copy>
</assign>

<reply partnerLink="issuer"
    portType="cns:cardIssuingPT"
    operation="issue"
    variable="issueCard">
    <correlations>
        <correlation set="applicationId"
            initiate="no"/>
    </correlations>
</reply>
</sequence>
</process>
BIBLIOGRAPHY


