

Eberhard-Karls Universität Tübingen  
Wilhelm-Schickard-Institut für Informatik  
Lehrstuhl Technische Informatik  
Sand 13  
72076 Tübingen

Tübingen, im Dezember 2005

Diplomarbeit im Fach Informatik

# Discovering and Classifying Regions in Workflow Graphs

vorgelegt von Nicolai Mainiero

Betreuer:

Prof. Dr.-Ing. Wilhelm G. Spruth  
Dipl.-Inf. Michael Friess



# Executive Summary

The design of correct, precise and executable business processes is more and more challenging. This diploma thesis shows a possibility to assist designers and developers during the creation of business processes. The goal of this thesis is to provide a set of several algorithms that can be applied in several ways, for example in the transformation of business processes. These algorithms provide the base of a toolset, to assist designer and implementer in producing correct business processes.

The main task during this work were the design of a Process Flow Graph (PFG) model to use for the analysis and the transformation of such graphs based on the detection of regions. Therefore it was necessary to evaluate several algorithms for the verification of the structural correctness and the detection of regions. Some of the found algorithms were suitable but slow, others were not usable to verify the structural correctness of PFG. So the extension of an already known algorithm was necessary. The algorithm found for the detection of Single Entry Single Exit (SESE) regions suits perfectly, even for the PFG model. To classify the new available regions the extension of the classification scheme from [12] was needed, but only the consideration of the new parallel regions was necessary.

The PFG model is realized as Eclipse Modeling Framework (EMF) model. The algorithms for the verification of the structural correctness and the detection and classification of regions were implemented. Even two algorithms that restructure the PFG to prepare it for the transformation into other models. All this algorithms are combined within an Eclipse plug-in to provide a modular architecture that is easily extendable for other algorithms.

For the future it is possible to implement different algorithms that do the same, so the user can select which algorithm suits in his situation best, or more algorithms could be added to do other analysis of the graph. Also the set of restructuring algorithms can be completed to allow the semi-automated transformation to Business Process Execution Language (BPEL) for example.

---

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebene Quellen und Hilfsmittel verwendet habe.

Tübingen, 20. Dezember 2005

Nicolai Mainiero

# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Intention of this Work . . . . .	1
1.3. Overview of this Work . . . . .	2
<b>2. Business Processes</b>	<b>3</b>
2.1. What are business processes? . . . . .	3
2.1.1. Definition . . . . .	3
2.1.2. Models . . . . .	3
2.2. The Business Process Execution Language . . . . .	4
2.3. Design Problems . . . . .	5
2.3.1. Design vs. Implementation . . . . .	5
<b>3. Process Flow Graphs</b>	<b>7</b>
3.1. Workflow . . . . .	7
3.1.1. Definition . . . . .	7
3.1.2. Workflow as a Graph . . . . .	7
3.1.3. Formal Definition of Workflow Graphs . . . . .	9
3.2. Control Flow Graph . . . . .	10
3.2.1. Formal Definition of Control Flow Graphs . . . . .	10
3.2.2. Workflow Graph vs. Control Flow Graph . . . . .	10
3.3. Process Flow Graph . . . . .	11
3.3.1. Formal Definition of Process Flow Graphs . . . . .	11
3.3.2. Extension from CFG to PFG . . . . .	12
3.4. Analysis of the PFG . . . . .	12
3.4.1. Regions within the PFG . . . . .	12
3.4.2. Classification Scheme . . . . .	14
<b>4. Algorithms</b>	<b>17</b>
4.1. Structural Correctness . . . . .	17
4.1.1. What is structural correctness? . . . . .	17
4.1.2. Instance Subgraph . . . . .	18
4.1.3. Algorithm . . . . .	21

4.1.4. Another possibility to verify structural correctness in cyclic graphs . . . . .	22
4.2. SESE-Regions . . . . .	23
4.2.1. What are SESE regions? . . . . .	23
4.2.2. Finding Canonical SESE regions . . . . .	24
4.3. T1-T2 Theorem . . . . .	26
4.3.1. T1-T2 Transformations . . . . .	26
4.3.2. Misuse of the T1-T2 Theorem . . . . .	26
<b>5. Implementation</b>	<b>29</b>
5.1. Architecture . . . . .	29
5.2. Eclipse Modeling Framework . . . . .	29
5.3. Explanation of the EMF Model . . . . .	29
5.4. How the plug-in works . . . . .	31
5.4.1. Verify the structural correctness . . . . .	31
5.4.2. Add and classify the regions in the PFG . . . . .	31
5.4.3. Restructure the PFG to prepare it for the conversion to BPEL . . . . .	35
5.4.4. Create a new BPEL graph from the PFG . . . . .	35
5.5. Extending the plug-in . . . . .	35
<b>6. The PFG to BPEL Transformation</b>	<b>37</b>
6.1. Transformation . . . . .	37
6.1.1. Transformation Library . . . . .	38
6.2. Problems . . . . .	41
6.2.1. Regions which need restructuring . . . . .	41
<b>7. Summary and Outlook</b>	<b>45</b>
7.1. Summary . . . . .	45
7.1.1. Limitations . . . . .	45
7.2. Outlook . . . . .	45
<b>A. Appendix</b>	<b>47</b>
A.1. Manual . . . . .	47
A.1.1. Installation . . . . .	47
A.1.2. Use . . . . .	47
A.1.3. Sourcecode Repository . . . . .	47
<b>List of Acronyms</b>	<b>49</b>
<b>Bibliography</b>	<b>51</b>

# List of Figures

3.1. Workflow Graph Symbols . . . . .	8
3.2. CFG-Classification Scheme . . . . .	11
3.3. TT-Region . . . . .	13
3.4. SESE-Regions . . . . .	13
3.5. TT to SESE transformation . . . . .	14
3.6. Classification scheme for Process flow graphs . . . . .	14
3.7. cross-over synchronized region and avalanche structure . . . . .	15
4.1. A graph with lack of synchronization . . . . .	17
4.2. A graph with a dead lock . . . . .	18
4.3. Example: Instance Subgraphs . . . . .	19
4.4. Examples for the instance subgraph . . . . .	20
4.5. Rule 4 and the extended version . . . . .	23
4.6. Possible SESE regions, canonical SESE regions and the PST . . . . .	25
4.7. T1-T2 transformation rules . . . . .	26
4.8. Comparison of while- and until-loops after application of T1-T2 clas- sification algorithm . . . . .	27
5.1. UML representation of the EMF model . . . . .	30
5.2. The UML visualization of the <code>abstract class TransformPFG</code> and the implementing classes . . . . .	32
5.3. Comparison between an until- and a while-loop . . . . .	35
5.4. The UML visualization of the popup action classes . . . . .	36
6.1. transformation . . . . .	38
6.2. A if-then -else region as PFG and in BPEL . . . . .	39
6.3. A case region as PFG and in BPEL . . . . .	39
6.4. A classical while loop as PFG and in BPEL . . . . .	40
6.5. An ACTIVITY node of the PFG and the possible representation in BPEL . . . . .	40
6.6. A parallel region as PFG and in BPEL . . . . .	41
6.7. A SESE Until Loop and both possible restructured while loops . . . . .	41
6.8. A multiple merge region and the corresponding restructured equivalent	42
6.9. A not reducible region . . . . .	43
6.10. A SESE not classical while loop . . . . .	43

## *List of Figures*

---

A.1. Context menu of the eclipse plug-in . . . . .	48
A.2. SVN configuration dialog from Eclipse . . . . .	48



# 1. Introduction

The following thesis is about the detection, classification and transformation of regions found in PFGs. First it will be investigated how one can split up the PFG in useful regions and how this regions can be classified, to get simple conversion functions for each detected region. For the classifying of the regions results from *Aspekte der Abbildung von Geschäftsprozessen, spezifiziert im Business Process Definition Metamodel, in BPEL4WS* by Kirsten Stöhr [12] were used and adapted were necessary. The results of the analysis will form a prototype which implements the found algorithms as an Eclipse plug-in and provides the possibility to execute this algorithms on a PFG and do the restructuring necessary for the transformation from PFG to BPEL which requires some of the restructuring methods introduced in [12].

## 1.1. Motivation

These days we have almost in every company so called *business processes*. These are simple step-by-step instructions which, put together, represent what the company produces or manufactures. But unfortunately the precise creation of such processes is complicated and therefore the concept of modeling such processes as flow was introduced, which provides a simple but efficient possibility to design business processes.

With the introduction of the computer more and more of these business processes were adapted to run on computers to assist the employees. At this time another problem raised: the gap between the designer of such a business process and the guy how has to implement it on a computer.

The new designed business processes became more and more complex, which also means it became more and more difficult to design a correct business process and implement it in currently available languages as BPEL for example.

## 1.2. Intention of this Work

The goal of this work is to provide a tool that assists in the design, analysis, restructuring and transformation process needed to adapt the process design in a specific language, for example BPEL. To achieve this several algorithms for the analysis of PFG were examined. The concept of regions was reconsidered and adapted. Finally the existing classification scheme fro regions from the previous work [12] was

extended were necessary. Actually were the following algorithms implemented: one for the verification of the structural correctness and one to detect and classify the regions, furthermore two simple restructuring rules for BPEL as target language were implemented. The design of this implementation is made such that it is extendable with other analysis methods or restructuring algorithms.

### 1.3. Overview of this Work

A short overview of the six chapters in this thesis.

In chapter 2 business processes are shown and which models are used to describe them. The chapter will give an overview of used models and implementations. Chapter 3 is about the definition of PFGs and which algorithms are useful for the analysis of such graphs. In chapter 4 the algorithms implemented during this work are shown and in the following chapter 5 the implementation is explained. Finally in chapter 6 the concrete transformation from PFG to BPEL is described. The last chapter 7 provides a short summary of what was done and an outlook of the future possibilities. In the appendix you will find a short manual how to install and use the provided plug-in.

## 2. Business Processes

In this chapter a short introduction into business processes is given. Then as a concrete example the structure of BPEL is explained. At last the difficulties during the design of precise business processes are mentioned.

### 2.1. What are business processes?

#### 2.1.1. Definition

A more informal definition of a business process can be found at [\[15\]](#).

**Definition 1** A business process is a recipe for achieving a commercial result. Each business process has inputs, methods and outputs. The inputs are a pre-requisite that must be in place before the method can be put into practice. When the method is applied to the inputs then certain outputs will be created.

A business process is a collection of related structural activities that produce a specific outcome for a particular customer.

A business process can be part of a larger, encompassing process and can include other business processes that have to be included in its method.

The business process can be thought of as a cookbook for running a business; “Answer the phone”, “place an order”, “produce and invoice” might all be examples of a business process.

This shows that business processes are used by nearly every company, but one must differentiate between human-executed and computer-implemented business processes.

In the following only the computer-implemented processes are relevant, but even with this limitation human interaction is possible.

#### 2.1.2. Models

Nowadays these business processes are modeled as flows, as a succession of different tasks that are executed in parallel or in sequence. Sometimes decisions are needed or a task has to be executed multiple times limited by a condition. All these different possibilities end up in a very complex definition of workflow models.

### 2.2. The Business Process Execution Language

The Business Process Execution Language 4 Web Services BPEL [2] currently at version 1.1 is an XML based language to describe executable business processes. BPEL represents a convergence of the ideas in the XLANG and WSFL specifications. Both XLANG (XLANG) and Web Services Flow Language (WSFL) are superseded by the BPEL4WS specification.

#### Overview

The key elements of BPEL are structured activities. These activities describe the order in which a collection of activities take place.

The structured activities of BPEL include:

- Ordinary sequential control between activities is provided by sequence, switch, and while.
- Concurrency and synchronization between activities is provided by flow.
- Nondeterministic choice based on external events is provided by pick.

#### sequence

A sequence activity contains one or more activities that are performed sequentially, in the order in which they are listed within the <sequence> element, that is, in lexical order. The sequence activity completes when the final activity in the sequence has completed.

#### switch

The switch structured activity supports conditional behavior in a pattern that occurs quite often. The activity consists of an ordered list of one or more conditional branches defined by case elements, followed optionally by an otherwise branch. The case branches of the switch are considered in the order in which they appear. The first branch whose condition holds true is taken and provides the activity performed for the switch. If no branch with a condition is taken, then the otherwise branch is taken. If the otherwise branch is not explicitly specified, then an otherwise branch with an empty activity is deemed to be present. The switch activity is complete when the activity of the selected branch completes.

#### while

The while activity supports repeated performance of a specified iterative activity. The iterative activity is performed until the given boolean while condition no longer holds true.

### **pick**

The pick activity awaits the occurrence of one of a set of events and then performs the activity associated with the event that occurred. The occurrence of the events is often mutually exclusive (the process will either receive an acceptance message or a rejection message, but not both). If more than one of the events occurs, then the selection of the activity to perform depends on which event occurred first. If the events occur almost simultaneously, there is a race and the choice of activity to be performed is dependent on both timing and implementation.

### **flow**

The flow construct provides concurrency and synchronization. The most fundamental semantic effect of grouping a set of activities in a flow is to enable concurrency. A flow completes when all of the activities in the flow have completed.

## **2.3. Design Problems**

With the increasing complexity of business processes it becomes more and more difficult to design them efficient and correct. There are also problems that are caused by the different limitations and possibilities between the design language and the implementing language.

### **2.3.1. Design vs. Implementation**

Typically neither the process designer knows something about implementation nor the one who implements knows much about the task of designing a process. So the task of designing and implementing a new business process is something, more than one person is involved and this is where most problems begin. The designer does not know which limitations the target language has, and the implementer has the problem to restructure the process design to meet the requirements of the target language. This restructuring process is very error-prone and difficult and this is the place where the developed tool assists the implementer.



## 3. Process Flow Graphs

This chapter is about Workflow, Control Flow and Process Flow. First the two more known models, workflow and control flow are presented. The the difference between them is shown, and the extension to the process flow graph is done. Then one main concept in the analysis of PFGs, the building of regions, is introduced. Combined with this concept a classification scheme for the regions is presented.

### 3.1. Workflow

#### 3.1.1. Definition

A nice definition what workflow is can be found at [17]:

**Definition 2 (Workflow)** Workflow is the operational aspect of a work procedure: how tasks are structured, who performs them, what their relative order is, how they are synchronized, how information flows to support the tasks and how tasks are being tracked. As the dimension of time is considered in Workflow, Workflow considers “throughput” as a distinct measure.

While the concept of workflow is not specific to information technology, support for workflow is an integral part of group-ware software.

#### 3.1.2. Workflow as a Graph

It is easy to define a graphical representation of workflow, the so called Workflow Graph (WFG). Therefore we use seven symbols representing an ACTIVITY, a FORK and the corresponding SYNCHRONIZER, a CHOICE and its MERGE, and of course the START and END symbols. For the rest of this work the symbols from figure 3.1 are used to represent this nodes.

**An ACTIVITY node** is the most simple modeling node and defines the order of task execution. It has at most one incoming and one outgoing node.

**A FORK (AND-Split) node** has exactly two outgoing edges and is used to represent concurrent paths within a workflow graph.

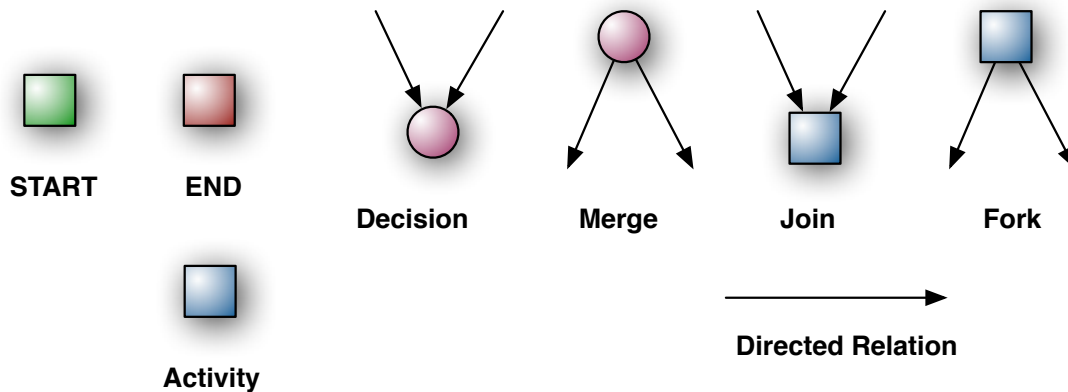


Figure 3.1.: The symbols used in workflow graphs to model parallel and alternative processes.

- A SYNCHRONIZER (AND-join) node** with exactly two incoming edges is applied to synchronize such concurrent paths. A task waits until all incoming transitions (edges) have been triggered.
- A DECISION (XOR-split) node** has exactly two outgoing edges and is used to model mutually exclusive alternative paths. At run-time, the workflow selects one of the alternative paths for a given instance of the business process by activating one of the transitions (edges) originating from the DECISION node. The DECISION node is exclusive and complete. The exclusive characteristic ensures that only one of the alternative paths is selected. The complete characteristic ensures that one of its outgoing flows will always be triggered.
- A MERGE (XOR-join) node** is opposite to the DECISION node. It has exactly two incoming edges and joins mutually exclusive alternative paths into one path.

By connecting nodes with edges a directed acyclic graph called workflow graph is created. This graph has at least one node with no incoming edge, its called START node and at least one node with no outgoing edge, called END. There are two kinds of structural conflicts in such graph models:

- **Deadlock** - Joining multiple paths opened by an exclusive DECISION with a SYNCHRONIZER node results in a deadlock conflict. At least one of the edges from a SYNCHRONIZER is not triggered, so the continuation of the workflow path is blocked.



- Lack of Synchronization - Joining multiple paths opened by a FORK with a MERGE node results in a lack of synchronization. In the consequence it is possible that the nodes followed the MERGE node are unintentionally multiple activated.

An important concept for the analysis of workflow graphs is the instance subgraph. An instance subgraph represents a subset of workflow tasks that may be executed for a particular instance of a workflow. It can be generated by traversing the graph and following only one outgoing edge of a DECISION node and all outgoing edges of a FORK node from the START node to the END node. The instance subgraph provides an easy approach to detect the above mentioned structural conflicts within WFG, with the following two correctness criteria:

**Correctness Criteria 1 - deadlock free instance subgraphs** An instance subgraph is free of deadlock structural conflicts if it does not contain only a proper subset of the incoming nodes of a SYNCHRONIZER node.

**Correctness Criteria 2 - lack of synchronization free instance subgraphs** An instance subgraph is free of lack of synchronization structural conflicts if it does not contain more than one incoming nodes of a MERGE node.

A workflow graph is only correct if and only if all instance subgraphs of the workflow graph meet the Correctness Criteria 1 and 2.

### 3.1.3. Formal Definition of Workflow Graphs

Now a more formal definition of a workflow graph.

**Definition 3 (Workflow Graph)** The workflow graph  $G = (N, E)$  is a simple directed acyclic graph where

1.  $N$  is a finite set of nodes
2.  $E$  is a finite set of directed edges representing transitions between two nodes
3.  $size[G] = size[N] + size[E]$  represents the total number of nodes and edges in  $G$ .

The graph  $G$  meets the following syntactical correctness properties:

- It uses only core modeling nodes, namely, ACTIVITY, DECISION, MERGE, FORK and SYNCHRONIZER.
- It does not contain any cycles, i.e.  $\forall n_i, n_j \in N$ , a path from  $n_i$  to  $n_j$  implies  $n_i \neq n_j$  (no self-loops) and no path from  $n_j$  to  $n_i$  exists (no cycles).

- It has exactly a single START node  $n_{start}$  with exactly one outgoing edge.
- It has exactly a single END node  $n_{end}$  with exactly one incoming edge.

**Definition 4** For each path  $p$  from  $n_i$  to  $n_j$  where  $n_i, n_j \in N$ , we define:  $pathNodes[p] = \{n_i, \dots, n_j\}$  represents a set of nodes contained within  $p$ .

As you can see in WFG no cycles are allowed, that means these graphs only allow step-by-step operations without loops.

## 3.2. Control Flow Graph

### 3.2.1. Formal Definition of Control Flow Graphs

The possibility of loops gives us the Control Flow Graph (CFG) but therefore it lacks him of the parallel activities. But first a formal definition what a CFG consists of:

**Definition 5 (Control Flow Graph)** The control graph  $G = (N, E)$  is a simple directed graph where

1.  $N$  is a finite set of nodes
2.  $E$  is a finite set of directed edges representing transitions between two nodes
3.  $size[G] = size[N] + size[E]$  represents the total number of nodes and edges in  $G$ .

The graph  $G$  meets the following syntactical correctness properties:

- It uses only core modeling nodes, namely, ACTIVITY, DECISION and MERGE.
- It has exactly a single START node  $n_{start}$
- It has exactly a single END node  $n_{end}$

The CFG is expatiated in the earlier work found at [12]. Figure 3.2 shows the result of the classification scheme developed in this work. For a complete overview with detailed explanation of the found regions look at [12].

### 3.2.2. Workflow Graph vs. Control Flow Graph

There are two differences between workflow graphs and control flow graphs. First only the WFG allows the realization of parallel executed activities. And second only the CFG allows the realization of repeated execution of activities. To allow both an extension of these models is needed, the PFG.

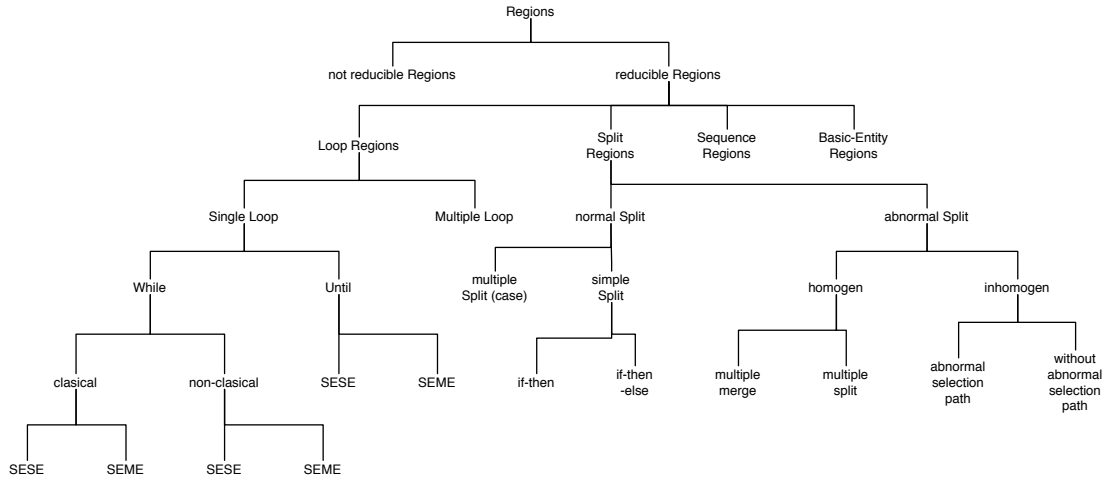


Figure 3.2.: The classification scheme for the CFG (not complete)

### 3.3. Process Flow Graph

The PFG allows the modeling of more sophisticated graphs, which contains loops and parallel activities.

#### 3.3.1. Formal Definition of Process Flow Graphs

**Definition 6 (Process Flow Graph)** The control flow graph  $G = (N, E)$  is a simple directed graph where

1.  $N$  is a finite set of nodes
2.  $E$  is a finite set of directed edges representing transitions between two nodes
3.  $size[G] = size[N] + size[E]$  represents the total number of nodes and edges in  $G$ .

The graph  $G$  meets the following syntactical correctness properties:

- It uses only core modeling nodes, namely, ACTIVITY, DECISION, MERGE, FORK and SYNCHRONIZER.
- It has exactly a single START node  $n_{start}$
- It has exactly a single END node  $n_{end}$

#### 3.3.2. Extension from CFG to PFG

The definition 6 is very similar to the definition of the CFG which leads to the assumption that the properties of CFGs and PFGs are also very similar. For this work it would be nice to reuse the classification scheme from [12] and extend it for the PFG were necessary. This leads to theorem 1.

**Theorem 1** *The control flow graph is a real subclass of the process flow graph.*

**PROOF (BY REDUCTION)** It is easy to see, that the definitions of the CFG (Definition: 5) and PFG (Definition: 6) only differ in the used core modeling nodes. So the control flow graph is a process flow graph without the FORK and SYNCHRONIZER nodes.

#### 3.4. Analysis of the PFG

The analysis of PFGs is based on the concept of regions. Every graph is divisible into a finite number of regions, and each of this region can be analyzed independently.

##### 3.4.1. Regions within the PFG

**Definition 7 (Region)** The term region is used here to describe a subgraph  $G' = (N', E')$ , with  $N' \subset N$  and  $E' \subset E$ .

These regions are used to structure the graph, but unfortunately not all such regions are adequate to do this. For the later analysis and transformation steps regions that are independently from each other are needed. So the idea is to use so called Two-Terminal regions known from the program analysis, which have one entry node  $a \in N'$  and one exit node  $b \in N'$  so that,

- $d_N(a, k)$  with  $k \in N'$
- $pd_N(b, k)$  with  $k \in N'$

**Definition 8 (Domination)** A node  $x$  is said to dominate node  $y$  in a directed graph if every path from start to  $y$  includes  $x$ . A node  $z$  is said to postdominate a node  $y$  if every path from  $y$  to end includes  $z$ . We write  $d_N(x, k)$  with  $k \in N'$  for dominating and  $pd_N(z, k)$  with  $k \in N'$  for postdominating.

The figure 3.3 shows how such a region can look like.

A special kind of Two-Terminal Region (TTRegion) is the SESE which adds a constraint to the definition of the TTRegion given above.

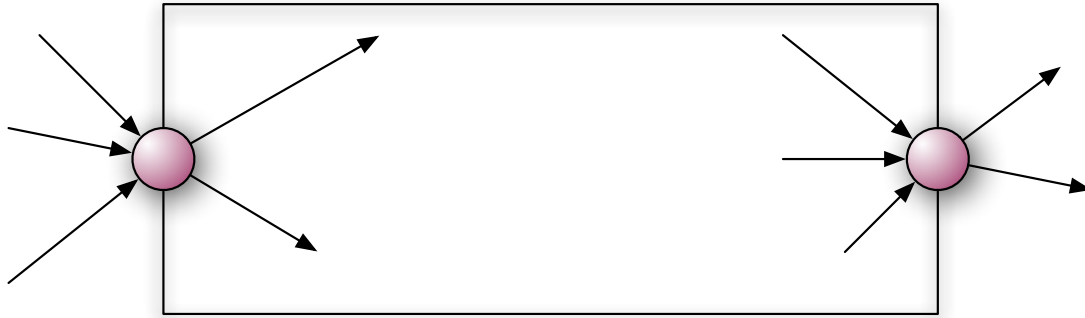


Figure 3.3.: Two-Terminal Region

**Definition 9 (SESE)** A SESE region is a subgraph beginning with a dominating node and ending with a postdominating node and between this two nodes there are no edges to any successor of the postdominating node or any predecessor of the dominating node and there is only one edge entering the region and only one edge leaving the region.

Figure 3.4 shows how such regions could look like.

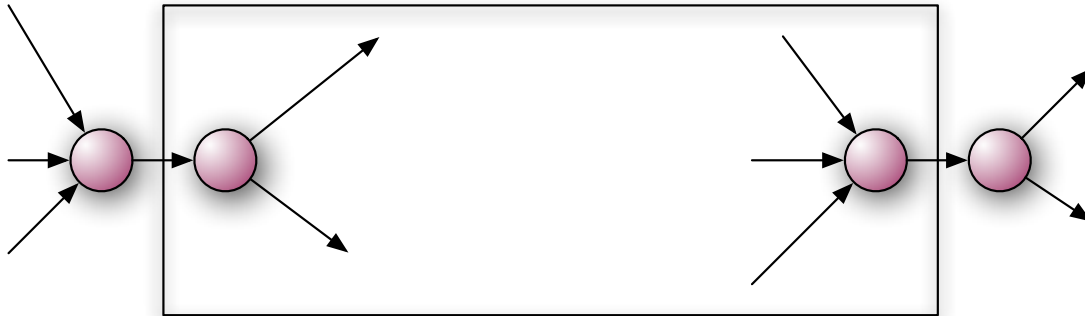


Figure 3.4.: Shows a Single Entry Single Exit Region

The two region types are equivalent as the transformation shown in figure 3.5 proofs. By inserting dummy-nodes after the dominating node of the TTRregion and before the postdominating node, we get a SESE region.

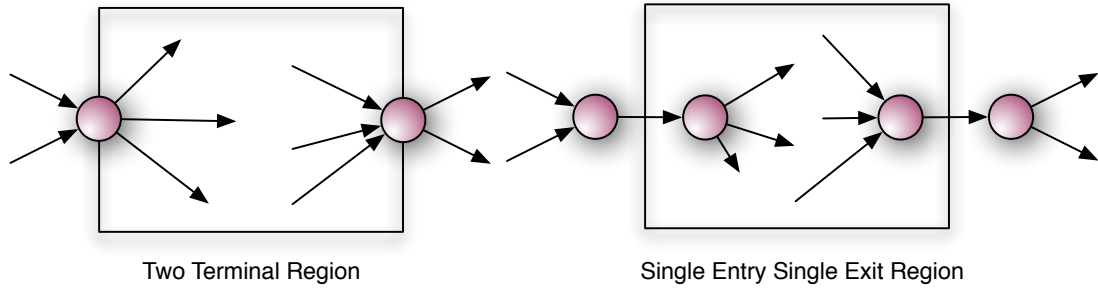


Figure 3.5.: The equivalent transformation from TT-Region into SESE-Region

#### 3.4.2. Classification Scheme

The classification scheme found in [12] is complete for the CFG but not for the PFG. So the classification scheme needs to be extended to fit also for PFGs. The extended classification scheme can be divided into three main branches: only regions that are possible in CFGs, only such that are possible in WFGs, and such that are not realizable in the one or the other. This leads to the following classification scheme:

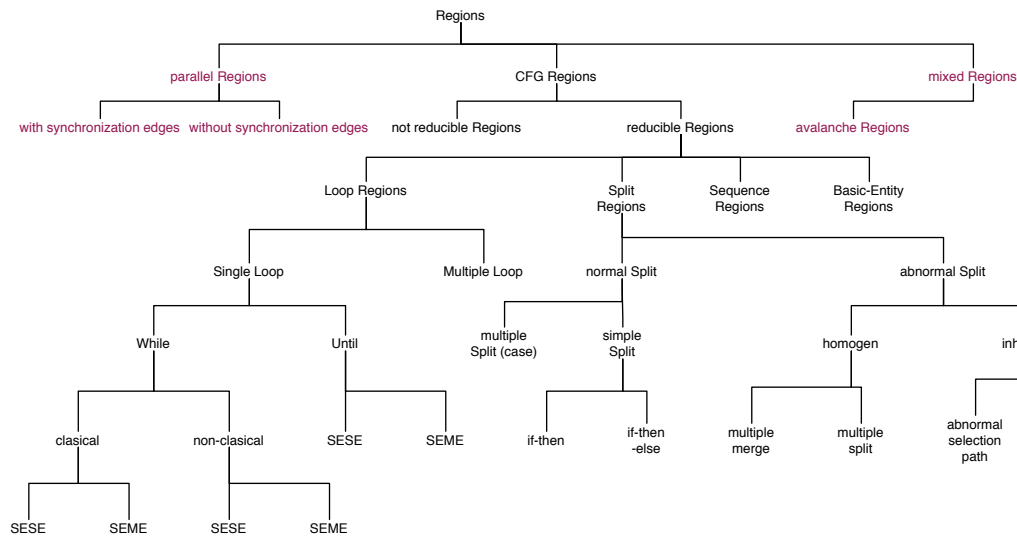


Figure 3.6.: Classification scheme for process flow graphs. The red marked region is new in comparison to the CFG

The regions can be split up into three big parts, the CFG regions, already explained in [12], the parallel regions and mixed regions. Within the parallel regions

only FORK and JOIN nodes are allowed, which reduces the possibilities of regions. But there are synchronizing edges allowed, which leads to regions like the cross over synchronized, explained later. The mixed regions allow the use of FORK and JOIN nodes as well as the use of DECISION and MERGE nodes. This rises the possibility of more unusual but structural correct regions, like the avalanche region, also explained and show here.

**cross over synchronized** In this region, there are several synchronization edges, so that all activities are executed in sequence and no parallel activity is left, it would be possible to restructure such regions during the transformation steps. Figure 3.7 shows an example for such regions.

**avalanche structure** This regions is not so familiar. It is more a theoretical construct than something naturally emerged during process design. But this construction is a very good test case for other algorithms that implements the verification of structural correctness. Figure 3.7 shows an example of such a region.

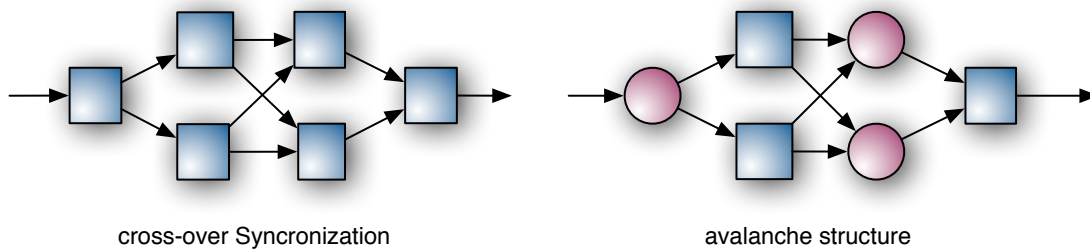


Figure 3.7.: The region on the left is a cross over synchronized structure, and on the right side is is an avalanche structure





## 4. Algorithms

In this chapter all for this work relevant algorithms are presented. This algorithms are mainly used for the analysis of the PFG and partly also for the transformation.

### 4.1. Structural Correctness

#### 4.1.1. What is structural correctness?

A consistent workflow graph is structurally correct, if from exactly one start transition exactly one end transition is reachable under the workflow rules. If a workflow graph contains decisions as well as synchronizations, two different structural problems may arise as mentioned in [13] and [10]: Deadlock and lack of synchronization.

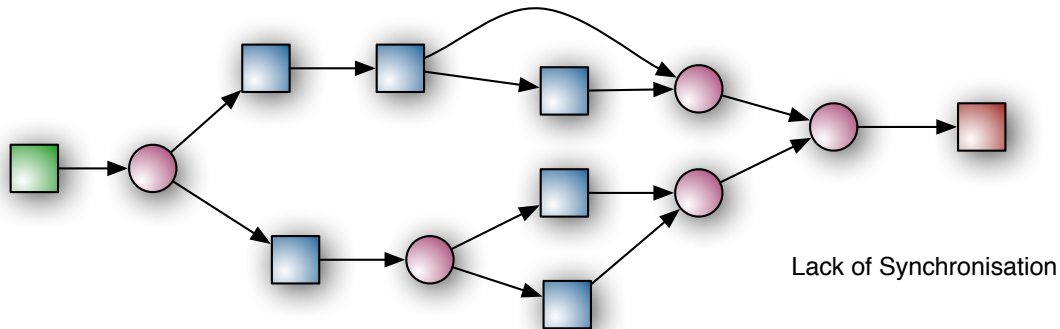


Figure 4.1.: A structural problem: A graph with lack of synchronization.

**Deadlock** A deadlock as shown in Figure 4.2 arises, if after a decision alternative activities are merged by a synchronization. In this case the synchronization activity can not be executed.

**Lack of Synchronization** A lack of synchronization as shown in Figure 4.1 arises, if asynchrony activities are merged by a contact. In this case the following activities would be executed more than once.

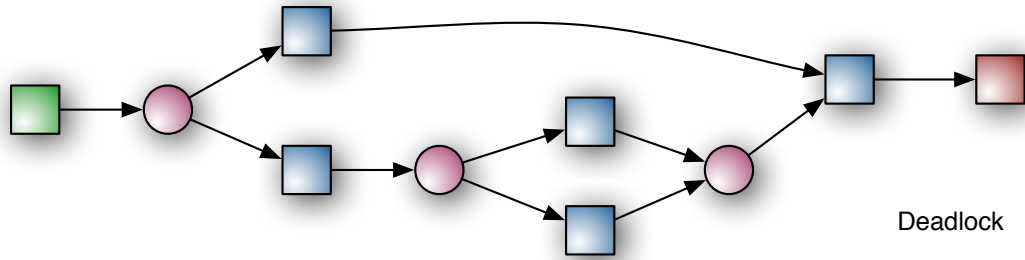


Figure 4.2.: A structural problem: A graph with a dead lock.

### 4.1.2. Instance Subgraph

A consistent workflow graph describes all possible workflows. In an acyclic workflow graph every single possible workflow can be described by an instance subgraph as a subgraph of the workflow graph. The structural correctness of the workflow graph implies the structural correctness of all instance subgraphs and vice versa.

**Definition 10 (Instance Subgraph)** A workflow graph  $W$  can be unambiguously divided into  $W_i$  instance subgraphs. Every instance subgraph describes one possible workflow without decisions. According to this, every transition of an instance subgraph except for the end transition has exactly one successor. No instance subgraph is a subgraph of another instance subgraph.

#### Structural Correctness

A acyclic workflow is structurally correct, if every instance subgraph is structurally correct. An instance subgraph is structurally correct, if all the following conditions are fulfilled:

- Every transition in the instance subgraph, except for the start transition, has exactly one predecessor.
- Every activity in the instance subgraph has the same predecessors as in the workflow graph.
- Every instance subgraph has exactly one end transition.

In figure 4.4 a workflow graph and its instance subgraphs are shown. If the first condition is not fulfilled a lack of synchronization exists and if the second condition is not fulfilled a deadlock exists in the workflow graph. If the third condition is broken, the workflow graph does not have a unique end.

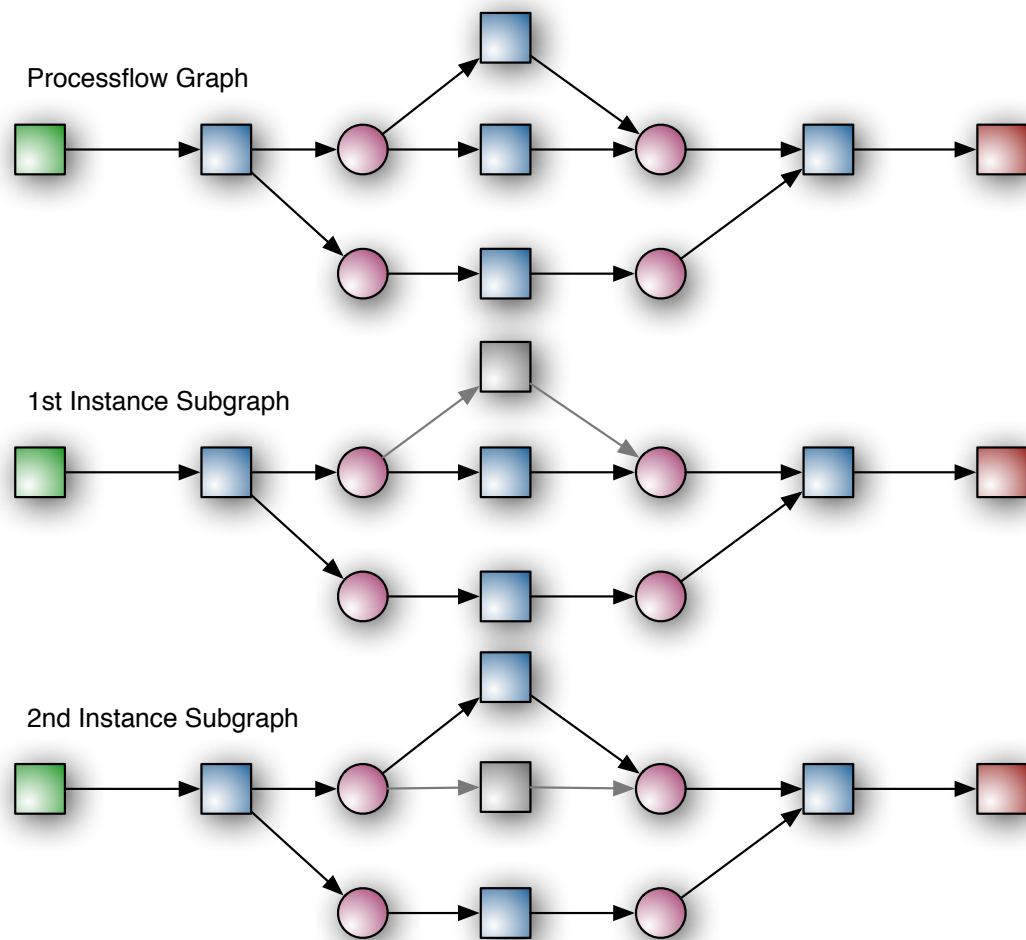


Figure 4.3.: An example graph with its instance subgraphs.

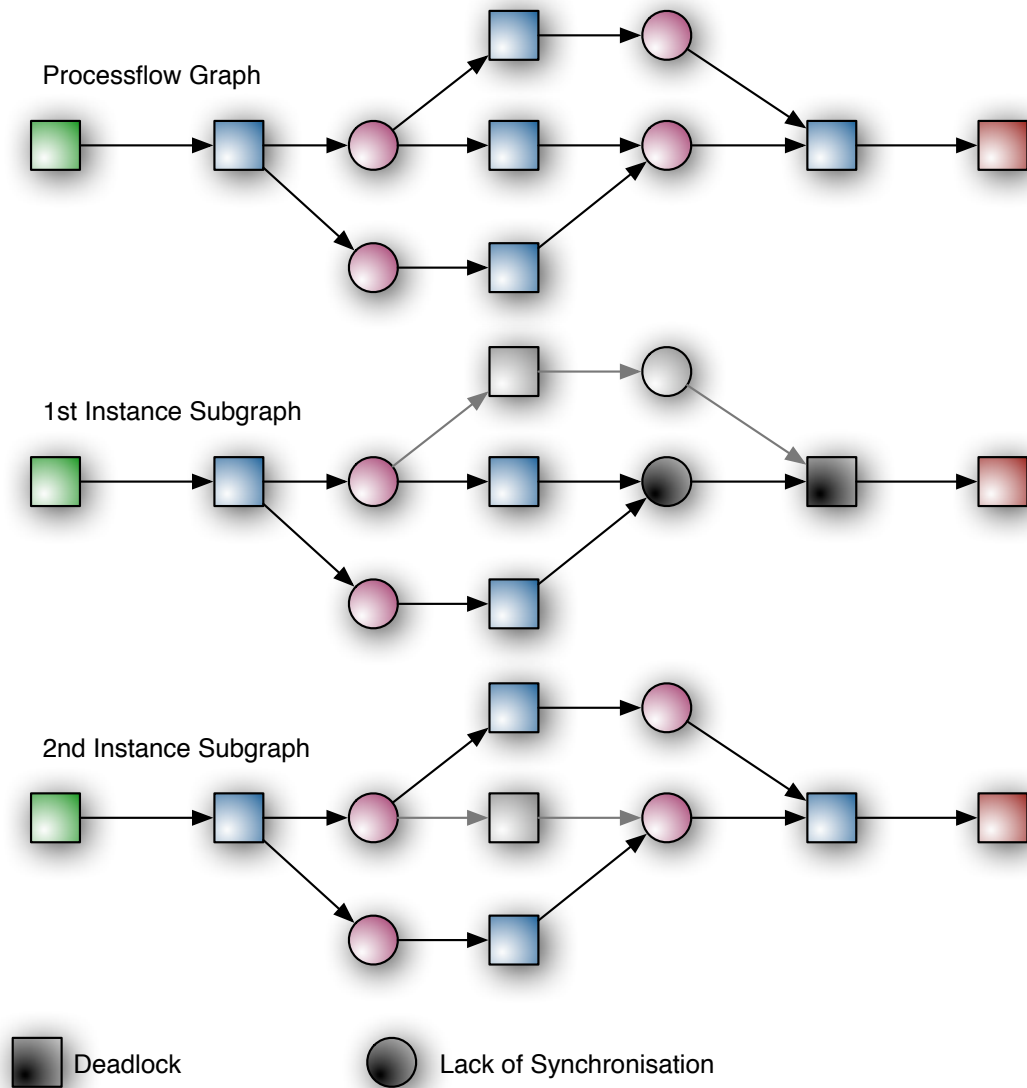


Figure 4.4.: An example for instance subgraphs of a workflow that is not structurally correct.

### 4.1.3. Algorithm

There are several algorithms to verify the structural correctness of such graphs, for example in [8] an extension of the already known algorithm from [11] to identify structural correctness is given. Both are based on various reduction rules that reduce the graph to an empty one if it is structural correct. Unfortunately both of this algorithms works only with acyclic graphs.

Another approach to verify the structural correctness is presented in [13]. The approach is not rule based as the two others but uses a transformation based approach. Therefore the graph will be transformed in a WF-net, some kind of Petri net, that allows to verify the structural correctness. Furthermore other analyses are possible with the corresponding analysis tool called Woflan which is also presented in that work.

For this work another algorithm was considered, which is based on instance sub-graphs and extends this approach to cyclic graphs.

#### Extension on non-acyclic graphs

So in [7] the known algorithms are discussed and the extension to non-acyclic graphs is done. I will short describe how the modification works: Two properties are needed to guarantee that the graph is structural correct:

1. The reduced workflow graph is structural correct
2. Every cyclic workflow component is structural correct

By replacing the strongly connected components through dummy nodes a reduced workflow graph is created, which can now processed as a simple acyclic graph, and tested for its structural correctness.

**Cyclic workflow component:** A cyclic workflow component is a cyclic subgraph with alternative input transitions and alternative output transitions. Input activities for cyclic workflow components are not allowed, because at least one predecessor is unreachable. Output activities are not allowed, because the following activities could be executed more than once.

**Cut of a workflow component:** A transition within a component with one or more predecessors from outside of this workflow component is called an input transition. A transition within the component with one or more successors from outside of this workflow component is called an output transition. Each input transition has to be cut and is replaced by one transition with all successors and one transition with all predecessors of the original input transition. Each output transition has to be cut and is replaced by one transition with all successors and one transition with all predecessors of the original output

transition. Transitions without predecessors are start transitions. Transitions without successors are end transitions.

When all input and output transitions are replaced, the workflow component is a new workflow graph. If this workflow graph is acyclic, it can be subdivided into instance subgraphs. If the workflow graph is not acyclic, it can be decomposed into strongly connected components and mapped to a reduced component graph. This component graph has to be checked for structural correctness as shown above. These steps will be repeated until no more strongly connected components are left.

**Compute the Instance subgraphs** An instance subgraph of a workflow graph can be created by following three simple rules:

1. Start at one start node and traverse the graph
2. For each FORK, the traversal is done for each successor
3. For each DECISION, the traversal needs only for one successor to be done

After the traversal of the workflow graph the instance subgraph contains all passed FORK and DECISION. For each start node and for each decision node exists one instance subgraph. In order to get all instance subgraphs, efficient algorithms based on depth-first search (DFS) and breadth-first search (BFS) can be used.

### 4.1.4. Another possibility to verify structural correctness in cyclic graphs

Another possibility is to extend the rule 4 (closed reduction rule) presented in [8] to allow cycles in such constructs.

The figure 4.5 illustrates the new rule, it allows to reduce cycles that occur between DECISION and MERGE nodes. I think, that this new rule does not break the proof given in [8], so all graphs that were reducible with this new rule are still structural correct. Of course further investigations and a formal proof is needed to make sure that there are no side-effects on the other rules.

If this rule does not break the others the verification of of any graph can be done in  $O((size[G])^2) \cdot O((size[N])^2)$ .

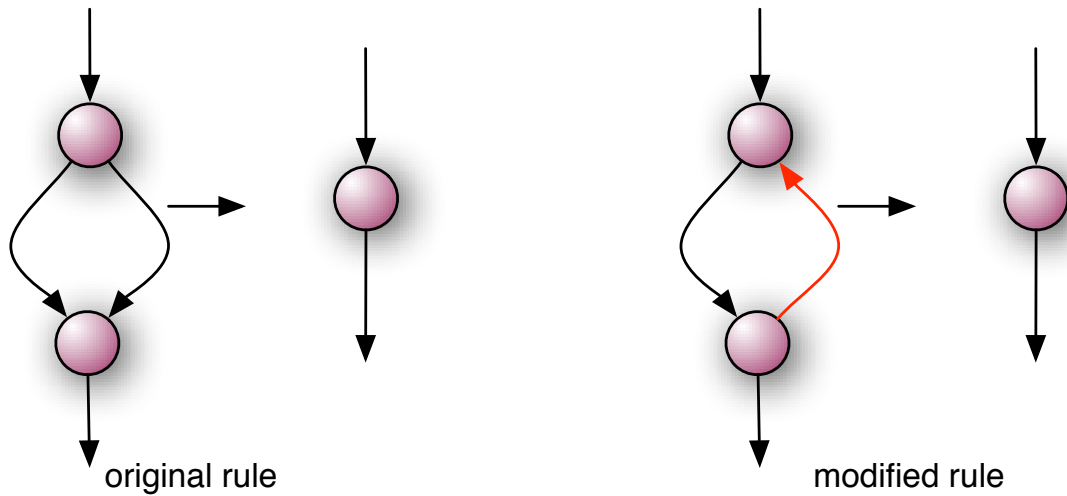


Figure 4.5.: Rule 4 and the extended version notice the modified red edge which allows also the application of Rule 4 in cyclic graphs under certain conditions

## 4.2. SESE-Regions

The second important algorithm that will be used is presented in [5]. It is a fast method to find the SESE regions of a graph. This one will be needed to find the different regions, for the classification process, that will follow.

### 4.2.1. What are SESE regions?

SESE-Regions are defined as follows:

**Definition 11 (SESE-Regions)** Two distinct nodes  $a$  and  $b$  in a CFG  $G$  enclose a single entry single exit region if

- $a$  dominates  $b$
- $b$  post-dominates  $a$
- $a$  and  $b$  are cycle equivalent in  $G$ .

With this definition it is possible that almost every node constitutes itself as a SESE region. But this is not the desired effect and so these trivial regions are excluded from the consideration by requiring, that  $a$  and  $b$  are distinct nodes.

The first two conditions ensure that control reaches  $b$  whenever it reaches  $a$  and vice versa. The third condition ensures that whenever control reaches  $a$ , it reaches

b before reaching a again and vice versa. The ordered pair (a,b) is used to denote a SESE region where a is the entry node and b the exit node.

### 4.2.2. Finding Canonical SESE regions

During the search for SESE regions it could be possible to come across with the following problem, the determination of non-trivial SESE regions of graphs.

As example if (a,b) is a SESE region and (b,c) is a SESE region, then (a,c) is a SESE region as well. Expressed differently, SESE regions have a certain transitivity property. So a graph with  $N$  nodes can have  $O(N^2)$  SESE regions - each of the  $N^2$  node pairs in a chain of  $N$  nodes encloses a SESE region. Normally a complete enumeration of all SESE regions is not useful. Instead, for each node  $x$  in the graph, it is necessary to find the smallest SESE regions, if they exist, for which  $x$  is an entry or an exit node. These regions will be called the canonical SESE regions associated with  $x$ . More formally:

1. Given a node  $x$ , find a node  $b$ , if it exists, such that,
  - (x,b) is a SESE region, and
  - if (x,b') is also a SESE region, then  $b$  dominates  $b'$ .
2. Given a node  $x$ , find a node  $a$ , if it exists, such that
  - (a,x) is a SESE region, and
  - if (a',x) is also a SESE region, then  $a$  post-dominates  $a'$ .

The following theorem, proofed in [5], is the key to solving this problem for all nodes in the graph:

**Theorem 2** *Let  $S$  be the strongly connected component constructed by adding an edge  $END \rightarrow START$  to a CFG  $G$ . Nodes  $a$  and  $b$  in  $G$  enclose a SESE region iff they are cycle equivalent in  $S$ .*

Now it is possible to find those canonical SESE regions, by using the algorithm to compute cycle equivalent nodes. Such an algorithm is presented in [5], explaining it here would go too far. This problem leads to the following definition:

**Definition 12** A node  $n$  in a graph  $G$  is contained within a SESE region (a,b) if  $a$  dominates  $n$  and  $b$  post-dominates  $n$ .

The node  $n$  is “between”  $a$  and  $b$  in the graph. This definition can be extended to the containment of SESE regions. The following theorem describes how canonical SESE regions are related:



**Theorem 3** *If  $R_1$  and  $R_2$  are two canonical SESE regions of a graph, one of the following statements applies:*

1.  $R_1$  and  $R_2$  are node disjoint.
2.  $R_1$  is contained within  $R_2$  or vice versa.
3. The exit node of  $R_1$  is the entry node  $R_2$  or vice versa.

Another problem that arise is the determination of the nesting of canonical SESE regions. In other words, canonical SESE regions cannot have partial overlap - if two regions have any nodes in common, they are either nested or in tandem. The proof of this theorem is found in [5].

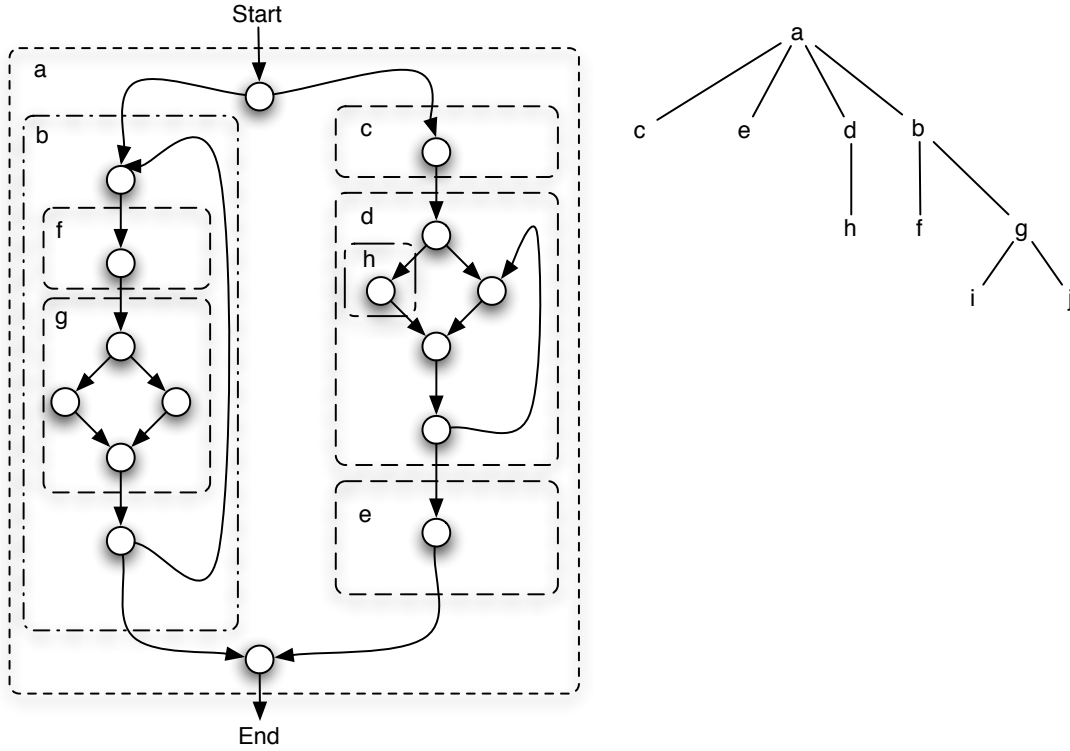


Figure 4.6.: Possible SESE regions, canonical SESE regions and the PST

As result of the permission of the nesting of such SESE regions another data structure emerges, the program structure tree (PST). It is a tree representation of the nesting of such SESE regions, and gives the possibility to traverse it and so classify every SESE region in the graph.

### Algorithm

The algorithm is presented in [6] and not further explained here. It can find the canonical SESE regions and compute the PST in  $O(E)$ .

## 4.3. T1-T2 Theorem

Another important analysis algorithm is the T1-T2 theorem presented in [4]. It is a simple algorithm to verify whether a given workflow graph or region is reducible. This algorithm will be necessary for the classification of regions, because the big two parts of the classification scheme are reducible and non-reducible regions, which will be handled differently in the transformation process.

### 4.3.1. T1-T2 Transformations

The theorem as presented in [4] introduces two simple transformation rules as you can see in figure 4.7.

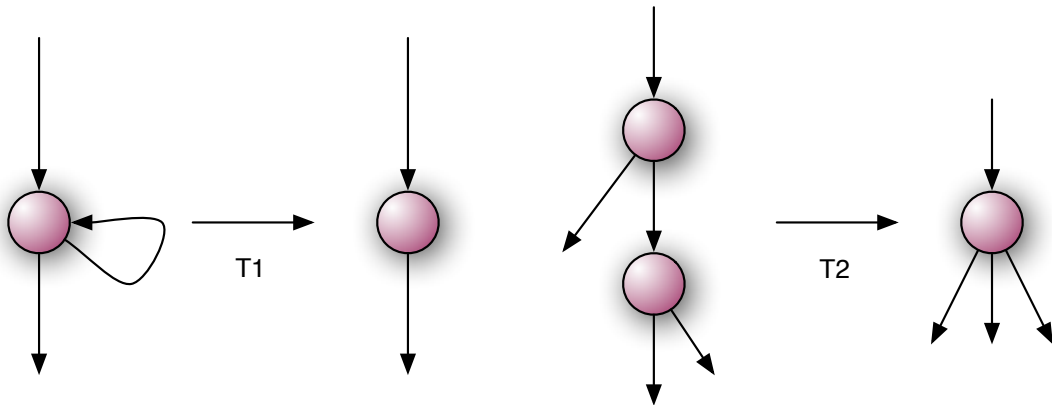


Figure 4.7.: The T1-T2 transformation rules.

This two simple rules allow to reduce if possible a workflow graph or region, and so allow to differentiate between reducible and non-reducible regions.

### 4.3.2. Misuse of the T1-T2 Theorem

The idea was to use the T1-T2 reduction rules to classify the current region, that is analyzed. Without loss of generality the T1 rule is used until it is no longer applicable, then the T2 rule is used until T1 can be used again. This is repeated

until the graph is reduced or it is detected that it is not reducible. During the application of the rules T1 and T2 the sequence of T1 and T2 is compared to a library, with all possible sequences for all known classified regions. So it would be easy to classify all regions during the test of reducibility. Unfortunately some different regions generate the same sequences. For example in figure 4.8, while- and until-loops are compared and as you can see, they result in the same sequence,  $(T2)^*$  until its a self-loop and the one T1. So you can not decide if it was a while- or until-loop that was detected. It might be possible to extend the algorithm with another rule T3 that detects when the outgoing edge of a region is moved, and with this knowledge differentiate between while and until loops for example.

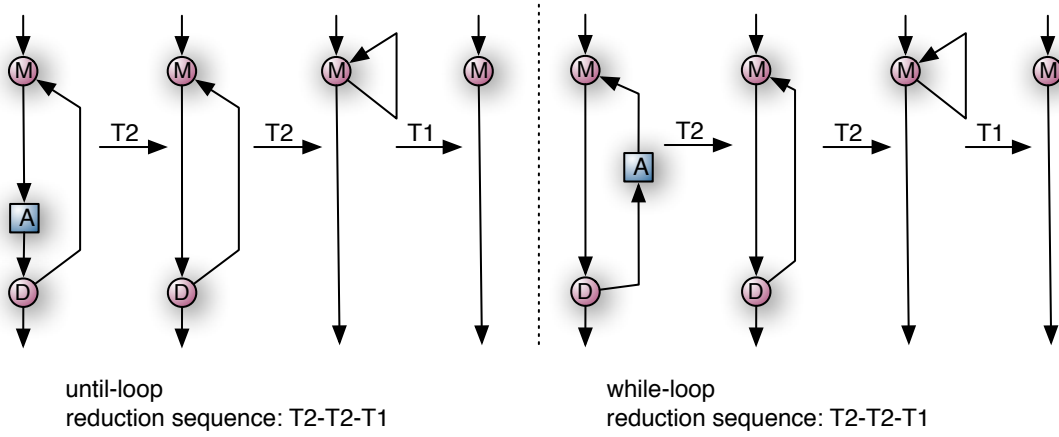


Figure 4.8.: Comparison of while- and until-loops after application of T1-T2 classification algorithm



## 5. Implementation

### 5.1. Architecture

The goal is to implement a modular architecture that allows to add different algorithms that work with such PFGs and to provide the possibility to transform from different models e.g. UML2 into PFG. These requisites led me to implement this as an Eclipse plug-in. The base of this plug-in will be an EMF model, that represents the PFG. It is introduced in the following subsection. The advantage of using EMF is that we can also use the Model Transformation Framework (MTF).

### 5.2. Eclipse Modeling Framework

The EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. This framework will build the base of the implementation. Different techniques can be used to build such a structured data model. In this context I will use Unified Modeling Language (UML) to design the model because some graphical tools provide support designing such a models.

### 5.3. Explanation of the EMF Model

As you can see the most important class is the **Node** which is the core modeling element of graphs. The class **Node** is extended by several subclasses like **Start** or **Decision**. This gives us the possibility to add fields or methods to specific nodes that are not a feature of the general **Node** class. The chart shows you also the top class, that contains all other, the **ProcessFlowGraph**. It contains the **Node** as well as the **Region**. And the last modeling class is the **Edge** which is contained by the **Node**. There are two more classes, which represent enumerations. The first class **MarkerColor** is used in several algorithms like DFS or BFS to mark already visited nodes. The second class is **regionType** which contains an entry for each known region type.

This model is very basic, but its intention is not to use it for modeling processes. More it is used as interface on different underlying modeling concepts, that all rely on the concept of nodes and edges. So it is necessary to model all common concepts

## 5. Implementation

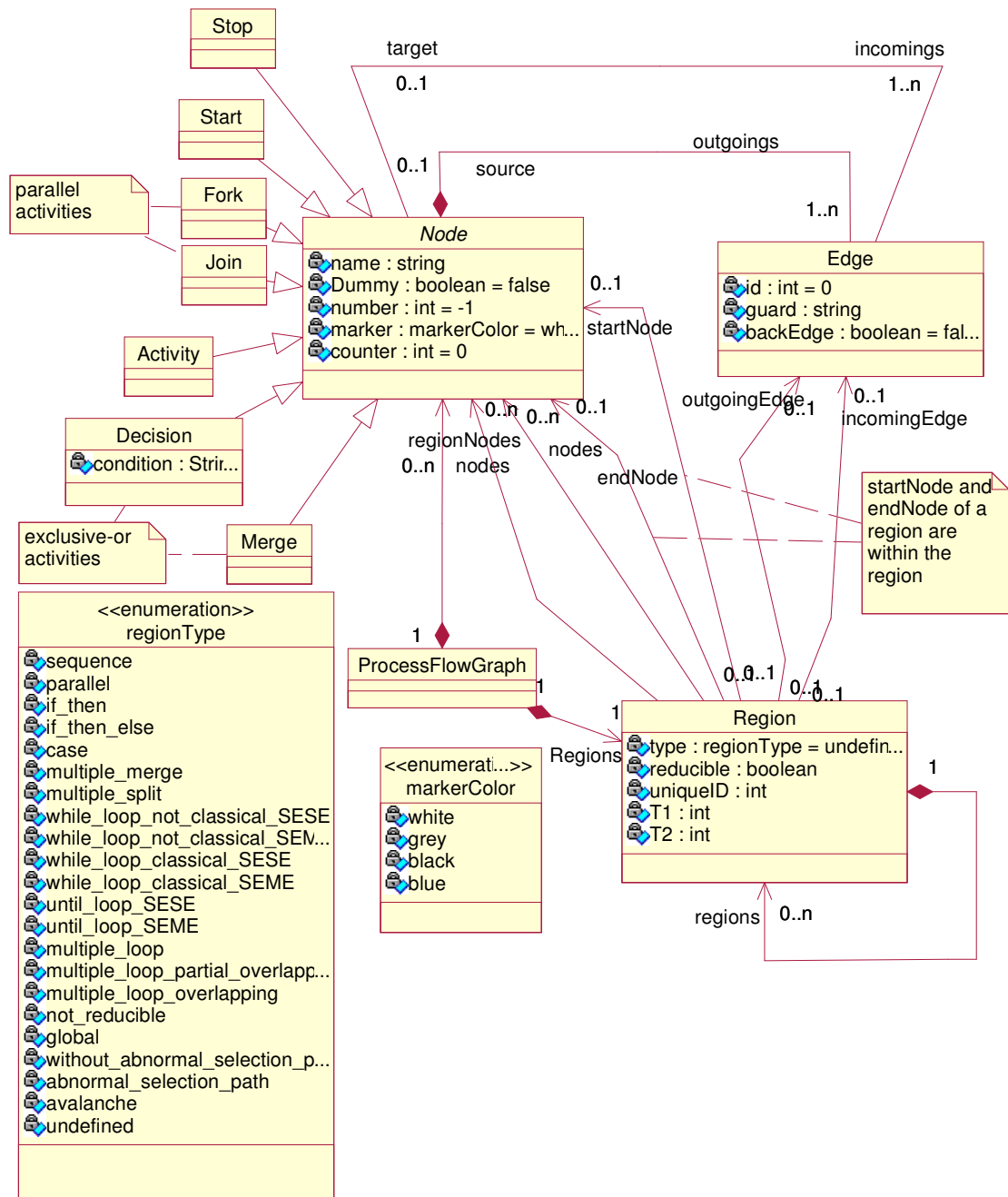


Figure 5.1.: This is the UML representation of the EMF model used for the implementation

from these underlying models to map them into this model. This contains such things as the `guard` attribute of the `Edge` or the extension of the `Decision` to have the `condition` attribute.

## 5.4. How the plug-in works

The plug-in provides several actions:

1. Create a PFG by hand
2. Verify the structural correctness of the PFG
3. Add and classify regions in the PFG
4. Restructure the PFG to prepare it for the conversion to BPEL
5. Create a new BPEL graph from the PFG

This four actions are separated into several classes as shown in figure 5.2. The following sections will show which task were implemented during this work.

### 5.4.1. Verify the structural correctness

To verify the structural correctness of a given PFG the rule based algorithm presented in [8] that reduces the graph was implemented. If the graph is reduced to the empty graph the graph is structural correct, if not it is not. This algorithm was extended as described in section 4.1.4 to do also the verification on cyclic graphs. The implementation is complete and full functional.

### 5.4.2. Add and classify the regions in the PFG

The detection and classification of the SESE regions in the PFG is the most complex task in this tool set. For this the algorithm presented in [5] was implemented. The classification is done by detecting incoming and outgoing node of such SESE regions, then all other nodes are investigated to classify the regions unambiguously. A rough overview of the different possible regions that can be detected only by consideration of the incoming and outgoing node are listed in the following table. This implementation is also complete and the classification is done completely.

The table below presents you a classification scheme based on the entry and exit node of a SESE-Region. As you can see some combinations like `DECISION` entry and `FORK` exit are not possible, and therefore they need no further investigation. The remaining can be split up into two big parts, concurrent and non-concurrent regions.

## 5. Implementation

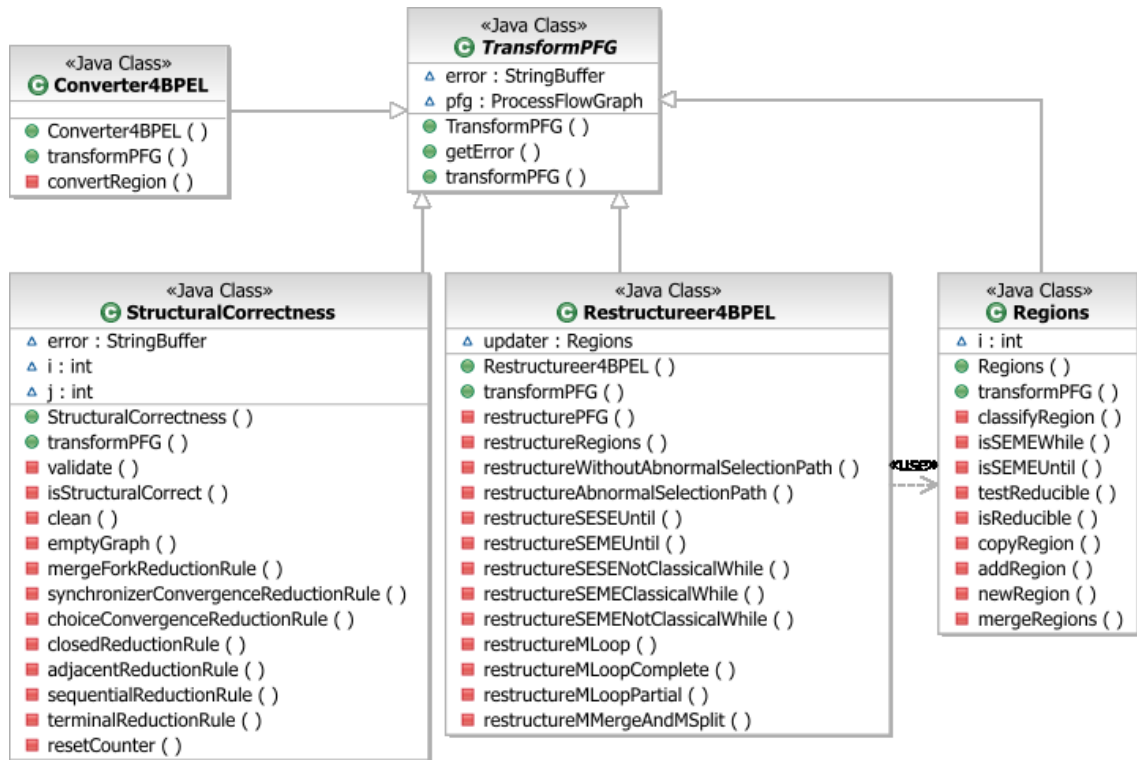
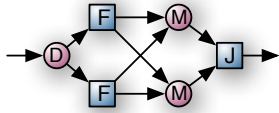
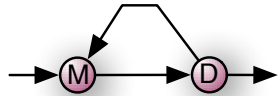
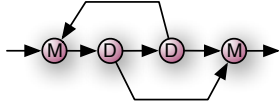


Figure 5.2.: The UML visualization of the abstract class **TransformPFG** and the implementing classes

Entry	Exit	Allowed	Region Classification	Example
Decision	Decision	yes	A traditional DECISION followed by a loop, which also ends in a DECISION	
Decision	Merge	yes	The typical XOR-Region, a DECISION followed by a MERGE	



Entry	Exit	Allowed	Region Classification	Example
Decision	Fork	no	Each fork has at least two outgoing edges, so at least one stays in the region. This path ends up in a join node, which is never activated because the fork is also never activated	
Decision	Join	yes	A not so familiar region where the FORKS are used to activate both MERGES to force that a JOIN is needed	
Merge	Decision	yes	The typical loop-Region	
Merge	Merge	yes	An overlapping XOR-and loop-Region	
Merge	Fork	no	same reason as Decision and Fork	
Merge	Join	no	no back-edge from the concurrent region to the merge node allowed	
Fork	Decision	no	no back-edge into the concurrent region from the decision node allowed	

## 5. Implementation

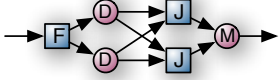
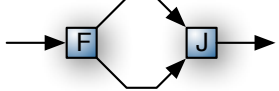
Entry	Exit	Allowed	Region Classification	Example
Fork	Merge	yes	A similar region to that one beginning with a DECISION and ending with a JOIN. The region is structural correct nevertheless it is possible that a Deadlock occurs during execution, so I reject it during my verification of structural correctness	
Fork	Fork	no	same reason as Decision and Fork	
Fork	Join	yes	typical concurrent region	
Join	Decision	no	never possible, the join will be never activated because, the region is never entered	
Join	Merge	no	same as above	
Join	Fork	no	same as above	
Join	Join	no	same as above	

Table 5.1.: Table of all possible combinations of ingoing and outgoing nodes for SESE region

The table 5.1 gives you an overview of possible regions, there may be different regions embedded, where in the table are only edges shown. So for example the loop region can be either a while-loop or until-loop, it depends where another region is embedded. In figure 5.3 an until- and a while-loop are compared. The until-loop has the decision at the end of the loop body, other the while-loop, which has first the decision and then the loop body. The loop body is of course any SESE region.

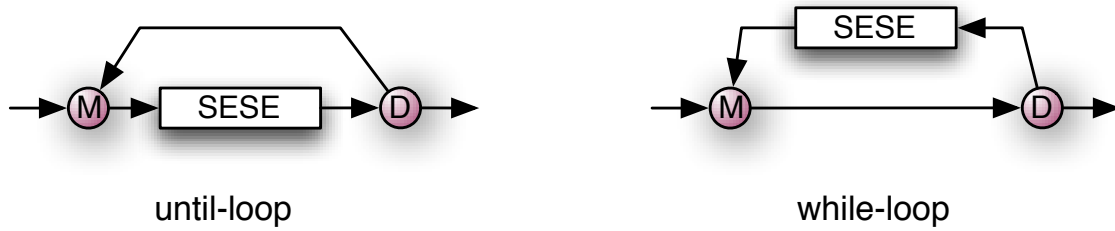


Figure 5.3.: Comparison between an until- and a while-loop

### 5.4.3. Restructure the PFG to prepare it for the conversion to BPEL

This task is done by traversing all regions and restructuring them, if they are not directly representable in BPEL. I used the restructuring proposals made in [12]. More details are in chapter 6. The implementation of this task is not complete, only until-loops and not classical while loops can be restructured.

### 5.4.4. Create a new BPEL graph from the PFG

In this task a new BPEL graph is created from the PFG. Details on this can be found in chapter 6.

## 5.5. Extending the plug-in

To extend the plug-in it just extend the abstract class `TransformPFG`, which works on the PFG model. In figure 5.2 you can see four classes that implement it. The `StructuralCorrectness`, `Restructureer4BPEL`, `Regions` and `Converter4BPEL`. All these classes overwrite the `transformPFG` method, and do their work in this method. They are instantiated and their methods are called from the corresponding class in the package `processflowgraph.convert.popup.actions`. They are shown in figure 5.4. So the class `AddRegions` instantiates the class `Regions` and calls the method `transformPFG()`. For the other actions it is similar.

So if you want to add a new algorithm that works on the PFG you have to do the following three steps:

1. Extend the `TransformPFG` class and overwrite the `transformPFG` method, any errors that occur during the execution of the `transformPFG` method should be written into the `StringBuffer` error. This new class should be placed in the `processflowgraph.convert.util` package.

## 5. Implementation

2. Add a new class that implements `IObjectActionDelegate` and is placed in the `processflowgraph.convert.popup.actions` package. Implement the method `run(IAction action)` and call `transformPFG`.
3. Add this new class to the `plugin.xml` as new extension.

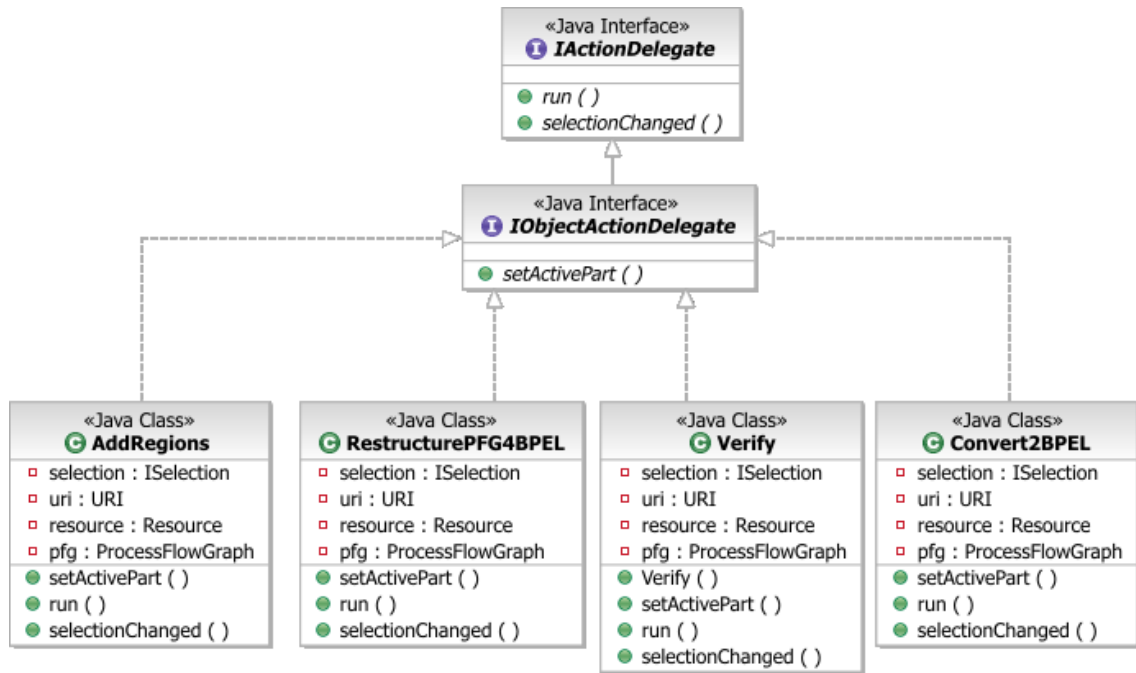


Figure 5.4.: The UML visualization of the popup action classes

## 6. The PFG to BPEL Transformation

This chapter describes how the transformation is actually done. During the transformation it can occur that there are regions, that are not "compatible" with BPEL. The complete enumeration, how each region is mapped into a BPEL equivalent is shown in section 6.1.1. But before the graph is converted into BPEL the restructuring is done. This is a PFG to PFG mapping that restructures all regions, that are not directly mappable in BPEL or if this is not possible the region will be marked and an error occurs.

### 6.1. Transformation

The transformation from PFG to BPEL is easy if all regions found in the PFG are directly mappable in the BPEL graph. During the transformation all the earlier introduced algorithms are executed. The figure 6.1 gives an impression of the execution of the single tasks. If everything is ok only only four steps are needed to convert any PFG into BPEL.

1. Verify the structural correctness of the PFG.
2. Detect all SESE regions
3. Transform every SESE into its BPEL compatible PFG equivalent
4. Convert every prepared SESE region into BPEL

Each of this steps is implemented as a own class, which provides the modular architecture that is required. The benefits of this approach are, if a new algorithm for one of the steps will be presented, it is easy to adapt the transformation queue to invoke the new knowledge acquired. Another use case could be if the BPEL standard changes it would be easy to adapt the new standard within the transformation library.

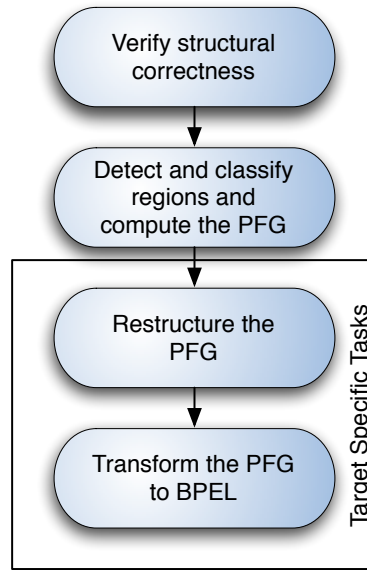


Figure 6.1.: Workflow of the invocation of the presented algorithms.

### 6.1.1. Transformation Library

Earlier in section 2.2 the elements BPEL consists of were introduced. This library contains for each pattern detected in the PFG a corresponding implementation in BPEL, it is based on the work of [12]. The following table gives you an overview. Some of the presented pattern require a modification as long as they are still PFGs.

PFG	BPEL
if-then region	<b>switch-activity</b> with one case
if-the-else region	<b>switch-activity</b> with one case and one otherwise
case region	<b>switch-activity</b> with multiple cases
classical while loop	<b>while-activity</b>
activity	<b>invoke-activity</b>
parallel	<b>flow-activity</b>
sequence	<b>sequence-activity</b>

Table 6.1.: The mapping from PFG to BPEL

The following sections will show you the PFG region and the corresponding modeling elements in BPEL. They are here not shown as XML-snippets but as images as they are shown in the Websphere Integration Developer.

### if-then-else regions

If-then and If-then-else regions are directly representable in BPEL using the **switch**-activity. If it is a simple if-then statement only the **case**-statement will be needed, if it is an if-then-else statement an **otherwise**-statement is added. The conditions are copied if possible.

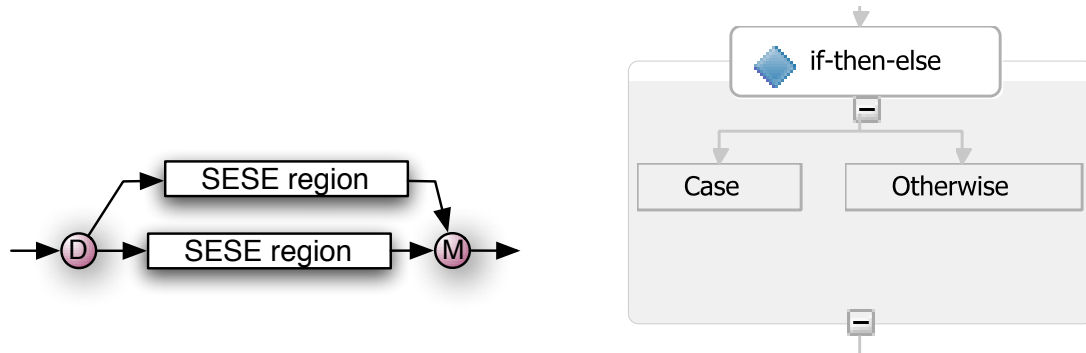


Figure 6.2.: A if-then-else region as PFG and in BPEL

### case

Case regions are similar to the if-then-else regions transformed. A **switch**-activity is added to the BPEL graph and for each case in the region a **case**-statement for the BPEL graph is generated. The conditions are copied if possible.

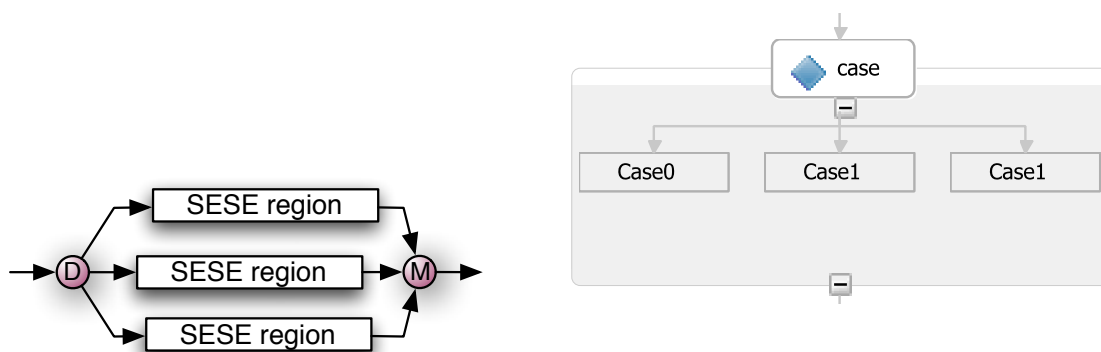


Figure 6.3.: A case region as PFG and in BPEL

### classical SESE while loop

While regions are easy to transform. For each detected while loop in the PFG we generate a **while**-activity in the BPEL graph. The condition of the while is copied from the condition of the DECISION node. Then all embedded regions are processed and added into the **while**-activity.

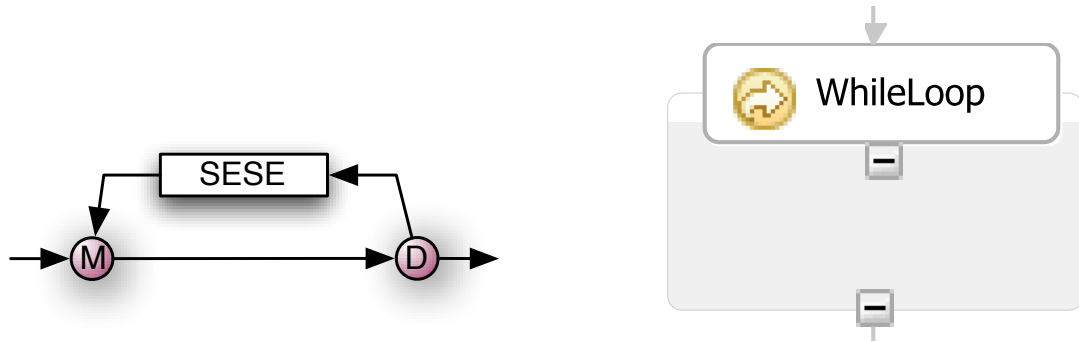


Figure 6.4.: A classical while loop as PFG and in BPEL

### activity

The transformation of an activity is straight forward. For each activity in the PFG an **invoke**-activity in the BPEL graph is generated. Then, if necessary edges are added to connect the BPEL activities.



Figure 6.5.: An ACTIVITY node of the PFG and the possible representation in BPEL

### parallel

Parallel regions with and without synchronize edges can be transformed very directly. We just generate a **sequence**-activity and then add each embedded region to this sequence, while keeping all parallel paths separated and we add the synchronize edges were necessary.



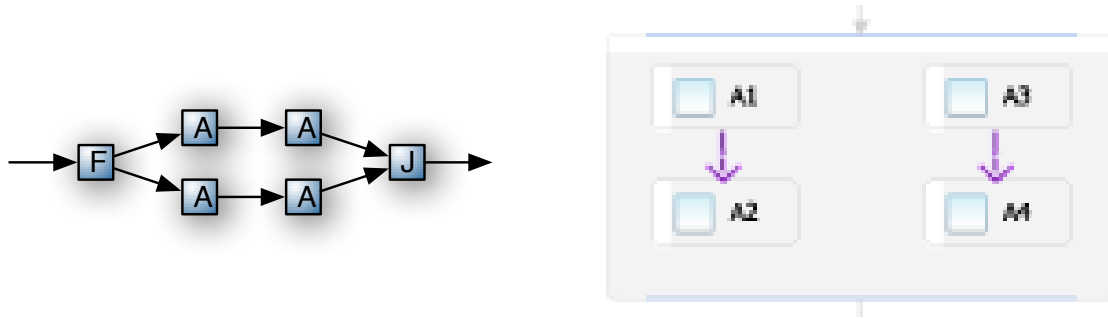


Figure 6.6.: A parallel region as PFG and in BPEL

## 6.2. Problems

The following region types are more problematic to transform them to BPEL. For the SESE until loop and the multiple merge- and -split regions a restructuring was implemented. This was also done for the SESE not classical while loop.

### 6.2.1. Regions which need restructuring

#### SESE Until Loop

One of these is the SESE until loop, which can not directly converted into BPEL because BPEL only allows while-loops, but no until-loops. Because of the equivalence of until- and while-loops we can transform such SESE until loops either by node-duplication or by the introduction of a new variable, which is only relevant for the transformation to BPEL, into SESE classical while loops.

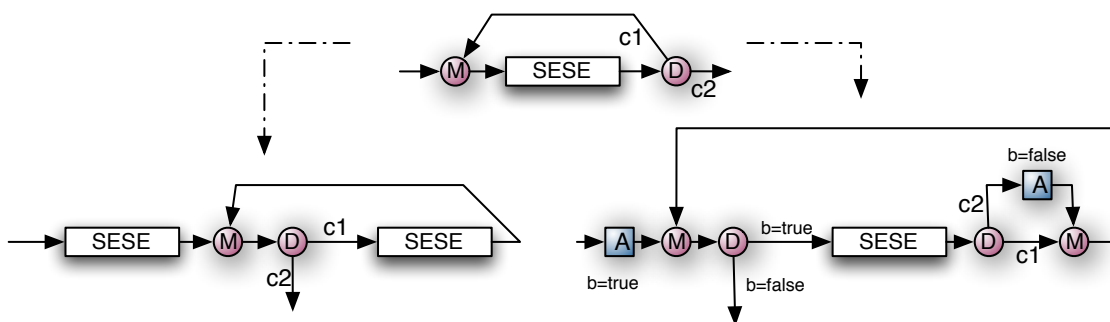


Figure 6.7.: A SESE Until Loop and both possible restructured while loops

The necessary steps are shown in figure 6.7. Now follows a short instruction how the transformation is done. I've implemented the variant which does the node

duplication, because the condition of the loop is untouched.

1. move DECISION from the end to behind the MERGE and update the origin of the “back-edge” to the last node of the last region contained in the loop.
  - a) either modify the condition of the DECISION and add another ACTIVITY to force that at least one time the loop body is executed.
  - b) or copy the whole loop body before the merge node, and let the condition of the DECISION untouched

### Multiple Merge- and -Split Regions

Figure 6.8 shows you how a multiple merge region could look like.

This kind of region is directly representable in BPEL. But we would have to use a **flow-activity** which becomes very complex with increasing number of edges. This is the reason why I have implemented a restructuring method to restructure the multiple merge- and -split regions into if-then-else regions. The algorithm is very easy to understand, as long as a DECISION has more than 2 outgoing edges I generate a new DECISION 1 node, and move two outgoing edges from the originating DECISION node to the new one. Then a new edge between the originating and new node is inserted and the condition of the new outgoing path is set to  $(\text{Cond1} \ || \ \text{Cond2})$  where **Cond1** and **Cond2** were the conditions of the outgoing edges of the originating node. The figure 6.8 gives you an impression how it looks like. Multiple Merge- and -Split regions are restructured in my implementation.

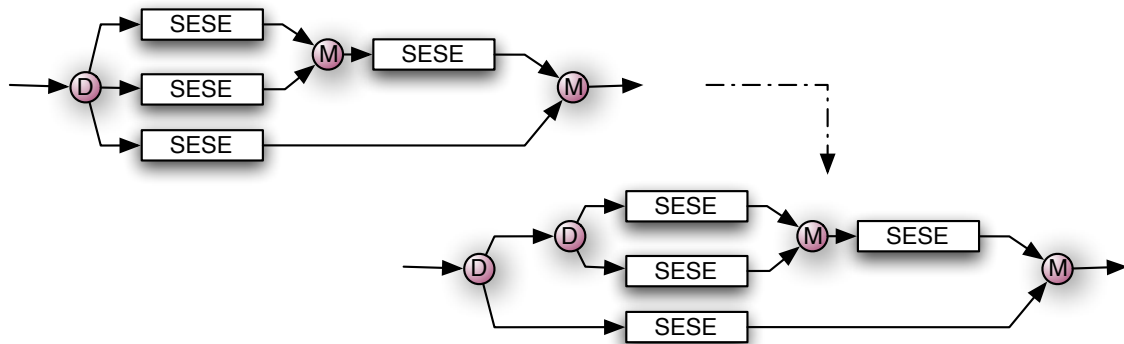


Figure 6.8.: A multiple merge region and the corresponding restructured equivalent

After each restructuring step it is necessary to re-run the algorithm for the detection and classification of the regions, because all this restructuring methods invalidate the current regions and their classification.

### Not Reducible Regions

This kind of region becomes marked as impossible to transform and a message is generated to tell the user that he has to restructure this region to make it possible, that it can be transformed into BPEL. In a previous work [12] other methods from [18, 3, 1] are mentioned, but none of them was implemented in this work.

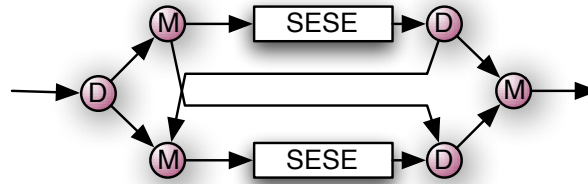


Figure 6.9.: A not reducible region

### SESE not classical While Loop

The restructuring of this region is done in two steps. First the region is transformed into an until loop, and then it is transformed like a classical until loop. The second step, the restructuring from an until loop to a while loop can also be done in two different ways, as stated above in section 6.2.1

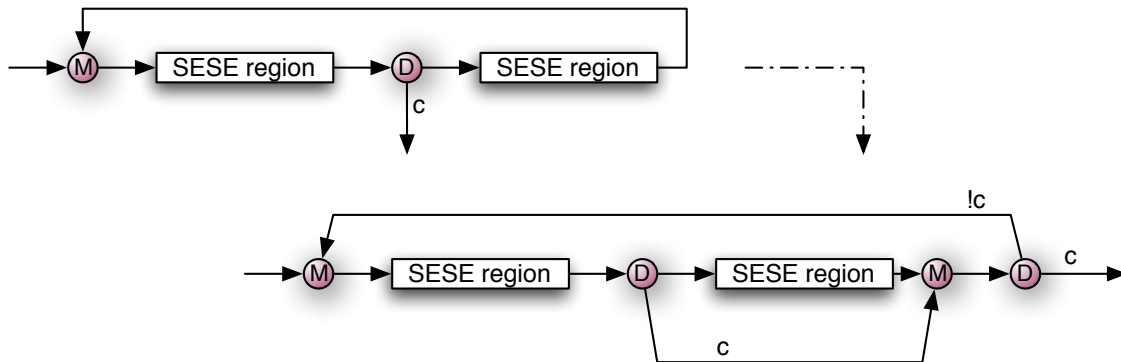


Figure 6.10.: A SESE not classical while loop

### Other regions

The following regions are also not directly mappable into BPEL and no restructuring is implemented for them.

- SEME classical While Loop
- SEME Until Loop
- SEME not classical While Loop
- regions without abnormal selection path
- regions with abnormal selection path
- multiple loop regions without overlapping
- multiple loop region with partial overlapping
- multiple loop regions with complete overlapping

# 7. Summary and Outlook

## 7.1. Summary

This diploma thesis is about the detection and classification of regions in workflow graphs. To solve this task a abstract model for the PFG was defined and an existing classification scheme for regions was extended. Several algorithms for the detection, analysis and classification of regions were examined and finally implemented.

For the implementation the abstract model was realized as an EMF model, and all algorithms implemented, work with this model. The resulting plug-in contains a small tool set, that allows to verify the structural correctness of such PFGs, to add and classify the regions and to restructure two regions for BPEL.

### 7.1.1. Limitations

Unfortunately not all implementations are complete, so the restructuring for example furthermore, the algorithm which is implemented to verify the structural correctness may be not correct. A prove of this is needed in any succeeding work.

Another problem is that a BPEL process is not only defined by its graph which can be generated easily from the PFG generated by any modeling software. There are other components, such as Partners and Variables, that are not contained in a PFG but needed to execute a BPEL process.

And last the model design for the PFG is not adequate. It contains much knowledge that is only needed for the execution of some algorithms but not an essential component of the graph.

## 7.2. Outlook

For the future it could be possible that an import or mapping from UML2 activity diagrams to PFG is possible. Therefore it would be wise to rework the EMF model and approximate the model of the activity diagrams.

Another task could be to implement a full set of restructuring methods and the generating of BPEL , perhaps even a full functional BPEL process.

It could also be possible for the application that the execution of the presented algorithms could be changed. First do the detection of regions and then do every analysis, such as structural correctness or reducibility, in the second step. Finally

## 7. *Summary and Outlook*

---

more analysis algorithms could be implemented and provided, a candidate for this could be another verification algorithm found at [\[9\]](#).

# A. Appendix

## A.1. Manual

### A.1.1. Installation

The plug-in is provided as sourcecode only. Please follow the instructions in section A.1.3. To compile the plugin you need the following prerequisites:

- EMF 2.1.0

### A.1.2. Use

Select in the resource view the file you want to convert and select one of the possibilities from the context menu as seen in figure A.1. You have to choose of

**Verify Structural correctness** Verifies the structural correctness. This task does not require a previous run of **Add and Classify Regions**

**Add and Classify Regions** Adds regions to the graph and classifies them. This task is needed for almost any analysis task.

**Restructure the PFG** Restructures the PFG to map it to BPEL.

**Convert the PFG to BPEL** Runs all the above tasks and finally creates a BPEL file (not implemented yet)

### A.1.3. Sourcecode Repository

The sourcecode of this plug-in is available from a Subversion repository. The URL is <https://opensvn.csie.org/Diplomarbeit>. You can simply add the repository location within your eclipse using the subversion plug-in. The login is: ibm, the password is: mssllap.

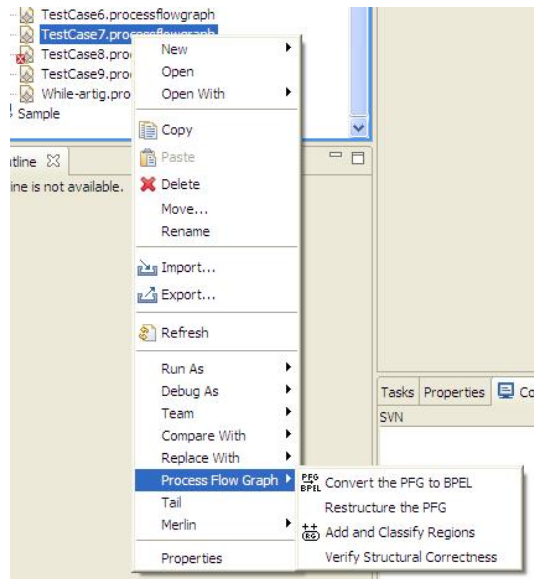


Figure A.1.: The context menu with the provided eclipse plug-in.

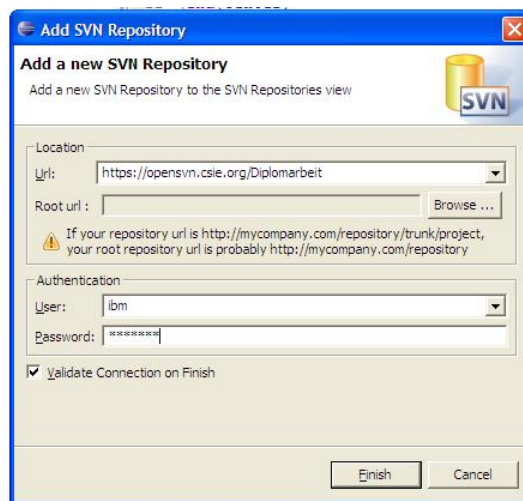


Figure A.2.: SVN configuration dialog from Eclipse



# List of Acronyms

## **breadth-first search (BFS)**

A search algorithm that considers neighbors of a vertex, that is, outgoing edges of the vertex's predecessor in the search, before any outgoing edges of the vertex. Extremes are searched last. 20, 26

## **Business Process Execution Language (BPEL)**

Is an XML grammar defining and standardizing structures necessary for web services orchestration. BPEL's focus on modern business processes, plus the histories of WSFL and XLANG, led BPEL to adopt web services as its external communication mechanism.[\[14\]](#) ii, 1–4, 28, 32, 34–40, 42, 44

## **Control Flow Graph (CFG)**

A directed graph consisting following node types: START, STOP, ACTIVITY, DECISION and MERGE 9, 11, 13, 21, 22

## **depth-first search (DFS)**

Any search algorithm that considers outgoing edges of a vertex before any neighbors of the vertex, that is, outgoing edges of the vertex's predecessor in the search. Extremes are searched first. 20, 26

## **Eclipse Modeling Framework (EMF)**

Eclipse Modelling Framework is a modeling framework and code generation facility for building tools and other applications based on a structured data model. ii, 26, 42

## **Model Transformation Framework (MTF)**

The Model Transformation Framework is a set of tools that helps developers make comparisons, check consistency, and implement transformations between Eclipse Modeling Framework models. The framework also supports persistence of a record of what was mapped to what by the transformation; this record can be used to support round-tripping, reconciliation of changes, or display of the results to a user. 26

### **Process Flow Graph (PFG)**

An extension to the Workflow Graph allowing cycles ii, 1, 2, 6, 9–11, 13, 15, 26, 28, 32, 34–37, 42, 44

### **program structure tree (PST)**

The PST is a hierarchical representation of the control structure of a program based on single entry single exit regions. 23, 24

### **Single Entry Single Exit (SESE)**

A part of a graph starting with a domination node and ending with an post-dominating. See also 8 ii, 11, 12, 21–24, 28, 31, 34, 37, 38

### **Two-Terminal Region (TTRegion)**

A Two-Terminal Region is a subgraph that is entered through only one node and left through only one node. 11, 12

### **Unified Modeling Language (UML)**

The Unified Modeling Language is a non-proprietary, third generation modeling and specification language. UML is not restricted to modeling software. As a graphical notation, UML can be used for modeling hardware (engineering systems) and is commonly used for business process modeling, systems engineering modeling, and representing organizational structure. [16] 26

### **Web Services Flow Language (WSFL)**

The Web Services Flow Language (WSFL) is an XML language for the description of Web Services compositions. 4

### **Workflow Graph (WFG)**

A directed (acyclic) graph consisting of the following node types: START, STOP, ACTIVITY, FORK, JOIN, DECISION and MERGE 6, 8, 9, 13

### **XLANG (XLANG)**

XLANG is an extension of the Web Service Definition Language. It provides both the model of an orchestration of services as well as collaboration contracts between orchestrations. 4

# Bibliography

- [1] AMMARGUELLAT, Z. A control-flow normalization algorithm and its complexity. *Software Engineering* 18, 3 (1992), 237–251.
- [2] ANDREWS, T., CURBERA, F., DHOLAKIA, H., AND GOLAND, Y. Business process execution language for web services (version 1.1). Tech. rep., IBM and Microsoft and BEA and Siebel Systems, 2003.
- [3] EROSA, A. M., AND HENDREN, L. J. Taming Control Flow: A Structured Approach to Eliminating GOTO Statements. In *ICCL* (1994).
- [4] HECHT, M. S., AND ULLMAN, J. D. Flow graph reducibility. In *STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing* (New York, NY, USA, 1972), ACM Press, pp. 238–250.
- [5] JOHNSON, R., PEARSON, D., AND PINGALI, K. Finding regions fast: Single entry single exit and control regions in linear time. Tech. rep., Cornell University, Ithaca, NY, 1993.
- [6] JOHNSON, R., PEARSON, D., AND PINGALI, K. The program structure tree: Computing control regions in linear time. In *SIGPLAN Conference on Programming Language Design and Implementation* (1994), pp. 171–185.
- [7] KLINGER, A., KÖNIG, M., AND BERKHAHN, V. Structural correctness of planning processes in building engineering. Tech. rep., University of Hannover, 2004.
- [8] LIN, H., ZHAO, Z., LI, H., AND CHEN, Z. A novel graph reduction algorithm to identify structural conflicts. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9* (Washington, DC, USA, 2002), IEEE Computer Society, p. 289.
- [9] PERUMAL, S., AND MAHANTI, A. A graph-search based algorithm for verifying workflow graphs. In *DEXA Workshops* (2005), IEEE Computer Society, pp. 992–996.
- [10] SADIQ, W., AND ORLOWSKA, M. E. Applying graph reduction techniques for identifying structural conflicts in process models. In *CAiSE '99: Proceedings of the 11th International Conference on Advanced Information Systems Engineering* (London, UK, 1999), Springer-Verlag, pp. 195–209.

- [11] SADIQ, W., AND ORLOWSKA, M. E. Analyzing process models using graph reduction techniques. *Inf. Syst.* 25, 2 (2000), 117–134.
- [12] STÖHR, K. Aspekte der Abbildung von Geschäftsprozessen, spezifiziert im Business Process Definition Metamodel, in BPEL4WS. Master's thesis, Universität Leipzig, 2005.
- [13] VAN DER AALST, W. M. P., HIRNSCHALL, A., AND VERBEEK, H. M. W. An alternative way to analyze workflow graphs. In *Advanced Information Systems Engineering, 14th International Conference, CAiSE 2002 Toronto, Canada, May 27-31, 2002 / A. Banks Pidduck, J. Mylopoulos, C.C. Woo, M. Tamer Ozsu (Eds.)* (May 2002), Springer Verlag, LNCS 2348, pp. 1–535pp. Internal-Note: Submitted by: hr.
- [14] WIKIPEDIA. Bpel — wikipedia, the free encyclopedia, 2005. [Online; accessed 30-August-2005].
- [15] WIKIPEDIA. Business process — wikipedia, the free encyclopedia, 2005. [Online; accessed 07-July-2005].
- [16] WIKIPEDIA. Unified modeling language — wikipedia, the free encyclopedia, 2005. [Online; accessed 30-August-2005].
- [17] WIKIPEDIA. Workflow — wikipedia, the free encyclopedia, 2005. [Online; accessed 07-July-2005].
- [18] WILLIAMS, M., AND OSSHER, H. Conversion of unstructured flow diagrams into structured form. *The Computer Journal* 21, 2 (1978), 161–167.