

MASTER

BPMN 2 BPEL

research on mapping BPMN to BPEL

Blox, J.

Award date:
2009

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Industrial Engineering

BPMN 2 BPEL
research on mapping BPMN to BPEL

By
Jeroen Blox

Supervisors:

R. Eshuis (TU/e)
L. Mühlberg (Logica)
J.C. van Gaalen (Logica)

Eindhoven, March 2009

Preface

This thesis is the final product of the graduation project I conducted at Logica Nederland, as the final part of the Business Information Systems master program at Eindhoven University of Technology. During my graduation period I was positioned within the Working Tomorrow program, which is a program enabling graduates to perform their graduation project on challenging and innovative topics. The theoretical basis created during my study, could be combined with the practical knowledge about the topic available at Logica.

In this preface, I would like to grasp the opportunity to express thanks to a number of people. First of all, I would like to thank my supervisor from Eindhoven University of Technology for his professional and critical approach in assisting me during my master project. Our discussions were of great value for both the research and this thesis. I also would like to thank Lambert Mühlenberg and Jan-Cees van Gaalen for their support in guiding me in the entire process, providing me with helpfull feedback and help me in defining two case studies.

Of course, I also want to thank my colleague students at Logica for their help in my project, but also for the pleasant time we had during and after office hours. Finally, I would like to express my gratitude to my girlfriend Noor, my family and friends for supporting me during my whole master project and master program.

Jeroen Blox

Eindhoven, March 20, 2009

Abstract

This thesis is the end result of a master project about automatic translation from BPMN to BPEL. Both BPMN and BPEL are relatively new languages which are still under development. In the last few years, a lot of research has been done in the field of mapping BPMN to BPEL. Different strategies and implementations have arisen, each with their own strengths and weaknesses. Three main categories can be stated: structured-based, link-based and event-action based strategies. Combining these three main categories results in implementations with a high level of readability and completeness. Existing implementations try to be complete and therefore combine strategies from the three categories, but often lack readability because of conflicting criteria.

We present a new approach for mapping BPMN to BPEL with both the readability and completeness criteria in mind, but focus on the readability aspect. The approach is based on an existing composition algorithm proposed by Eshuis [EG09]. In this approach, BPMN models are analyzed and synchronization dependencies are temporarily eliminated from the process. The remaining model is preprocessed for finding structural properties and different concepts like immediate dominator, loop header and follow sets are calculated. Together with the structural properties, the synchronization links are processed by an algorithm which creates a structured composition. Such a structured composition represents a BPEL model and can be expressed in the official BPEL syntax.

Our approach is implemented in an Eclipse plugin and we evaluated our approach by doing two case studies and comparing the results with other implementations. We can conclude that our algorithm for finding synchronization dependencies and the level of structure our generated BPEL models has, are contributing to current strategies and implementations. Other implementations translate unstructured components with the link-based or event-action based strategies, while our approach eliminates synchronization dependencies and uses the structured-based strategy. Different researchers agree that structured-based translation results in more readable BPEL code.

Unstructured loops cannot be translated by our approach, while implementations like the BABEL-tool and the Eclipse plugin can handle these unstructured

loops. Because our approach requires a well-formed business process diagram as input model, we evaluated these well-formedness restrictions and conclude that some of these restrictions can be relaxed since the preprocessing step of the approach inserts dummy nodes when needed.

Also some ideas for future work are presented. The field of mapping BPMN to BPEL is changing rapidly and more implementations arise. We are convinced that future work will narrow the gap between the design of business processes and the implementation of these processes.

Contents

Preface	i
Abstract	iii
1 Introduction	1
1.1 Problem statement	1
1.2 Approach	2
1.3 Outline	3
2 State-of-the-art: BPMN and BPEL	5
2.1 BPMN analyzed	5
2.1.1 BPMN explained	5
2.1.2 Evaluation of BPMN	7
2.2 BPEL analyzed	8
2.2.1 BPEL explained	8
2.2.2 Evaluation of BPEL	10
2.3 Research on BPMN to BPEL	12
2.3.1 Differences between BPMN and BPEL	12
2.3.2 Current research on mapping BPMN to BPEL	14
2.3.3 Current implementations	18
2.4 Conclusions	18
3 Mapping BPMN to BPEL	21
3.1 Business Process Diagram	22
3.2 Synchronization dependencies	24
3.2.1 Finding synchronization dependencies	25
3.2.2 The complaint handling process	29
3.3 Dominators, loop headers and follow sets	30
3.4 BPEL grammar	31
3.5 The algorithm	32
3.5.1 Adjustment of the algorithm	32
3.5.2 The complaint handling process	34
3.6 BPEL syntax	35
4 An Eclipse plugin	39

4.1	Creating BPMN models in Eclipse	39
4.2	Running the algorithm	39
4.3	Plugin evaluation	41
5	Case studies	45
5.1	Loan request for an insurance company	45
5.1.1	Manual translation	47
5.1.2	Translating via the plugin	48
5.2	Total credit risk calculation	51
5.2.1	Manual translation	53
5.2.2	Translating via the plugin	54
5.3	Conclusion	57
6	Evaluation of the mapping	59
6.1	Strengths & Weaknesses	59
6.2	Relation with other strategies/implementations	60
6.2.1	Case study 1	62
6.2.2	Case study 2	63
6.2.3	Conclusion	64
6.3	Evaluation of well-formed BPD restrictions	65
6.4	Future work	69
7	Conclusions	71
7.1	Answers to the research questions	71
7.1.1	What is the state-of-the-art with respect to BPMN and BPEL?	71
7.1.2	How can BPMN be mapped to BPEL?	72
7.1.3	How can our translation be evaluated compared to existing translations?	72
7.2	Threats to validity	73
7.3	Future work	74
A	BPMN explained	A-1
B	BPEL explained	A-5
C	Existing BPMN 2 BPEL approaches	A-9
D	Business Process Diagram	A-11
E	Input model for other implementations	A-13

List of Figures

2.1	The basic elements used in BPMN[WAD ⁺ 06b]	6
2.2	A complaint handling process, adapted from [ODHA08]	7
2.3	The complaint handling process in BPEL syntax	9
2.4	Transformation strategies by Mendling et al.[MLZ06]	15
2.5	The same model decomposed twice in a non-deterministic way [VVK08]	16
2.6	Translation of structured BPMN components to BPEL [OADH06]	17
3.1	BPD elements displayed in a class diagram	22
3.2	An incorrect synchronization link, which causes a deadlock	24
3.3	Two parallel constructs in a sequence; SESE-regions are necessary.	26
3.4	Algorithm for finding synchronization dependencies	28
3.5	A complaint handling process, adapted from [ODHA08]	29
3.6	Algorithm for constructing structured compositions	33
3.7	Inserting unique <i>FOLLOW</i> node after <i>current</i> (l. 31 of Fig. 3.6)	34
3.8	Adding synchronization dependencies to the block structure	34
3.9	The structured composition for the complaint handling process	37
4.1	The BPMN modeling tool in Eclipse	40
4.2	Graphical flowchart how the plugin works	40
4.3	Example of an error message of Oracle's BPEL Process Manager	42
4.4	Screenshot of the execution environment	43
5.1	Case study 1: Loan request for an insurance company	46
5.2	Structured composition of case study 1	49
5.3	Part of the BPEL syntax for case study 1	49
5.4	Structured composition tree of case study 1	50
5.5	Case study 2: total credit risk calculation	52
5.6	Structured composition of case study 2	55
5.7	Part of the BPEL syntax for case study 2	55
5.8	Structured composition tree of case study 2	56
6.1	Case study 1 translated with the Eclipse implementation [GB08]	62
6.2	Case study 1 translated with the BABEL-tool [OADH08]	63
6.3	Case study 1 translated with the Intalio-tool	63
6.4	Case study 2 translated with the Eclipse implementation [GB08]	64

6.5	Case study 2 translated with the Intalio-tool	64
6.6	Start and end events with respectively an out- and indegree of more than one.	67
6.7	Tasks and intermediate events that have an in- or outdegree of more than one.	67
6.8	Gateways with an in- and outdegree of one can be transformed to a task node.	68
A.1	The basic elements used in BPMN[WAD ⁺ 06b]	A-1
E.1	Case study 1 adjusted input model for the Eclipse implementation	A-13
E.2	Case study 1 adjusted input model for the BABEL-tool	A-13
E.3	Case study 2 adjusted input model for the Eclipse implementation	A-14

List of Tables

2.1	Most common BPEL elements.	11
2.2	BPMN and BPEL compared with workflow patterns [WADH05, WADH03]	13
2.3	Evaluation of current approaches	20
3.1	<i>before(g)</i> - and <i>after(g)</i> -set for gateways of our example	29
3.2	Dominators, loop headers and follow sets	35
3.3	Structured composition to BPEL syntax	36
5.1	<i>before(g)</i> - and <i>after(g)</i> -set for gateways of case study 1	47
5.2	Dominators, loop headers and follow sets for case study 1	48
5.3	<i>before(g)</i> - and <i>after(g)</i> -set for gateways of case study 2	53
5.4	Dominators, loop headers and follow sets for case study 2	54
6.1	Evaluation of implementations	65
6.2	Evaluation of well-formed BPD restrictions	66
7.1	Evaluation of current approaches	72
7.2	Evaluation of implementations	73
C.1	Existing approaches for translating BPMN to BPEL	A-9

Chapter 1

Introduction

The business process modeling notation (BPMN) is a graph-oriented language often used by domain analysts [WAD⁺06a] and supported by more than 50 tools [Web08d]. The business process execution language for web services (WS-BPEL or BPEL in short) is a block-structured language, which is more focused on software developers. BPEL is also supported by several execution platforms of international players [Web08c] and therefore is the standard in process-based service description languages. BPMN can be used to design and discuss business processes with both process analysts and software engineers, while BPEL can be used for implementation purposes. A mapping between both languages, BPMN and BPEL, would narrow the gap between design and implementation for both business analysts and software developers.

The thesis at hand is the result of my master project conducted at Logica Nederland B.V. in Eindhoven. Logica is a leading IT and business services company, employing 39,000 people across 36 countries. They enable business transformation for their customers through the innovative use of technology.

In this chapter we will discuss the problem statement and the research questions we answer in this thesis. The approach we followed during the master project is explained and in the outline a description of further chapters is given.

1.1 Problem statement

Recently some research has been done in the field of transforming a BPMN-model to structured BPEL-code [MLZ06, Whi05, OADH08, Web08a, ODHA08, RM06]. Until now, no common agreed method exists and most implementations do not support unstructured BPMN-models. Only structured business processes that meet a list of requirements specified by a tool, can be transformed by these tools. Because implementations which support unstructured business process often result in translations which are not readable by humans

[OADH08, ODHA08, MLZ06], research is necessary on this field. Completeness and readability are two important requirements for a mapping between BPMN and BPEL and will be used frequently in this thesis.

In association with Logica and Eindhoven University of Technology, we want to investigate the state of the art in the field of translating BPMN to BPEL. By analyzing current research in translating BPMN to BPEL and by evaluating the specifications of both languages, the strengths and weaknesses of current approaches can be identified. Next, we want to extend an existing algorithm presented by Rik Eshuis [EG09] to make it suitable for mapping BPMN to BPEL. Currently this existing algorithm is used for creating block structure from dependency graphs according to composition theory. This algorithm will be evaluated against the strengths and weaknesses discovered by analyzing the state-of-the-art. Therefore, three main research question are formulated:

1. What is the state-of-the-art with respect to BPMN and BPEL?
2. How can BPMN be mapped to BPEL using the algorithm of [EG09]?
3. How can our translation be evaluated and compared to existing translations?

1.2 Approach

To answer the first research question, we first explain the concepts of BPMN and BPEL. Then a literature research is conducted for both languages and the strengths and weaknesses of each language are evaluated. Also the possibilities and limitations of each language are analyzed, which is needed for judging current translations. After describing both languages, current research in mapping BPMN to BPEL is analyzed and an overview is made of different strategies and implementations currently available. We summarize the constraints, strengths and weaknesses of different strategies (and implementations) which are used to develop our own algorithm.

The second research question is answered both by literature research and by our own approach we present. An existing algorithm of Rik Eshuis [EG09] is used to investigate the possibilities in mapping BPMN to BPEL. Adjustments on the existing algorithm should result in a new approach which is suitable for translation from BPMN to BPEL. Together with adjusting the algorithm, an implementation in Eclipse is developed and the algorithm is evaluated during the development. In this phase of the master project, we work according to the idea of the spiral model of Barry Boehm [Boe88]. First basic models are mapped correctly by the plugin, before we extend the algorithm with synchronization dependencies and loops.

An algorithm is developed for finding synchronization dependencies in single entry, single exit-regions. The introduction of an algorithm for these synchronization dependencies result in less unstructured components which have to be translated. We also evaluate well-formedness restrictions on the input BPMN model, to discuss the need of these restrictions.

For answering the third research question, we gather two different case studies and use these to determine the performance of an algorithm or implementation. After defining some requirements, our own algorithm and implementation is compared to other existing implementations or strategies described by other researchers.

1.3 Outline

This master thesis is structured to the presented approach. Chapter 2 contains a critical analysis of BPMN and BPEL to answer the first research question. The state-of-the-art with respect to BPMN and BPEL is given, but also a critical analysis on research done in the field of mapping those languages, is conducted.

In chapter 3 our own solution for mapping BPMN to BPEL is presented. The algorithm is based on an existing algorithm of Rik Eshuis. His algorithm is used for creating structured block diagrams with dependency graphs as input model. The algorithm is adjusted for our purpose and extended to trace synchronization dependency relations in concurrency situations and translate them to BPEL control links.

An implementation of our algorithm is discussed in chapter 4. We first describe the possibilities of modeling BPMN in Eclipse and then explain how our algorithm is implemented as an Eclipse plugin. The plugin is evaluated by comparing results with manual translations we have done according to the same algorithm as implemented.

We also present two case studies in chapter 5, which are gathered from some clients of Logica. Using these case studies we evaluate our own algorithm and in chapter 6 we compare these results with results from other implementations and strategies. In this way we are able to identify strengths and weaknesses of our mapping and we can propose possible improvements for mapping BPMN to BPEL. In chapter 6 we also discuss how the well-formedness restrictions which are required by the presented approach can be relaxed.

Chapter 2

State-of-the-art: BPMN and BPEL

In this chapter, both BPMN and BPEL are discussed and critically analyzed to describe the state-of-the-art. Also a brief analysis of current research on mapping BPMN to BPEL is given. Based on the analysis, we develop requirements and restrictions for our own algorithm presented in chapter 3.

2.1 BPMN analyzed

2.1.1 BPMN explained

In May 2004 the first version of the Business Process Modeling Notation was proposed by the Business Process Modeling Initiative (BPMI) to provide a standard visual notation for business processes. The Object Management Group (OMG) approved this first version in February 2006 as a final adopted specification [OMG08b]. The intention of BPMN was to create a notation which could be used by both domain analysts and software developers. Also a mapping between the graph-oriented language and the executable language BPEL should be possible when using BPMN. One of their goals is to eliminate the gap between domain and technical analysts.

BPMN is designed to support the modeling of business processes and doesn't support the modeling of organizational structures, functional breakdowns, data models, strategy and business rules. Within the modeling aspect for business processes, three main types of process models can be defined [OMG08b]:

1. Private (internal) business processes
2. Abstract (public) processes
3. Collaboration (global) processes

Private business processes describe a set of activities inside an organization. These models will not be published in order to keep the process confidential. When different parties want to collaborate and use the different private processes, abstract processes need to be described first. Abstract processes are used to describe that part of a process which is interacting with a third party. Internal activities, which do not have a direct interface functionality, are left outside the scope of these models for simplicity and reasons related to confidentiality. Collaboration processes are abstract processes of different parties modeled within one single model. A collaboration process describes a process of interaction between multiple participants from an external view.

At this moment, a request for proposal is placed for a next version of the BPMN specification. Currently the official released version is BPMN 1.1, while BPMN 2.0 is under development and available as a draft. The vision of BPMN 2.0 is to have one single specification for notation, meta model and interchange format [OMG08a]. Because BPMN 2.0 is still under development, we will use BPMN 1.1 in our further research.

BPMN is built from a standard set of elements to create the Business Process Diagrams (BPD). These elements are basically the control flow elements of BPMN. Swimlanes and artifacts are elements designed for representing data and resource utilization. In Figure 2.1 the basic elements used in BPMN are combined in one graphical overview. This overview is not a complete set of elements, e.g. there are some derivatives of event-elements like timer and message events. Also an XOR-gateway can be modeled with or without a cross, both are correct as long as they are used consistently. An explanation of the basic elements can be found in Appendix A. In Figure 2.2 an example BPD is given of a complaint handling process, which will be explained and used to explain our translation in further sections.

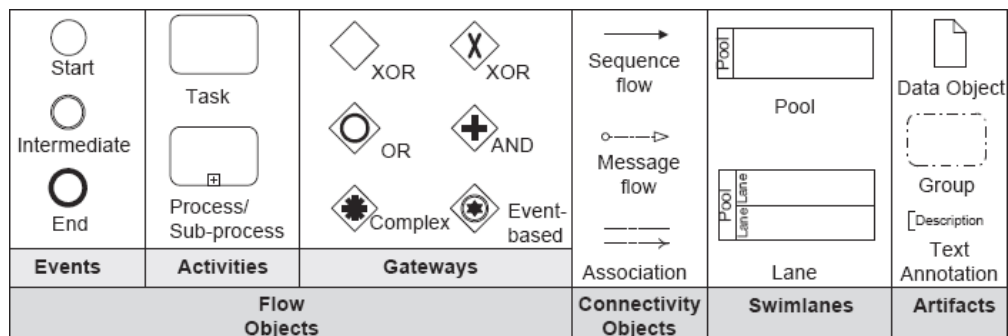


Figure 2.1: The basic elements used in BPMN[WAD⁺06b]

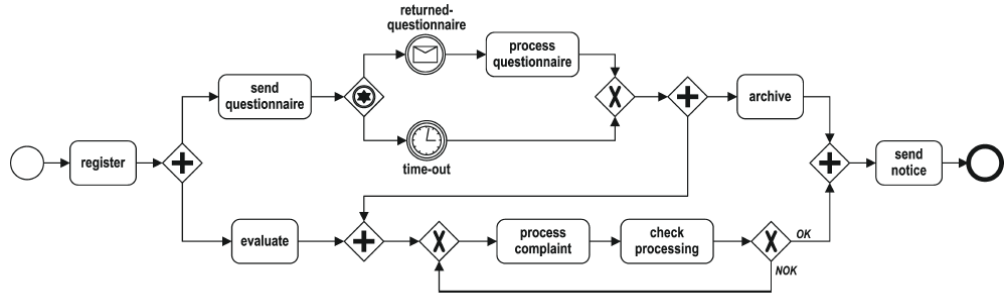


Figure 2.2: A complaint handling process, adapted from [ODHA08]

2.1.2 Evaluation of BPMN

Recently some research has been done in the field of BPMN and several advantages and disadvantages of using BPMN as a modeling language have been identified.

Different researchers identify a standard set of benefits when using BPMN as a modeling language. The most important benefits are stated by Muehlen [MH08], who mentioned the freedom and ease of use to read the BPMN models for both business analysts and technical analysts. Muehlen uses two different case studies to identify these benefits which are often stated in literature.

Also, because BPMN is implemented in more and more tools, more industry reference models will be available. These reference models will help the end user by creating and understanding models expressed in BPMN [MH08].

Ou-Yang et al. [OYL08] and Dijkman et al. [DDO08] are also listing these benefits, but indicate that BPMN models should be checked for feasibility since the freedom in modeling might cause problems. Deadlock situations and livelocks could occur by improper modeling. They suggest a method to check a model for liveness, reachability and consumed time to validate models and improve them. This method is based on Petri nets, which have a high mathematical proof strength. Dijkman et al. [DDO08] emphasize the need for formal analysis on BPMN models, since problems in these models are among the most costly and hardest to correct.

Some other drawbacks can be identified. Wohed et al. [WAD⁺06a] discuss all known control flow patterns in the field of workflow patterns [Web08b] and conclude that not all known patterns are covered by BPMN. Most of the patterns are possible to create in BPMN, but full support is not available. One of the missing patterns is the milestone phenomenon, which ensures that a specific activity can only start if a previous activity is completed. The reason for this missing pattern can be found in the lack of explicit notion of a "state".

Often, business analysts misuse elements of BPMN in order to increase the readability of the BPMN model. Simulations will be false in case of misuse and different BPMN dialects are arising, which is not what the developers of BPMN intended [MH08]. The aspect of resources is thereby completely missing in BPMN, according to the specifications [OMG08b], but the presence of pools and lanes gives the impression of supporting resources.

Because of the freedom to misuse elements in BPMN and the high coverage of workflow patterns, BPMN still has some issues. The benefits which make the language easy to understand and easy to use, are also risks when the specification lacks clearness.

2.2 BPEL analyzed

2.2.1 BPEL explained

OASIS (Organization for the Advancement of Structured Information Standards), the organization behind BPEL, is a consortium that drives development, convergence and adoption of open standards. OASIS has developed several web services standards since their foundation in 1993. Originally the following basic specifications formed the web services space: SOAP, WSDL and UDDI [OAS07]. SOAP is a definition of an XML message protocol for basic service interoperability. WSDL (Web Services Description Language) is a standard for describing services. UDDI (Universal Description, Discovery and Integration), developed by OASIS, basically provides the infrastructure which can publish all services in a systematic way in order to discover web services. With UDDI, web services can invoke other web services discovered via the infrastructure.

These specifications together provide the possibility to interact between web services following a loosely coupled, platform independent model. However system integration requires more than only simple interactions; a specification for describing the communication between services is needed. The specification needs to consider data-dependent behavior, exceptional conditions and their consequences, and long-running interactions [OAS07]. WS-BPEL (Web Services Business Process Execution Language), in short BPEL, describes the full interaction possibilities between web services and the ordering of these web services in a process. In our research, we consider version 2.0 of the BPEL specification.

Within the BPEL specification two different ways of applying the BPEL concepts can be found; the abstract and executable way. Abstract processes are partially specified processes which are not intended to be executed. Executable processes are intended to be executed and therefore need to be fully specified.

```

<process name="complaint handling">
  <sequence name="tc0">
    <invoke name="register">
      <flow name="tc1">
        <sequence name="tc3">
          <invoke name="send questionnaire".../>
          <pick name="tc2">
            <onMessage operation="returned-questionnaire"...>
              <invoke name="process questionnaire".../>
            </onMessage>
            <onAlarm for='P14DT'>
              <empty/>
            </onAlarm>
          </pick>
        </sequence>
        <scope name="tc4">
          <onEvent Start(C4)>
            <invoke end(g1)/>
          </onEvent>
          <onEvent switch(g3,g1,NOK)>
            <invoke end(g1)/>
          </onEvent>
          <onEvent end(g1)>
            <sequence>
              <sequence name="tc1">
                <invoke name="process complaint".../>
                <invoke name="evaluation".../>
              </sequence>
              <invoke end(tc1)/>
            </sequence>
          </onEvent>
          <onEvent end(tc1)>
            <switch name="g2">
              <case condition="DONE">
                <invoke switch(g2,g4,DONE)/>
              </case>
              <case condition="CONT">
                <invoke switch(g2,a8,CONT)/>
              </case>
            </switch>
          </onEvent>
          <onEvent switch(g2,a8,CONT)>
            <sequence>
              <invoke name="check processing".../>
              <invoke end(a8)/>
            </sequence>
          </onEvent>
          <onEvent end(a8)>
            <switch name="g3">
              <case condition="OK">
                <invoke switch(g3,g4,OK)/>
              </case>
              <case condition="NOK">
                <invoke switch(g3,g1,NOK)/>
              </case>
            </switch>
          </onEvent>
          <onEvent switch(g2,g4,DONE)>
            <invoke end(g4)/>
          </onEvent>
          <onEvent switch(g3,g4,OK)>
            <invoke end(g4)/>
          </onEvent>
          <invoke Start(C4)/>
        </scope>
      </flow>
      <invoke name="archive">
    </sequence>
  </process>

```

Figure 2.3: The complaint handling process in BPEL syntax

The available concepts for executable processes are also available for abstract processes, but abstract processes also have the possibility to hide some of the required concrete operational details. Abstract processes have a descriptive role and can give information about an executable process.

In Figure 2.3 the complaint handling example of Figure 2.2 is given in BPEL code. BPEL syntax is structured and uses nesting. Each element can have attributes and other nested child elements. The BPEL specification contains a lot of elements, which can be used for specific purposes. A full list of BPEL elements can be found in [OAS07]. The most common elements can be found in Table 2.1 and are categorized into the following classes; basic activities, structured activities and partner links [OAS07]. Activities perform the process logic in BPEL. A distinction is made between basic activities and structured activities. Basic activities describe elemental steps of the process behavior, while structured activities describe the control-flow logic. Structured activities can contain other activities. Partner links represent the interactions between different services or business processes. A more detailed description of all elements with respect to BPEL can be found in Appendix B.

2.2.2 Evaluation of BPEL

Different researchers investigated the BPEL standard and compared the language with other web services languages.

Van der Aalst and Ter Hofstede [Web08b] created a list of 20 different workflow patterns. Wohed et al. [WADH06] compared these workflow patterns with BPEL and created a coverage overview. This research concluded direct support for most of the patterns is implemented. Four major patterns which are not covered within BPEL directly are: Discriminator, Multi-Merge, Arbitrary Cycles and Milestones. Milestones can be covered using a work-around, but since this work-around is rather complex this implementation might be improved. Because states are not supported by BPEL, variables must be introduced to keep track of the state. In section 2.3.1 the coverage of both BPMN and BPEL is discussed.

Another drawback of BPEL is the possible overlap of constructs, which results in different implementations. Some notations deliver more readable output, while other notations only result in correct output which is not comprehensible for a human being. In Figure 2.4 different BPEL structures are displayed for the same BPMN input model. A sequence of activities can be modeled within the `<sequence>` concept for example, but can also be modeled using a `<flow>` concept with control links for every relation. In the BPEL structure generated by the event-condition-action-rules strategy the control flow logic is not visible in the structure anymore and is therefore harder to understand.

Table 2.1: Most common BPEL elements.

Basic activities	
Invoke	Used to invoke Web Service operations.
Receive and Reply	A (sub)process waits for a specific message and will send a reply to the initiator of the process.
Assign	The assign activity allows variables to be updated during the process.
Throw and Rethrow	For fault handling, both elements can explicitly throw a fault to the fault handler.
Wait	When a business process needs a delay for a specific time, the wait activity can be used.
Empty	The empty activity does nothing, but can be used to generate valid and readable BPEL-code.
Exit	The exit activity immediately terminate all activities in the process instance.
Structured activities	
Sequence	Activities which are executed one by one in a given order, are placed in a sequence control activity.
If	For conditional behavior.
While and repeatUntil	Provide a repeated set of activities.
Pick	Waits for an occurrence of one event and then executes the activities related to this event. Other events where the Pick was listening to are neglected.
Flow	Makes synchronization and concurrency possible.
ForEach	ForEach makes it possible to process multiple branches. Depending on the design of this activity, the children of this activity can be executed sequential or in parallel.
Partner Links	Describe cross enterprise business interactions through Web Service interfaces. By describing the partner links, also roles are assigned to different participants.

The readability aspect is a criterion which is often discussed by describing strategies for translating BPMN to BPEL. Most researchers, including [OADH08, ODHA08, MLZ06], agree on the high readability of structure transformation strategies compared to other strategies as we will see in the next section. The definition of readability is however not given and an empirical study to support the researchers assumptions cannot be found. We adapt the assumptions on readability from the other researchers, but underline the need for an empirical study in the near future.

2.3 Research on BPMN to BPEL

In the past years, researchers tried to develop an algorithm for the transformation from BPMN to BPEL. Some strategies arose, but are also critically reviewed by other researchers. In section 2.3.1 we first discuss the differences between BPMN and BPEL. Then an evaluation of current research will be given in section 2.3.2. In section 2.3.3 current implementations are discussed.

Both readability and completeness will be used as two main criteria for evaluating existing transformation strategies and implementations. Together with other researchers, including [OADH08, ODHA08, MLZ06], we adopt the assumption that more structure results in a higher readability. As indicated before, we underline the need for an empirical study to confirm this assumption. We define completeness as the set of requirements on the input model. We also want to translate unstructured input models to corresponding BPEL structure.

2.3.1 Differences between BPMN and BPEL

According to several researchers, including Recker et al. [RM06], there is a mismatch between BPMN and BPEL. Three different reasons are given [RM06, Web08a]. First the two standards are used in different phases in the life cycle of a business process. BPMN is normally used in the design phase, where business analysts design and discuss a business process with the use of graphical notations. BPEL is used in the implementation phase, which is more related to technical analysts and programmers who will use the language. The second reason is the difference between business analysts and technical analysts who have their own perspective of the two languages. Finally, BPMN is a graph oriented notation and BPEL a block oriented language, which results in a conceptual mismatch between both standards.

Having a conceptual mismatch can be a problem, but tools for mapping BPMN to BPEL can narrow down the mismatch. Both process analysts and software engineers have BPMN and BPEL models available, which can help in understanding models and will improve communication. The conceptual mismatch as discussed by Recker et al. [RM06] might therefore be an extra reason for researching mappings from BPMN to BPEL instead of a problem.

Recker et al. [RM06] compared BPMN and BPEL by listing the level of support of the workflow patterns from [Web08b]. In Table 2.2 an overview is given. A + sign indicates direct support of the pattern by the language, a +/- indicates indirect support and a - sign indicates that there is no support for the specific pattern. The most important differences are the multiple merge, discriminator, arbitrary cycles and MI with a priori Runtime Knowledge patterns. These

Table 2.2: BPMN and BPEL compared with workflow patterns [WADH05, WADH03]

Workflow Patterns	BPMN	BPEL	Workflow Patterns (ctd.)	BPMN	BPEL
<i>Basic Control Flow</i>			11. Implicit Termination	+	+
1. Sequence	+	+	<i>Multiple Instances Patterns</i>		
2. Parallel Split	+	+	12. MI without Synchronization	+	+
3. Synchronization	+	+	13. MI with a priori Design Time Knowledge	+	+
4. Exclusive Choice	+	+	14. MI with a priori Runtime Knowledge	+	-
5. Simple Merge	+	+	15. MI without a priori Runtime Knowledge	-	-
<i>Adv. Synchronization</i>			<i>State-Based Patterns</i>		
6. Multiple Choice	+	+	16. Deferred Choice	+	+
7. Synchronizing Merge	+/-	+	17. Interleaved Parallel Routing	+/-	+/-
8. Multiple Merge	+	-	18. Milestone	-	-
9. Discriminator	+	-	<i>Cancellation Patterns</i>		
<i>Structural Patterns</i>			19. Cancel Activity	+	+
10. Arbitrary Cycles	+	-	20. Cancel Case	+	+

patterns are explained in an article by [WAD⁺06a]:

1. WCP8: *Multiple merge* - the ability to represent the unsynchronised convergence of two or more distinct branches. If more than one branch is active, the activity following the merge is started for every activation of every incoming branch;
2. WCP9: *Discriminator* - the ability to depict the convergence of two or more branches such that the first activation of an incoming branch results in the subsequent activity being triggered and subsequent activations of remaining incoming branches are ignored. It is a special case of *N-out-of-M* pattern, where N is equal to one;
3. WCP10: *Arbitrary cycles* - the ability to represent loops that have multiple entry or exit points;
4. WCP14: *MI with a priori runtime knowledge* - the ability to initiate multiple instances of an activity within a given process instance. The number of instances varies but is known at runtime before the instances must be created. Once all instances have completed, a subsequent activity is initiated.

In BPMN these four patterns can be modeled, while in BPEL there is no support for these patterns. The main reason for this is the lack of multiple threads in BPEL and the awareness of states. The influence this difference in supporting patterns has, depends on the purpose of the translation and the input model. The patterns mentioned above, might never exist in BPMN models which are used as input models. However, users should be aware of this restriction when modeling in BPMN with the purpose of generating a BPEL model.

2.3.2 Current research on mapping BPMN to BPEL

Mendling et al. [MLZ06] discuss five strategies for transforming BPMN to BPEL. They even discuss some reversed transformation strategies, which might be interesting for researchers investigating round-trip engineering. The five strategies discussed are: element preservation, element minimization, structure identification, structure maximization and event-condition-action-rules. We will now briefly explain these strategies. In Figure 2.4 examples of these strategies are given.

Element preservation - This strategy maps all process graph elements to a flow, maps connectors to `<empty>` elements and maps arcs to `<links>`. The advantage of this strategy is the ease of implementation and the one-to-one correspondence between the nodes and original process graph. However, the BPEL control flow includes more elements and the flow might be difficult to understand. This strategy is only feasible if the process does not contain loops. When readability is important, the strategy should not be applied to large processes.

Element minimization - This strategy is an extension on the element preservation strategy and eliminates the `<empty>` elements. This results in BPEL code which is less intuitive than the first strategy, but still is in the spirit of a BPEL flow. The strategy can be used when as few nodes as possible should be generated, because of performance matters.

Structure identification - For processes which are structured, structure identification is a good way to map BPMN to BPEL. The process is folded to one single component while mapping. Different rules are described by Mendling et al. which can be used directly for this purpose. The advantage is that all control flow is translated to structured activities which results in readable BPEL code. However, the relation with the original BPMN model might not be intuitive.

Structure maximization - The structure maximization strategy is an extension of the structure identification strategy and makes it possible to translate also non-structured process graphs as long as they do not have arbitrary cycles. The element-preservation or element-minimization strategy can be used for those parts which can not be translated with the structure identification strategy. A drawback of this strategy is that multiple strategies need to be implemented.

Event-condition-action-rules - Another strategy based on the structure identification strategy is the event-condition-action-rules strategy. Structured parts of the process model are translated with the structure identification strategy and other parts will be translated with the help of event handlers. These

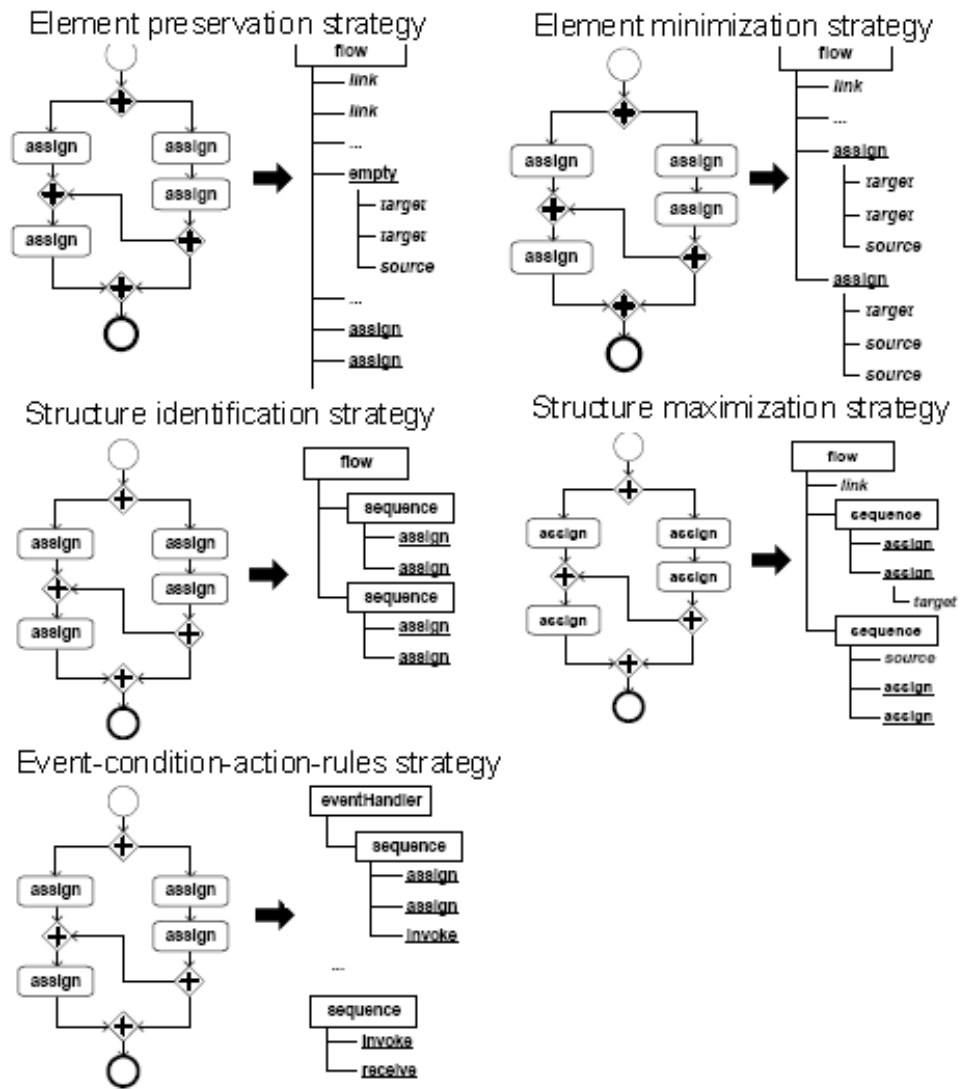


Figure 2.4: Transformation strategies by Mendling et al.[MLZ06]

event handlers are similar to `goto`-statements in software engineering. In Figure 2.4 an example of this strategy can be found; the event handler is not directly connected to the control flow. The advantage of this strategy is that almost any process graph, including those with unstructured loops, can be transformed. The use of event handlers however makes the generated BPEL code less understandable because the control flow crosses the border of event handlers.

Ouyang et al. [OADH06] suggest a method for transformation with an event-action rule-based algorithm. First a Business Process Diagram (BPD) is decomposed into components. Well structured components are translated with the structure identification strategy, see also Figure 2.6. Components which cannot be translated with this strategy are transformed with the event-action rule-based translation. Preconditions are defined which generate event-action rules. These event-action rules can then be transformed to BPEL code.

In another research of Ouyang et al. [OADH08] the described method is slightly adjusted. Petri nets are used to check the soundness and safeness of the BPD. Also dependencies between activities are described as control links, before the event-action rule-based strategy is executed. This will result in BPEL code with a higher level of readability compared to the case of using events, because control links are enclosed in the control flow within the generated BPEL code. In another research article [ODHA08], a new phase in the transformation process is proposed, where quasi-structured patterns are refined into well-structured patterns.

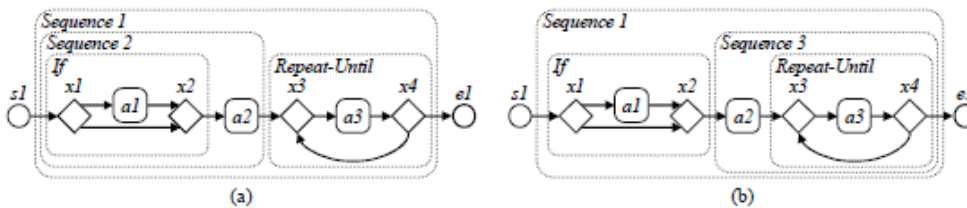


Figure 2.5: The same model decomposed twice in a non-deterministic way [VVK08]

The approach of [OADH08] is not the ideal approach, since the approach is not deterministic. Multiple translations of the same input model can result in different BPEL models. Multiple sequences can occur in sequence of each other and result in different combinations as shown in Figure 2.5. Components *sequence2* and *sequence3* differ from each other and result in different implementations. We don't want an approach which is non-deterministic, because this influences the readability of the translated BPMN model. Vanhatalo et al.

[VVK08] present the refined Process Structure Tree approach which disregards non-maximal sequences in the process structure tree and find as much structure as possible. Vanhatalo et al. prove that the presented decomposition is unique and modular.

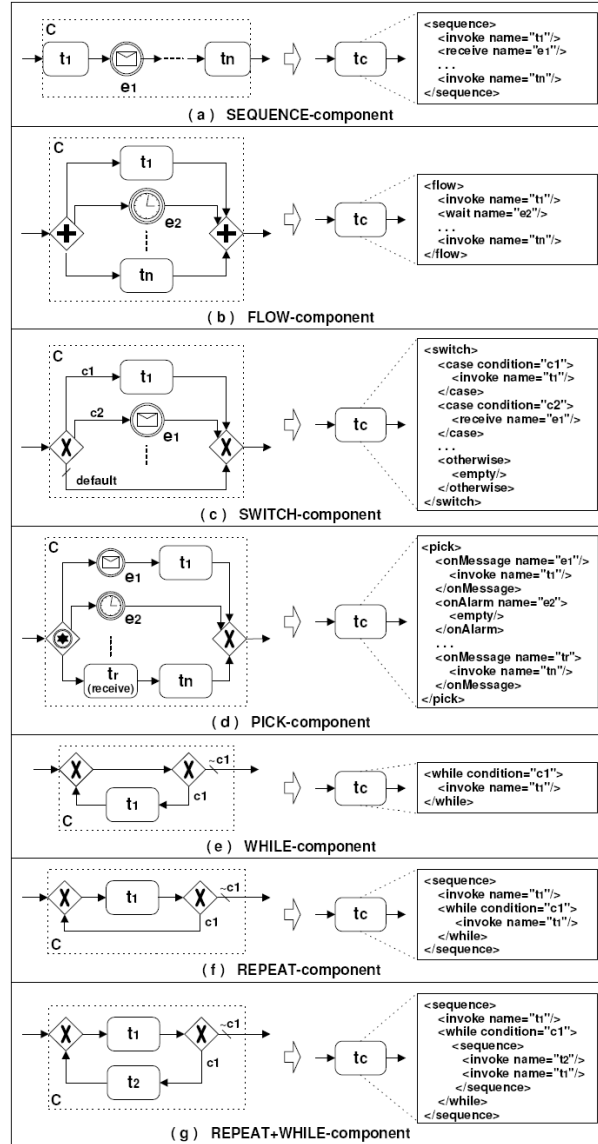


Figure 2.6: Translation of structured BPMN components to BPEL [OADH06]

White [Whi05] describes a strategy which is focused on the flow element of BPEL. In that case BPMN is mapped to the graph structure of BPEL instead of the block structure (sequence element). Therefore, the links are not only

used for synchronization purposes, but also for other relations between activities. Mapping of unstructured processes might therefore be possible, but the readability of the generated BPEL code is an issue.

2.3.3 Current implementations

Two open source BPMN2BPEL tools are currently available. BABEL¹ is the best known tool of both and is developed based on an article of Ouyang et al. [OADH08]. However, this tool is not maintained anymore since there were too many bugs, there was too little documentation and a non-deterministic single-entry single-exit (SESE) decomposition approach was not implemented while it was necessary [Dum09]. Vanhatalo et al. [VVK08] discuss the refined process structure tree (PST) which is a decomposition approach that is unique and modular. Adjusting the BABEL-tool to this PST-approach was not possible according to M. Dumas [Dum09].

A new implementation based on the Process Structure Tree decomposition was developed as an Eclipse plugin available on Google code². This implementation uses the SESE decomposition approach presented by [VVK08] to identify SESE regions and determine process components with structure. These regions are nested, such that they can be arranged to form a structure tree. Each leaf is then translated by the tool to a corresponding BPEL construct. García-Bañuelos [GB08], developer of the Eclipse plugin, explains how to identify patterns and structure a BPMN model into SESE regions.

Both implementations are based on the idea of combining different strategies for translating BPMN to BPEL. First structured components are identified and translated, then all elements which can be translated by the link-based approach are translated and finally the event-action based approach is applied. The Eclipse plugin is not based on the BABEL-tool, but the website of the BABEL-tool suggests otherwise. After a private conversation with the authors, both tools seem to be independent from each other.

2.4 Conclusions

BPMN might become an industry standard soon, but still has some issues which hopefully have been improved in BPMN 2.0. Due to the possible ambiguity of models, misunderstanding can occur and models should be checked for feasibility. However, BPMN is a modeling notation which can be understood by both business analysts and technical analysts. This results in better material for discussing business processes and will have a positive effect on implementations.

¹BABEL: <http://www.bpm.fit.qut.edu.au/projects/babel/tools/>

²Google code: <http://code.google.com/p/bpmn2bpel/>

BPEL is a well developed standard in the field of web service specifications. A broad range of workflow patterns can be supported by BPEL. However, there are multiple ways of implementing the same processes, which can result in BPEL code which is hard to understand.

Different strategies for mapping BPMN to BPEL are explained in the previous section. Each strategy has some benefits and drawbacks. Often completeness and readability are the criteria to evaluate the quality of a strategy. Both should be high for an optimal transformation strategy, but can be conflicting criteria. At this moment combining strategies is the best way to create a complete and readable model in BPEL.

One common problem of translating BPMN to BPEL is the improper modeling of an input BPMN model. Since BPMN can contain infinite loops or deadlocks, implementations can be generated when translating BPMN to BPEL which cannot be executed. Therefore it would be important to check for soundness and liveness in a BPMN model. Dijkman et al. [DDO08] propose an algorithm for translating BPMN to Petri nets. With the help of Petri nets the soundness and safeness can be checked in a formal way.

A main distinction between structured and unstructured components and processes can be found in current approaches. Often, structured components are mapped to associated BPEL constructs for improved readability. Unstructured components are mapped to control links or event handlers. Both the control links and event handlers are found to be less readable compared to the structured component translations [OADH08, ODHA08, MLZ06], but support a broader range of input process models. Restrictions on the input process model like requiring a well-formed business process diagram can be relaxed in these less readable approaches. As said, combining strategies will probably lead to a translation with the best readability and highest completeness.

Table C.1 in Appendix C, summarises the strengths, weaknesses, constraints and completeness of existing approaches discussed earlier. This overview will also be used for evaluating our own algorithm, which we will present in chapter 3. In table 2.3 we visualised the possibilities of different approaches with a matrix. A '+' means that the mapping supports the component (or is found to be readable or complete by literature), where a '-' means that the component is not supported. A '+/-' indicates a partial support for a component.

Table 2.3: Evaluation of current approaches

Component \ Mapping	A	B	C	D	E	F	G	H
Readability	-	+/-	-	-	+	+	-	+/-
Completeness	+/-	+	+/-	+/-	-	+/-	+	+
Structured components	+	+	+	+	+	+	+	+
Structured loops	-	+	-	-	+	+	+	+
Unstructured components	+	+	+	+	-	+	+	+
Unstructured loops	-	+	-	-	-	-	+	+
Synchronization links	+	+/-	+	+	-	+	+/-	+/-

The following mappings are displayed in table 2.3.

A: Graph structure approach (White) [Whi05]

B: A combined approach (Ouyang et al.) [OADH08]

C: Element-preservation (Mendling et al.) [MLZ06]

D: Element-minimization (Mendling et al.) [MLZ06]

E: Structure-identification (Mendling et al.) [MLZ06]

F: Structure-maximization (Mendling et al.) [MLZ06]

G: Event-condition action-rules (Mendling et al.) [MLZ06]

H: Pattern identification and Classification (García-Bañuelos) [GB08]

Chapter 3

Mapping BPMN to BPEL

We have seen the differences in translations of BPMN to BPEL for business processes. Structured and unstructured business processes are mapped according to different approaches. We assume that a combination of approaches will lead to the translation with the highest level of completeness and readability. Because both requirements are conflicting, we find readability more important than completeness because this implies more structure in the generated BPEL code. The implementation of García-Bañuelos [GB08] mentioned in section 2.3, already combines structured and unstructured approaches. First the structured components, which are identified using single entry single exit regions, are processed. The resulting part of the process is then translated using control links. We will compare this approach with a modified algorithm of Rik Eshuis [EG09], which composes services into structured processes. The generated structured process model can be translated to BPEL syntax according to the structured identification strategy proposed by Mendling et al. [MLZ06].

We will start with using a well-formed Business Process Diagram (see section 3.1), but will evaluate these constraints during the development of the algorithm. Starting with structured processes will match current research in the field of BPMN to BPEL. While evaluating well-formed restrictions, we can discuss if the requirements for input processes can be relaxed and if input models with less structure can be transformed. We also want to adjust the algorithm of Eshuis with an algorithm for finding synchronization links. These synchronization links can be mapped to the BPEL control links. The remaining process will be structured which is possible due to the restrictions of a well-formed Business Process Diagram. We focus only on the control flow aspect of BPMN models and will not focus on data aspects. Our goal is to develop a mapping which is able to translate BPMN models, that meet the well-formedness restrictions, to valid BPEL syntax. Also we are interested in the added value the existing algorithm of Eshuis can offer in the process of translating BPMN to BPEL.

3.1 Business Process Diagram

BPMN uses Business Process Diagrams (BPDs) to describe business process. BPDs are built of a core subset of BPMN elements and described by [OADH08, ODHA08]. A BPD contains a set of objects \mathcal{O} which can be partitioned into disjoint sets of tasks \mathcal{T} , events \mathcal{E} and gateways \mathcal{G} . Events can be partitioned into disjoint sets of start events \mathcal{E}^S , intermediate events \mathcal{E}^I and end events \mathcal{E}^E . The intermediate events can be partitioned further into a disjoint, complete set of intermediate message events $\mathcal{E}_{\mathcal{M}}^I$ and timer events $\mathcal{E}_{\mathcal{T}}^I$. Gateways can be partitioned into disjoint sets of parallel fork gateways \mathcal{G}^F , parallel join gateways \mathcal{G}^J , data-based XOR decision gateways \mathcal{G}^D , event-based XOR decision gateways \mathcal{G}^V , and XOR merge gateways \mathcal{G}^M . Relations between objects are defined in the control flow relation $\mathcal{F} \subseteq \mathcal{O} \times \mathcal{O}$.

A graphical representation can be found in the class diagram in Figure 3.1, where the formal definition [OADH08, ODHA08] can be found in Appendix D.

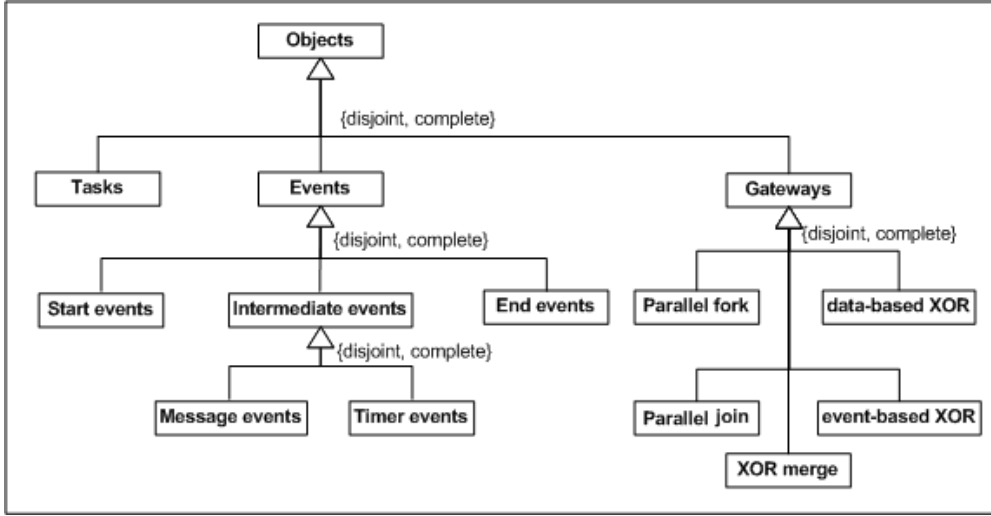


Figure 3.1: BPD elements displayed in a class diagram

The relation \mathcal{F} defines a directed graph with nodes(objects) and arcs(sequence flows). However, the definition allows graphs which are unconnected, not having start or end events, containing objects without input or output etc. Because we only want a model without deadlocks and livelocks, we need to restrict the definition to a well-formed BPD, before we can use an algorithm on the model.

A so called well-formed BPD is a BPD which meets specific requirements. The remaining of this section is adapted from [OADH08, ODHA08]. All tasks and events, represented by o , can only have an indegree, $in(o)$, of at most one and

an outdegree, $out(o)$, of at most one.

$$in(o) = \{x \mid (x, y) \in \mathcal{F} \wedge y = o\}$$

$$out(o) = \{y \mid (x, y) \in \mathcal{F} \wedge x = o\}$$

Start events and end events have an indegree and outdegree respectively of zero. Gateways however, can have more incoming or outgoing relations. Fork or decision gateways have an outdegree of more than 1, while join or merge gateways have an indegree of more than 1. Also every object of the BPD must be on a path from start till end event.

Definition 1. *A core BPD is well-formed if relation \mathcal{F} satisfies the following requirements [OADH08, ODHA08]:*

- $\forall s \in \mathcal{E}^S, in(s) = \emptyset \wedge |out(s)| = 1$, *i.e. start events have an indegree of zero and an outdegree of one,*
- $\forall e \in \mathcal{E}^E, out(e) = \emptyset \wedge |in(e)| = 1$, *i.e. end events have an outdegree of zero and an indegree of one,*
- $\forall x \in \mathcal{T} \cup \mathcal{E}^I, |in(x)| = 1$ *and* $|out(x)| = 1$, *i.e. tasks and intermediate events have an indegree of one and an outdegree of one,*
- $\forall g \in \mathcal{G}^F \cup \mathcal{G}^D \cup \mathcal{G}^V, |in(g)| = 1 \wedge |out(g)| > 1$, *i.e. fork or decision gateways have an indegree of one and an outdegree of more than one,*
- $\forall g \in \mathcal{G}^J \cup \mathcal{G}^M, |out(g)| = 1 \wedge |in(g)| > 1$, *i.e. join or merge gateways have an outdegree of one and an indegree of more than one,*
- $\forall g \in \mathcal{G}^V, out(g) \subseteq \mathcal{E}^I \cup \mathcal{T}^R$, *i.e. event-based XOR decision gateways must be followed by intermediate events or receive tasks,*
- $\forall g \in \mathcal{G}^D, \exists x \in out(g), Cond((g,x)) = \neg \bigwedge_{y \in out(g) \setminus \{x\}} Cond((g,y))$, *i.e. (g,x) is the default flow among all the outgoing flows from g .*
- $\forall x \in \mathcal{O}, \exists (s,e) \in \mathcal{E}^S \times \mathcal{E}^E, s\mathcal{F}^*x \wedge x\mathcal{F}^*e$, *i.e. every object is on a path from a start event to an end event.*

For developing a new algorithm which can translate BPMN models to BPEL code, we will first focus only on well-formed BPDs. Later on, the constraints of a well-formed BPD will be evaluated to see whether or not such a constraint can be eliminated without any unintended effect on the result of our algorithm.

3.2 Synchronization dependencies

In our algorithm, we want to support the presence of synchronization links. If a task can only be executed after another task is finished and both tasks are in a different branch of a concurrency situation, a synchronization link can be identified. In all other situations synchronization links are not possible, since otherwise deadlock situations could occur. For example, waiting for a task in another branch of a deferred choice to be finished will never end, which results in a deadlock situation. Figure 3.2 gives an example, where the filled elements represent the flow which is already executed. The process can not execute further, because task A is not and will never be executed. Of course, translating a business process model which contains deadlocks make no sense, because this results in improper BPEL code. However, the example in Figure 3.2 demonstrates that the synchronization link in this model is only correct if all gateways are parallel gateways.

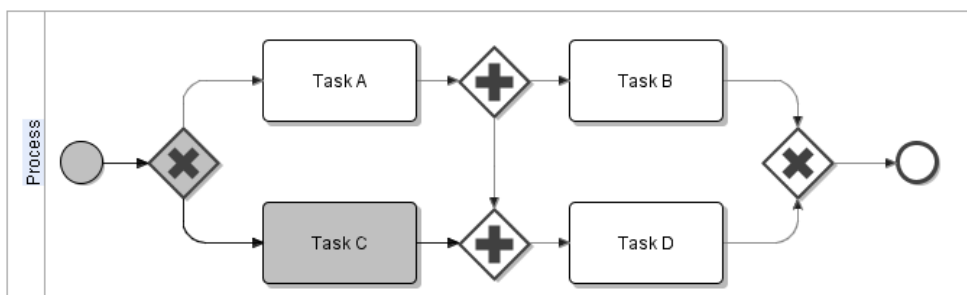


Figure 3.2: An incorrect synchronization link, which causes a deadlock

Because synchronization links in BPEL are modeled by the `<link>` components, some other requirements have to be met, in order to meet the specification constraints [OAS07]. It is not allowed that a synchronization link causes cyclic behavior because then the synchronization source has the target activity as a preceding activity. This implies that the scope of the synchronization link is never a loop. Synchronization links are also not allowed to cross boundaries of repeatable constructs like a loop. If a loop contains a synchronization link, the scope will be restricted to the concurrency construct `<flow>`. This implies that the synchronization link must be activated each execution of the loop instance. A synchronization dependency in a BPMN model must therefore meet the following constraints:

- The synchronization source should not have the target as preceding activity.
- Boundaries of repeatable constructs should not be crossed.

- Both the source and target of the dependency should be preceded by a parallel split gateway.

The concept of Dead-Path-Elimination (DPE) is also an important issue for synchronization dependencies, since the `<link>` source components must be assigned a value before the targets can be processed and terminated in BPEL. With DPE the activities which are not executed can be 'garbage collected' by assigning the value 'false'. The business process as a whole is then processed and the process can be correctly terminated when necessary. The problem of DPE is the risk of (unintended) side effects which can occur. Van Breugel et al. [BK05] discuss these side effects and propose a modification for DPE to solve these issues. The BPEL syntax offers a possibility to use DPE by setting the value of the `<process>`-attribute `suppressJoinFailure`.

A synchronization link candidate can be easily marked, since this needs to be a direct relation between a parallel fork gateway and a parallel join gateway. Other modeling notations for synchronization links do not match the well-formed BPD criteria as discussed earlier. Using multiple incoming or outgoing arcs on a node for example is not allowed because of the restrictions of a well-formed BPD. We develop a method to determine if a synchronization link candidate really is a synchronization link. The reason for making a distinction between normal control flow relations and synchronization relations is the way we process both types further in our algorithm. Synchronization relations are not used for determining the structure of the business process, but are only used in one of the last phases of the mapping. We will add the synchronization dependencies in the structured composition created in the algorithm.

3.2.1 Finding synchronization dependencies

Synchronization dependencies are modeled by a direct relation between a parallel split and a parallel join gateway. If there is no direct relation between two of these gateways, there are no synchronization dependencies.

First the set of tasks and events which precede a gateway and the set of tasks and events which follow a gateway are captured for both parallel fork gateways and join gateways. This results in two sets of objects for a gateway $g \in \mathcal{G}^{\mathcal{F}} \cup \mathcal{G}^{\mathcal{J}}$, the *before(g)* and *after(g)*-set. The *before(g)*-set contains all tasks and events which are always executed before the gateway can be visited. The *after(g)*-set contains all tasks and events which are always executed after the gateway is visited. The union of both sets describes the whole branch from entry node to exit node in form of tasks and events. The set *branch(g)* represents this union and is defined as $before(g) \cup after(g)$.

During the evaluation of our algorithm, a lot of models adapted from literature were used to evaluate our algorithm for finding synchronization dependencies.

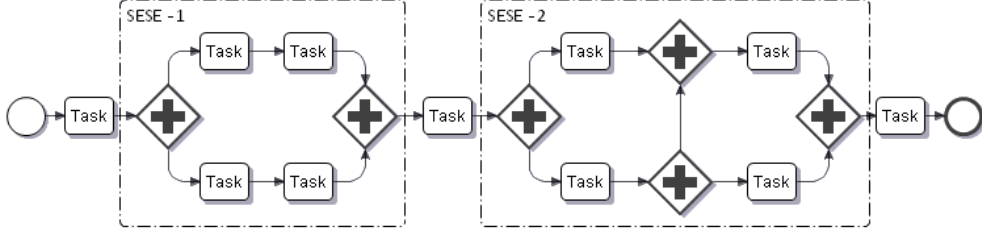


Figure 3.3: Two parallel constructs in a sequence; SESE-regions are necessary.

Also two case studies described in chapter 5 were used and show the need for single entry, single exit (SESE) regions. The algorithm we propose worked fine as long as there were no two parallel constructs in a sequence of each other. To support these situations, which we modeled in Figure 3.3, we have done some finetuning in the algorithm and added the restriction that the gateways we are evaluating are in the same SESE-region. A SESE-region, or component, can be defined in the following definition[OADH06]. To facilitate the definitions, Ouyang et al. specified an auxiliary function over a domain of singletons, i.e., if $X = \{x\}$, then $\text{elt}(X) = x$.

Definition 2. Let $\mathcal{BPD} = (\mathcal{O}, \mathcal{F}, \text{Cond})$ be a well-formed core BPD. $C = (\mathcal{O}_c, \mathcal{F}_c, \text{Cond}_c)$ is a component(SESE-region) of BPD if and only if:

- $\mathcal{O}_c \subseteq \mathcal{O} \setminus (\mathcal{E}^S \cup \mathcal{E}^E)$,
- $|\cup_{x \in \mathcal{O}_c} \text{in}(x) \setminus \mathcal{O}_c| = 1$, i.e. there is a single entry point outside the component, which can be denoted as $\text{entry}(C) = \text{elt}(\cup_{x \in \mathcal{O}_c} \text{in}(x) \setminus \mathcal{O}_c)$,
- $|\cup_{x \in \mathcal{O}_c} \text{out}(x) \setminus \mathcal{O}_c| = 1$, i.e. there is a single exit point outside the component, which can be denoted as $\text{exit}(C) = \text{elt}(\cup_{x \in \mathcal{O}_c} \text{out}(x) \setminus \mathcal{O}_c)$,
- there exists a unique source object $i_c \in \mathcal{O}_c$ and a unique sink object $o_c \in \mathcal{O}_c$ and $i_c \neq o_c$, such that $\text{entry}(C) \in \text{in}(i_c)$ and $\text{exit}(C) \in \text{out}(o_c)$,
- $\mathcal{F}_c = \mathcal{F} \cap (\mathcal{O}_c \times \mathcal{O}_c)$,
- $\text{Cond}_c = \text{Cond}(\mathcal{F}_c)$, i.e. the Cond function where the domain is restricted to \mathcal{F}_c .

The $\text{before}(g)$, $\text{after}(g)$ and $\text{branch}(g)$ -sets can be defined as:

$$\text{before}(g) = \{x \mid x \in \mathcal{O}_c \wedge \text{entry}(C) \mathcal{F}^* x \wedge x \mathcal{F}_c^* g\}$$

$$\text{after}(g) = \{y \mid y \in \mathcal{O}_c \wedge g \mathcal{F}^* y \wedge y \mathcal{F}_c^* \text{exit}(C)\}$$

$$\text{branch}(g) = \text{before}(g) \cup \text{after}(g) \cup \{g\}$$

For each parallel fork gateway $f \in \mathcal{G}^{\mathcal{F}}$, beginning with the one with the smallest *before*(f)-set, a matching parallel join gateway $j \in \mathcal{G}^{\mathcal{J}}$ will be found when both sets contain the same elements, so $branch(f) = branch(j)$ or, if there is no join gateway with the same set as the fork gateway, one of the sets is a subset of the other ($branch(f) \subseteq branch(j)$ or $branch(f) \supseteq branch(j)$). A relation between a matching fork and join gateway is not a synchronization link and the join gateway is removed from the set which is used to determine synchronization links. The algorithm in Figure 3.4 shows this approach and orders the set of gateways using a while-construct [Esh05], where the smallest and largest gateways are taken for the fork and join gateway respectively. This while-construct will be terminated when all gateways are processed. By introducing the ordering, the algorithm can be executed more efficiently because matching gateways are found earlier. A join gateway j , which can not be matched to any fork gateway f and where a direct relation between the fork gateway f and the join gateway j exists, can be marked as target of the synchronization link. The source of the synchronization link will be the fork gateway f of the direct relation.

All synchronization links will be stored in the set $\mathcal{F}_S \subseteq \mathcal{F}$ and will be processed different from other relations in the next step. We will now first explain the algorithm for finding synchronization and use an example of a complaint handling process, which is presented in the next section.

The algorithm for finding synchronization links is listed in figure 3.4. It requires as input a set $\mathcal{G}^{\mathcal{F}}$ of parallel fork gateways which will be ordered by the algorithm in ascending order of the *before*(g)-set size as explained before. Also a set $\mathcal{G}^{\mathcal{J}}$ of parallel join gateways, which will be ordered in descending order of the *before*(g)-set size, is required. The algorithm looks for matching *branch*(g)-sets first and in case there is no matching, also for subsets of the fork and join gateway-*branch*(g)-sets. Matching gateways are skipped when further processing the algorithm. In case there is no match between a fork gateway f and a join gateway j , a synchronization link will be identified when a direct relation between both gateways can be found in \mathcal{F} .

After introducing the *before*(g), *after*(g) and *branch*(g)-sets we can explain the need for SESE-regions once more, since in Figure 3.3 all gateways which are not part of the synchronization dependency have the same *branch*(g)-set. Because of the ordering in our algorithm, the two outer gateways are matched which is not the situation modeled. When the algorithm is executed within a SESE-region, this problem situation can not occur.

Alternatives for the presented algorithm for finding synchronization dependencies are asking the user for clearness or process the whole component which contain a synchronization dependency as an unstructured component. Because

```

1: procedure FINDINGSYNCHRONIZATION( $(G^F, G^J, \mathcal{F})$ )
2:    $links := \emptyset$ 
3:   if  $\exists (f, j) \in \mathcal{F} \mid f \in G^F \wedge j \in G^J$  then
4:      $forktovisit := G^F$ 
5:     while  $forktovisit \neq \emptyset$  do
6:        $f :=$  the gateway with the smallest  $before(g)$ -set of  $forktovisit$ 
7:        $SearchSubsets = true$ 
8:        $jointovisit := G^J$ 
9:       while  $jointovisit \neq \emptyset$  do
10:         $j :=$  the gateway with the largest  $before(g)$ -set of  $jointovisit$ 
11:        if  $branch(f) = branch(j)$  then
12:           $G^J := G^J \setminus \{j\}$ 
13:           $SearchSubsets = false$ 
14:          break;
15:        end if
16:         $jointovisit := jointovisit \setminus j$ 
17:      end while
18:      if  $SearchSubsets = true$  then
19:        while  $jointovisit \neq \emptyset$  do
20:           $j :=$  the gateway with the largest  $before(g)$ -set of  $jointovisit$ 
21:          if  $branch(f) \supseteq branch(j) \wedge (f, j) \notin \mathcal{F}$  then
22:             $G^J := G^J \setminus \{j\}$ 
23:            break;
24:          else if  $branch(f) \subseteq branch(j) \wedge (f, j) \notin \mathcal{F}$  then
25:             $G^J := G^J \setminus \{j\}$ 
26:            break;
27:          else if  $(f, j) \in \mathcal{F}$  then
28:             $links := links \cup \{(f, j)\}$ 
29:          end if
30:           $jointovisit := jointovisit \setminus j$ 
31:        end while
32:      end if
33:       $forktovisit := forktovisit \setminus f$ 
34:    end while
35:  end if
36:  return  $links$ 
37: end procedure

```

Figure 3.4: Algorithm for finding synchronization dependencies

we want to create an approach which requires as little user input as necessary, the second alternative is the only interesting one. [GB08] use this second alternative and processes the whole (un)structured component which has a synchronization dependency as an unstructured component. This results in less readable BPEL code, since all relations within this unstructured component are translated as links.

3.2.2 The complaint handling process

We have chosen the complaint handling process model [ODHA08] to explain our algorithm by an example. The process contains concurrency, exclusive choice, events, a synchronization link and a repeatable construct in terms of a loop. The process therefore has different constructs, but does not violate the constraints for a well-formed BPD.

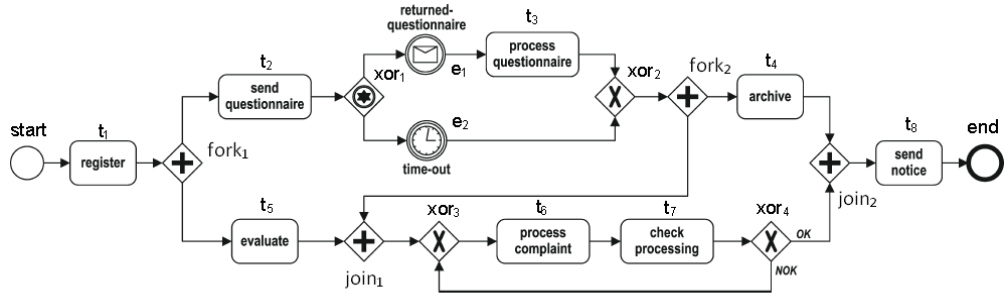


Figure 3.5: A complaint handling process, adapted from [ODHA08]

First, for every parallel fork and join gateway, the $before(g)$ -set and $after(g)$ -set will be calculated, which results in table 3.1. The set G^F of parallel fork gateways is ordered in ascending order of the $before(g)$ -set size and is therefore equal to $G^F = \{fork_1, fork_2\}$. The set G^J of parallel join gateways is ordered in descending order of the $before(g)$ -set size and is equal to $G^J = \{join_2, join_1\}$.

Table 3.1: $before(g)$ - and $after(g)$ -set for gateways of our example

Gateway:	$before(g)$ -set	$after(g)$ -set
$fork_1$	—	$t_2, e_1, t_3, e_2, t_4, t_5, t_6, t_7$
$fork_2$	t_2, e_1, t_3, e_2	t_4, t_6, t_7
$join_1$	t_2, e_1, t_3, e_2, t_5	t_6, t_7
$join_2$	$t_2, e_1, t_3, e_2, t_4, t_5, t_6, t_7$	—

There exists a direct relation between a parallel fork and join gateway which satisfies the condition on line 3 of the algorithm. Node $fork_1$ is taken and will be matched to $join_2$ by the algorithm because of matching $branch(g)$ -sets. Node $fork_2$ will then be processed and can not be matched to an equal $branch(g)$ -set of a remaining join-gateway. Also $branch(fork_2)$ is not a subset of $branch(join_1)$ and vice versa. Since the relation between $fork_2$ and $join_1$ is indeed a direct relation, it is marked as a synchronization link.

3.3 Dominators, loop headers and follow sets

Since synchronization links will be processed different from other relations in this step, we first need to define an adjusted Business Process Diagram. This adjusted BPD is cleared from synchronization links and gateways might have been replaced by dummy nodes. The new set of relations $\mathcal{F}^{\mathcal{N}}$ is the set of relations excluding the elements in the synchronization links set $\mathcal{F}^{\mathcal{S}}$, so $\mathcal{F}^{\mathcal{N}} = \mathcal{F} \setminus \mathcal{F}^{\mathcal{S}}$. Some gateways might transform into gateways with only one incoming and one outgoing arc, which conflicts with the definition of a well-formed BPD. By replacing these gateways with dummy tasks, the resulting BPD will be well-formed again.

For finding structure in BPDs, some concepts like dominators, loop headers and follow-sets have been introduced. We adapted this section from Eshuis et al. [EG09]. Follow-sets can be calculated based on the dominators and loop headers. The result is a description of the structured properties of the BPD. As we will see later on, these concepts can be used in our algorithm.

The concept of dominator is used to identify nesting structure and is taken from [EG09]. Let p be a node in \mathcal{O} , q be a node in \mathcal{O} and $start$ be a node in $\mathcal{E}^{\mathcal{S}}$. Node p dominates node q if every path from $start$ to q passes through p and this path is free of synchronization links:

$$p, q \in \mathcal{O}, start \in \mathcal{E}^{\mathcal{S}}, start \mathcal{F}^{\mathcal{N}*} p \wedge p \mathcal{F}^{\mathcal{N}*} q, \text{ i.e. every path from } start \text{ to } q \text{ passes through } p.$$

Every node except the start node has at least one dominator. Node p is the immediate dominator of node q if every dominator of q other than p also dominates p . Let $DOM(p)$ denote the immediate dominator of p . A node p can only have one $DOM(p)$ and therefore $DOM(p)$ is unique for p [ASU86].

Next, loops need to be identified and loop headers need to be determined. A loop is identified with the help of back edges. A back edge is an edge (x, y) in $\mathcal{F}^{\mathcal{N}}$ where y dominates x . The natural loop of a back edge can be computed, which is the set of nodes that can reach x without going through y , plus y .

Node y is the header of the natural loop, which is defined as $HEAD(n) = y$. A node that is target of some back edge is called a loop node. If a node is not a loop node, it's $HEAD(n)$ is undefined.

When we both determined the dominators and loop headers of a BPD, we can calculate the follow-sets. A follow-set of p contains all the nodes that are immediately after p , at the same level of nesting, in the structured composition.

Definition 3. *The follow-set of p , $FOLLOW(p)$, can be calculated with the following rules:*

For a fork node p , so $p \in \mathcal{G}^{\mathcal{F}} \cup \mathcal{G}^{\mathcal{D}} \cup \mathcal{G}^{\mathcal{V}}$, let

$$FOLLOW(p) = \{ q \mid q \in \mathcal{G}^{\mathcal{J}} \cup \mathcal{G}^{\mathcal{M}} \wedge p = DOM(q) \wedge HEAD(p) = HEAD(q) \}$$

For a loop node p , so some back edge enters p , define

$$FOLLOW(p) = \{ q \mid DOM(q) \text{ is in a natural loop headed by } p \}$$

For each other node p , define

$$FOLLOW(p) = \{ q \mid HEAD(p) = HEAD(q) \wedge p = DOM(q) \}$$

3.4 BPEL grammar

When we determined the synchronization links, dominators, loop headers and follow sets, the necessary input for the algorithm is gathered. The algorithm will result in a description of the BPEL process in the grammar presented below. To keep the algorithm readable and comprehensible we haven chosen for this grammar, adapted from Eshuis [EG09], instead of direct BPEL code to output. We also investigated a grammar based on process algebra presented by [BK05], but due to the restriction of the binary grammar we have chosen not to use this. In the end, a mapping between our grammar and the BPEL syntax can be used to generate BPEL code.

Let \mathcal{T} be a set of tasks, ranged over by t , let \mathcal{E} be a set of events, ranged over by e and let \mathcal{D} be a set of dummy objects, ranged over by d . The language of structured processes, ranged over by P , is generated by the following grammar:

$$P ::= seq \mid \text{and}\{seq, seq, \dots, seq, linkset\} \mid \text{xor}\{grd \triangleright seq, grd \triangleright seq, \dots, grd \triangleright seq\} \\ \mid \text{repeat } seq \text{ until } grd \mid \text{atomic}$$

$$\text{atomic} ::= t \mid e \mid d$$

$$seq ::= \langle P, P, \dots, P \rangle$$

$$grd ::= \text{in}(\text{atomic}) \mid grd \vee grd \mid \text{true}$$

$$linkset ::= \{\text{link}(atomic, atomic), \dots, \text{link}(atomic, atomic)\}$$

The expression $\langle P, P, \dots, P \rangle$ indicates that the elements in the list are executed one by one, according to the order specified in the list. Expression $\text{and}\{seq, seq, \dots, seq, linkset\}$ specifies that the elements in the set are executed in parallel and the branches can have multiple synchronization dependencies which are defined in $linkset$. Expression $\text{xor}\{grd \triangleright seq, grd \triangleright seq, \dots, grd \triangleright seq\}$ specifies that exactly one of the guarded expressions in the set is executed, while $\text{repeat } seq \text{ until } grd$ specifies that expression seq is executed until condition grd holds. The guard $\text{in}(atomic)$ is true if $atomic$ was done previously. The $linkset$ can contain multiple synchronization dependencies, which can be defined as $\text{link}(atomic, atomic)$. This indicates a synchronization dependency between two atomic elements, where the first element is the source of the synchronization link and the second element is the target. The target can only be processed when the link is activated by the source element. Both $atomic$ elements of a link-construct should be within the same scope, which is the lowest common ancestor and-construct.

3.5 The algorithm

3.5.1 Adjustment of the algorithm

The structured composition algorithm adapted from Eshuis et al. [EG09] is listed in Figure 3.6. The algorithm is slightly adjusted to fit to the input parameters of a well-formed BPD and to add synchronization links in the scope of and-constructs. It requires as input a set \mathcal{O} of objects, a set \mathcal{F}^N of relations, a set \mathcal{F}^S of synchronization links, a partial function grd that maps relations to guard expressions, and the node $current$ that is to be processed. The algorithm needs a well-formed BPD as input as explained earlier. For each fork a subsequent join of the same type needs to be present. It is also possible that one fork belongs to multiple joins or one join belongs to multiple forks, as long as they have the same type. All loops need a single entry gateway $g^{entry} \in \mathcal{G}^M$ and a single exit gateway $g^{exit} \in \mathcal{G}^D \cup \mathcal{G}^V$. Finally, it is required that parallelism does not cross the border of loops, because such behaviour cannot be expressed in structured processes.

The algorithm for finding synchronization links should be executed before this structured composition algorithm. This structured composition algorithm returns a sequential block that starts with $current$ and is based on the BPEL grammar presented before. Further explanation of the algorithm is given in [EG09].

The procedure for adding a synchronization link is given in Figure 3.8. This procedure is called from the algorithm of Eshuis and requires a set of objects

```

1: procedure STRUCTUREDCOMPOSITION( $(\mathcal{O}, \mathcal{F}^N, \mathcal{F}^S, \text{grd}, \text{current})$ )
2:   if  $\text{current} \in \mathcal{G}^F \cup \mathcal{G}^D \cup \mathcal{G}^V$  then
3:      $\text{children} := \emptyset$ 
4:     for  $n \in \text{post}(\text{current})$  do
5:       if  $n$  not in any FOLLOW set then
6:          $C_n := \text{StructuredComposition}(\mathcal{O}, \mathcal{F}^N, \mathcal{F}^S, \text{grd}, n)$ 
7:       else
8:          $C_n := \langle \text{dummy}_{\text{current}, n} \rangle$ 
9:       end if
10:      if  $\text{current} \in \mathcal{G}^D \cup \mathcal{G}^V$  then
11:         $C_n := \text{grd}(\text{current}, n) \triangleright C_n$ 
12:      end if
13:       $\text{children} := \text{children} \cup \{C_n\}$ 
14:    end for
15:    if  $\text{current} \in \mathcal{G}^D \cup \mathcal{G}^V$  then
16:       $P_{\text{comp}} := \text{xor children}$ 
17:    else
18:       $C_n := \text{AddingSynchronization}(\text{children}, \mathcal{F}^S)$ 
19:       $\text{children} := \text{children} \cup \{C_n\}$ 
20:       $P_{\text{comp}} := \text{and children}$ 
21:    end if
22:     $P := \langle P_{\text{comp}} \rangle$ 
23:  else if  $\text{current}$  is loop node with successor node  $x$  and FOLLOW node  $n$  then
24:     $P_x := \text{StructuredComposition}(\mathcal{O}, \mathcal{F}^N, \mathcal{F}^S, \text{grd}, x)$ 
25:     $P := \langle \text{repeat } P_x \text{ until in}(\text{dummy}_{\text{current}, n}) \rangle$ 
26:  else
27:     $P := \langle \text{current} \rangle$ 
28:  end if
29:  if  $\text{FOLLOW}(\text{current}) \neq \emptyset$  then
30:    if  $|\text{FOLLOW}(\text{current})| > 1$  then
31:      insert unique FOLLOW node after  $\text{current}$ 
32:    end if
33:     $\text{next} :=$  the unique node following  $\text{current}$ 
34:     $Q := \text{StructuredComposition}(\mathcal{O}, \mathcal{F}^N, \mathcal{F}^S, \text{grd}, \text{next})$ 
35:     $P := P \frown Q$ 
36:  end if
37:  return  $P$ 
38: end procedure

```

Figure 3.6: Algorithm for constructing structured compositions

in an and-construct (*children*) and a set of synchronization links which are not placed in the process description. If all synchronization links are already placed into the process description, an empty set remains and the procedure is skipped. Otherwise the *children* set is analyzed and the fork and join gateways can be found. If both gateways are in the *children* set, the link should be placed in this scope.

```

1:  $f :=$  a fresh node not in  $\mathcal{O}$ 
2:  $\mathcal{O} := \mathcal{O} \cup \{f\}$ 
3:  $E_{follows} := \{ (x, y) \mid (x, y) \in \mathcal{F}^{\mathcal{N}} \wedge x \in \mathcal{O} \wedge y \in FOLLOW(current) \}$ 
4:  $E_{new} := \{ (x, f), (f, y) \mid (x, y) \in E_{follows} \}$ 
5:  $E := (\mathcal{F}^{\mathcal{N}} \setminus E_{follows}) \cup E_{new}$ 
6: if  $current \in \mathcal{G}^{\mathcal{D}} \cup \mathcal{G}^{\mathcal{V}}$  then
7:    $grd := grd \cup \{(f, y) \mapsto formulaPreCondition(y) \mid (x, y) \in E_{follows}\}$ 
8: end if

```

Figure 3.7: Inserting unique *FOLLOW* node after *current* (l. 31 of Fig. 3.6)

Because this procedure is first called at the deepest level of nesting of an and-construct, the scope of the link is always as narrow as possible.

```

1: procedure ADDINGSYNCHRONIZATION( $(children, \mathcal{F}^{\mathcal{S}})$ )
2:   if  $(\mathcal{F}^{\mathcal{S}} = \emptyset)$  then
3:     return  $\emptyset$ 
4:   else
5:      $linkset := \emptyset$ 
6:     for  $(f, j) \in \mathcal{F}^{\mathcal{S}}$  do
7:       if  $f \in children \wedge j \in children$  then
8:          $\mathcal{F}^{\mathcal{S}} := \mathcal{F}^{\mathcal{S}} \setminus (f, j)$ 
9:          $linkset := linkset \cup \{link(f, j)\}$ 
10:      end if
11:    end for
12:    return  $linkset$ 
13:  end if
14: end procedure

```

Figure 3.8: Adding synchronization dependencies to the block structure

3.5.2 The complaint handling process

To illustrate the algorithm as a whole, we will further process the translation of the BPMN model presented in 3.2.2. First the set of dominators, loop headers and follow sets is determined and is given in table 3.2. Note that we use the set $\mathcal{F}^{\mathcal{N}}$ of relations instead of \mathcal{F} , since we do not want to process synchronization links at this moment. We can see that each last node of a specific level of nesting has an empty *FOLLOW*(*n*)-set, all other nodes have one element in the *FOLLOW*(*n*)-set. A loop can be identified between the nodes *xor*₃ and *xor*₄.

With these concepts, the algorithm can be executed and results in a structured composition. In Figure 3.9 we show this structured composition. In total, two dummy nodes have been added to support the loop in the structured model. Also the synchronization link detected earlier is added to the structured com-

Table 3.2: Dominators, loop headers and follow sets

Node n	$DOM(n)$	$HEAD(n)$	$FOLLOW(n)$
<i>start</i>	-	-	<i>t</i> ₁
<i>t</i> ₁	<i>start</i>	-	<i>fork</i> ₁
<i>fork</i> ₁	<i>t</i> ₁	-	<i>join</i> ₂
<i>t</i> ₂	<i>fork</i> ₁	-	<i>xor</i> ₁
<i>xor</i> ₁	<i>t</i> ₂	-	<i>xor</i> ₂
<i>e</i> ₁	<i>xor</i> ₁	-	<i>t</i> ₃
<i>t</i> ₃	<i>e</i> ₁	-	-
<i>e</i> ₂	<i>xor</i> ₁	-	-
<i>xor</i> ₂	<i>xor</i> ₁	-	<i>fork</i> ₂
<i>fork</i> ₂	<i>xor</i> ₂	-	<i>t</i> ₄
<i>t</i> ₄	<i>fork</i> ₂	-	-
<i>t</i> ₅	<i>fork</i> ₁	-	<i>join</i> ₁
<i>join</i> ₁	<i>t</i> ₅	-	<i>xor</i> ₃
<i>xor</i> ₃	<i>join</i> ₁	-	<i>t</i> ₆
<i>t</i> ₆	<i>xor</i> ₃	<i>xor</i> ₃	<i>t</i> ₇
<i>t</i> ₇	<i>t</i> ₆	<i>xor</i> ₃	<i>xor</i> ₄
<i>xor</i> ₄	<i>t</i> ₇	<i>xor</i> ₃	-
<i>join</i> ₂	<i>fork</i> ₁	-	<i>t</i> ₈
<i>t</i> ₈	<i>join</i> ₂	-	<i>end</i>
<i>end</i>	<i>t</i> ₈	-	-

position again. We see the synchronization link with source *fork*₂ and target *join*₁ in the scope of the and-construct.

3.6 BPEL syntax

The generated structured composition represent a business process in BPEL, but has a different notation than the BPEL syntax presented before. Therefore a last translation step is needed. Each element of the structured composition can be translated to BPEL syntax. For example a SEQ-node can be translated to a <sequence> element, an AND-node can be translated to a <flow> element etc. In Table 3.3 the constructs used in the structured composition are mapped to the constructs of the BPEL syntax. In the last column, the extra information which is needed for a valid BPEL syntax is stated. This information cannot be gathered from the BPMN model directly and should be entered manually by an end-user.

In other literature research, mapping a deferred choice is often done by using the <switch> construct. Since this construct is deprecated in the BPEL 2.0 specification, we use the <if> construct instead. In Table 3.3 we also see two

possible implementations for the XOR-construct. If a race condition occurs between multiple events, the choice of the event is modeled by a `<pick>` construct. When choice is not made based on events but on data, the `<if>` construct will be used.

Table 3.3: Structured composition to BPEL syntax

Structured composition	BPEL construct	Information needed
task	<code><invoke></code>	partnerLink, portType, operation
SEQ	<code><sequence></code>	
AND	<code><flow></code>	
XOR	<code><if></code> , <code><condition></code> <code><elseif></code> , <code><condition></code> <code><else></code> <code><pick></code>	Condition Condition
LOOP	<code><repeatUntil></code>	Condition
LINK	<code><links></code> , <code><link></code> <code><sources></code> , <code><source></code> <code><targets></code> , <code><target></code>	Name linkName linkName

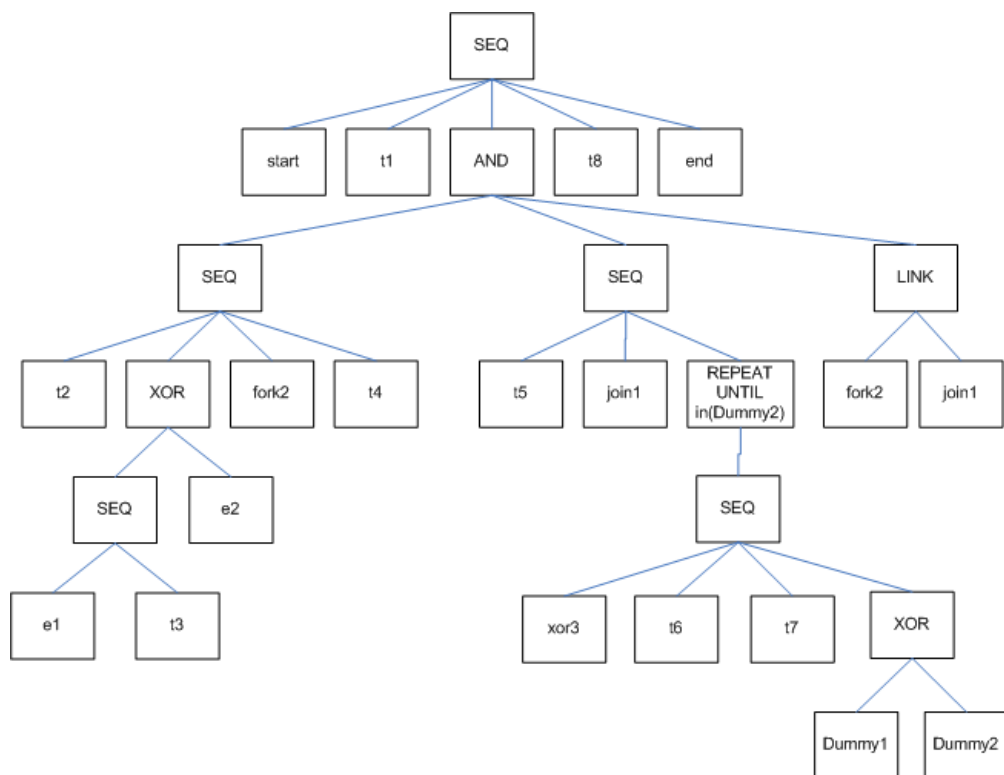


Figure 3.9: The structured composition for the complaint handling process

Chapter 4

An Eclipse plugin

The algorithm presented in chapter 3 is implemented as a plugin in the Eclipse development environment. We have chosen for this open source environment because there already exists an implementation for translating BPMN to BPEL by [GB08] and within Eclipse it is possible to model processes according to the BPMN standard.

4.1 Creating BPMN models in Eclipse

Before the translation from BPMN to BPEL can be developed, first a tool should be available for modeling business processes in the BPMN standard. We have chosen for the plugin available for Eclipse because we are going to develop the translation process also in Eclipse. For modeling business processes according to the BPMN standard, the ‘STP BPMN modeler’ is needed. This modeler can be installed by installing the Eclipse plugins for ‘Graphical Editors and Frameworks’, ‘Models and Model Development’ and the ‘STP BPMN modeler’.

In Eclipse, a BPMN model can be drawn by selecting the needed constructs and connect them with arrows. In Figure 4.1 an example of a BPMN model drawn in Eclipse is given. However, the plugin does not check the model for correctness. A user can create an incorrect or incomplete model which can not be translated by the algorithm. This restriction of the plugin should be taken into account while creating models. The user must check the constraints of a well-formed BPD, which we mentioned before, by himself.

4.2 Running the algorithm

The algorithm discussed in chapter 3 is implemented by extending an existing implementation by Eshuis. His implementation requires a dependency graph

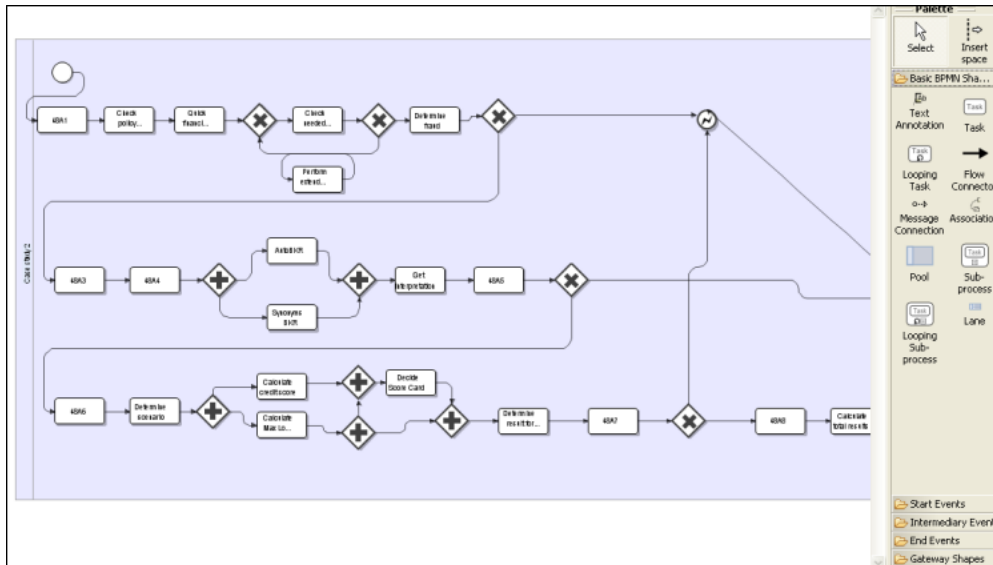


Figure 4.1: The BPMN modeling tool in Eclipse

represented by nodes and edges as input model and translates this to a structured composition. In Figure 4.2 the plugin is described in a flow chart. The implementation follows the algorithm presented in [EG09] but has still some functionalities unimplemented like loop constructs. The existing implementation generates a structured tree and eventually an ESML export. In our implementation, we will adjust both functionalities to BPEL generation.

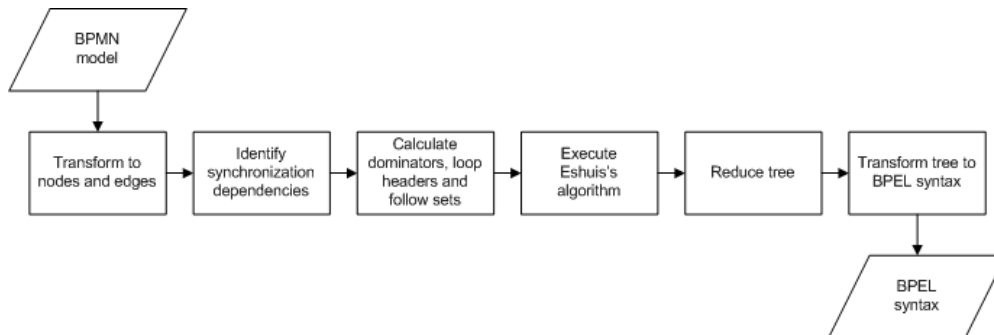


Figure 4.2: Graphical flowchart how the plugin works

First the input requirement should be adjusted to require a BPMN model as input instead of a dependency graph. The BPMN model should be analyzed and expressed in terms of nodes and edges. In this way, the BPMN model can be seen as an extended type of a dependency graph. Therefore the current implementation can be reused and only needs some adjustments which will be

discussed later. To prevent losing information about the BPMN model, extra information is added to the concept of a node. A node can have different types like a task, a parallel fork gateway etc.

A new algorithm which is implemented is the algorithm to find synchronization dependencies. We presented this algorithm in Figure 3.4. The algorithm is implemented as a function and searches for synchronization dependencies. If synchronization dependencies are found, these edges are moved to a temporary variable. These edges are used again when the algorithm of Eshuis has been executed and the structured process composition is generated.

Some functionalities are improved in the implementation. Loops for example were not covered in the way the algorithm of Eshuis described, but were parsed as sequences. We adjusted the implementation such that loops with one entry and one exit point are implemented according to the algorithm of Eshuis. A dummy node is inserted and the loop is repeated until the dummy node is executed. Some minor bugs we encountered during the adjustment are fixed and the implementation is extended with the algorithm for finding synchronization dependencies. User input in the implementation is not necessary anymore, since the function 'ask for feedback by unclearness' can be replaced with the information available from the input model.

The structured composition, which is one of the generated results of the Eclipse plugin, is also exported in BPEL syntax. This BPEL syntax can be used by Oracle's BPEL Process Manager, which will be discussed later. To ensure valid BPEL syntax, adding extra information is necessary. WSDL data needs to be added for defining namespaces, partnerlinks, porttypes and possible variables. Currently, this information is hardcoded in the implementation.

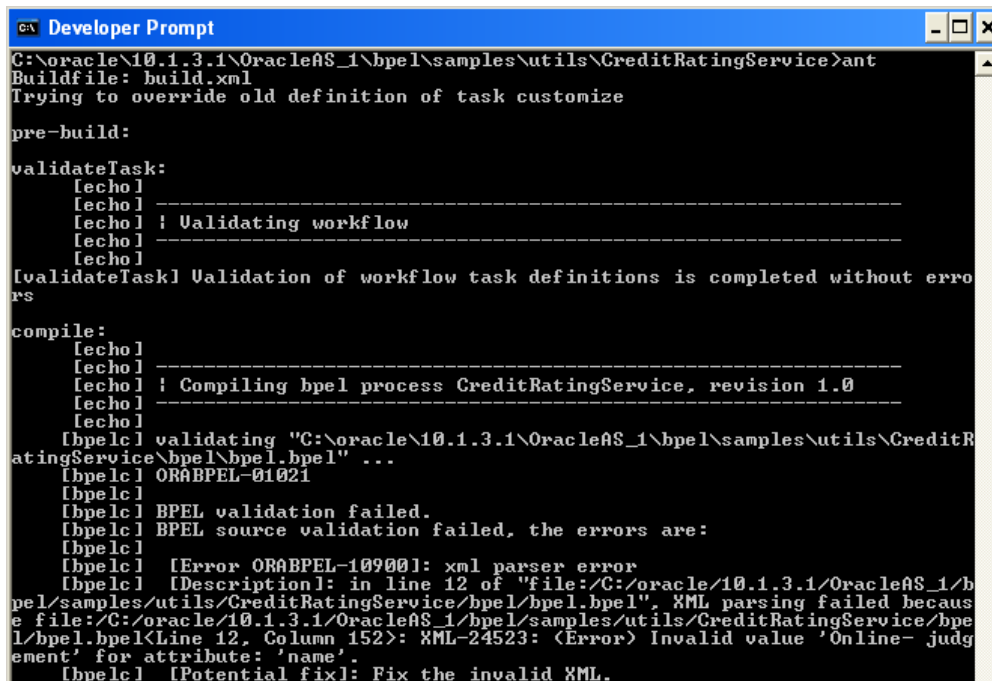
The Eclipse plugin can be executed by running it from the commandline or running the program via Eclipse. The initial goal was to adjust the interface of the BPMN modeling tool such that the plugin is directly executed. Because of unforeseen problems we are not able to solve this issue. However, when running the plugin via commandline with the filename of the BPMN model as argument, the plugin works fine. This small workaround is acceptable for our master project.

4.3 Plugin evaluation

Development of the plugin was done according to the spiral model of Barry Boehm [Boe88]. First basic BPMN models must be mapped correctly to a BPEL process, then the algorithm for finding synchronization dependencies was

implemented and finally some adjustments regarding loops and multiple gateways were implemented. Each iteration, the plugin was tested with both case studies we present in chapter 5. Testing with two real world business processes showed us some flaws in the plugin which we didn't identify with examples from literature. Loops with an exit gateway with more than two outgoing arcs for example, were not processed correctly by the plugin. At this time, we have one known problem at hand, where the structured composition cannot be created according to the algorithm, because multiple decision gateways belong to one or more merge gateways. Case study 2 has such a situation and is discussed in section 5.2.1.

The generated BPEL syntax is validated with help of the Oracle BPEL Process Manager. This tool validates the BPEL syntax and offers a process manager interface to simulate the modeled processes and check whether the BPEL process represent the BPMN model in the right way.



```

Developer Prompt
C:\oracle\10.1.3.1\OracleAS_1\bpel\samples\utis\CreditRatingService>ant
Buildfile: build.xml
Trying to override old definition of task customize

pre-build:
validateTask:
  [echo]
  [echo] -----
  [echo] ! Validating workflow
  [echo] -----
  [echo]
[validateTask] Validation of workflow task definitions is completed without errors

compile:
  [echo]
  [echo] -----
  [echo] ! Compiling bpel process CreditRatingService, revision 1.0
  [echo]
  [echo]
[bpelc] validating "C:\oracle\10.1.3.1\OracleAS_1\bpel\samples\utis\CreditRatingService\bpel\bpel.bpel" ...
[bpelc] ORABPEL-01021
[bpelc]
[bpelc] BPEL validation failed.
[bpelc] BPEL source validation failed, the errors are:
[bpelc]
[bpelc] [Error ORABPEL-10900]: xml parser error
[bpelc] [Description]: in line 12 of "file:/C:/oracle/10.1.3.1/OracleAS_1/bpel/samples/utis/CreditRatingService/bpel/bpel.bpel", XML parsing failed because file:/C:/oracle/10.1.3.1/OracleAS_1/bpel/samples/utis/CreditRatingService/bpel/bpel.bpel<Line 12, Column 152>: XML-24523: <Error> Invalid value 'Online-judgment' for attribute: 'name'.
[bpelc] [Potential fix]: Fix the invalid XML.

```

Figure 4.3: Example of an error message of Oracle's BPEL Process Manager

To validate the BPEL syntax, we use the developer prompt of Oracle's BPEL Process manager. By loading the generated BPEL syntax into this developer prompt, the process is built and can be executed with the Process Manager. By building the process, the syntax is validated according to the BPEL specification. In some cases, user input is necessary to judge an error message of

the validation done, because Oracle uses BPEL 1.1 for validating the syntax and we use BPEL 2.0 in our master project. In Figure 4.3 an example of the error reporting of BPEL Process Manager is given. When the generated BPEL syntax is validated by the Process Manager, we can conclude that our generated process has a correct syntax. Simulation in the process manager is necessary for concluding if the generated process is indeed representing the original process as described in the BPMN model. In Figure 4.4 a screenshot of the execution environment is given.

Dashboard			
BPEL Processes		Instances	Activities
Deployed BPEL Processes		In-Flight BPEL Process Instances 1 - 3	
Name	Instance	BPEL Process	Last Modified ↑
CreditRatingService	13 : Instance #13 of StarLoan	StarLoan (v. 1.0)	7/22/06 10:03:36 AM
LoanFlowPlus	11 : Loan Flow Plus- John Smith	LoanFlowPlus (v. 1.0)	7/22/06 10:03:36 AM
StarLoan	9 : Instance #9 of LoanFlowPlus	LoanFlowPlus (v. 1.0)	7/22/06 10:01:22 AM
TaskActionHandler			
TaskManager			
UnitedLoan			
Recently Completed BPEL Process Instances (More...)			
	✓ 14 : Instance #14 of UnitedLoan	UnitedLoan (v. 1.0)	7/22/06 10:03:34 AM
	✓ 12 : Instance #12 of CreditRatingService	CreditRatingService (v. 1.0)	7/22/06 10:03:33 AM
	✓ 10 : Instance #10 of CreditRatingService	CreditRatingService (v. 1.0)	7/22/06 10:01:20 AM
	7 : Instance #7 of myCreditFlow	myCreditFlow (v. 1.0)	7/21/06 4:23:28 PM
	8 : Instance #8 of CreditRatingService	CreditRatingService (v. 1.0)	7/21/06 4:23:28 PM
<input type="button" value="Deploy New Process"/>			

Figure 4.4: Screenshot of the execution environment

Chapter 5

Case studies

In this chapter, we discuss two case studies which are gathered at Logica. We will describe the case studies, translate them from BPMN to BPEL manually according to the algorithm presented in chapter 3 and finally compare these results with the results of the developed plugin (chapter 4).

Both case studies are based on real business processes of financial companies in the Netherlands. For reasons of confidentiality we are not mentioning the names of these companies. In both situations, the software architecture used within the company has a high number of orchestration-layers. Each component invokes other services and uses the response for further processing. Such situations match the purpose of BPEL and the case studies are therefore quite useful for evaluating our algorithm.

5.1 Loan request for an insurance company

Our first case study represents a loan request of a customer via a direct or intermediate channel. Both channels use the same underlying process. After the request is received and saved in the database, the request is judged via an online interface. This interface can respond with three different cases. The request can be accepted, declined or referred.

When a request needs to be referred, it can be due to a fraud suspicion, synonyms misunderstanding or technical error. Depending on the reason, an activity is started to solve this issue or check for more information. With this information, the loan request can again be judged via the online interface.

If a loan request is accepted, the process will inform the customer and wait for an acceptance. The accepted offer is processed further in concurrency; the contract is prepared while documents are gathered. When all documents are gathered and processed, the contract is finished and the loan is provided.

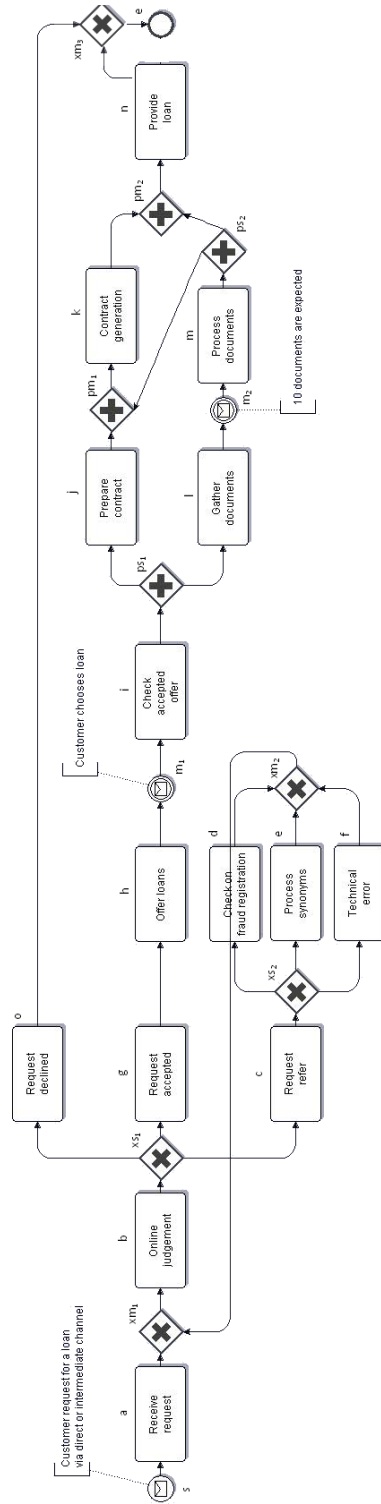


Figure 5.1: Case study 1: Loan request for an insurance company

5.1.1 Manual translation

In this paragraph we manually translate the case study described above, according to the presented approach, to a structured composition representing the BPEL process. First the algorithm for finding synchronization dependencies is executed. After this, the adjusted algorithm of Eshuis is executed and the structured composition is presented. We compare the result of the manual translation according to the presented approach, with our ideal result in mind. The result of this manual translation is also used to compare with the result of the implementation described in chapter 4.

Before the algorithm for finding synchronization dependencies is executed, we first determine if a direct relation between a parallel fork gateway and a parallel join gateway exists. If not, the algorithm can be skipped because of the restriction we presented in section 3.2. In Table 5.1 the *before(g)*-sets and *after(g)*-sets are given for the parallel gateways.

Table 5.1: *before(g)*- and *after(g)*-set for gateways of case study 1

Gateway:	<i>before(g)</i> -set	<i>after(g)</i> -set
ps_1	-	$j, pm_1, k, l, m_2, m, ps_2, pm_2$
ps_2	ps_1, l, m_2, m	pm_1, k, pm_2
pm_1	ps_1, j, l, m_2, m, ps_2	k, pm_2
pm_2	$ps_1, j, pm_1, k, l, m_2, m, ps_2$	-

With the sets in Table 5.1 we can execute the algorithm for finding synchronization dependencies. We see that ps_1 matches to pm_2 and that no match exists between ps_2 and pm_1 . Because a direct relation between those two gateways exists, we can identify the relation (ps_2, pm_1) as a synchronization link.

Next we calculate the concepts of dominators, loop headers and follow sets. In Table 5.2 the result of these calculations are summarized.

The algorithm delivers us a structured description of the process in the BPEL grammar we formalised in section 3.4. The process can be described as:

$$P = \langle s, a, \text{repeat } P_x \text{ until in } (xs_1^{dum}), \text{xor}\{o, \langle g, h, m_1, i, \text{ and } \langle j, pm_1, k \rangle, \langle l, m_2, m, ps_2 \rangle, \{\text{link}(ps_2, pm_1)\}\}, pm_2, n \rangle, xm_3, e \rangle$$

$$P_x = \langle b, \text{xor}\{xs_1^{dum}, \langle c, \text{xor}\{d, e, f\}, xm_2 \rangle\} \rangle$$

This process is visualised in the structured composition in Figure 5.2. The structured composition matches the ideal translation we have in mind, except

Table 5.2: Dominators, loop headers and follow sets for case study 1

Object (O)	$DOM(O)$	$HEAD(O)$	$FOLLOW(O)$
s	—	—	a
a	s	—	xm_1
xm_1	a	<i>Loopheader</i>	xs_1^{dum}
b	xm_1	xm_1	xs_1
xs_1	b	xm_1	c
c	xs_1	xm_1	xs_2
xs_2	c	xm_1	xm_2
d	xs_2	xm_1	—
e	xs_2	xm_1	—
f	xs_2	xm_1	—
xm_2	xs_2	xm_1	xm_1
g	xs_1^{dum}	—	h
h	g	—	m_1
m_1	h	—	i
i	m_1	—	ps_1
ps_1	i	—	pm_2
j	ps_1	—	pm_1
pm_1	j	—	k
k	pm_1	—	—
l	ps_1	—	m_2
m_2	l	—	m
m	m_2	—	ps_2
ps_2	m	—	—
pm_2	ps_1	—	n
n	pm_2	—	—
o	xs_1^{dum}	—	—
xm_3	xs_1^{dum}	—	e
e	xm_3	—	—

for the merge-gateways. These gateways are not needed in the structured composition, because they do not represent an activity. For Figure 5.2, the nodes xm_2 , pm_2 and xm_3 can be removed.

5.1.2 Translating via the plugin

When we translate the same case study with the plugin we developed, a structured composition tree given in Figure 5.4 is generated. This tree is similar to the graphical structured composition of the manual composition, except for some elements which represent the join or merge gateways. Some of these tasks are eliminated in the plugin, since they do not represent a real activity. There is some future work in removing these gateways, since not all gateways are re-

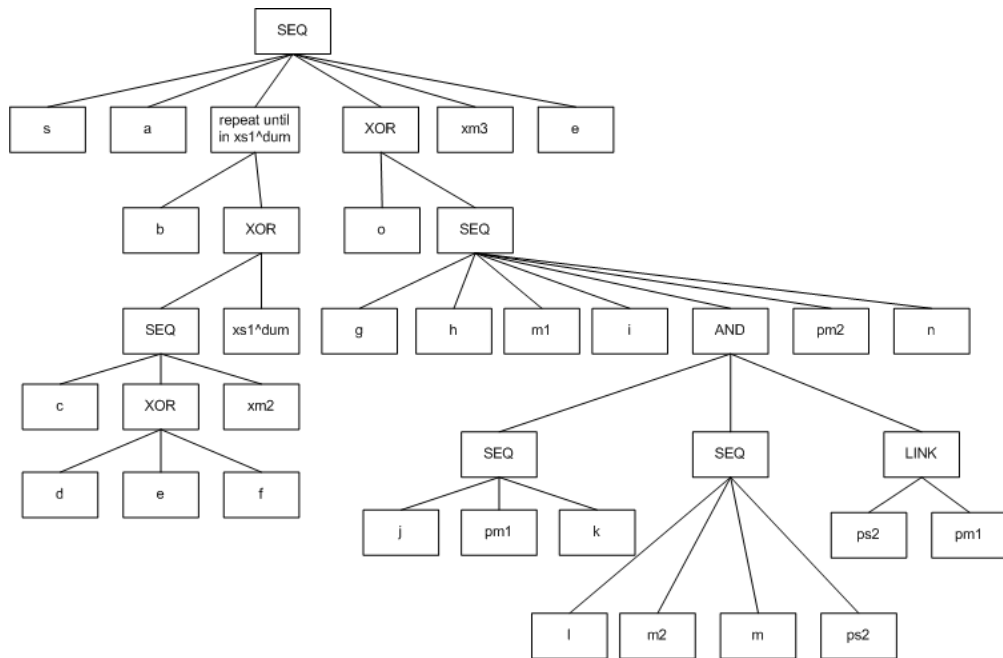


Figure 5.2: Structured composition of case study 1

moved yet by the plugin and in the manual translation these gateways remains in the structured composition.

```

<sequence>
  <invoke name="start" partnerLink="client" portType="tns:PaymentProcessorServiceCallback" oper
  <invoke name="Receive request" partnerLink="client" portType="tns:PaymentProcessorServiceCall
  <sequence>
    <invoke name="XOR-merge-2" partnerLink="client" portType="tns:PaymentProcessorServiceCallbe
  <sequence>
    <invoke name="Online judgement" partnerLink="client" portType="tns:PaymentProcessorServic
    <if min_duration="POD">
      <invoke name="XOR-split-3-DUMMY" partnerLink="client" portType="tns:PaymentProcessorSer
    <sequence>
      <invoke name="Request refer" partnerLink="client" portType="tns:PaymentProcessorServi
      <if min_duration="POD" />
      <if min_duration="POD">
        <sequence>
          <invoke name="Technical error" partnerLink="client" portType="tns:PaymentProcess

```

Figure 5.3: Part of the BPEL syntax for case study 1

The plugin is also able to export the structured composition tree to BPEL syntax as specified in the official BPEL specification. This generated code can be directly imported into Oracle BPEL Process Manager for example. In Figure 5.3 a preview of the BPEL syntax is given.

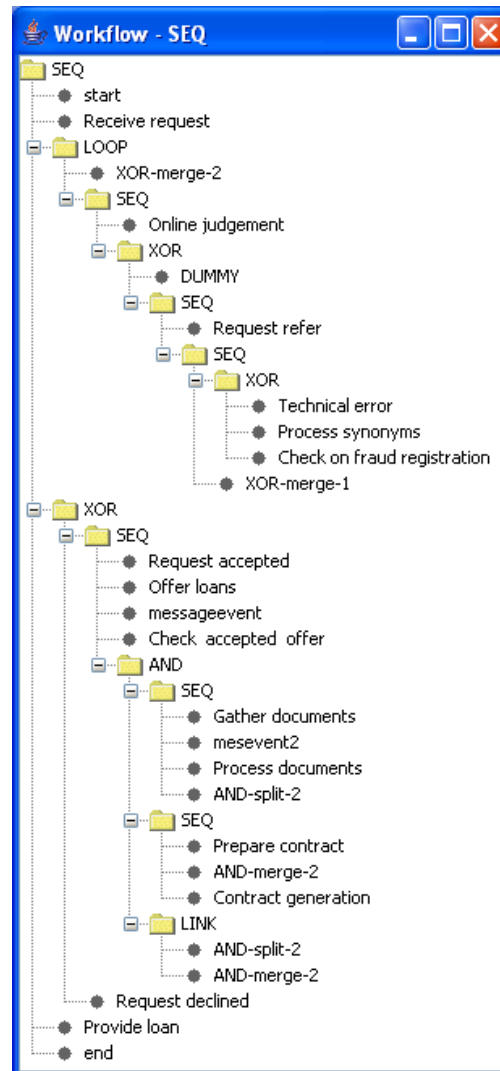


Figure 5.4: Structured composition tree of case study 1

5.2 Total credit risk calculation

Our second case study represents the process of a total credit risk calculation. The process is adapted from an insurance company which is a client of Logica. The process represents an orchestration layer between the end-user applications and functional services which are implemented in the software architecture of the company. In Figure 5.5 the BPMN model of this orchestration layer is given.

A request for a credit risk calculation is done by a customer or an insurance agency. After this request, some activities are invoked in sequence. First the internal activity 48A.1 is invoked and the policy restrictions are checked by another invocation. A quick financial check is then executed. The result of this check is evaluated to determine if all necessary information is obtained. If not, an extended check is performed and evaluated again. This loop continues until all necessary information is obtained.

The complete set of information is then evaluated and the process determines the risk on fraud. In case of a high risk on fraud, the process returns an error to the initiator of the process. In other cases, the orchestration layer invokes more internal activities and then performs a credit check. Both the service ‘AutoBKR’ and ‘SynonymsBKR’ are invoked in parallel. The results of both services are combined and interpreted when both services are finished. Internal activity 48A.5 evaluates this interpretation and decides if more checks need to be executed or a result can already be given to internal activity 48A.10.

In case more checks need to be executed, a scenario is determined and handled concurrently by two other processes. The credit score and the max loan capacity are calculated. After calculating the credit score, a score card needs to be decided, but this can only be done when the max loan capacity is also known. Therefore a synchronization link is introduced in the model. The result is determined for the scenario and the process might be terminated because of an error message, or the total results are calculated and processed as either an error or a positive result. The positive result will be processed further by internal activity 48A.10 and will be returned to the initiator of the process.

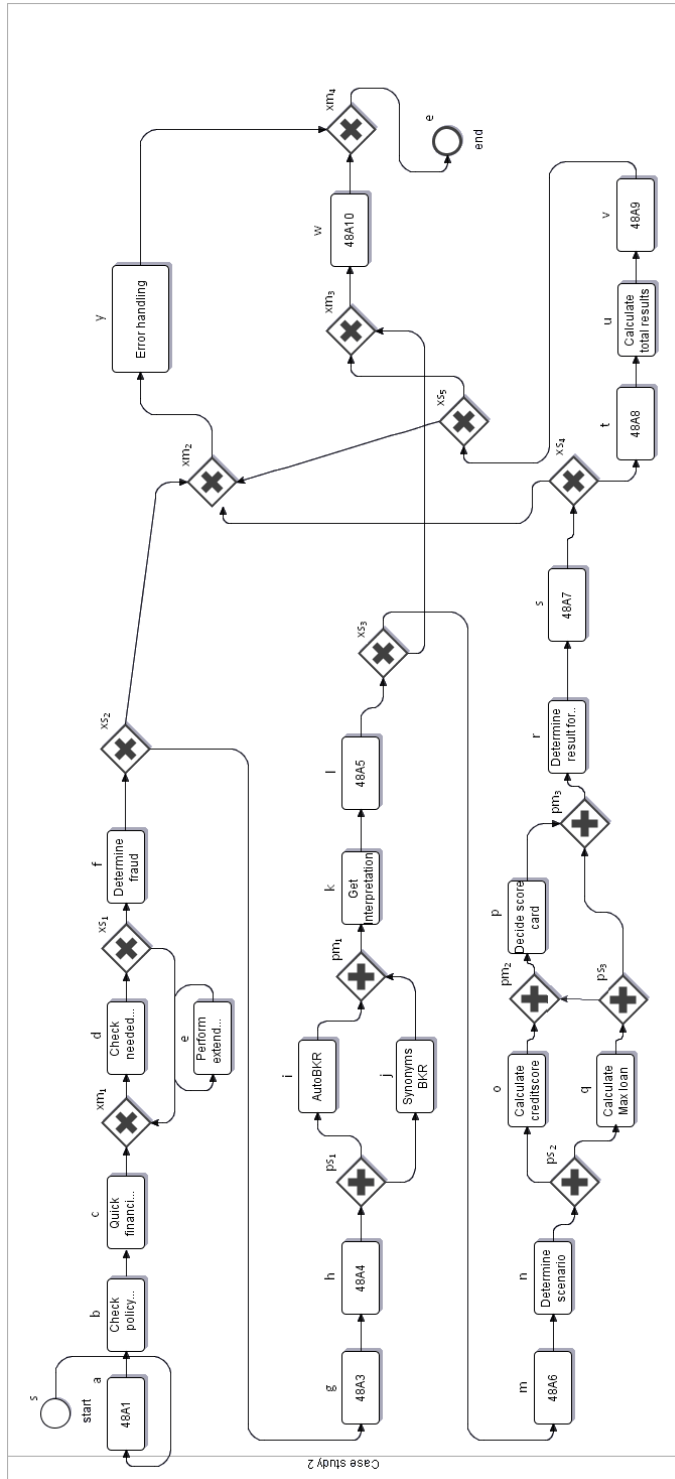


Figure 5.5: Case study 2: total credit risk calculation

5.2.1 Manual translation

In this paragraph we manually translate the case study described above to a structured composition representing the BPEL process. First the algorithm for finding synchronization dependencies is executed. After this, the adjusted algorithm of Eshuis is executed and the structured composition is presented. Again we compare the result of the manual translation according to the presented approach, with our ideal result in mind. The result of this manual translation is also used to compare with the result of the implementation described in chapter 4.

Before the algorithm for finding synchronization dependencies is executed, we first determine if a direct relation between a parallel fork gateway and a parallel join gateway exists. If not, the algorithm can be skipped because of the restriction we presented in section 3.2. In Table 5.3 the *before(g)*-sets and *after(g)*-sets are given for the parallel gateways.

Table 5.3: *before(g)*- and *after(g)*-set for gateways of case study 2

Gateway:	<i>before(g)</i> -set	<i>after(g)</i> -set
ps_1	-	i, j, pm_1
ps_2	-	$o, pm_2, p, q, ps_3, pm_3$
ps_3	ps_2, q	pm_2, p, pm_3
pm_1	ps_1, i, j	-
pm_2	ps_2, o, q, ps_3	p, pm_3
pm_3	$ps_2, o, pm_2, p, q, ps_3$	-

With the sets in Table 5.3 we can execute the algorithm for finding synchronization dependencies. We see that ps_1 matches to pm_1 , ps_2 matches to pm_3 and that no match exists between ps_3 and pm_2 . Because a direct relation between those two gateways exists, we can identify the relation (ps_3, pm_2) as a synchronization link.

Next we calculate the concepts of dominators, loop headers and follow sets. In Table 5.4 the result of these calculations are summarized. The algorithm delivers a structured description of the process in the BPEL grammar we formalised in section 3.4. The process can be described as:

$$P = \langle s, a, b, c, \text{repeat } P_x \text{ until in } (xs_1^{dum}), f, \text{xor}\{P_y, xs_2^{dum}\}, xm_2, y, xm_4, e \rangle$$

$$P_x = \langle d, \text{xor}\{e, xs_1^{dum}\} \rangle$$

$$P_y = \langle g, h, \text{and}\{i, j\}, pm_1, k, l, \text{xor}\{P_z\}, xm_3, w \rangle$$

$$P_z = \langle xs_3^{dum}, \langle m, n, \text{and}\{\langle o, pm_2, p \rangle, \langle q, ps_3 \rangle, \{\text{link}(ps_3, pm_2)\}\}, pm_3, r, s, \text{xor}\{xs_4^{dum}, \langle t, u, v, \text{xor}\{xs_5^{dum1}, xs_5^{dum2}\} \rangle \rangle \rangle$$

Table 5.4: Dominators, loop headers and follow sets for case study 2

Obj. (<i>O</i>)	<i>DOM</i> (<i>O</i>)	<i>HEAD</i> (<i>O</i>)	<i>FOLLOW</i> (<i>O</i>)	Obj. (<i>O</i>)	<i>DOM</i> (<i>O</i>)	<i>HEAD</i> (<i>O</i>)	<i>FOLLOW</i> (<i>O</i>)
<i>s</i>	-		<i>a</i>	<i>ps₂</i>	<i>n</i>		<i>pm₃</i>
<i>a</i>	<i>s</i>		<i>b</i>	<i>o</i>	<i>ps₂</i>		<i>pm₂</i>
<i>b</i>	<i>a</i>		<i>c</i>	<i>pm₂</i>	<i>o</i>		<i>p</i>
<i>c</i>	<i>b</i>		<i>xm₁</i>	<i>p</i>	<i>pm₂</i>		-
<i>xm₁</i>	<i>c</i>	<i>loopnode</i>	<i>f</i>	<i>q</i>	<i>ps₂</i>		<i>ps₃</i>
<i>d</i>	<i>xm₁</i>	<i>xm₁</i>	<i>xs₁</i>	<i>ps₃</i>	<i>q</i>		-
<i>xs₁</i>	<i>d</i>	<i>xm₁</i>	-	<i>pm₃</i>	<i>ps₂</i>		<i>r</i>
<i>e</i>	<i>xs₁</i>	<i>xm₁</i>	-	<i>r</i>	<i>pm₃</i>		<i>s</i>
<i>f</i>	<i>xs₁</i>		<i>xs₂</i>	<i>s</i>	<i>r</i>		<i>xs₄</i>
<i>xs₂</i>	<i>f</i>		<i>xm₂; xm₄</i>	<i>xs₄</i>	<i>s</i>		-
<i>g</i>	<i>xs₂</i>		<i>h</i>	<i>t</i>	<i>xs₄</i>		<i>u</i>
<i>h</i>	<i>g</i>		<i>ps₁</i>	<i>u</i>	<i>t</i>		<i>v</i>
<i>ps₁</i>	<i>h</i>		<i>pm₁</i>	<i>v</i>	<i>u</i>		<i>xs₅</i>
<i>i</i>	<i>ps₁</i>		-	<i>xs₅</i>	<i>v</i>		-
<i>j</i>	<i>ps₁</i>		-	<i>xm₂</i>	<i>xs₂</i>		<i>y</i>
<i>pm₁</i>	<i>ps₁</i>		<i>k</i>	<i>xm₃</i>	<i>xs₃</i>		<i>w</i>
<i>k</i>	<i>pm₁</i>		<i>l</i>	<i>w</i>	<i>xm₃</i>		-
<i>l</i>	<i>k</i>		<i>xs₃</i>	<i>y</i>	<i>xm₂</i>		-
<i>xs₃</i>	<i>l</i>		<i>xm₃</i>	<i>xm₄</i>	<i>xs₂</i>		<i>e</i>
<i>m</i>	<i>xs₃</i>		<i>n</i>	<i>e</i>	<i>xm₄</i>		-
<i>n</i>	<i>m</i>		<i>ps₂</i>				

This process is visualised in the structured composition in Figure 5.6. The structured composition matches the ideal translation we have in mind globally, except for the merge-gateways. The merge-gateways are not needed in the structured composition, because they do not represent an activity. For Figure 5.6, the nodes *pm₁*, *pm₃*, *xm₂*, *xm₃* and *xm₄* can be removed.

5.2.2 Translating via the plugin

Translating the second case study via the plugin was more difficult than for the first case study. Problems occurred when processing the *xm₂* decision gateway. Both the algorithm and the plugin must be improved at this point, which we suggest as future work. In Figure 5.8 the structured composition which is currently the result of the mapping is given. This result should be improved on two aspects. First the synchronization dependency algorithm must only be executed within one SESE-region and finally decision and merge gateways must be investigated and improved.

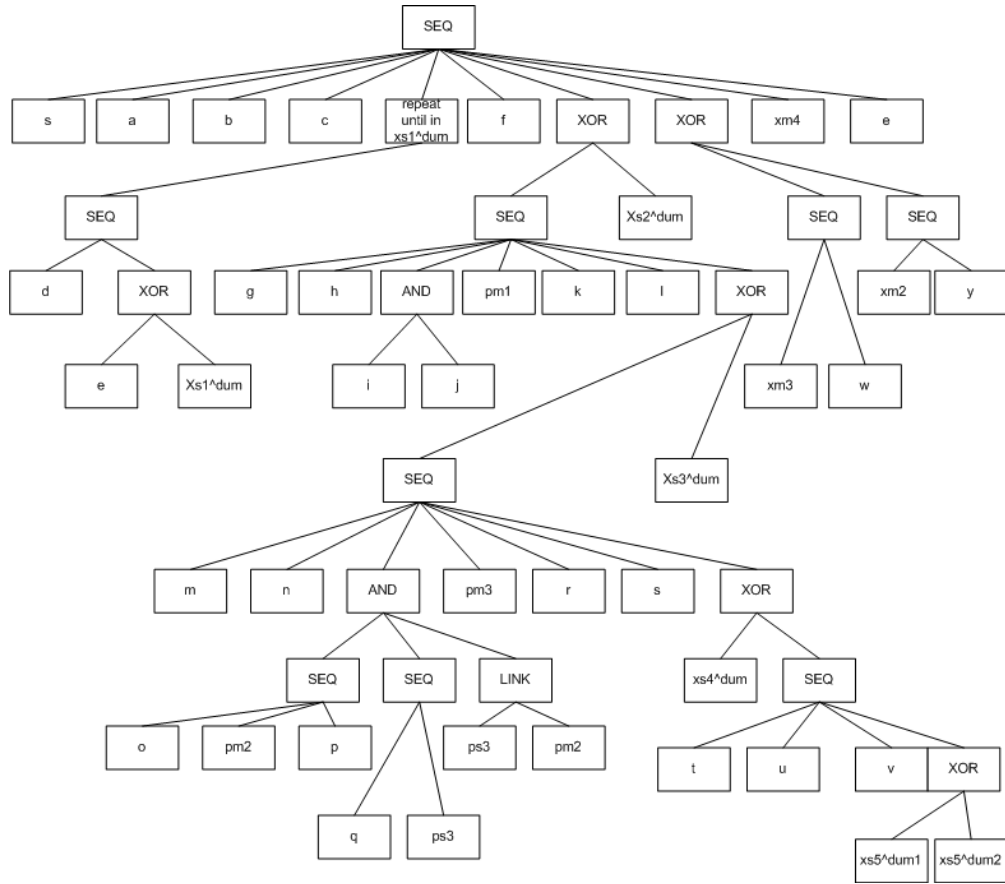


Figure 5.6: Structured composition of case study 2

```

</partnerLinks>
<sequence>
  <invoke name="start" partnerLink="client" portType="tns:PaymentProcessorServiceCallback" oper
  <invoke name="48A1" partnerLink="callback" portType="tns:PaymentProcessorServiceCallback" oper
  <invoke name="Check policy restrictions" partnerLink="client" portType="tns:PaymentProcessorS
  <invoke name="Quick financial check" partnerLink="client" portType="tns:PaymentProcessorServ
  <sequence>
    <invoke name="XOR-merge-1" partnerLink="client" portType="tns:PaymentProcessorServiceCallb
    <sequence>
      <invoke name="Check needed information" partnerLink="client" portType="tns:PaymentProces
      <if min_duration="POD">
        <invoke name="Perform extended check" partnerLink="client" portType="tns:PaymentProces
        <invoke name="Determine fraud-DUMMY" partnerLink="client" portType="tns:PaymentProces
      </if>
    </sequence>
  </sequence>
</sequence>

```

Figure 5.7: Part of the BPEL syntax for case study 2

The plugin is also able to export the structured composition tree to BPEL syntax as specified in the official BPEL specification. This generated code matches the generated structured composition and therefore has the same problems as

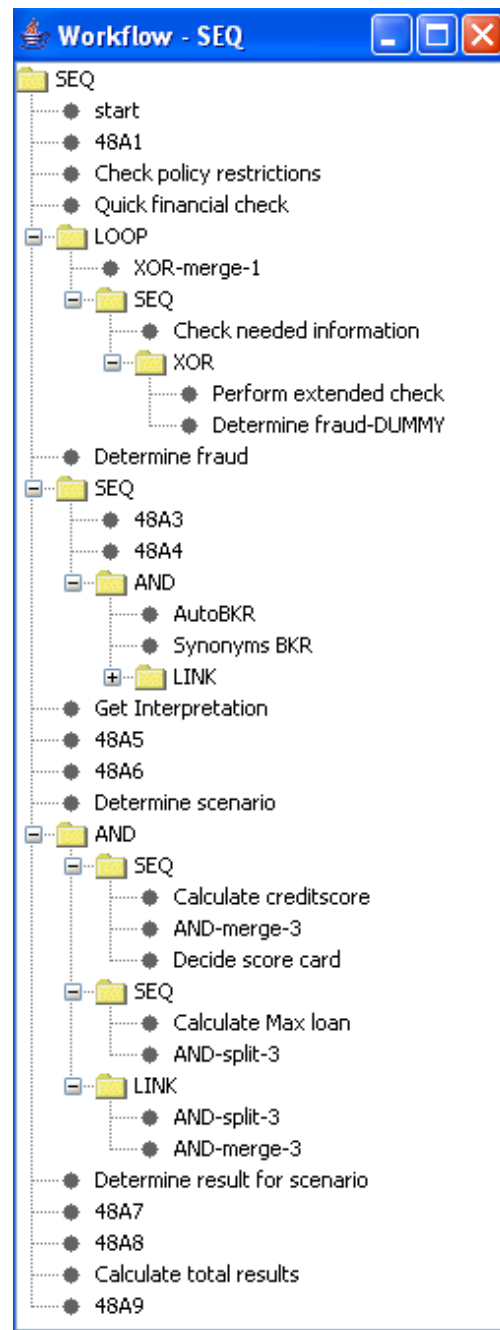


Figure 5.8: Structured composition tree of case study 2

the translation to the structured composition. In Figure 5.7 a part of the BPEL syntax is given.

5.3 Conclusion

The purpose of introducing two case studies is the evaluation of both our algorithm and the implementation in Eclipse. The first case study was known at forehand and is used during the development of the mapping discussed in chapter 3. The second case study is used after we described the mapping, but resulted in some adjustments to our algorithm. An example of such an adjustment is the extension of the algorithm for finding synchronization dependencies with so called SESE-regions.

We also discovered that our implementation currently has some differences with the algorithm in the field of multiple subsequent decision and merge gateways. A manual translation of the case study results in a corresponding BPEL model, but a translation via the plugin has some issues as described in the section of case study 2.

We can conclude that our algorithm can generate BPEL models from input BPMN models which satisfy the restrictions of a well-formed BPD. In chapter 6 we evaluate these restrictions and will see that some of the restrictions can be relaxed.

Chapter 6

Evaluation of the mapping

In this chapter, we will evaluate our own mapping presented in chapter 3. We will discuss the strengths and weaknesses of the presented mapping and also the relation with other strategies and implementations. Because our algorithm uses a well-formed BPD as inputmodel, we will evaluate the restrictions of a well-formed BPD to investigate if we can also process more unstructured input models. In the last section of this chapter, we will propose some future work for our own algorithm based on the evaluation done.

We wanted to extend an existing algorithm presented by Rik Eshuis to make it suitable for mapping BPMN to BPEL. In chapter 2 we discussed different strategies and concluded that combining the strengths of different approaches would result in the best mapping. The developed algorithm works with structured components, but can also identify relations as synchronization dependencies. Control links, which are the main constructs of other strategies, are used only when this is most appropriate. The existing algorithm presented by Rik Eshuis is adjusted to make it suitable for mapping most BPMN models to BPEL and implemented as a plugin for Eclipse.

6.1 Strengths & Weaknesses

In chapter 2 we evaluated different strategies and mentioned some strengths and weaknesses of each approach. In section 6.2 we will compare our approach with these strategies. Before we can compare our own approach with these strategies, we will discuss the main strengths and weaknesses of our approach presented in chapter 3.

Our approach supports input models with non-structured components, except for arbitrary cycles. Not only structured input models can be processed, but also unstructured models which satisfy the well-formedness restrictions of a BPD can be processed. The approach can deal with all parallel gateways and

decision gateways with arbitrary topology, however the implemented Eclipse plugin still has some problems with these situations. The generated BPEL model is comprehensible and readable because it is largely structure based. The generated BPEL is extended with synchronization dependencies, which are also comprehensible as long as they are only used for synchronization purposes.

Our approach consists of different steps like preprocessing, calculating concepts as the immediate dominator, the original algorithm and the mapping to the BPEL syntax. These different steps make it more difficult to implement the approach in a plugin for example. The plugin which we developed during our master project has still some issues. Trivial models are mapped correctly, but complex models can result in some errors. Manual translation does not have these problems and results in a correct BPEL model.

Another important issue is the presence of merge gateways. In the original algorithm of Rik Eshuis, no separate merge gateways exist, since these are combined with normal tasks. As a result, these merge gateways are processed as tasks in our approach and cause unnecessary 'empty' BPEL constructs. The implementation can easily be adjusted to filter these merge gateways, but the algorithm should also be adjusted.

6.2 Relation with other strategies/implementations

First, we want to compare our own approach with the strategies mentioned in chapter 2. Our own approach scores better on readability than the link-based or event-based approaches. Since the structured components are translated to their subsequent BPEL constructs, the generated BPEL model can be maintained a lot easier. Also the concept of a repeatable construct like a loop is not supported in pure link-based strategies. Structure-based and event-based strategies do cover this construct. Our implementation also support loops with a single entry and a single exit point. Compared with structure-based strategies, our approach requires input models with less constraints and supports unstructured models.

We also want to compare our approach and implementation with the two opensource implementations mentioned in chapter 2, the BABEL-tool and the BPMN 2 BPEL Eclipse plugin. The BABEL-tool is not maintained anymore, but is an implementation often discussed in literature. The Eclipse plugin is a newly developed implementation, based on the global ideas of the BABEL-tool, but improved with SESE-regions. Both existing implementations and our own implementation generate BPEL models with a high level of readability, since they all identify structured blocks first. All implementations support

the presence of loops, but our own implementation only supports structured loops with one entry and one exit point. The BABEL-tool and Eclipse plugin translate unstructured loops via the event-action based approach, which is not implemented in our mapping. An advantage of our approach compared to the existing implementations is the algorithm for finding synchronization dependencies. Structured components which contain a synchronization dependency are mapped as structured components in our own algorithm. In other implementations, this structured component is translated as an unstructured component, which results in less readable output.

Also a short evaluation is done on a commercial tool available from Intalio¹. We are interested which strategy is used in this commercial tool and how synchronization links, (un)structured loops and multiple gateways are handled. Unfortunately, we are not able to investigate the underlying source code or extended documentation since both are not available. By modeling some models from literature and both case studies we discovered some strengths and weaknesses from the tool and can conclude that the implementation is focused mainly on structure identification. Because the modeling tool already checks the model on valid BPMN and restrictions set by the tool, all models which can be created within the tool can also be translated. The validation functionality of the tool is therefore it's main strength. Together with the possibility to generate correct WSDL information and the 'on the fly' translation of models, the tool has some great strengths. However, there are some general issues and restrictions with the tool. Both structured and unstructured loops can not be modeled and translated by the tool. Only tasks which represent one activity which is executed more than once can be modeled, but backward edges are not allowed. Synchronization links are also not processed correctly and are parsed as different flows of a parallel construct. The problem this solution has, is the fact that one task occurs several times in the generated BPEL code. The commercial tool has some nice benefits on validation and WSDL information, but has some significant restrictions in creating and translating models.

Next, we will evaluate the existing implementations by translating both case studies described in chapter 5. In this chapter, we already translated the process manually and via our own developed implementation. Both our own implementation and the BPMN2BPEL Eclipse plugin use a BPMN model created in Eclipse as input model. The BABEL-tool has another input format for describing a process. The commercial tool of Intalio is based on Eclipse, but has it's own environment to create the models. Both case studies need to be available in both formats, so both case studies can be translated by all implementations. We will now discuss both case studies in separate sections.

¹Intalio Designer: <http://bpms.intalio.com/tutorials/intalio-bpms-designer-5.0-modeling-tutorial-beginner.html>

6.2.1 Case study 1

Case study 1 is translated in chapter 5 by our own algorithm and implementation and matches the original input model as expected. It was not needed to adjust the input model for a correct translation. Both other implementations have some problems with translating our input model. First, the Eclipse plugin has some problems with the intermediate events in the original case study and only worked after we splitted our first decision gateway manually. The synchronization dependency is supported, but the whole parallel flow is translated as an unstructured component, which result in less readable code. Also the mapping uses empty constructs instead of invoke constructs. In Figure 6.1 a part of the generated BPEL syntax is given for the Eclipse plugin. The resulting process matches the adjusted input model, which we included in Appendix E.

```

        <bpws:empty name="Technical&#xd; &#xa;error"/>
      </bpws:elseif>
    </bpws:if>
  </bpws:sequence>
  <bpws:empty name="online&#xd; &#xa;judgement"/>
</bpws:sequence>
</bpws:while>
</bpws:sequence>
<bpws:if name="null-null">
  <bpws:empty name="declined"/>
<bpws:elseif>
  <bpws:sequence name="Request&#xd; &#xa;accepted-Provide&#xd; &#xa;loan">
    <bpws:empty name="Request&#xd; &#xa;accepted"/>
    <bpws:empty name="offer loans"/>
    <bpws:empty name="check &#xd; &#xa;accepted &#xd; &#xa;offer"/>
    <bpws:flow name="and1-and4">
      <bpws:links>
        <bpws:link name="null_null"/>
        <bpws:link name="Process&#xd; &#xa;documents__null"/>
        <bpws:link name="Process&#xd; &#xa;documents__and4"/>
        <bpws:link name="null_Contract&#xd; &#xa;generation"/>
      </bpws:links>
    </bpws:flow>
  </bpws:sequence>
</bpws:elseif>
</bpws:if>
</bpws:sequence>
</bpws:sequence>

```

Figure 6.1: Case study 1 translated with the Eclipse implementation [GB08]

The BABEL-tool can also translate the input model, but also needs some slight adjustments. The synchronization dependency is not supported by the implementation and a different result is created depending on the implicit gateway which is added. If the implicit gateway is added to the input model, readable BPEL syntax is generated as displayed in Figure 6.2. However, if the implicit gateway is not added, the whole process after this gateway is translated via the action-based approach and results in less readable output. The input model used for the translation via the BABEL-tool is also included in Appendix E.

Intalio Designer, the commercial tool, can translate the input model only if we remove the looping situation. Also the synchronization dependency is processed in an unwanted manner and result in double occurrence of task 'contract generation'. In Figure 6.3 part of the generated BPEL code is given. In our opinion, the use of structured components could have been done in a smarter way which would have resulted in more readable and comprehensible BPEL code.

```

<sequence name="sequenceComponent_9">
  <receive name="ProcessInstantiation" partnerLink="client" portType="localPT" operation="local
  <invoke name="a" partnerLink="local" portType="localPT" operation="a" inputVariable="a_data_
  <sequence>
    <invoke name="b" partnerLink="local" portType="localPT" operation="b" inputVariable="b_data
    <while condition="a1">
      <sequence>
        <invoke name="b" partnerLink="local" portType="localPT" operation="b" inputVariable="b
        <sequence name="sequenceComponent_6">
          <invoke name="c" partnerLink="local" portType="localPT" operation="c" inputVariable="
          <switch>
            <case condition="b1">
              <invoke name="d" partnerLink="local" portType="localPT" operation="d" inputVariab
            </case>
          </switch>
        <case condition="b2">

```

Figure 6.2: Case study 1 translated with the BABEL-tool [OADH08]

```

<bpel:empty bpmn:label="check accepted offer" bpmn:id="_c11-5v2UEd2q1_-TGbuS4g"/>
<bpel:flow bpmn:label="Parallel_Gateway" bpmn:id="_dQYJoP2UEd2q1_-TGbuS4g">
  <bpel:sequence>
    <bpel:empty bpmn:label="prepare contract" bpmn:id="_fL73ZP2UEd2q1_-TGbuS4g"/>
    <bpel:empty bpmn:label="contract generation" bpmn:id="_jzobRP2UEd2q1_-TGbuS4g"/>
  </bpel:sequence>
  <bpel:sequence>
    <bpel:empty bpmn:label="gather documents" bpmn:id="_gph2wf2UEd2q1_-TGbuS4g"/>
    <bpel:empty bpmn:label="process documents" bpmn:id="_h5H2ZP2UEd2q1_-TGbuS4g"/>
    <bpel:flow bpmn:label="Parallel_Gateway" bpmn:id="_d_erUP2UEd2q1_-TGbuS4g">
      <bpel:empty/>
      <bpel:sequence>
        <bpel:empty bpmn:label="contract generation" bpmn:id="_jzobRP2UEd2q1_-TGbuS4g"/>
      </bpel:sequence>
    </bpel:flow>
  </bpel:sequence>

```

Figure 6.3: Case study 1 translated with the Intalio-tool

6.2.2 Case study 2

Case study 2 is also translated by our own algorithm and implementation but still has some problems left with multiple merge gateways. Both other implementations have also some problems with translating our input model. First, the Eclipse plugin has some problems with the multiple decision and merge gateways. We removed one merge gateway and one split gateway and added a dummy node between two remaining gateways which were connected directly. The synchronization dependency is supported again, but the whole parallel flow is translated as an unstructured component, which results in less readable code. In Figure 6.4 a part of the generated BPEL syntax is given for the Eclipse plugin. The resulting process matches the adjusted input model, which we included in Appendix E.

The BABEL-tool has also problems with our original input model and with the adjusted input model used for the Eclipse plugin. Even after removing the synchronization dependencies as we did for case study 1, the model could not

```

<?xml version="1.0" encoding="UTF-8"?>
<bpws:process exitonStandardFault="yes" name="Case study 2"
  suppressJoinFailure="yes" targetNamespace="http://tempuri.org/plks"
  xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable" xmlns:tns="http://tem
  <bpws:sequence name="start-end">
  <bpws:empty name="start"/>
  <bpws:empty name="48A1"/>
  <bpws:empty name="Check&#xd;&#xa;policy&#xd;&#xa;restrictions"/>
  <bpws:empty name="quick financial check"/>
  <bpws:sequence name="null-null_firstiter">
  <bpws:empty name="Check needed information"/>
  <bpws:while name="null-null">
  <bpws:sequence>
  <bpws:empty name="Perform extended check"/>
  <bpws:empty name="Check needed information"/>
  </bpws:sequence>
  </bpws:while>
  </bpws:sequence>
  <bpws:empty name="determine fraud"/>
  <bpws:if name="null-null">
  <bpws:sequence name="48A2_48A10">

```

Figure 6.4: Case study 2 translated with the Eclipse implementation [GB08]

be translated. By debugging the input model, we found that the cause of the problem can be found in the parallel flow construct.

Intalio Designer can correctly translate the BPMN model, except for the loop construct and the synchronization dependencies. No problems occur with the multiple subsequent decision and merge gateways. The only serious issue which we already mentioned is the occurrence of the same task in the generated BPEL more than once. In Figure 6.5 a preview of the generated BPEL is given.

```

<bpel:sequence>
  <bpel:empty bpmn:label="y" bpmn:id="_7nNL5P2ZEd2q1_-TGbuS4g"/>
</bpel:sequence>
<bpel:else>
  <bpel:sequence>
  <bpel:empty bpmn:label="t" bpmn:id="_34NxpF2ZEd2q1_-TGbuS4g"/>
  <bpel:empty bpmn:label="u" bpmn:id="_4MIMJf2ZEd2q1_-TGbuS4g"/>
  <bpel:empty bpmn:label="v" bpmn:id="_4nE_cf2ZEd2q1_-TGbuS4g"/>
  <bpel:if>
  <bpel:condition>true()</bpel:condition>
  <bpel:sequence>
  <bpel:empty bpmn:label="y" bpmn:id="_7nNL5P2ZEd2q1_-TGbuS4g"/>
  </bpel:sequence>
  <bpel:else>
  <bpel:sequence>
  <bpel:empty bpmn:label="w" bpmn:id="_8WLKtP2ZEd2q1_-TGbuS4g"/>

```

Figure 6.5: Case study 2 translated with the Intalio-tool

6.2.3 Conclusion

In Table 6.1 we summarize the evaluation in a table with the same components as the tabel presented in section 2.4. The BABEL-tool is not maintained anymore and the Eclipse plugin is a new implementation based on [GB08] which deals with the deterministic issues from the BABEL-tool discussed in chapter 2. However, the Eclipse plugin should use a synchronization dependency al-

gorithm for generating more readable BPEL-code. Our own implementation is very strong in finding synchronization dependencies and can contribute to existing implementations on this part. Our own implementation does not support unstructured loops and can not generate BPEL via the action-based approach.

We also evaluated a commercial tool and observed the lack of identifying synchronization dependencies. Intalio Designer parsed parallel fork gateways as flow-constructs, even if the gateway represent a synchronization dependency. In this solution, activities occur more than once in the generated BPEL model. Our algorithm can therefore also trigger improvements in commercial tools.

Table 6.1: Evaluation of implementations

Component \ Impl.	BABEL [OADH08]	Eclipse plugin [GB08]	Intalio	Own approach
Readability	+	+/-	+/-	+
Structured components	+	+	+	+
Structured loops	+	+	-	+
Unstructured loops	+	+	-	-
Synchronization links	-	+/-	-	+

6.3 Evaluation of well-formed BPD restrictions

BPMN models which are used as input for our mapping between BPMN and BPEL must satisfy the well-formedness constraints of a Business Process Diagram. In Definition 1 these eight restrictions are given. Our proposed mapping is developed based on these restrictions and models which do not satisfy all well-formedness restrictions might not be processed in a correct manner.

One of the goals of our mapping was to develop an approach which can also handle unstructured input models. Models which satisfy all well-formedness constraints cannot be called unstructured anymore. Therefore we want to discuss all eight restrictions and eliminate these restrictions when possible. We will also discuss some workarounds to make an unstructured inputmodel more structured in a way that it can be processed by our mapping.

We will now discuss all eight restrictions which we summarized in Table 6.2. A '+' indicates that the restriction can easily be relaxed because the algorithm already deals with it and/or the user can easily adjust the input model. A '+/-' indicates possibilities to relax the restriction in some way, but it is not possible to remove the restriction completely. A '-' represent the fact that the restriction can not be relaxed.

Table 6.2: Evaluation of well-formed BPD restrictions

Restriction:	Supported by algorithm	Possible by (user)input
1. Start events have an indegree of zero and an outdegree of one	+/-	+/-
2. End events have an outdegree of zero and an indegree of one	+/-	+/-
3. Tasks and intermediate events have an indegree of one and an outdegree of one	+	+
4. Fork or decision gateways have an indegree of one and an outdegree of more than one	+/-	+
5. Join or merge gateways have an outdegree of one and an indegree of more than one	+/-	+
6. Event-based XOR decision gateways must be followed by intermediate events or receive tasks	-	-
7. There must be a default flow for a data-based decision gateway	-	+
8. Every object is on a path from a start event to an end event	-	-

1. Start events have an indegree of zero and an outdegree of one.

This restriction consists of two different parts. First the restriction that a start event has an indegree of zero can not be eliminated. If the start event has one or more incoming edges, there must be another event which can start the process. The start event with one or more incoming edges is then an intermediate event instead of the real start event. The second part of the restriction however, can be made more flexible. A dummy gateway can be inserted to facilitate an outdegree of more than one. If a dummy node is inserted, there should be an extra check to be sure that for each fork gateway in the model, a subsequent join gateway of the same type must be present. In Figure 6.6 this situation is modeled. The preprocessing steps presented in [EG09] will insert dummy nodes and check for subsequent gateways.

2. End events have an outdegree of zero and an indegree of one.

The second restriction has also two different parts. For the first part, end events must have an outdegree of zero, the same explanation as discussed in the previous restriction results in the conclusion that end events must always have an outdegree of zero. The second restriction can be relaxed with the use of dummy gateways as discussed earlier. For each dummy join gateway, a subsequent fork gateway of the same type must be present. In Figure 6.6 this situation is modeled. The preprocessing steps presented in [EG09] will insert

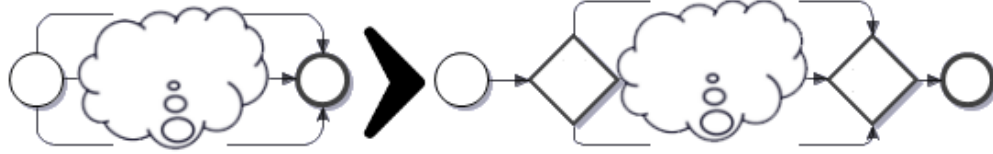


Figure 6.6: Start and end events with respectively an out- and indegree of more than one.

dummy nodes and check for subsequent gateways.

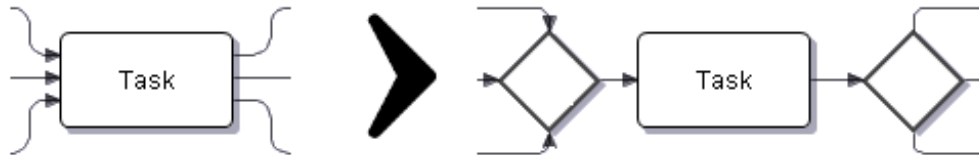


Figure 6.7: Tasks and intermediate events that have an in- or outdegree of more than one.

3. Tasks and intermediate events have an indegree of one and an outdegree of one.

The algorithm of Eshuis[EG09] is developed for dependency graphs. Dependency graphs can be seen as a network of nodes and edges, where a node can represent a task and edges represent relations between tasks. Nodes can have multiple incoming or outgoing edges, but are preprocessed by inserting loop and fork nodes. These loop and fork nodes must be dummy services and are not allowed to represent tasks or intermediate events. Therefore, having tasks or intermediate events with an indegree or outdegree of more than one should be no problem, but dummy nodes must be inserted before the execution of the algorithm. This restrictions can be eliminated by the preprocessing step. In Figure 6.7 this situation is modeled. Note that tasks or intermediate events must have at least one incoming and one outgoing relation, it is not allowed to have an indegree or outdegree of zero.

4. Fork or decision gateways have an indegree of one and an outdegree of more than one.

Fork or decision gateways should indeed have an outdegree of more than one, otherwise it would not be a fork or decision gateway but a normal node. If gateways modeled with an indegree of one and an outdegree of one are found in a process, the algorithm can further process these gateways as normal nodes (see Figure 6.8). Because the algorithms presented do check if a gateway is a fork or decision gateway, the gateway should be replaced by a task for correct

processing. Fork or decision gateways which have an indegree of more than one can be processed as normal fork or decision gateways, because merge and join gateways are processed as normal tasks in the algorithm. For finding synchronization dependencies, fork gateways which have an indegree of more than one should also be processed as join gateways. Replacing gateways with an indegree and outdegree of exactly one with task nodes can only be done manually in the input model. Handling fork or decision gateways with an indegree and outdegree of more than one can be automatically processed by the algorithm.



Figure 6.8: Gateways with an in- and outdegree of one can be transformed to a task node.

5. Join or merge gateways have an outdegree of one and an indegree of more than one.

Restriction 5 can be evaluated as restriction 4, the restriction can be made more flexible by transforming the gateway as task node or by process the gateway as a fork or decision gateway. Fork or decision gateways have a different processing in the algorithm, while join or merge gateways do not.

6. Event-based XOR decision gateways must be followed by intermediate events or receive tasks.

Event-based XOR decision gateways must indeed be followed by intermediate events or receive tasks. If not, the decision gateway is data-based and must be modeled by a data-based gateway instead of an event-based gateway. The input model must be corrected if this restriction is not satisfied.

7. There must be a default flow for a data-based decision gateway.

According to the BPEL syntax, a default flow is not necessary for a data-based decision gateway. However, there must be at least one flow which is followed when a decision gateway is processed. If the modeller can ensure at least one flow is followed, no default flow need to be specified. This restriction cannot be eliminated from the algorithms perspective, but can be eliminated when the user is sure one flow is followed as default.

8. Every object is on a path from a start event to an end event.

We want our input model to be safe and sound because every object must be processed in a correct manner. Objects are related to each other and with the help of immediate dominators, loop headers and follow sets the whole input model is processed. Every object must be reachable from the start event to be

able to calculate these concepts. Also every object must be on a path to an end event, to prevent deadlock situations. Therefore this restriction cannot be released.

As we can see, most restrictions can be relaxed by introducing some preprocessing steps. Inserting dummy nodes and possibly transforming gateways to tasks or other gateways make some restrictions more flexible. Because calculations are based on models which are well-formed, we need the introduction of these preprocessing steps, since these steps transform an unstructured process model to a well-formed BPD. The modeller only must ensure at least one flow is followed in case of a data-based decision gateway.

6.4 Future work

In this chapter we evaluated our mapping presented in chapter 3. We have seen that some well-formed BPD restrictions can be relaxed and there is still some work left in the field of unstructured loops. The event-condition action-rules based approach presented by Mendling et al. might be implemented to be able to handle unstructured loops. Both the BABEL-tool and Eclipse plugin have such an approach implemented. The problem we currently encounter in the implementation (not in the algorithm) regarding multiple decision and merge gateways should also be investigated further for a stable implementation.

Another possible extension to the implementation is the data perspective. In our research, we left this aspect out of scope, since the data object in BPMN is currently only a representation of an object. There are no rules about the data itself.

Chapter 7

Conclusions

In this final chapter we summarize our findings by answering the research questions defined in the introduction of this thesis. After doing this we cover some general threats to the validity of the conclusions and give some suggestions for future work.

7.1 Answers to the research questions

In this section we will answer the research questions using the conclusions and answers given in the other chapters of this thesis. The first research question can be answered using the results from the literature research, while the other two research questions can be answered using the results of our own mapping and the evaluation of the mapping by discussing the case studies.

7.1.1 What is the state-of-the-art with respect to BPMN and BPEL?

We have seen that both BPMN and BPEL are relatively new languages which are still under development. In our research we focused on BPMN 1.1 and BPEL 2.0, which means that some examples from literature are not relevant anymore because they use BPEL 1.0 constructs which are deprecated. New features and possibilities that BPMN 2.0 will bring us, are not used because they are still in a draft status.

In the last few years, a lot of research has been done in the field of mapping BPMN to BPEL. Different strategies and implementations arose, each with its own strengths and weaknesses. Three main categories can be stated: structured-based, link-based and event-action based strategies. We can conclude that combining these three main categories will result in implementations with the highest level of readability and completeness. In table 7.1 an overview of the evaluation of different strategies can be found.

Table 7.1: Evaluation of current approaches

Component \ Mapping	A	B	C	D	E	F	G	H
Readability	-	+/-	-	-	+	+	-	+/-
Completeness	+/-	+	+/-	+/-	-	+/-	+	+
Structured components	+	+	+	+	+	+	+	+
Structured loops	-	+	-	-	+	+	+	+
Unstructured components	+	+	+	+	-	+	+	+
Unstructured loops	-	+	-	-	-	-	+	+
Synchronization links	+	+/-	+	+	-	+	+/-	+/-

The following mappings are displayed in table 2.3.

- A: Graph structure approach (White) [Whi05]
- B: A combined approach (Ouyang et al.) [OADH08]
- C: Element-preservation (Mendling et al.) [MLZ06]
- D: Element-minimization (Mendling et al.) [MLZ06]
- E: Structure-identification (Mendling et al.) [MLZ06]
- F: Structure-maximization (Mendling et al.) [MLZ06]
- G: Event-condition action-rules (Mendling et al.) [MLZ06]
- H: Pattern identification and Classification (García-Bañuelos) [GB08]

7.1.2 How can BPMN be mapped to BPEL?

By answering the first research question, we have seen different strategies and implementations which can map BPMN to BPEL. Translations are done according to a specific strategy and each has its restrictions, strengths and weaknesses. We have investigated if a mapping from BPMN to BPEL is also possible based on the algorithm of Eshuis, which translates dependency graphs to structured compositions. We adjusted the algorithm in a way that BPMN models are translated to nodes and edges. Then a new algorithm for finding synchronization dependencies is executed and the resulting model will be used for calculating dominators, loop headers and follow sets. These concepts can be used in the algorithm which creates a structured composition. This composition can be mapped to BPEL constructs and will result in BPEL code.

The approach we presented needs a well-formed BPD as input model and will execute some preprocessing steps. These preprocessing steps can relax some well-formed BPD restrictions. In chapter 6 we evaluated these restrictions and discussed which restrictions can be relaxed.

7.1.3 How can our translation be evaluated compared to existing translations?

Compared to other implementations and strategies, our own mapping is not able to process input models which contain unstructured loops. A loop must

have one single entry and one single exit point. Implementations and strategies which can process unstructured loops use event-action based strategies for the translation. Our own mapping is not able to handle these event-action based strategies. Our own implementation can find synchronization dependencies and eliminate these synchronization links temporarily. In this way, unstructured components, which are completely processed via the link-based approach by other implementations, are processed as structured components by our own mapping. In chapter 6 we discussed all benefits and drawbacks of our mapping compared to existing mappings. We can conclude that our own approach can be improved by learning from other implementations, but other implementations can also learn from our approach and improve their processing of synchronization dependencies.

In table 7.2 our implementation is compared to BABEL and the existing Eclipse plugin based on the same criteria used in the evaluation of current strategies and approaches.

Table 7.2: Evaluation of implementations

Component	BABEL [OADH08]	Eclipse plugin [GB08]	Own implementation
Readability	+	+/-	+
Completeness	+	+	+/-
Structured components	+	+	+
Structured loops	+	+	+
Unstructured components	+	+	+/-
Unstructured loops	+	+	-
Synchronization links	+/-	+/-	+

Because still no common agreed method exist in translating BPMN to BPEL, we can only do a suggestion which combines the strengths of the implementations evaluated earlier. We think that combining our approach with the existing Eclipse plugin will result in a stable and decent mapping. However, both implementations are still under development and need some refinement first.

7.2 Threats to validity

Some threats can be identified which can influence the validity of the conclusions drawn. Therefore we mention the following threats:

- There might be some bugs or uncompleted implementations which we used for evaluating purposes. The Eclipse plugin for example is developed in 2008 and is still under construction. It is possible that our case studies cannot be processed correctly due to these incompleteness.

- Both case studies are from the financial sector. Other case studies from other sectors might introduce new situations which cause problems that are not covered yet by either the algorithm or the implementation. For both case studies, we tried to find a real business processes which cover as many workflow patterns [Web08b] as possible.

7.3 Future work

We see some needs and possibilities for future work, as discussed in chapter 6 and extended here:

- Currently there are some problems in the implementation regarding multiple decision and merge gateways. These issues must be resolved for further usage of the developed plugin.
- Some functionalities within the plugin need to be reimplemented. Due to the medium level of JAVA knowledge available, some functions might have been implemented in a less efficient way. Performance of the translation in terms of speed and complexity was not in the scope of the project, since the plugin was not the main goal of the master project.
- The well-formed BPD restrictions which are described in chapter 6 and which can be relaxed, might be implemented in the plugin.
- The data perspective is not implemented yet. Research to the possibilities of translating the data perspective from BPMN to BPEL might be interesting.
- Round-trip-engineering will need automated translations between BPMN to BPEL and vice versa. Research into translating BPEL to BPMN might therefore be an interesting subject. When both translations become more mature, we are convinced that the gap between process design and implementation will narrow down.

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Industrial Engineering

BPMN 2 BPEL

Appendix

By
Jeroen Blox

Supervisors:

R. Eshuis (TU/e)
L. Mühlenberg (Logica)
J.C. van Gaalen (Logica)

Eindhoven, February 2009

Appendix A

BPMN explained

Events

An event is something that "happens" during the business process and has a relation with the environment or other parties. Events affect the flow of the process and can have a cause(trigger) or an impact(result). There are different types of events: Start, Intermediate and End.

Start event A start event indicates when a process will start and can not have any incoming arcs.

Intermediate event An intermediate event occurs somewhere between the start and end event and does not start or terminate a process. Examples are message events or timer events.

End event An end event will terminate the process and therefore can not have any outgoing arcs.















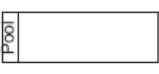




 Start  Intermediate  End	 Task  Process/ Sub-process	 XOR  OR  Complex  XOR  AND  Event-based	 Sequence flow  Message flow  Association	 Pool  Lane	 Data Object  Group  Text Annotation
Events	Activities	Gateways	Connectivity Objects	Swimlanes	Artifacts
Flow Objects					

Figure A.1: The basic elements used in BPMN[WAD⁺06b]

Activities

Activities are tasks or (sub)processes within a company which represent a specific amount of work which has to be done. These activities can be atomic(tasks) or non-atomic. Processes and sub-processes are defined in more detail, while tasks can not be broken down into finer activities.

Gateways

Gateways are used to create a control flow other than the standard sequence flow. Branching, forking, merging and joining of paths becomes possible with the introduction of gateways. For specific functions of a gateway, different markings have been designed.

Exclusive decision	Exclusive decisions can be based on data or events. The outcome of the decision is not known at design time.
Inclusive decision	Inclusive decision making is a decision based on a condition, which is known at design time.
Complex decision	Decision situations which can not be placed under other gateways.
Parallel gateway	Parallel forking and joining will handle activities which can be executed concurrently. Joining after a forking is necessary to ensure the safeness of the model.

Connectivity objects

Within the connectivity objects a differentiation can be made between normal sequence flow, message flow and association. Normal sequence flows show the order in which activities will be executed in a process, message flows show the direction of messages between participants in the process and associations link information and flow objects.

Swimlanes

Swimlanes are divided in lanes and pools. Pools indicate the different subprocesses and participants within a larger process, where lanes can organize and categorize activities.

Artifacts

Three different artifacts can be identified in the BPMN specification. Data objects provide information about the requirements for executing activities and about what they produce, but do not have any direct effect on the sequence or message flow [OMG08b]. Groups are used for documentation and display purposes only, they also do not affect the flow. Text annotations are also only meant for additional information to the reader of the BPMN-models.

Appendix B

BPEL explained

The BPEL specification contains a lot of elements, which can be used for specific purposes. A full list of BPEL elements can be found in [OAS07]. The most common elements can be categorized into the following classes; basic activities, structured activities and partner links [OAS07].

Basic activities

Activities perform the process logic in BPEL. A distinction is made between basic activities and structured activities. Basic activities describe elemental steps of the process behavior, while structured activities describe the control-flow logic. Structured activities can contain other activities.

Invoke - The invoke activity is used to invoke Web Service operations. These operations can be other web services, or can be activities which need to be executed by human people.

Receive and Reply - Business processes provide services to their partners through inbound message activities and corresponding reply activities. A (sub)process waits for a specific message, executes its activities and will send a reply to the original initiator of the process. Multiple initiations can take place at the same time.

Assign - The assign activity allows variables to be updated during the process.

Throw and Rethrow - For fault handling in BPEL, both the `<throw>` and `<rethrow>` activities can explicitly throw a fault to the fault handler. The rethrow activity can throw the original fault message again to a fault handler after catching the fault.

Wait - When a business process needs a delay for a specific time, the wait activity can be used. With the `<for>` or `<until>` element, a fixed duration or a fixed deadline can be specified.

Empty - The empty activity does nothing, but can be used to generate valid and readable BPEL-code.

Exit - The exit activity immediately terminates all activities in the process instance.

All activities can have two standard containers; `<sources>` and `<targets>`, with elements `<source>` and `<target>`. They are used to establish synchronization relationships through links.

Structured activities

Structured activities describe the control-flow logic of a business process. The order in which activities are executed is determined with these class of activities.

Sequence - Activities which are executed one by one in a given order, are placed in a sequence control activity. The sequence activity completes when the last activity of the sequence is completed.

If - For conditional behavior, the `<if>` activity can be used. With the elements `<elseif>`, `<else>` and `<condition>` a conditional behavior can be described. Other (basic) activities will be placed in the different containers.

While and repeatUntil - The `<while>` activity provides a repeated set of activities and at the beginning of each cycle, the `<condition>` element must be evaluated successfully to true. Therefore it is possible that the `<while>` iteration is not visited. The `<repeatUntil>` activity evaluates the condition after executing it's content and is always executed at least once.

Pick - The `<pick>` activity waits for an occurrence of one event and then executes the activities related to this event. Other events where the `<pick>` activity was listening to are neglected. Only one event can be processed with this activity.

Flow - The `<flow>` activity makes synchronization and concurrency possible. Different groups of activities are initiated at the same moment. With the `<link>` elements, synchronization links can be identified and used with the `<source>` and `<target>` elements.

ForEach - ForEach makes it possible to process multiple branches. Depending on the design of this activity, the children of this activity can be executed sequential or in parallel.

Partner Links

Partner links describe cross enterprise business interactions through Web Service interfaces. By describing the partner links, also roles are assigned to different participants.

Appendix C

Existing BPMN 2 BPEL approaches

Table C.1: Existing approaches for translating BPMN to BPEL

Graph structure approach [Whi05]	
Strength	The graph structure, which is used in BPMN models, is maintained.
Weakness	Generated BPEL might be hard to understand. Approach is not explained in detail, but is only loosely described.
Constr.	Global process, namespace and variable information is needed, since a BPMN-model normally does not contain this information.
Compl.	In his paper, White does not support al concepts. However it can be assumed that a high level of completeness can be obtained as long the restrictions of the <link>-constructs are satisfied.
The BPMN to BPEL way [OADH08, ODHA08]	
Strength	Arbitrary cycles are supported by using an event-based translation. The authors discuss semantics and correctness in a concise and unambiguous manner by introducing Petri nets. The approach can deal with all parallel gateways and xor gateways with arbitrary topology. Generated BPEL-code is readable, because block-structured components are mapped before flow-based translation is used.
Weakness	The approach uses events for unstructured (sub)processes, which results in less readable and comprehensible BPEL-code. Because of the combination of three approaches, implementation might be more difficult.
Constr.	The BPMN model must satisfy the well-formed BPD-constraints. The BPMN model must be checked on deadlocks and multiple instances are not possible. For translation using control links, these links may not result in cycles, may not have event-based gateways and must be safe and sound.
Compl.	Any component can be translated, first well structured components are mapped, then the link-based and event-based approaches are used for resulting components.
Element-preservation strategy [MLZ06]	
Strength	Easy to implement and both BPEL and the original process graph are graph-based and easy to compare.
Weakness	Very difficult to understand because of the high number of elements and control links.
Constr.	The process graph must be acyclic.
Compl.	Not all models can be translated using this approach, because of the restrictions in the flow-based approach.
Element-minimization strategy [MLZ06]	
Strength	In the spirit of the BPEL flow, which is graph-based. Also a good strategy when performance matters, since only really necessary elements are included.
Weakness	By removing elements, it becomes less intuitive to identify correspondences.

Constr.	The process graph must be acyclic.
Compl.	Not all models can be translated using this approach, because of the restrictions in the flow-based approach.
Structure-identification strategy [MLZ06]	
Strength	Whole process graph translated to structured activities, which results in best understanding of the generated BPEL-code.
Weakness	The relation with the process graph might be less intuitive, since BPEL now is block-structured.
Constr.	The process graph must be structured, since every component must be mapped to structured activities.
Compl.	Structured processes can be translated, unstructured processes not.
Structure-maximization strategy [MLZ06]	
Strength	This approach also support unstructured processes, as long as loops can be reduced by the structure-identification part.
Weakness	Two different implementations are needed.
Constr.	Only cycles with a single entry and a single exit point are allowed in unstructured processes.
Compl.	Almost any process can be translated, except for arbitrary cycles.
Event-condition action-rules strategy [MLZ06]	
Strength	Any process graph can be translated, even unstructured loops.
Weakness	BPEL-code is difficult to understand and it is not clear how the approach behaves with semantic problems.
Constr.	No specific constraints on input model.
Compl.	Any process can be translated.
The Petri nets approach [AL08]	
Strength	Petri Nets have a strong theoretical foundation. Assuming the input Petri Net is sound and safe, it can be guaranteed that a correct BPEL is generated.
Weakness	BPMN should first be mapped to Petri Nets.
Constr.	The input net should be safe and sound, it should have only one source, one sink and every node is on a path from source to sink.
Compl.	The mapping between BPMN and Petri Nets is missing.

Appendix D

Business Process Diagram

In this appendix, we present the formal definition of a Business Process Diagram (BPD) adapted from Ouyang et al. [OADH08, ODHA08]. In section 3.1 we introduced the concept of BPD in order to define a well-formed BPD. A well-formed BPD will be used as input model for the algorithms presented, unless stated otherwise.

Definition 4. *A core BPD is a tuple $BPD = (\mathcal{O}, \mathcal{T}, \mathcal{E}, \mathcal{G}, \mathcal{T}^R, \mathcal{E}^S, \mathcal{E}^I, \mathcal{E}^E, \mathcal{E}_{\mathcal{M}}^I, \mathcal{E}_{\mathcal{T}}^I, \mathcal{E}_{\mathcal{T}}^E, \mathcal{G}^F, \mathcal{G}^J, \mathcal{G}^D, \mathcal{G}^M, \mathcal{G}^V, \mathcal{F}, Cond)$ where:*

- \mathcal{O} is a set of objects which can be partitioned into disjoint sets of tasks \mathcal{T} , events \mathcal{E} and gateways \mathcal{G} ,
- $\mathcal{T}^R \subseteq \mathcal{T}$ is a set of receive tasks,
- \mathcal{E} can be partitioned into disjoint sets of start events \mathcal{E}^S , intermediate events \mathcal{E}^I and end events \mathcal{E}^E ,
- \mathcal{E}^I can be partitioned into disjoint sets of intermediate message events $\mathcal{E}_{\mathcal{M}}^I$ and timer events $\mathcal{E}_{\mathcal{T}}^I$,
- $\mathcal{E}_{\mathcal{T}}^E \subseteq \mathcal{E}^E$ is a set of end terminate events,
- \mathcal{G} can be partitioned into disjoint sets of parallel fork gateways \mathcal{G}^F , parallel join gateways \mathcal{G}^J , data-based XOR decision gateways \mathcal{G}^D , event-based XOR decision gateways \mathcal{G}^V , and XOR merge gateways \mathcal{G}^M ,
- $\mathcal{F} \subseteq \mathcal{O} \times \mathcal{O}$ is the control flow relation,
- $Cond: \mathcal{F} \cap (\mathcal{G}^D \times \mathcal{O}) \mapsto \mathcal{C}$ is a function mapping sequence flows emanating from data-based XOR decision gateways to the set of all possible conditions.

Appendix E

Input model for other implementations

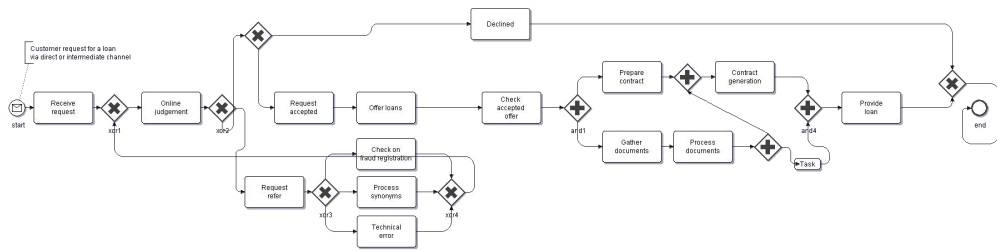


Figure E.1: Case study 1 adjusted input model for the Eclipse implementation

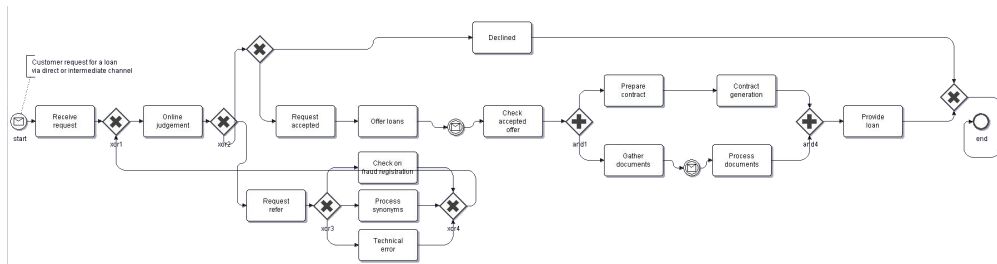


Figure E.2: Case study 1 adjusted input model for the BABEL-tool

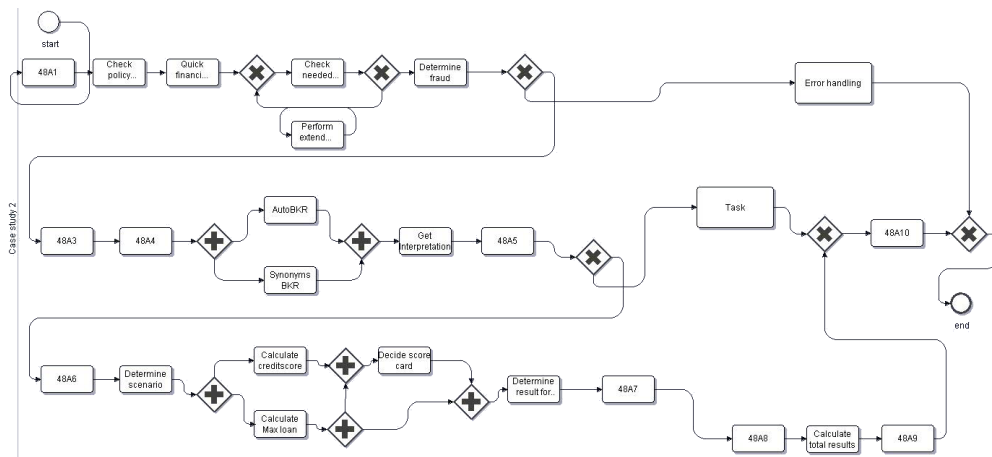


Figure E.3: Case study 2 adjusted input model for the Eclipse implementation

Bibliography

- [AL08] W.M.P. van der Aalst and K.B. Lassen. Translating unstructured workflow process to readable BPEL: Theory and implementation. *Information and Software Technology*, Volume 50 , Issue 3:131–159, 2008.
- [ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 1986.
- [BK05] F. van Breugel and M. Koshkina. Dead-path-elimination in BPEL4WS. *Fifth International Conference on Application of Concurrency to System Design*, 2005.
- [Boe88] B.W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, May 1988.
- [DDO08] R.M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in bpmn. *Information and Software Technology*, 50(12):1281–1294, 2008.
- [Dum09] M. Dumas. Private conversation. 2009.
- [EG09] R. Eshuis and P.W.P.J. Grefen. Composing services into structured processes. *International Journal of Cooperative Information Systems*, to be appear, 2009.
- [Esh05] R. Eshuis. Statecharting petri nets. *BETA Working Paper Series WP 153*, 2005.
- [GB08] L. García-Bañuelos. Pattern identification and classification in the translation from bpmn to bpel. In *OTM Conferences (1)*, pages 436–444, 2008.
- [MH08] M. zur Muehlen and D.T. Ho. Service process innovation: A case study of BPMN in practice. *Hawaii International Conference on System Sciences*, Proceedings of the 41st Annual:372–372, 2008.

- [MLZ06] J. Mendling, K.B. Lassen, and U. Zdun. On the transformation of control flow between block-oriented and graph-oriented process modeling languages. *Int. J. Business Process Integration and Management*, 2006.
- [OADH06] C. Ouyang, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. From BPMN process models to BPEL web services. *Proceedings of the 4th International Conference on Web Services*, IEEE Computer Society:pages 285–292, 2006.
- [OADH08] C. Ouyang, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. From business process models to process-oriented software systems: The BPMN to BPEL way. *ACM Transactions on Software Engineering and Methodology*, 2008.
- [OAS07] OASIS. *Web Services Business Process Execution Language Version 2.0*. OASIS Standard, 2007.
- [ODHA08] C. Ouyang, M. Dumas, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Pattern-based translation of BPMN process models to BPEL web services. *International Journal of Web Services Research*, 5(1):42–62, 2008.
- [OMG08a] OMG. *Business Process Model and Notation (BPMN) Specification 2.0*. 2008.
- [OMG08b] OMG. *Business Process Modeling Notation, V1.1*. 2008.
- [OYL08] C. Ou-Yang and Y. D. Lin. BPMN-based business process model feasibility analysis: a petri net approach. *International Journal of Production Research*, 46:3763 – 3781, 2008.
- [RM06] J. Recker and J. Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. *Proceedings 18th International Conference on Advanced Information Systems Engineering*, pages 521–532, 2006.
- [VVK08] J. Vanhatalo, H. Völzer, and J. Koehler. The refined process structure tree. *BPM*, pages 100–115, 2008.
- [WAD⁺06a] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. On the suitability of BPMN for business process modelling. *Lecture notes in computer science*, ISSN 0302-9743, 2006.
- [WAD⁺06b] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. Pattern-based analysis of BPMN - an extensive evaluation of the control-flow, the data and the resource perspectives. *BPM Center Report*, BPM-06-17, 2006.

-
- [WADH03] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of web services composition languages: The case of bpeL4ws. In *Song, I.Y., Liddle, S.W., Ling, T.W., Scheuermann, P., eds.: Conceptual Modeling - ER2003*, 2813:200–215, 2003.
- [WADH05] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-based analysis of bpmn - an extensive evaluation of the control-flow, the data and the resource perspectives. *BPM Center Report*, BPM-05-26, 2005.
- [WADH06] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of web services composition languages: The case of BPEL4WS. *Proceedings 22nd International Conference on Conceptual Modeling (ER)*, pages 200–215, 2006.
- [Web08a] Website. *Transforming BPMN into BPEL: Why and How*. <http://www.oracle.com/technology/pub/articles/dikmans-bpm.html?rrsid=rss>, 12 September 2008.
- [Web08b] Website. *Workflow patterns*. <http://www.workflowpatterns.com/>, 25 September 2008.
- [Web08c] Website. *BPEL Implementations*. <http://www.oasis-open.org/-news/oasis-news-2007-04-12.php>, 29 September 2008.
- [Web08d] Website. *BPMN Implementations*. http://www.bpmn.org/BPMN_Supporters.htm, 29 September 2008.
- [Whi05] S.A. White. Using BPMN to model a BPEL process. *IBM white paper*, 2005.