Product-Based Design and Support of Workflow Processes

CIP-DATA LIBRARY EINDHOVEN UNIVERSITY OF TECHNOLOGY

Vanderfeesten, Irene T.P.

Product-Based Design and Support of Workflow Processes / by Irene T.P. Vanderfeesten. –
Eindhoven: Eindhoven University of Technology, 2008. - Proefschrift. –

A catalogue record is available from the Eindhoven University of Technology Library.

ISBN: 978-90-386-1506-6
NUR: 982
Keywords: Business Process Redesign (BPR) / Workflow Management / Product Based
Workflow Design (PBWD) / Product Data Model (PDM) / process models

The research described in this thesis has been carried out under the auspices of Beta Research
School for Operations Management and Logistics. Beta Dissertation Series D116.

This research was financially supported by the Dutch Technology Foundation STW (6446).

Printed by Printservice, Eindhoven University of Technology
Cover photograph "Callicarpa" (schoonvrucht): Jac Vanderfeesten
Cover design: Joyce Vanderfeesten, Irene Vanderfeesten

Product-Based Design and Support of Workflow Processes

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr.ir. C.J. van Duijn,
voor een commissie aangewezen door het College voor Promoties
in het openbaar te verdedigen
op dinsdag 3 februari 2009 om 16.00 uur

door

Irene Toos Peter Vanderfeesten

geboren te Weert

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. W.M.P. van der Aalst

Copromotor:
dr.ir. H.A. Reijers

Preface

The PhD thesis you are holding is the result of the research I conducted during the past four years. Many persons have helped me and supported me along the way and I would like to express my thanks to them here.

First of all, I would like to thank my supervisors: Wil van der Aalst, for stimulating me to explore and exceed my own limits and to get the best out of myself, and Hajo Reijers, for believing in my capabilities at the time I did not believe in myself. I am also grateful for the pleasant collaboration with some of my colleagues abroad. Jorge Cardoso gave me the opportunity to visit the University of Madeira to work together on a preliminary version of the CC-metric. With Jan Mendling (Humboldt-University Berlin) I have had very pleasant and fruitful discussions on business process metrics. Our cooperation eventually led to the CC-metric presented in Chapter 7. Moreover, I would like to thank Barbara Weber (University of Innsbruck) and Dominic Müller (University of Ulm) for the discussions we had. My gratitude also goes to two of the Master students I co-supervised during my PhD-project: Jan Vogelaar, who helped me with the case study at the Provincie Noord-Brabant, and Johfra Kamphuis, who implemented three of the seven algorithms to automatically generate process models. I am indebted to the members of the STW user committee for their valuable feedback and enthusiasm: Robert van der Toorn (ING IM), Martin de Bijl (Dimpact), Paul Berens, Paul Eertink, Dolf Grünbauer, Maarte van Hattem, and Remmert Remmerts de Vries (Pallas Athena).

I also thank my colleagues in Eindhoven. The time I spent in the IS group has been very pleasant and I will certainly miss the positive working atmosphere, the valuable discussions and the nice social activities. I would like to express my special thanks to colleagues with whom I have worked more closely together or who have become my friends. Monique, my officemate, and Mariska, my projectmate, thank you for sharing all 'ups' and 'downs' with me during the past years; Ana Karla, Anke, and Ting, thank you for the nice dinners we had and the enjoyable trips we made; Paul, thank you for the conversations and good advice; Ineke, Ada, and Annemarie, thanks for your valuable secretarial support; Jochem, Eric, Boudewijn, and Christian, thanks for your technical support and your willingness to help with any issue; and Marcel, thank you for your explanations of Markov theory.

There are also a number of persons who have been a great support on a personal level. I am grateful to my friends Remco, Rogier, Karin and Mark, Deborah and Martijn, Marcel and Claudia, Jeanien, Mark and Shiva, Danny, Roland and Gonnie, Wilfred, Koos, and Frank and Paula for the good times we spent together and for their patience and understanding during the most stressful moments of my project. I also would like to thank my friends at the dancing school for many enjoyable moments. Special thanks to my dancing partners Robert, Wil, and René, for making me forget my troubles by leading me around the floor.

My final words are for the ones dearest to me. Pap en mam, thank you for your love, support and confidence in me. Without you I would not have become the person I am now.

Joyce, thanks for helping me to put things into perspective and for your contribution to the cover of this thesis. Ron, thank you for your help and understanding. During the last year of this project, you have been my soul mate and a very special coach with whom I could share all my happiness and worries.

Irene Vanderfeesten
December 2008

Contents

Chapter 1

Introduction

In today's fast growing and competitive market, companies face the challenge of providing their products and services cheaper, while maintaining high quality, and minimizing throughput times. Therefore, many companies are continuously improving their business processes to become more efficient and more effective. From the beginning of the 1990s, the value of focusing on cross-departmental business processes to improve performance has been recognized as a powerful alternative to improving the functionality of departments within the company in isolation [79, 121]. The organization of entire business processes is reconsidered across the various departmental borders and the business processes are redesigned in the context of so-called *Business Process Redesign* (BPR) programs. These programs aim for a performance improvement by changing the structural backbone of business processes. Or, as Hammer and Champy [121] define BPR, it is *"... the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical contemporary measures of performance, such as cost, quality, service, and speed"*. The application of information technology (IT), and in particular of Process Aware Information Systems (PAIS) [89], plays an important role in realizing these process redesigns. With new technologies, information can be digitally stored and exchanged. This enables radical changes in the business process, especially for *administrative processes*, also known as *workflow processes*. The subject of the research described in this thesis is a method for BPR that specifically focuses on these workflow processes: *Product Based Workflow Design (PBWD)*. In this first chapter, the context of the research and the problem statement are introduced. First of all, the stages of a BPR program are explained in Section 1.1, followed by an overview of existing BPR methods. Then, an introduction to the field of *workflow management* is given. The term workflow management refers to the ideas, methods, techniques, tools and software programs used to support workflow processes [13]. Section 1.4 introduces the ideas behind the PBWD method. This chapter concludes with an overview of the research contributions in this thesis, an explanation of the research methodology used, and a road map which summarizes the content of each chapter in this thesis.

1.1 Business Process Redesign

A BPR program typically consists of a number of stages which are summarized in the BPR life cycle of Figure 1.1. In the *process design phase*, the business process is designed (or redesigned) based on a requirements analysis. The result of this phase is a (high-level) process model that describes the business process. In the *process implementation phase*, the process

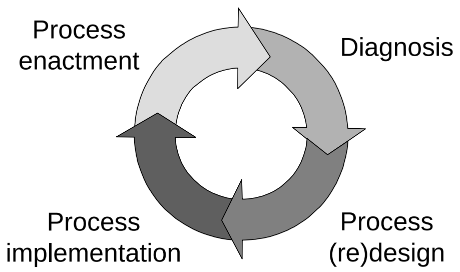


Figure 1.1: *The BPR life cycle [13, 89].*

model is refined into an operational process supported by a software system. This is achieved by configuring a generic infrastructure of a process aware information system, such as a workflow management system. In the *process enactment phase*, the operational process is executed using the configured system. In the *diagnosis phase*, the operational process is analyzed to identify problems and to find aspects that can be improved. Based on the identified improvement opportunities, the BPR-life cycle can be repeated.

In this thesis we mainly focus on the process design phase of the BPR life cycle. We also aim at a specific type of business process: the so-called *workflow process* [231]. Designing a process may be a complex task. Over the years many IT projects have failed because the information system that was developed was actually not supporting the process due to an incorrectly modeled process. Therefore, many methods have been developed to facilitate and support the design of business processes. In the next section, an overview of the existing BPR methods is given.

1.2 BPR Methods

There are many methods to support the process (re)design phase of a BPR project, e.g. industry prints, best practices, heuristics. However, these various methods often lack a sound foundation [7, 231]. This makes a BPR investment an adventure and illustrates the need for better and theoretically sound guidance [231].

As a first step towards this guidance, many researchers have classified the existing BPR methods. Kettinger, Teng and Guha compiled a list of 102 different tools to support redesign projects [149]. Building on this study, Al-Mashari, Irani and Zairi classified BPR-related tools and techniques in 11 major groups [32]. These groups cover activities such as project management, process modelling, problem diagnosis, business planning, and process prototyping. Gunasekaran and Kobu [117] reviewed the literature from 1993-2000 and came to the following classification of modelling tools and techniques for BPR: (i) conceptual models, (ii) simulation models, (iii) object-oriented models, (iv) IDEF models, (v) network models, and (vi) knowledge-based models. More recently, Attaran [36] linked the various available IT systems and tools to three different phases in a BPR program: (i) before a process is designed, a tool can act as an enabler (e.g. as an inspirator for a new strategic vision), (ii) while the process is being designed, a tool can be a facilitator (e.g. for mapping the process, gathering performance data, and simulation), and (iii) after the design is complete, it can act as an implementor (e.g. for project planning and evaluation).

We use the classification of Kettinger et al. [149] to further explain the focus of our research. Kettinger et al. distinguish three levels of abstraction for BPR methods: *tools*, *techniques* and *methodologies*. Reijers [231] has used this classification to present an overview of existing process design methods.

1.2.1 Tools

Tools are software packages to support BPR techniques and methodologies. First of all, there are a number of tools available that facilitate the performance analysis of a business process, e.g. the business process cockpit [252], and ARIS Process Performance Manager [135]. Secondly, many software tools exist that focus on the modeling and evaluation of a business process. For example, Protos [209] and ARIS [254] are modeling tools, while ExSpect [11] and CPN Tools [74] are tools which support the evaluation of a business process by means of simulation. Although analysis, modeling and evaluation are important to support the (re)design of business processes, they do not provide any design guidance. Only a few tools are available that systematically capture knowledge about the redesign direction or to support existing creativity techniques, e.g. [42, 162, 197]. Finally, some tools exist that support the discovery of process models from execution logs, e.g. the ProM framework [84, 223].

1.2.2 Techniques

Design techniques are precisely described procedures for achieving a standard task. They are at a higher level of abstraction than tools. Most existing BPR techniques focus on diagnosis, modeling and evaluation, e.g. fishbone diagramming, Pareto diagramming for diagnosis, and flowcharting, IDEF, activity-based costing for modeling and/or evaluation of business processes. These topics are relevant in the context of the research described in this thesis, but give no guidance for the final design of a process. Techniques that somehow support process designers in making a design are mostly creativity techniques, such as out-of-the-box-thinking, and the Delphi method. Moreover, several techniques have been developed for the discovery of process models from execution logs, e.g. workflow mining [31], the α-algorithm for process mining [24], heuristic process mining [294], multi-phase process mining [83], and genetic process mining [19, 34].

1.2.3 Methodologies

A design methodology is defined as a collection of problem-solving methods governed by a set of principles and a common philosophy for solving targeted problems. In this thesis we focus on a specific design methodology. Two approaches to the design of process models can be distinguished [231]:

· *Evolutionary approaches*
Evolutionary approaches take the existing process model as a starting point. This model is then gradually refined or improved by using a set of best practices or rules. In [163, 164, 231, 232], a survey of *best practices* for evolutionary process improvement of workflow process models is given. The survey includes 29 best practices that are mainly derived from literature, e.g. from [7, 213, 221, 247]. The 29 best practices can be categorized in six groups: (i) *task rules*, which focus on optimizing single tasks in the process model, (ii) *routing rules*,

which try to improve the routing structure in the process model, (iii) *allocation rules*, which involve a particular allocation of resources, (iv) *resource rules*, which focus on the types and number of resources, (v) *rules for external parties*, which try to improve the collaboration and communication with external parties, and (vi) *integral workflow rules*, which apply to the workflow process model as a whole. A best practice generally consists of three parts. It contains a construct or pattern that can be distinguished in the process model, an alternative for the selected part exists, and a context-sensitive justification for the use of this alternative can be given.

· *Revolutionary approaches*
Revolutionary approaches design the process completely from scratch, i.e. there is no a priori model used which is changed. Examples of revolutionary approaches include the DEMO approach [237], linear and dynamic programming approaches [33, 203], visual strategies [122], business process intelligence [112], process mining [23, 31, 72, 78], and PBWD [233].

Many of these tools, techniques and methodologies are presented as 'intelligent' [60, 181], although they do not actively guide the user in designing the business process. This has been recognized by Nissen [199], for instance, who states that despite the plethora of tools for modeling and simulation of enterprise processes, "such tools fail to support the deep reengineering knowledge and specialized expertise required for effective redesign". Similarly, Bernstein, Klein and Malone [42] observe that "today's business process design tools provide little or no support for generating innovative business process ideas". Gunasekaran and Kobu [117] indicate that only few knowledge-based models for BPR have been developed. Some of these exceptions are: the process recombinator tool [42, 167], grammatical models [162, 212], the KOPeR tool [197, 199] and the ProM framework [84, 223].

The most important reason to develop 'intelligent' tools for process redesign is that process designers can be guided in the design process and new design alternatives can be developed in a simpler manner [117], more cost-effectively [197], quicker [167, 199] and more systematically [167, 212, 308]. This may lead to better redesigns and more successful BPR projects.

The subject of our research is the revolutionary BPR methodology PBWD. This thesis focuses on the development of 'intelligent' tools for PBWD. PBWD aims at the improvement of *workflow processes*. Therefore, first an introduction to workflow processes and workflow management is given before the PBWD methodology is further explained.

1.3 Workflow Management

The term *workflow management* refers to the ideas, methods, techniques, tools and software programs used to support workflow processes [13]. A *workflow process* is an administrative business process, such as the handling of an insurance claim, a loan application, a subsidy request, or the registration of a new patient [13, 231]. Workflow processes focus on the processing of information instead of physical parts and therefore are considerably more flexible in their lay-out than manufacturing business processes [220]. For example, copying files or documents is relatively straightforward, which in principle enables the concurrent execution of steps in the process. A workflow process has two important characteristics: (i) it is a *make-to-order* process because a specific order or trigger initiates the process, production-to-stock is therefore not possible, and (ii) it is *case-driven* since each execution of a step in a workflow process can be attributed to exactly one specific case and batch processing is not possible [231].

Workflow processes can be (re)designed using BPR methods. In the next two sections we focus on the process design phase and the process implementation phase of the BPR life cycle for workflow processes.

1.3.1 Workflow Process Design

A workflow process can be modeled by a *process model*. Such a process model describes the control flow of the process, i.e. which *activities* must be performed in which order to successfully complete a *case*. The possible routes from start to completion of the process are described by the process model. It consists of activities, conditions, and sub processes. By using constructs such as AND-splits, AND-joins, OR-splits, OR- joins, XOR-splits, and XOR-joins parallel and alternative flows can be defined [13].

Many modeling languages exist to design a process model. Most of these languages also have a visual representation that supports the communication about the design among process designers and other stakeholders in a BPR project. Figures 1.2(a)-1.2(c) show some examples of process models in different languages. Some of the process modeling languages have a formal basis, e.g. Petri nets [82], Workflow nets [2], YAWL models [15] and Pi calculus [180], enabling their formal analysis and verification. Other languages have a (partly) informal basis, e.g. EPCs [4, 146], UML Activity Diagrams [246], and BPMN [299], which may lead to process models with unclear semantics and execution.

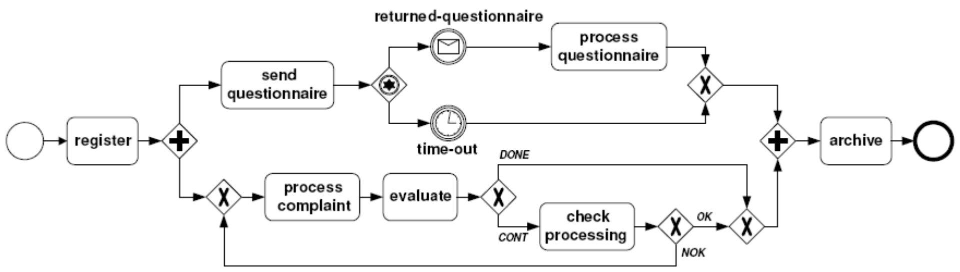
There are also many tools that support the design of process models with these languages, e.g. Protos [209] and ARIS [253]. However, as we have mentioned in Section 1.2 most of these tools only facilitate the modeling process and do not offer any design guidance in the development of process models.

When a process model has been created for a workflow process in the process design phase, this process model may be refined in the process implementation phase to an operational process, the execution of which can be supported by a workflow management system.

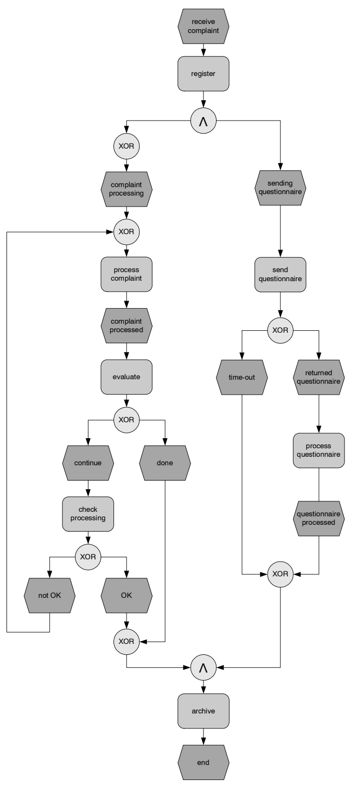
1.3.2 Workflow Management Systems

A workflow management system is a software package that provides support for the definition, management, and execution of workflow processes, together with certain interfaces to its environment and its users [13, 69]. It originates from earlier technologies such as office automation, document management, database management and electronic messaging sytems [89]. A workflow management system supports the execution of a workflow process based on the process model designed. There are many dedicated workflow management systems, such as TIBCO Staffware [268], FileNet [97], IBM WebSphere [134], and COSA [73], but many ERP-systems also include workflow components, e.g. SAP R/3 [127], and Oracle [200].

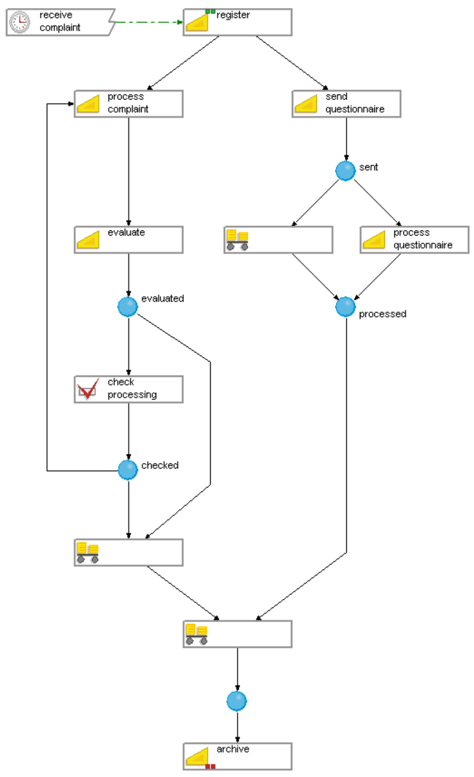
A workflow management system typically consists of a number of components (see the reference model of the Workflow Management Coalition (WfMC) in Figure 1.3). The heart of the system is the *workflow enactment service*, which comprises one or more workflow engine(s). The *workflow engine* is responsible for the control and execution of the various cases that have to be processed. Each *case* is a separate instance of the workflow process and is executed according to the process model which is defined in the *process definition tool*. A *work item* is the piece of work that has to be done to execute an activity in the process model for a specific case. Work items are executed by humans via a *worklist*, and by *workflow client applications*, which



(a) A process model in the BPMN language (reproduced from [205]).



(b) A process model in the EPC language.



(c) A process model in the Protos language.

Figure 1.2: *Some example process models.*

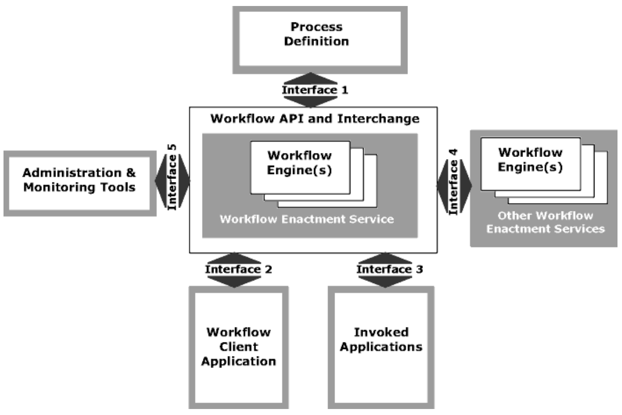

Process Definition
Interface 1
Workflow API and Interchange
Administration & Monitoring Tools
Interface 5
Workflow Engine(s)
Workflow Enactment Service
Interface 4
Workflow Engine(s)
Other Workflow Enactment Services
Interface 2
Interface 3
Workflow Client Application
Invoked Applications

Figure 1.3: *The WfMC reference model [69].*

support the user by executing his or her[1] task. Work items can also be executed automatically by *invoked applications*. The execution of the workflow process can be monitored through the *administration and monitoring tools* which may provide e.g. information on the workload, utilization, average throughput time, and the number of cases in the process. Finally, the workflow enactment service may exchange work with other workflow management systems through a communication interface with other workflow enactment services.

Traditional workflow management systems are characterized by a number of limitations in terms of flexibility and adaptability [18]. These limitations can be associated with the dominant paradigm for process modeling found in these systems, which is almost exclusively activity-centric [89]. The lack of flexibility and adaptability leads to many problems and inhibits a broader use of workflow technology. In recent years many authors have discussed the problem [18, 30, 66, 93, 128, 150, 151, 234] and different solution strategies have been proposed. There are many ways to provide more flexibility [278, 277], e.g. *dynamic change* [93, 227, 242], *worklets* [27], and *case handling* [10, 25]. Dynamic change focuses on the changes that can be made to the process model at runtime, either with respect to a single case, or with respect to all running cases. ADEPT [227, 242] is one of the systems that supports dynamic change. Worklets [27] allow for late binding of process fragments, i.e. it is decided at runtime which application or subprocess is used to execute an activity. YAWL [15] and Staffware [265] are systems that have implemented these ideas. Case handling systems (e.g. FLOWer [207, 208]) are much more data driven than traditional workflow management systems. They support knowledge intensive processes and focus on what can be done instead of on what should be done.

Now the concepts behind BPR and workflow management have been introduced, it is time to focus on the specific BPR methodology for workflow processes that is the subject of our research: the PBWD method.

[1] In this thesis we further refer to persons, e.g. users, modelers, stakeholders, process designers, as being male, although these persons can be female as well as male.

1.4 Product Based Workflow Design

PBWD is a revolutionary (re)design methodology, in which the workflow *product* is the central concept in the design process instead of the *activities* in the business process. This shift in focus is driven by the idea that the structure of a process model is dictated by the constraints on the product that is produced [1, 5, 37]. Therefore, the design of a process model should start with the definition of the artifact to be created in the process [37].

PBWD starts from an analysis of the (informational) product that is produced in the workflow process, e.g. the allowance of a mortgage, the decision on an insurance claim, the grant of a subsidy, etc. The structure of the workflow product is described by a *Product Data Model (PDM)*, which is similar to the concept of a Bill-of-Material (BoM) [201] in the manufacturing domain (see Figure 1.4). A PDM describes the processing of information to achieve the end product from the data that is provided as input to the process, e.g. to calculate the amount of mortgage for a client the client's gross annual income is used as input. The PDM consists of *data elements* and *operations* on these data elements, which specify how input data is processed to achieve new information. The PDM is then used to derive a process model.

A typical PBWD project is executed in the design phase of the BPR life cycle and consists of four steps [231]. In the *scoping* phase the process to be (re)designed is selected by identifying the product of the process. The product is then analyzed and decomposed into a product description containing the data elements and operations in the *analysis* phase; a PDM is the result of this phase. In the *design* phase, several process models are derived from this PDM. Each activity in the process model retrieves data and produces new data. Finally, in the *evaluation* phase, the alternative process models are further analyzed, verified, and validated with end users. Then the best design is selected from these alternatives and is used for the implementation in a workflow management system.

The most important advantages of PBWD are [231]:

1. *Radicalism* - The clean sheet approach that is taken allows for maximal space to establish performance improvements. Approaches that use the existing process will to some extent copy constructions from the current process, repeating existing errors or undesirable constructs.

2. *Rationality* - PBWD is rational. In the first place, because a product specification is taken as the basis for a workflow design, each recognized data element and each operation on a set of data elements can be justified and verified with this specification. As a result, no redundant steps are executed in the process, multiple registrations of the same information will no longer happen, and it becomes clear which data manipulations can be automated. Secondly, the ordering of activities themselves is completely driven by the performance targets of the design effort (e.g. shorter throughput time, or lower production cost). This allows for a process execution that is not governed by more or less arbitrary updates to the process in the past, but by the drivers that are important to an organization today.

3. *System integration* - The analytical approach of PBWD renders detailed deliverables suitable for system development purposes. Based on the PBWD deliverables, it is possible to develop functional models of the information system to be developed. The PDM describes the data processing in the workflow process. Operations can be seen as functional specifications for services the information system should offer. Data elements can be considered as attributes or entities and can be modeled in a data model

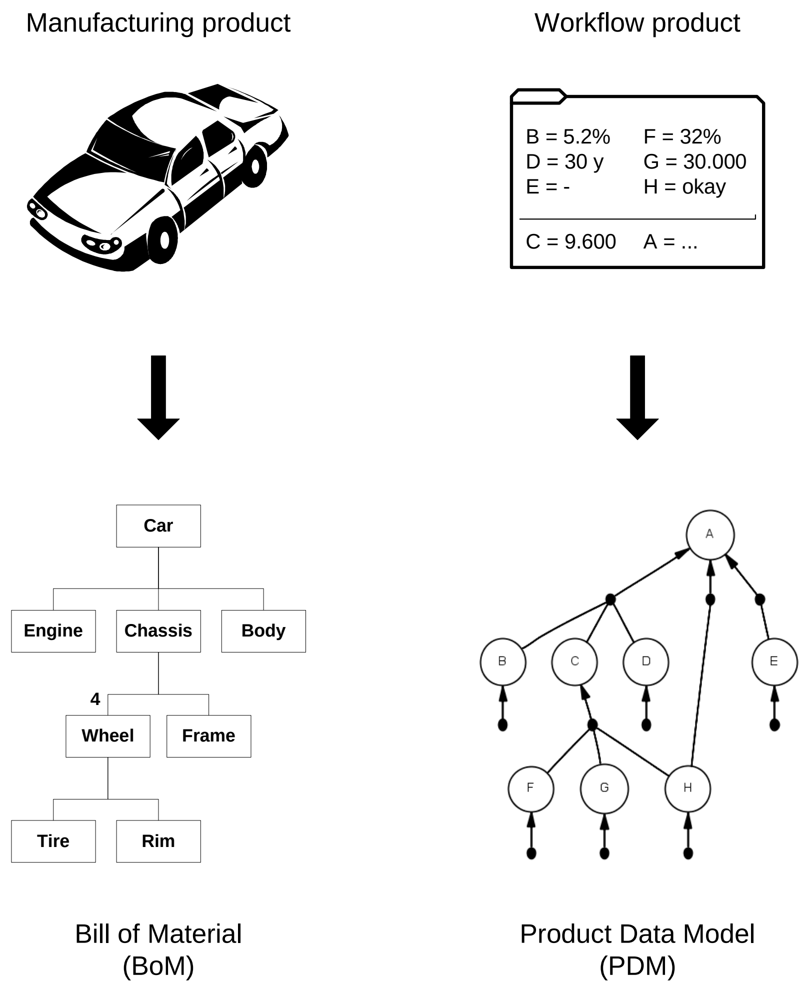


Figure 1.4: *The PDM is similar to the BoM. Both models describe the structure of a product. The BoM describes how a physical product is assembled from its parts. A PDM describes the information that is processed to determine an informational product.*

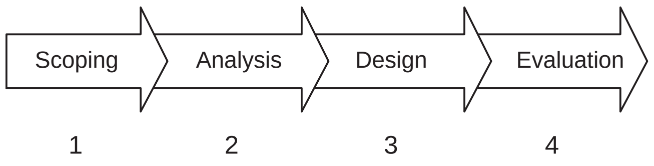


Figure 1.5: *The four phases of a PBWD project: (i) the scoping phase, (ii) the analysis phase, (ii) the design phase, and (iv) the evaluation phase.*

4. *Objectivity* - The focus on the product can help to create consensus between different stakeholders. It gives a clear and objective representation of the workflow product that can help in discussing the process. The product of a process is more easily understood than the process itself.

Besides these advantages one must also be aware of the main drawbacks of PBWD before using this approach to redesign a business process [231]:

1. The application of PBWD requires a clear concept of the product that is produced in the process. After all, if there is no product specification the basis for PBWD is missing. Thus, the application of PBWD is restricted to those processes in which a clear product can be distinguished.
2. The application of PBWD is an intensive effort because of the thorough analysis of the product specification that is required. A PBWD project may therefore be time-consuming. If this time is not available or only gradual improvements of a process are desirable, PBWD may not be suitable.
3. The PBWD method is not driven by technology-oriented analyses and approaches, but by a business-oriented analysis. This may change the role and responsibilities of persons and departments that are involved in BPR projects. Such a change may need to be managed and business as well as IT people may need to be convinced of their new role.
4. When applying PBWD, it may be hard for internal process experts to forget the existing process and to think in terms of the product of the workflow process. Thus, substantial education and training on the PBWD method may be needed.

PBWD has a theoretical basis and has proven its potential by a number of successful practical applications. For example, PBWD was used to redesign the process of awarding unemployment benefits in the Netherlands. This redesign was conducted within the UWV agency (formerly known as GAK). As a result, the average throughput time of the process was reduced by 73% and 10% of all cases now require no human intervention at all [231, 233]. Another successful application of PBWD aimed at redesigning the process of handling credit applications for commercial parties at a large bank in the Netherlands. The evaluation of the project showed an efficiency increase of 40% [231].

Previous PBWD projects have all been executed manually [231, 233], i.e. after a product specification was made the design of alternative process models was done completely by hand. Although the theory of PBWD is rather mature and has proven its practical relevance, no tools are available yet to support this design methodology. The development of these tools is one of the goals of this thesis.

1.5 Contributions

The research described in this thesis focuses on the development of 'intelligent' tools to support the product based design of workflow processes. Five main contributions are reported in this thesis:

· A *formal definition* of a PDM (based on earlier definitions in [5, 231]) together with a graphical notation to visualize a PDM. Based on this formal definition of a PDM, we also define a *basic notion of correctness* which can be used to verify whether a process model is consistent with the given PDM. This basic correctness notion can be used to evaluate a process model design in the evaluation phase of a PBWD project.

- The definition of *seven algorithms* to semi-automatically generate process models from a PDM. The algorithms generate process models with different structures and characteristics. Based on these differences a classification framework for the algorithms is defined. The algorithms can be used in the design phase of a PBWD project and support a process designer in deriving an initial process model based on a given PDM.
- The support of the *direct execution* of a PDM. A PDM can be directly executed without first deriving a process model first. Based on the data available for the case, it can be decided which steps can be executed next. The direct execution of a PDM makes the design of a process model superfluous and provides more dynamic and flexible support for the workflow process. This approach does not fit in the four phases that are normally distinguished in a PBWD project. However, it is a way to realize a workflow management system in a more direct manner (see the third phase in the BPR life cycle in Figure 1.1).
- The introduction of two sets of *business process metrics*. The first set of metrics is defined on process models derived from a PDM and focuses on the cohesiveness of the content of the activities and the coupling between activities in the process model. It does so by looking at the relationships between data elements. The second set of metrics focuses on the structure of the process model itself and tries to capture the cognitive effort needed for a process designer to understand a process model by measuring the connectivity of all model parts. These business process metrics can be used to evaluate a (manually) designed process model or to compare alternative process models in the evaluation phase of a PBWD project.
- Finally, for each of these ideas a *prototype* of a tool supporting the idea has been developed. These prototypes are implemented in the ProM Framework [223] and show the feasibility of the ideas.

The above contributions are described in detail in chapters 4-7 of this thesis.

1.6 Methodology

The methodology used to conduct the research described in this thesis is called *design-oriented research* [287] or *design science research* [129], and is different from behavioral science research [129]:

> *Design science creates and evaluates IT artifacts intended to solve identified organizational problems. Such artifacts are represented in a structured form that may vary from software, formal logic, and rigorous mathematics to informal natural language descriptions. Design science addresses research through the* building *and* evaluation *of artifacts designed to meet the identified business need. The goal of behavioral science research is truth. The goal of design science research is utility.*

In this thesis, we present three directions for the development of tools to facilitate the product based design and support of workflow processes. We evaluate our ideas by showing their feasibility. For each of them a prototype is built as a plugin in the ProM framework [223]. Moreover, we have included a number of case studies to evaluate the practical application of the tools.

1.7 Outline

In this section, we briefly describe the structure of this thesis by giving a short summary of the content of each chapter. The references indicate earlier publications on these subjects co-authored by the PhD candidate.

Chapter 2 provides the background knowledge that is needed to understand the research presented in this thesis.

Chapter 3 introduces the notion of product data models and shows the relation of a product data model to a process model. This chapter is partly based on [278].

Chapter 4 presents a formal definition of the product data model and defines a basic notion of correctness of a process model with respect to a PDM.

Chapter 5 describes seven algorithms to generate a process model from a product data model. The basis of this chapter has been published in [143].

Chapter 6 focuses on the direct execution of a product data model and provides two approaches to guide execution decisions. The ideas have been published in [280, 281].

Chapter 7 presents two sets of business process metrics to evaluate and compare process designs. This chapter is based on [235, 275, 276, 279, 282].

Chapter 8 contains an evaluation of the ideas presented in the previous chapters. It is partly based on the work described in [235, 276].

Chapter 9 summarizes the contributions of this thesis and presents an outlook to future work.

Each chapter has the same structure. After a short introduction, the core ideas of the chapter are presented and discussed. Next, a description is given of the tool support that has been developed to show the feasibility of the ideas presented. Finally, a section on related work is presented together with a summary.

Chapter 2

Preliminaries

In the remainder of this thesis a number of concepts and theories are used, which we first introduce and formalize in this chapter. We start with an introduction to propositional logic and set theory in sections 2.1 and 2.2. The notion of a graph is explained in Section 2.3. Then, a number of business process modeling languages are introduced: Section 2.4 deals with the basic theory on Petri nets and WorkFlow nets, Section 2.5 introduces the YAWL language, and Section 2.6 explains the so-called Event-driven Process Chains. Next, the theory of Markov chains and Markov decision processes is introduced in Section 2.7. These are mathematical models for the analysis of discrete-time stochastic processes. Finally, Section 2.8 elaborates on the technical infrastructure that is used to develop the prototypes for the research described in this thesis.

2.1 First-Order Logic

Logic is used to reason based on statements. If we use propositional logic, a statement is called a *proposition* or *propositional formula*. Each proposition has a *truth value*; it can either be *true* or *false*. For instance *'snow is white'* is a proposition which is true, while the proposition *'grass is purple'* is false. Propositions are usually expressed by *propositional letters* such as p and q, where e.g. p stands for *'snow is white'* and q for *'grass is purple'* [98]. A proposition of only one propositional letter is an *atomic formula* and is the smallest proposition possible. With atomic formulas, larger propositions can be formed by using a number of logical operators.

Definition 2.1.1 (Logical Operators)**.** The basic logical operators on propositions are:

- Negation - A negation $\neg p$ is true if and only if p is false.
- Conjunction - A conjunction $p \wedge q$ is true if and only if both p and q are true.
- Disjunction - A disjunction $p \vee q$ is true if and only if p is true or q is true. That is, either p or q is true or both are true.
- Implication - An implication $p \Rightarrow q$ is false if and only if p is true and q is false.
- Equivalence - An equivalence $p \Leftrightarrow q$ is true if and only if both $p \Rightarrow q$ and $q \Rightarrow p$. That is, if p and q have the same truth value.

If q stands for *'grass is purple'* and p stands for *'snow is white'*, then q = false, $\neg q$ = true, and $p \wedge q$ = false, for example. A *literal* is an atomic formula or its negation. Thus, p and $\neg p$

are literals. When a propositional formula is in conjunctive normal form or disjunctive normal form it has a specific structure.

Definition 2.1.2 (Conjunctive Normal Form [132, 194])**.** A propositional formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses, where a clause is a disjunction of literals, e.g. $(p_1 \vee p_2) \wedge (\neg p_3 \vee p_4)$ is in CNF. The only propositional operators in CNF are $\wedge$, $\vee$, and $\neg$. The $\neg$ operator can only be used as part of a literal.

Definition 2.1.3 (Disjunctive Normal Form [132, 194])**.** A propositional formula is in *disjunctive normal form (DNF)* if it is a disjunction of one or more conjunctions of one or more literals, e.g. $(p_1 \wedge p_2) \vee (\neg p_3 \wedge p_4)$ is in DNF. The only propositional operators in DNF are $\wedge$, $\vee$, and $\neg$. The $\neg$ operator can only be used as part of a literal.

As we have seen above, propositional logic deals with simple declarative propositions. *First-order logic* (or *predicate logic*) extends propositional logic by introducing predicates and quantifications. A *predicate* is a proposition that contains a variable. It assigns a truth value to each element of a set to indicate whether a certain property is true or false for the element. For instance, if we consider all kind of birds and the predicate 'is able to fly', then we can say that the predicate is true for a sparrow and for an eagle, but it is false for an emu and a penguin. By quantification one can specify the quantity of elements that satisfy a certain property. Thus, we may truthfully say that 'there is a bird that can fly', but if the statement that 'all birds can fly' would not be true. To express these quantifications, two special symbols are used. These symbols are called *quantifiers*.

Definition 2.1.4 (Quantifiers)**.** There are two quantifiers in first-order logic:

- $\forall$, which is the universal quantifier and means 'for all',
- $\exists$, which is the existential quantifier and means 'there exists a'.

With these quantifiers logical sentences can be formed that contain variables. For instance, if we want to express that there is a bird that is able to fly we can use the following formula: $\exists x[(x \text{ is a bird}) \wedge (x \text{ is able to fly})]$.

2.2 Sets

Sets are a fundamental concept in mathematics and computer science. This section gives an introduction to set theory; a more detailed overview can be found in [196]. A set is a collection of distinct objects considered as a whole, e.g. the set of natural numbers ($\mathbb{N} = \{0, 1, 2, 3, ...\}$), or the set of characters in the alphabet ($\{a, b, c, ..., x, y, z\}$). The members of a set can be specified in two ways: (i) by listing all members of the set, or (ii) by using a semantic description. For instance, the sets $\{0, 1, 4, 9, 16, 25\}$ and $\{x^2 \mid 0 \leq x \leq 5 \wedge x \in \mathbb{N}\}$ contain the same elements. We use a number of notations to denote sets and operators on these sets.

Definition 2.2.1 (Set notations)**.** A number of standard operators for sets are defined as follows:

- Let s_1 and s_2 be two elements. We construct a set S of these two elements by stating $S = \{s_1, s_2\}$, i.e. we use $\{$ and $\}$ to enumerate the elements in a set.
- If an element s is contained in S, we say s is a *member* of S: $s \in S$.
- The *cardinality* or size of a set S, denoted by $|S|$, is the number of elements in S.

- S_1 is a *subset* of S, $S_1 \subseteq S$, if S_1 is contained in S.
- S_1 is a *proper subset* of S, $S_1 \subset S$, if $S_1 \subseteq S \wedge S_1 \neq S$.
- The *power set* of a set S is the set of all subsets of S, i.e. $\mathcal{P}(S) = \{S_1 \mid S_1 \subseteq S\}$.
- The *union* (S) of two sets (S_1, S_2) is defined as $S = S_1 \cup S_2$, i.e. S contains all elements of S_1 and S_2.
- The *intersection* (S) of two sets (S_1, S_2) is defined as $S = S_1 \cap S_2$, i.e. S contains only the elements that are members of both S_1 and S_2.
- Two sets S_1 and S_2 are *equal*, $S_1 = S_2$, if $S_1 \subseteq S_2 \wedge S_2 \subseteq S_1$. The order in which the elements of a set appear does not matter, i.e. $\{s_1, s_2\} = \{s_2, s_1\}$.
- The *difference* between two sets S_1 and S_2, $S_1 - S_2$, is defined by: $S_1 - S_2 = \{s | s \in S_1 \wedge s \notin S_2\}$.
- The *Cartesian product*, denoted by $S = S_1 \times S_2$, is defined by: $S = \{(s_1, s_2) | s_1 \in S_1 \wedge s_2 \in S_2\}$.
- $\emptyset$ denotes the empty set. For all sets S it holds that $\emptyset \in S$. If a set contains more elements than just the empty set, it is said to be *non-empty*.

A relation relates the members of sets to each other.

Definition 2.2.2 (Relation)**.** Let S_1 and S_2 be two non-empty sets, then $R \subseteq S_1 \times S_2$, is a relation between S_1 and S_2. The set S_1 is called the *domain* of relation R and S_2 is called the *range* of relation R.
Let R be a relation on S, i.e. $R \subseteq S \times S$. Then, a number of properties are defined:

- R is reflexive: $\forall_x[x \in S \Rightarrow (x, x) \in R]$
- R is irreflexive: $\forall_x[x \in S \Rightarrow \neg(x, x) \in R]$
- R is symmetric: $\forall_{x,y}[(x, y) \in R \Rightarrow (y, x) \in R]$
- R is asymmetric: $\forall_{x,y}[(x, y) \in R \Rightarrow \neg(y, x) \in R]$
- R is antisymmetric: $\forall_{x,y}[((x, y) \in R \wedge (y, x) \in R) \Rightarrow x = y]$
- R is transitive: $\forall_{x,y,z}[((x, y) \in R \wedge (y, z) \in R) \Rightarrow (x, z) \in R]$

For instance the relation $R = \{(a, a), (b, b), (c, c), (a, b), (b, c)\}$ on set $S = \{a, b, c\}$ is reflexive, and antisymmetric (but *not* irreflexive, symmetric, antisymmetric, asymmetric, transitive, or intransitive).

A partial order is a reflexive, antisymmetric, and transitive relation.

Definition 2.2.3 (Partial order)**.** A relation $R \subseteq S \times S$ is a partial order if R is: (i) reflexive, (ii) antisymmetric, and (iii) transitive.

Thus, $R = \{(a, a), (b, b), (c, c), (a, b), (b, c)\}$ is not a partial order, since it is not transitive. However, if we add (a, c) to R such that $R' = \{(a, a), (b, b), (c, c), (a, b), (b, c), (a, c)\}$, then R' is a partial order. The transitive closure of a relation $R \subseteq S \times S$ is the smallest transitive relation on S that contains R. The transitive closure can be composed by iteration of the relation over itself.

Definition 2.2.4 (Transitive closure)**.** Let $R \subseteq S \times S$ be a relation on S. The *transitive closure* of R, denoted by R^+, is defined as: $R^+ = \bigcup_{i \in \mathbb{N} \setminus \{0\}}[R^i] = R^1 \cup R^2 \cup R^3$... where R^i denotes the i-th relational iteration, i.e. the composition of R on itself i times.

In our example, $R' = \{(a,a),(b,b),(c,c),(a,b),(b,c),(a,c)\}$ is the transitive closure of R, i.e. $R^+ = R'$.

A function is a special kind of relation in which every element of the first set is mapped to exactly one element from the second set.

Definition 2.2.5 (Function)**.** Let S_1 and S_2 be two sets. A *function* f from S_1 to S_2, denoted by $f : S_1 \rightarrow S_2$, is defined as follows:

- $f \subseteq S_1 \times S_2$
- $\forall_{s_1 \in S_1}[\exists_{s_2 \in S_2}[(s_1, s_2) \in f]]$
- $\forall_{s_1,s_2,s_3}[((s_1, s_2) \in f \wedge (s_1, s_3) \in f) \Rightarrow s_2 = s_3]$

Functions may have special properties. A function is said to be injective if there is at most one element in the domain for each element in the range of the function. A function is said to be surjective if there is at least one element in the domain for each element in the range. A function is bijective if it is both injective and surjective.

Definition 2.2.6 (Injection, surjection, bijection)**.** Let Let S_1 and S_2 be two sets, and f be a function from S_1 to S_2, denoted by $f : S_1 \rightarrow S_2$. Then f is a:

- Injection, if and only if: $\forall_{s_1,s_1' \in S_1}\forall_{s_2 \in S_2}[((s_1, s_2) \in f \wedge (s_1', s_2) \in f) \Rightarrow s_1 = s_1']$
- Surjection, if and only if: $\forall_{s_2 \in S_2}\exists_{s_1 \in S_1}[(s_1, s_2) \in f]$
- Bijection, if and only if: f is injective and f is surjective.

A partial function is a relation that associates each element of the first set to *at most* one element from the second set, i.e. not every element from the domain S_1 has to be associated with an element from the range S_2.

Definition 2.2.7 (Partial function)**.** Let S_1 and S_2 be two sets. A *partial function* f from S_1 to S_2, denoted by $f : S_1 \not\rightarrow S_2$, is defined as follows:

- $f \subseteq S_1 \times S_2$
- $\forall_{s_1,s_2,s_3}[(s_1, s_2) \in f \wedge (s_1, s_3) \in f \Rightarrow s_2 = s_3]$

A multi-set is a generalization of a set. In a multi-set, a member can have more than one occurrence.

Definition 2.2.8 (Multi-set)**.** A *multi-set* or bag X on a set S is a function of S to the natural numbers, i.e. $X : S \rightarrow \mathbb{N}$. We use square brackets to denote the enumeration of elements of the multi-set, e.g. $[a^2, b, c^5]$ is a multi-set on $S = \{a, b, c, d\}$, with $X(a) = 2$, $X(b) = 1$, $X(c) = 5$, $X(d) = 0$. Let $X : S_1 \rightarrow \mathbb{N}$ and $Y : S_2 \rightarrow \mathbb{N}$ be two multi-sets. We use a number of notations:

- a is an *element* of X if and only if $a \in S_1$ and $X(a) > 0$.
- The *cardinality* or size of X is defined by: $|X| = \sum_{a \in S_1} [X(a)]$.
- X is a sub multi-set of Y, $X \leq Y$ if and only if $\forall_{a \in S_1}[X(a) \leq Y(a)]$.
- X and Y are *equal*, $X = Y$, if and only if $\forall_{a \in S_1 \cup S_2}[X(a) = Y(a)]$.
- The *union* of X and Y, denoted by $Z = X \cup Y$, is a function from $S_1 \cup S_2$ to $\mathbb{N}$, $Z : S_1 \cup S_2 \rightarrow \mathbb{N}$, where $Z(a) = \max(X(a), Y(a))$.

- The *intersection* of X and Y denoted by $Z = X \cap Y$, is a function from $S_1 \cup S_2$ to $\mathbb{N}$, $Z : S_1 \cup S_2 \rightarrow \mathbb{N}$, where $Z(a) = \min(X(a), Y(a))$.
- The *sum* of X and Y, denoted by $Z = X \uplus Y$, is a function $Z : S_1 \cup S_2 \rightarrow \mathbb{N}$ where for all $a \in S_1 \cup S_2$ holds that $Z(a) = X(a) + Y(a)$.
- The *difference* of X and Y, denoted by $Z = X - Y$, is a function $Z : S' \rightarrow \mathbb{N}$ with $S' = \{a \in S_1 | X(a) - Y(a) > 0\}$ and for all $a \in S'$ holds that $Z(a) = X(a) - Y(a)$.
- A *partial order* is defined on two multi-sets X and Y, denoted by $X \leq Y$, if and only if $\forall_{a \in S_1}[X(a) \leq Y(a)]$.

A sequence and a tuple both are ordered lists of elements. A sequence usually contains elements from the same set, while a tuple can be a list of any kind of elements.

Definition 2.2.9 (Sequence)**.** Let S be a set of elements. A *sequence* of the elements of S is a function $\sigma : \{1, 2, ..., n\} \rightarrow S$. It is represented by an ordered list of zero or more elements of S, i.e. $\sigma = \langle s_1, s_2, ..., s_n \rangle$. By $\sigma(i)$, $1 \leq i \leq n$, we denote the element at index i in the sequence σ. S^* is the set of all sequences over S.

Definition 2.2.10 (Tuple)**.** A *tuple* is a (finite) ordered list of values. The values are called the *components* of the tuple and can be any kind of object, e.g. an element, a set, or a function. In contrast to a set or multi-set, the order in which the components of the tuple appear is important. A tuple with two components is often called an *ordered pair*. In general, a tuple with n components is called a n*-tuple*. We use (and) to denote a tuple, e.g. (S, s, f) is a 3-tuple with the set S as the first component, element s as the second component, and function f as the third component.

Tuples are often used to describe mathematical objects such as graphs.

2.3 Graphs

A graph is a mathematical structure, which has a clear graphical representation. Graphs are used to visualize models (e.g. process models). We introduce the theory of graphs starting from a very general notion of a graph, the hypergraph [41, 100, 116].

Definition 2.3.1 (Directed hypergraph)**.** A *directed hypergraph* H is defined as a pair (N, E) where N is a finite set of elements ($N = \{n_1, n_2, ..., n_N\}$) and E is a relation on the subsets of N, i.e. $E \subseteq \mathcal{P}(N) \times \mathcal{P}(N)$. The elements of N are called the *nodes* of the hypergraph and the elements in E are called the *hyperarcs*. Let $(X, Y) \in E$, then X is called the *source* and Y the *destination* of the hyperarc (X, Y).

Figure 2.1(a) shows an example of such a directed hypergraph with nine nodes and three hyperarcs:

$$\begin{aligned} N &= \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\} \\ E &= \{(\{n_1\}, \{n_2\}), (\{n_3, n_6\}, \{n_5\}), (\{n_7, n_8\}, \{n_4, n_9\})\}. \end{aligned}$$

The nodes in a hypergraph are depicted by dots while the edges are groups of nodes indicated by a curve. Note that the direction of the hyperarcs matters. In an undirected hypergraph, the direction of the edges is not important.

Definition 2.3.2 (Undirected hypergraph)**.** An *undirected hypergraph* H is a directed hypergraph $H = (N, E)$ in which E is symmetric, i.e. $\forall_{X,Y \in \mathcal{P}(N)}[(X, Y) \in E \Rightarrow (Y, X) \in E]$. The elements in E are called the *hyperedges*.

Figure 2.1(b) shows an example of a undirected hypergraph with nine nodes and six edges:

$$\begin{aligned} N &= \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\} \\ E &= \{(\{n_1\}, \{n_2\}), (\{n_3, n_6\}, \{n_5\}), (\{n_7, n_8\}, \{n_4, n_9\}), (\{n_2\}, \{n_1\}), (\{n_5\}, \{n_3, n_6\}), \\ &\quad (\{n_4, n_9\}, \{n_7, n_8\})\} \end{aligned}$$

A directed graph is a specific case of a directed hypergraph in which each hyperarc has exactly one source node and one destination node.

Definition 2.3.3 (Directed graph)**.** Let $H = (N, E)$ be a hypergraph. H is called a *directed graph* if and only if $\forall_{(X,Y)\in E}[|X| = |Y| = 1]$. Thus, the origin and destination sets of each arc contain exactly one element.

Note that a directed graph can be represented by (N, E) with $E \subseteq N \times N$.

A undirected graph, or simply graph, is a undirected hypergraph in which each hyperedge only has two nodes.

Definition 2.3.4 (Undirected Graph)**.** Let $H = (N, E)$ be a hypergraph. H is called an *undirected graph* (or simply *graph*) if and only if:

- E is symmetric,
  i.e. $\forall_{X,Y \in \mathcal{P}(N)}[(X, Y) \in E \rightarrow (Y, X) \in E]$, and
- the origin and destination sets for each edge contain exactly one element,
  i.e. $\forall_{(X,Y)\in E}[|X| = |Y| = 1]$.

Thus, an undirected graph can be represented by the pair (N, E) with $E \subseteq N \times N$. In figures 2.2(a) and 2.2(b), a directed graph and an undirected graph are shown. The directed graph of Figure 2.2(a) is represented by the following set of nodes (N) and the set of edges (E):

$$\begin{aligned} N &= \{n_1, n_2, n_3, n_4, n_5, n_6\} \\ E &= \{(n_2, n_1), (n_3, n_2), (n_3, n_4), (n_4, n_5), (n_5, n_6), (n_1, n_6), (n_2, n_5)\} \end{aligned}$$

The undirected graph of Figure 2.2(b) contains some extra edges:

$$\begin{aligned} N &= \{n_1, n_2, n_3, n_4, n_5, n_6\} \\ E &= \{(n_2, n_1), (n_3, n_2), (n_3, n_4), (n_4, n_5), (n_5, n_6), (n_1, n_6), (n_2, n_5), \\ &\quad (n_1, n_2), (n_2, n_3), (n_4, n_3), (n_5, n_4), (n_6, n_5), (n_6, n_1), (n_5, n_2)\} \end{aligned}$$

Definition 2.3.5 (Structure graph of a directed hypergraph [116])**.** Let $H = (N, E)$ be a directed hypergraph. The *structure graph*, H_S, associated with H is the directed graph $H_S = (N \cup U, F)$, where $U = (E \times \{1, 2\})$, and the elements of U are denoted by e_i, with $e \in E$ and $i \in \{1, 2\}$; and $F = F_O \cup F_D \cup \{e_1, e_2 | e \in E\}$, where F_O and F_D are defined as:

$$\begin{aligned} F_O &= \{(n, e_1) | \exists_{(X,Y)\in E}[e = (X, Y) \wedge n \in X]\} \\ F_D &= \{(e_2, n) | \exists_{(X,Y)\in E}[e = (X, Y) \wedge n \in Y]\} \end{aligned}$$

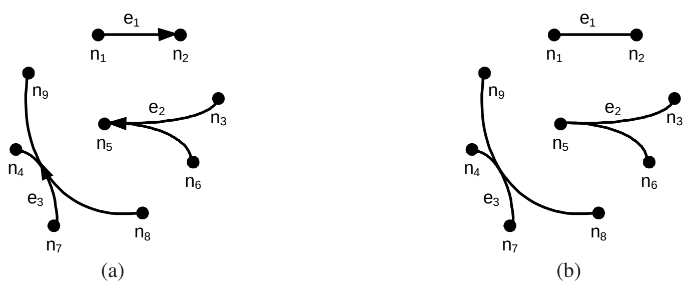


Figure 2.1: *A directed (a) and an undirected (b) hypergraph.*

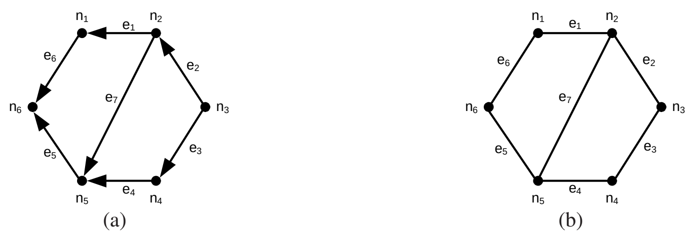


Figure 2.2: *A directed graph (a) and an undirected graph (b).*

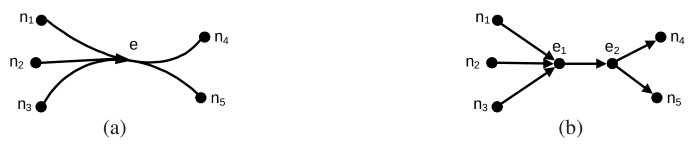


Figure 2.3: *A hypergraph (a) and its structure graph (b).*

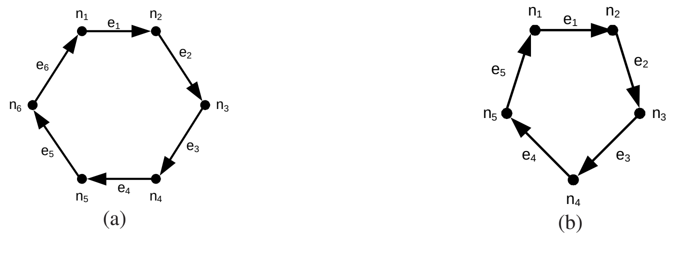


Figure 2.4: *A cyclic, bipartite graph (a) and a graph which is not bipartite (b).*

Figure 2.3 shows a directed hypergraph and its structure graph. Directed graphs can have a number of properties that are discussed below. First of all, we look at the degree of a node in a graph.

Definition 2.3.6 (Indegree, Outdegree, Degree)**.** Let $H = (N, E)$ with $E \subseteq N \times N$ be a directed graph and $n \in N$ be a node in H.

- The *indegree* of node n is the number of arcs directed to n, i.e. $indegree(n) = |\{(x, n) \mid x \in N \wedge (x, n) \in E\}|$;
- The *outdegree* of node n is the number of arcs directed from n, i.e. $outdegree(n) = |\{(n, x) \mid x \in N \wedge (n, x) \in E\}|$;
- The *degree* of n is the indegree of n plus the outdegree of n, i.e. $degree(n) = |\{(x, n) \mid x \in N \wedge (x, n) \in E\} \cup \{(n, x) \mid x \in N \wedge (n, x) \in E\}|$;
- n_1 is an input node of n if and only if there is a directed arc from n_1 to n, i.e. $(n_1, n) \in E$. The set of input nodes of n is denoted by $\bullet n$, i.e. $\bullet n = \{n_1 \mid (n_1, n) \in E\}$;
- n_1 is an output node of n if and only if there is a directed arc from n to n_1, i.e. $(n, n_1) \in E$. The set of output nodes of n is denoted by $n\bullet$, i.e. $n\bullet = \{n_1 \mid (n, n_1) \in E\}$.

For node n_2 in Figure 2.2(a) holds that: $indegree(n_2) = 1$, $outdegree(n_2) = 2$, $degree(n_2) = 3$, $\bullet n_2 = \{n_3\}$, and $n_2\bullet = \{n_1, n_5\}$.

Following the direction of the arcs in a directed graph one can ‘walk’ through the graph from one node to the other. Such a sequence of nodes and edges is called a *path*.

Definition 2.3.7 (Directed path, Closed path, Length of a path, Cycle)**.** Let $H = (N, E)$ be a directed graph.

- A *directed path*, or simply *path* is an alternating sequence of nodes and edges, $\sigma = \langle n_0, e_1, n_1, ..., e_x, n_x\rangle$ following the direction of the edges.
- A path, $\sigma = \langle n_0, e_1, n_1, ..., e_x, n_x\rangle$, is *closed* if the initial node is also the final node, i.e. $n_0 = n_x$.
- The *length* of a path σ, is the number of edges (including repetitions), i.e. $|\{e \mid e \in E \wedge e \in \sigma\}|$.
- A *cycle* is a closed path of length 1 or higher.

The path from node n_3 to node n_4 in Figure 2.2(a) is $\sigma = \langle n_3, e_3, n_4\rangle$. Note that there are two paths from n_3 to n_5: $\sigma_1 = \langle n_3, e_3, n_4, e_4, n_5\rangle$ and $\sigma_2 = \langle n_3, e_2, n_2, e_7, n_5\rangle$. In Figure 2.4(a), the path $\sigma = \langle n_1, e_1, n_2, e_2, n_3, e_3, n_4, e_4, n_5, e_5, n_6, e_6, n_1\rangle$ is a cycle of length 6.

Based on the notions of a path, a cycle, and the length of a path some other properties can be defined for graphs.

Definition 2.3.8 (Acyclic Graph)**.** Let $H = (N, E)$ be a directed graph. H is acyclic if it contains no cycles.

The graph in Figure 2.2(a) is an acyclic graph.

Definition 2.3.9 (Distance)**.** Let $H = (N, E)$ be a directed graph and let $n_1, n_2 \in N$ be two nodes in H. The *distance* from node n_1 to n_2 is the length of the shortest directed path from n_1 to n_2.

The distance between node n_1 and node n_6 in Figure 2.4(a) is 5.

Definition 2.3.10 (Eccentricity, Radius, Diameter)**.** Let $H = (N, E)$ be a directed graph and $n_1 \in N$ be a node in H. The *eccentricity* of n_1 is the greatest distance between n_1 and any other node $n \in N$. The *radius* of H is the minimum eccentricity of any node $n \in N$. The *diameter* of a graph is the maximum eccentricity of any node $n \in N$.

The eccentricity of node n_3 in the graph in Figure 2.2(a) is 3 and the eccentricity of n_1 is 1. The radius of the graph is 1, and the diameter of the graph is 3.

Definition 2.3.11 (Connected, Strongly connected)**.** Let $H = (N, E)$ be a directed graph. H is *connected* if between every pair of nodes there is a path, regardless of the direction of this path. A graph is *strongly connected* if a directed path exists from each node to each other node.

The graph in Figure 2.2(a) is connected but not strongly connected. The graph in Figure 2.4(a) is strongly connected.

Definition 2.3.12 (Bipartite graph)**.** Let $H = (N_1 \cup N_2, E)$ be a graph with two disjoint sets of nodes N_1 and N_2. H is said to be *bipartite* if every edge in E is of the form (n_1, n_2) or (n_2, n_1), where $n_1 \in N_1$ and $n_2 \in N_2$. Thus, $E \subseteq (N_1 \times N_2) \cup (N_2 \times N_1)$.

The graphs of figures 2.2(a), 2.2(b), and 2.4(a) are bipartite graphs, while the graph of Figure 2.4(b) is not. A tree is a special kind of graph.

Definition 2.3.13 (Directed tree, Rooted tree, Root, Leaf)**.** Let $H = (N, E)$ be a directed graph. H is called a *directed tree* if it is connected and a-cyclic, and $|E| = |N| - 1$. H is a *rooted tree* if one node has been designated as the *root*, i.e. $root \in N$, such that the root element has no ingoing arcs ($indegree(root) = 0$), and a directed path from the root towards each other node exists, i.e. $\forall_{n \in (N \setminus root)}[(root, n) \in E^+]$. A node n with no outgoing arcs ($outdegree(n) = 0$) is called a *leaf* of the tree.

For a more detailed introduction to graph theory we refer to [113, 173]. Moreover, in [41] a complete description of hypergraphs can be found. Graphs are used to present the model of many different business process modeling languages. Some of these modeling languages are introduced in the next sections. Also, the notion of a hypergraph is used in Chapter 4 to formally describe the structure of the workflow product.

2.4 Petri Nets and WorkFlow Nets

In this section two related modeling languages are discussed. First, Petri nets are formally introduced, followed by an explanation of WorkFlow nets, which are a special kind of Petri nets.

2.4.1 Petri Nets

This section introduces the basic terminology and notations for Petri nets, the formalism which was introduced in 1962 by Carl Adam Petri. For an elaborate introduction to Petri nets we refer to [82, 191, 239].

A Petri net is a mathematical model of a concurrent system and has a clear mathematical and graphical representation [82]. Petri nets are used to model all kinds of dynamic systems,

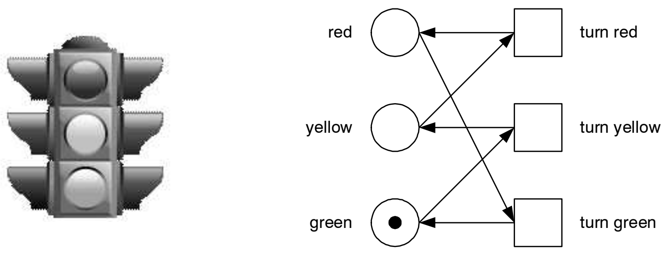


Figure 2.5: *An example of a Petri net describing the behavior of a traffic light. In the initial marking the place 'green' is marked with one token.*

including workflow processes. Since the Petri net language has formal and executable semantics, processes modeled in terms of a Petri net can be executed by an information system. Also, the sound mathematical basis of this modeling language makes it possible to analyse and verify processes modeled as Petri nets.

The classical Petri net [82] is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are graphically represented by circles and transitions by rectangles (see Figure 2.5 for an example of a Petri net). A Petri net can be defined formally, as follows [191, 239, 238].

Definition 2.4.1 (Petri net)**.** A *Petri net* is a triple (P, T, F) where:

· P is a finite set of *places*

· T is a finite set of *transitions* ($P \cap T = \emptyset$)

· $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation)

Note that $(P \cup T, F)$ is a bipartite directed graph. The elements of $P \cup T$ are called *nodes*. A node x is an *input node* of another node y if and only if there is a directed arc from x to y, i.e., $(x, y) \in F$. Node x is an *output node* of y if and only if $(y, x) \in F$. We use the notation $\bullet x$, with $x \in P \cup T$, to denote the set of input nodes to x: $\bullet x = \{y \mid (y, x) \in F\}$ (cf. Definition 2.3.6). Similarly, $x\bullet$ is the set of nodes sharing x as an input: $x\bullet = \{y \mid (x, y) \in F\}$.

At any time, a place contains zero or more *tokens*, drawn as black dots. The *state* of a Petri net, often referred to as *marking*, is the distribution of tokens over places, i.e., a multiset $M : P \to \mathbb{N}$. A state is represented as follows: $[p_1, p_2^2, p_3]$ is the state with one token in place p_1, two tokens in p_2, and one token in p_3. To compare states we define a partial ordering on these multisets. For any two states M_1 and M_2, $M_1 \leq M_2$ if and only if for all $p \in P$: $M_1(p) \leq M_2(p)$, cf. Definition 2.2.8.

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

1. A transition t is said to be *enabled* if and only if each input place p of t contains at least one token.
2. An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

Given a Petri net (P, T, F) and a state M_1, we have the following notations:

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\sigma} M_n$: the *firing sequence* $\sigma = \langle t_1, t_2, t_3, \ldots, t_{n-1} \rangle$ leads from state M_1 to state M_n via a (possibly empty) set of intermediate states $M_2, ...M_{n-1}$, i.e., $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} ... \xrightarrow{t_{n-1}} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) if and only if there is a firing sequence σ such that $M_1 \xrightarrow{\sigma} M_n$. Note that the empty firing sequence is also allowed, i.e. by definition $M_1 \xrightarrow{*} M_1$. We use (PN, M) to denote a Petri net PN with an initial state M. A state M' is a *reachable state* of (PN, M) if and only if $M \xrightarrow{*} M'$.

Now we can define some standard properties for Petri nets. First, some properties related to the dynamics of a Petri net are given.

Definition 2.4.2 (Live)**.** A Petri net (PN, M) is *live* if and only if for every reachable state M' and every transition $t \in T$ there is a state M'' reachable from M' which enables t.

A Petri net is *structurally live* if an initial state exists such that the net is live.

Definition 2.4.3 (Bounded, Safe)**.** A Petri net (PN, M) is *bounded* if and only if for each place $p \in P$ there is a natural number n such that for every reachable state the number of tokens in p is less than n. The net is *safe* if and only if for each place the maximum number of tokens does not exceed 1.

A Petri net is *structurally bounded* if the net is bounded for any initial state.

For $PN = (P, T, F)$ we also define some standard structural properties.

Definition 2.4.4 (Strongly connected)**.** A Petri net is *strongly connected* if and only if for every pair of nodes x and y, there is a directed path leading from x to y (cf. Definition 2.3.11).

Definition 2.4.5 (Free-choice)**.** A Petri net is *free-choice* if and only if for every two transitions $t_1 \in T$ and $t_2 \in T$, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$.

Definition 2.4.6 (State machine)**.** A Petri net is a *state machine* if and only if each transition has at most one input place and at most one output place, i.e., for all $t \in T$: $|\bullet\, t| \leq 1$ and $|t \,\bullet\,| \leq 1$.

Definition 2.4.7 (Marked graph)**.** A Petri net is a *marked graph* if and only if each place has at most one input transition and at most one output transition, i.e., for all $p \in P$: $|\bullet\, p| \leq 1$ and $|p \,\bullet\,| \leq 1$.

2.4.2 WorkFlow Nets

A *WorkFlow net* (WF-net) is a special kind of Petri net which models the control-flow dimension of a workflow. It has been introduced in [2]. In a WF-net the transitions correspond to activities in the business process that is represented by the WF-net. Places correspond to pre- and post-conditions of these activities. A WF-net has a *unique source place* and a *unique sink place*. Additionally, in a WF-net all nodes (i.e. places and transitions) should be on a directed path from source to sink. Figure 2.6 shows an example of a WF-net.

A WF-net can be formally defined as follows [2]:

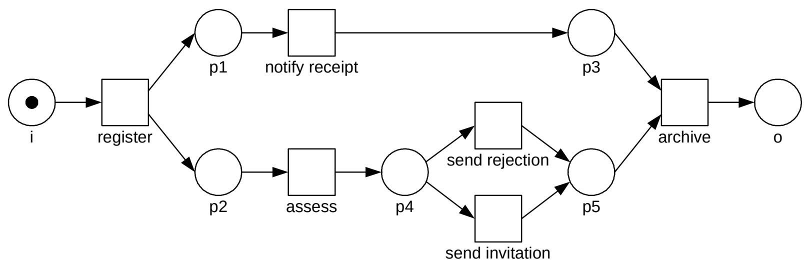


Figure 2.6: *An example of a WF-net. The WF-net describes a simple version of processing job applications. Note the special source and sink places i and o.*

Definition 2.4.8 (WF-net)**.** A Petri net $PN = (P, T, F)$ is a WF-net (WorkFlow net) if and only if:

- There is one source place $i \in P$ such that $\bullet i = \emptyset$
- There is one sink place $o \in P$ such that $o\bullet = \emptyset$
- Every node $x \in P \cup T$ is on a directed path from i to o

Given the definition of a WF-net it is easy to derive some properties of WF-nets [6]. For instance, if PN is a WF-net and we add a transition t^* to PN which connects sink place o with source place i (i.e., $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is *strongly connected*.

Another important property of WF-nets is *soundness*. Soundness was introduced as a correctness criterion for process models in [2, 3, 6]. It is a desirable property and can be formally verified. We first deal with the classical notion of soundness of WF-nets before introducing some variants to this basic concept.

Definition 2.4.9 (Soundness)**.** A WF-net $PN = (P, T, F)$ is sound if and only if:

1. There is an *option to complete*: For every state M reachable from the initial state M_i (one token in place i), there exists a firing sequence leading from state M to state M_o. Formally:

$$\forall_M [(M_i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} M_o)]$$

2. *Proper completion* is possible: State M_o is the only state reachable from state M_i with at least one token in place o. Formally[1]:

$$\forall_M [(M_i \xrightarrow{*} M \ \wedge \ M \geq [o]) \Rightarrow (M = M_o)]$$

3. There are *no dead transitions* in (PN, M_i). Formally:

$$\forall_{t \in T} \, \exists_{M,M'} [\, M_i \xrightarrow{*} M \xrightarrow{t} M']$$

[1]Note that the second requirement (proper completion) follows from the first requirement. However, it is included here to simplify the practical interpretation of soundness

Soundness ensures that the process can always terminate with a single token in the end place and that all other places are empty. In addition, there is no dead transition.

Many variants of this classical notion of soundness exist in literature. Here, only four common variants of soundness are addressed. A more detailed overview can be found in [14, 270]. The first variant of soundness discussed here is k-soundness [124, 125]. k-soundness focuses on the first requirement in Definition 2.4.9, i.e. the option to complete. In the k-soundness definition this notion is parameterized by a variable k which indicates the initial number of tokens in the source place.

Definition 2.4.10 (k-soundness)**.** A WF-net $PN = (P, T, F)$ is k-sound if and only if for every state M reachable from the initial state M_i with k tokens in place i ($M_i = [i^k]$), a firing sequence exists leading from state M to state M_o with k tokens in place o ($M_o = [o^k]$). Formally:

$$\forall_M [(M_i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} M_o)]$$

A special variant of k-soundness, 1-soundness, is also known as weak soundness [169]:

Definition 2.4.11 (Weak soundness)**.** A WF-net $PN = (P, T, F)$ is weak sound if and only if PN is 1-sound.

The notion of relaxed soundness, where proper termination is possible but not guaranteed, is introduced in [80, 81, 95].

Definition 2.4.12 (Relaxed soundness)**.** A WF-net $PN = (P, T, F)$ is relaxed sound if and only if every transition is in some firing sequence that starts in state M_i and ends in state M_o. Formally:

$$\forall_{t \in T}\, \exists_{M,M'} [\, M_i \xrightarrow{*} M \xrightarrow{t} M' \xrightarrow{*} M_o]$$

Relaxed soundness is intended to represent a more pragmatic view on correctness which is weaker (in a formal sense) but more easily applicable to application-oriented modeling. Intuitively, relaxed soundness means that enough executions exist which terminate properly. Enough means at least so many that every transition is covered [81].

The soundness notions we have discussed so far all focus on ending in a state with no tokens in a place other than the sink place. The notion of lazy soundness [224, 225] weakens this requirement. Tokens may be left behind in the model as long as the sink place is marked precisely once.

Definition 2.4.13 (Lazy soundness)**.** A WF-net $PN = (P, T, F)$ is lazy sound if and only if the following two requirements are satisfied:

1. There is an *option to complete*: For every state M reachable from the initial state M_i (one token in place i), a firing sequence exists that leads from state M to state M' with one token in place o (and possibly also tokens in other places). Formally:

$$\forall_M [(M_i \xrightarrow{*} M) \Rightarrow \exists_{M'} [(M \xrightarrow{*} M' \wedge M'(o) = 1)]]$$

2. *Proper completion* is possible: Every state M, reachable from the initial state M_i, has at most one token in place o (and possibly tokens in other places). Formally:

$$\forall_M [(M_i \xrightarrow{*} M) \Rightarrow (M(o) \leq 1)]$$

The various notions of soundness of WF-nets are used to verify the correctness of a process model. More details about the application of soundness to process model analysis and verification can be found in [2, 3, 6, 80, 81, 95, 124, 125, 169, 224, 225].

2.5 YAWL

YAWL (Yet Another Workflow Language) is a workflow language based on Petri nets and extends these Petri nets with additional features to facilitate the modeling of complex workflows [15]. Compared to other languages, YAWL is more expressive and has clear and unambiguous semantics. YAWL has been developed since Petri nets (and all other workflow languages) have some known limitations for the support of workflow patterns [17, 302].

A process model in YAWL (*workflow specification*) is a set of extended workflow nets (EWF-nets) which form a hierarchy. Each EWF-net consists of tasks (either *atomic tasks* or *composite tasks*) and conditions which can be interpreted as places. Each EWF-net has one unique input condition and one unique output condition. A task can have multiple instances. The notion of AND- and XOR-joins and -splits can be easily mapped on WF-nets. However, the notion of an OR-split and OR-join, introduced in the YAWL language, is a real extension that cannot be mapped on classical Petri nets. Finally, YAWL provides the ability to remove tokens from places irrespective of how many tokens there are (*cancellations regions*) [15]. This notation cannot be mapped on classical Petri nets either and is closely linked to the notion of reset nets [87, 88].

EWF-nets have been introduced to suitably deal with workflow patterns [15, 17, 302] since Petri nets only have a limited support for these patterns. An EWF-net extends the notion of a workflow net with OR-joins, cancellation, and multiple instances of activities. It can be formalized as follows [15].

Definition 2.5.1 (EWF-net)**.** An extended workflow net (EWF-net) N is a tuple $(C, i, o, T, F, split, join, rem, nofi)$ such that

- C is the set of conditions,
- $i \in C$ is the input condition,
- $o \in C$ is the output condition,
- T is a set of tasks,
- $F \subseteq (C \setminus \{o\} \times T) \cup (T \times C \setminus \{i\}) \cup (T \times T)$ is the flow relation,
- every node in the graph $(C \cup T, F)$ is on a directed path from **i** to **o**,
- *split*: $T \rightarrow \{AND, XOR, OR\}$ specifies the split behavior of each task,
- *join*: $T \rightarrow \{AND, XOR, OR\}$ specifies the join behavior of each task,
- *rem*: $T \not\rightarrow \mathcal{P}(T \cup C \setminus \{i, o\})$ specifies the additional tokens to be removed by emptying a part of the workflow, and
- *nofi*: $T \not\rightarrow \mathbb{N} \times \mathbb{N}^{inf} \times \mathbb{N}^{inf} \times \{dynamic, static\}$ specifies the multiplicity of each task (minimum, maximum, threshold for continuation, and dynamic/static creation of instances).

Note that the tuple (C, T, F) corresponds to a Petri net where C corresponds to places, T corresponds to transitions, and F is the flow relation. However, there are two additional requirements. First of all, an EWF-net has two special places: **i** and **o**. Secondly, the flow relation also allows for direct connections between transitions (called tasks in YAWL) [15]. Moreover, an EWF-net extends Petri nets with multiple instances, composite tasks, OR-joins, and removal of tokens. In Figure 2.7 the graphical representation of each of these concepts is shown. Figure 2.8 contains an example YAWL model.

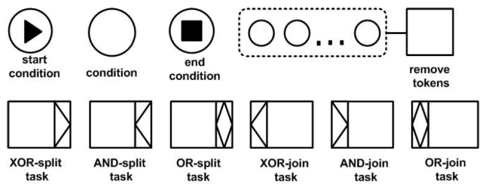


Figure 2.7: *Symbols in YAWL, reproduced from [305].*

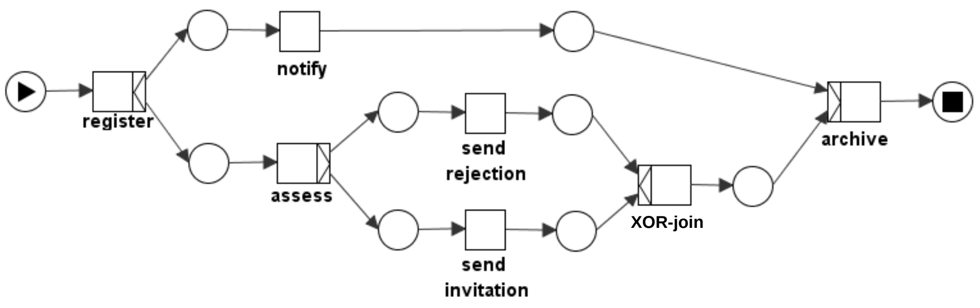


Figure 2.8: *An example of a YAWL model.*

Since YAWL has a formal basis, its workflow specifications can be checked for correctness. This correctness verification can be done based (amongst others) on four desirable properties for processes with cancellation regions and OR-joins: (i) soundness, (ii) weak soundness, (iii) irreducible cancellation regions, and (iv) immutable OR-joins [304]. The first two properties are soundness properties (cf. the different kinds of soundness for WF-nets in Section 2.4.2) and focus on the correctness of a process model. The latter two properties focus on detecting the existence of unnecessary cancellation regions and OR-joins in the process model [305].

The YAWL language is supported by a YAWL workflow system, which consists of a number of components including a workflow engine and an editor. YAWL workflow specifications can be designed using the YAWL editor and deployed in the YAWL engine for execution [305, 307].

2.6 Event-Driven Process Chains

Event-driven Process Chains (EPCs) are an intuitive graphical language to describe business processes, introduced by Keller, Nuttgens and Scheer [146]. The language is aimed at describing processes at the business level and not necessarily at a formal specification level. An EPC represents the control flow structure of the process as a chain of events and functions [4].

Definition 2.6.1 (Event-driven process chain)**.** An event-driven process chain is a five-tuple (E, F, C, T, A):

· E is a finite set of events,

· F is a finite set of functions,

· C is a finite set of logical connectors,

· $T \in C \to \{\wedge, XOR, \vee\}$ which maps each connector to a connector type,

· $A \subseteq (E \cup F \cup C) \times (E \cup F \cup C)$ is a set of arcs.

The graphical representation of an EPC is shown in Figure 2.9. Events are depicted as hexagons, functions as rectangles and connectors as circles.

Since EPCs were originally introduced informally [146], several approaches towards the formalization of EPCs exist, leading to different constraints and requirements. For instance, Langner et al. [160] require that the EPC contains no arcs between nodes of the same type, and Keller and Teufel [147] restrict their EPCs to graphs with (i) at least three nodes (one start event, one end event and one function in between) and (ii) no cyclic connections between connectors. Mendling [175] provides a comprehensive overview of these different EPC formalizations. Like the soundness property defined for workflow nets, also a notion of correctness for EPCs has been developed based on these formal definitions [4].

In this thesis we do not formulate explicit requirements for EPCs. Thus, nodes of the same type may be connected with each other, there may be less than three nodes, and cyclic connections between nodes may exist as well. The reader should note that such constraints are not required for the application of the work described in this thesis, but for most EPC verification, analysis and assessment techniques these constraints might be enforced.

2.7 Markov Chains and Markov Decision Processes

Markov chains and Markov Decision Processes (MDPs) are mathematical models of systems that change over time. A Markov chain descibes such a system, while an MDP extends the first model with a control aspect: The evolution of the system can be influenced by taking decisions.

2.7.1 Markov Chains

The notion of a Markov chain was devised by the Russian mathematician Andrei A. Markov (1856-1922). Nowadays, Markov chains are widely used to model uncertainty in systems that evolve over time [269]. They have applications in many areas, including operations research, biology, and computer science.

At the basis of a Markov chain is a *stochastic process*. A stochastic process is a probabilistic model of a system that evolves randomly. If the system is observed at discrete time points ($n = 0, 1, 2, ...$), and X_n is the state of the system at time n, then $\{X_n \mid n \geq 0\}$ is a (discrete-time) stochastic process describing this system [155]. The set of possible states the system can be in is called the *state space*. The behavior of the process is described by probability distributions. This means that even if the initial state is known, various possibilities exist to which state the process may go to in the future. A probability distribution is given by a distribution function $P : \{X_n \mid n = 0, 1, 2, ...\} \to [0, 1]$. Then, $P(X_n = i)$ denotes the probability that the system is in state i at time n.

Definition 2.7.1 (Stochastic process)**.** A stochastic process $\{X_n \mid n = 0, 1, 2, ...\}$ is a collection of random variables $X_0, X_1, X_2, ...$ which describe the state of a system that evolves randomly. The random variables are indexed by the discrete time points ($n = 0, 1, 2, ...$) at which the system is observed.

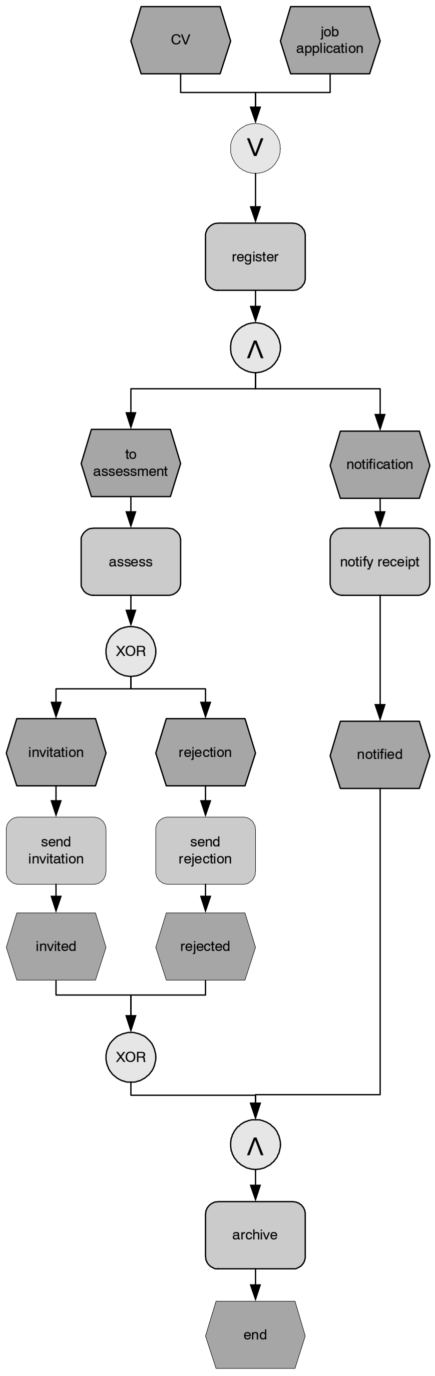


Figure 2.9: *An example of an EPC describing a simple job application process.*

A Markov chain is a discrete-time stochastic process, having the property that the next state of the process is merely dependent on the present state and not on the previous states of the system (Markov property) [68, 226, 269]. At each point in time, the state of the system may change from the current state to another state, or it may remain in the same state, according to a given probability distribution. These changes of state are called *transitions* and the associated probabilities of state changes are termed as *transition probabilities* [68, 226, 269].

Definition 2.7.2 (Markov chain)**.** The stochastic process $\{X_n \mid n = 0, 1, 2, ...\}$ with state space S is called a discrete-time Markov chain if, for each $n \geq 0$,

$$P(X_{n+1} = i_{n+1} \mid X_0 = i_0, ..., X_n = i_n) = P(X_{n+1} = i_{n+1} \mid X_n = i_n)$$

for all possible values of $i_0, ..., i_{n+1} \in S$.

From this definition it is clear that a Markov chain is *memoryless*. The probability of being in state i_{n+1} at time $n + 1$, is only dependent on the state at time n (current state) and not on the earlier history of states $(i_0, i_1, ..., i_{n-1})$ at times $0, 1, ..., n - 1$.

Markov chains with *time-homogeneous* transition probabilities are processes where transition probabilities do not depend on the specific point in time at which a transition occurs, i.e. for $n \geq 1$:

$$P(X_{n+1} = i_{n+1} \mid X_n = i_n) = P(X_n = i_{n+1} \mid X_{n-1} = i_n).$$

Let us assume that the Markov chain is time-homogeneous. Then, the one-step transition probability from state i to state j can be denoted by:

$$p_{ij} = P(X_{n+1} = j \mid X_n = i), \quad \text{for } i, j \in S, \text{ and independent of } n.$$

These probabilities satisfy the following properties:

$$p_{ij} \geq 0, \qquad \text{for } i, j \in S,$$

and

$$\sum_{j \in S} p_{ij} = 1, \qquad \text{for } i \in S.$$

Figure 2.10 shows an example of a Markov chain with a finite state space. For this Markov chain, e.g. $p_{12} = 1$, $p_{13} = 0$, $p_{01} = \frac{1}{2}$, and $p_{55} = 1$, and

$$\sum_{j \in S} p_{1j} = p_{12} = 1, \qquad \text{and} \qquad \sum_{j \in S} p_{0j} = p_{01} + p_{05} = \frac{1}{2} + \frac{1}{2} = 1.$$

A transition from state i to state j can also happen in more than one step. The n-step transition probabilities are defined by:

$$p_{ij}^{(n)} = P(X_n = j \mid X_0 = i), \quad \text{for } i, j \in S.$$

For example, in the Markov chain of Figure 2.10(a), it is possible to get from state 0 to state 3 in three steps. Note that $p_{ij}^{(1)} = p_{ij}$. In a Markov chain with a finite state space, the transition probability distribution can be represented by a matrix **P**, the *transition matrix*. In this matrix, the (i, j)-th element of **P** is equal to p_{ij}. The n-step transition probabilities can be computed by taking the n-th power of the transition matrix: $\mathbf{P}^n$.

$$\begin{tikzcd}
\boxed{0} \arrow[r, "1/2"] \arrow[d, "1/2"'] & \boxed{1} \arrow[r, "1"] & \boxed{2} \arrow[d, "1"] \\
\boxed{5} \arrow[loop below, "1"] & \boxed{4} \arrow[u, "1"] & \boxed{3} \arrow[l, "1"]
\end{tikzcd}$$

(a) An example of a Markov chain.

$$\mathbf{P} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) The transition matrix for the Markov chain.

Figure 2.10: *An example of a Markov chain with a finite state space, $S = \{0, 1, 2, 3, 4, 5\}$. The transition probabilities are represented by the transition matrix in (b). States 1, 2, 3, 4, and 5 are recurrent states, while state 0 is a transient state. State 5 is the only absorbing state in this Markov chain. States 1, 2, 3, and 4 have period 4. There are two closed classes: $\{1, 2, 3, 4\}$ and $\{5\}$.*

A state $j \in S$ is called *reachable* or *accessible* from state $i \in S$ if there is a path of length $n \geq 0$ of subsequent states starting in state i and going to state j, i.e. there is an $n \geq 0$ with $p_{ij}^{(n)} > 0$. We denote the reachability of state j from state i with $i \rightarrow j$. State i and state j are said to *communicate* if state i and state j are reachable from each other, i.e. $i \leftrightarrow j$. Two states that communicate are said to be in the same *class*. A class is *closed* if it is not possible to leave the class, e.g. the Markov chain in Figure 2.10(a) has two closed classes: $\{1, 2, 3, 4\}$ and $\{5\}$. A Markov chain is said to be *irreducible* if all states of its state space belong to the same class.

Many applications of Markov chains involve chains in which some of the states are *recurrent* and other states are *transient*. To define the concepts of recurrent and transient states, we first need to introduce the notion of *first-passage time probabilities* [269]. Let $\{X_n \mid n = 0, 1, 2, ...\}$ be a discrete-time Markov chain with finite state space S and one-step transition probabilities p_{ij}, $i, j \in S$. For any $n = 1, 2, ...$, the first-passage time probability $f_{ij}^{(n)}$ is defined by:

$$f_{ij}^{(n)} = P(X_1 \neq j, X_2 \neq j, ..., X_{n-1} \neq j, X_n = j \mid X_0 = i).$$

Thus, $f_{ij}^{(n)}$ denotes the probability that the first visit to state j from state i occurs at time n. Then, the probability that starting from state i the process *ever* visits state j is defined as:

$$f_{ij} = \sum_{n=1}^{\infty} f_{ij}^{(n)}.$$

A state i is called *recurrent* if $f_{ii} = 1$, i.e. the system will eventually always return to state i. State i is said to be *transient* if $f_{ii} < 1$. For example, in Figure 2.10(a), state 0 is a transient state, while the other states are recurrent states. A special case of a recurrent state is an *absorbing* state. State i is an absorbing state if it is impossible to leave this state, i.e. $p_{ii} = 1$. Note that state 5 in Figure 2.10(a) is an absorbing state. A closed class only contains recurrent states and is therefore also called a *recurrent class*. A class that is not closed is a *transient class* since the process will certainly leave this class again. A Markov chain with in each closed class exactly one absorbing state is called a *transient chain*.

A state i is said to be *periodic* if it has a period d if $p_{ii}^{(n)} = 0$ whenever n is not divisible by d, and d is the largest integer with this property. A state with period 1 is said to be *aperiodic*.

Periodicity is also a property of a class; all states in a class have the same period.

Markov chains are the basis of another mathematical model, a Markov Decision Process, on which the next section elaborates.

2.7.2 Markov Decision Processes

A Markov Decision Process (MDP) [51, 226, 269] is an extension of a Markov chain with decisions. In each state of the Markov chain, several decisions can be taken. The transition matrix is dependent on these decisions. Each decision is associated with a reward (or cost), which depends on the state of the system. The goal of an MDP is to find a function, called a *strategy*, which specifies which decision to take in each state, so as to optimize some function of the sequence of rewards (minimize cost/maximize reward) [269].

To solve the problem of optimizing the reward function, a dynamic programming approach can be used. This is a mathematical approach for analyzing sequential decision processes with a finite horizon. It was developed in the 1950s by Richard Bellman (1920-1984) and can be computed with e.g. a linear programming algorithm or a value iteration algorithm [40, 269].

The application of MDPs in this thesis focuses on minimizing cost. Therefore, we will only elaborate on minimizing the cost function. However, this approach can be adopted in a similar way to maximize rewards.

Definition 2.7.3 (Markov Decision Process)**.** A Markov Decision Process (MDP) can be represented by a tuple (S, T, A, P, c, q) where

- S is the finite state space ($S = \{0, 1, 2, ..., M\}$).
- T is the set of discrete time points with a finite horizon ($T = \{0, 1, 2, ..., N\}$).
- A is the finite decision space. In each state, $i \in S$, the set of decisions that can be taken (A_i) is a subset of the decision space (i.e. $A_i \subseteq A$).
- P is the transition function: $p_{ij}(a)$ is the probability that decision a in state i leads to state j at the next time point.
- c is the immediate cost function such that $c^a(i)$ is the immediate cost received in state i when decision a is taken:

$$c^a(i) := s_i + \sum_{j \in S} p_{ij}(a) \cdot cst^a(i, j)$$

  where s_i denotes the sojourn cost of being in state i for one time step and $cst^a(i, j)$ represents the cost whenever the process is in state i, decision a is taken and the process moves to state j.
- q is the terminal cost function. $q(i)$ denotes the terminal cost when the process finishes in state i at the time horizon $t = N$.

At each time point n the process is in one of the states $i \in S$. Then, a decision from the set A_i has to be taken and the process moves to another state. We are interested in choosing the decisions, that should be taken in a specific state at a specific point in time, such that the expected total cost over a finite period is minimized. Any prescription for taking decisions is called a *strategy*. In general, the decisions to be taken may depend on time n and state i_n, but also on the complete history of the process. A *decision rule* prescribes a strategy for all states, all time points and all possible histories.

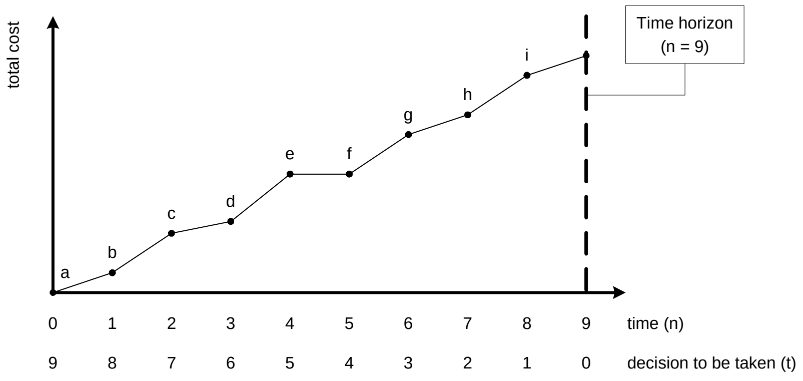

total cost
Time horizon
(n = 9)
a
b
c
d
e
f
g
h
i
0 1 2 3 4 5 6 7 8 9 time (n)
9 8 7 6 5 4 3 2 1 0 decision to be taken (t)

Figure 2.11: *This figure describes a time line of an MDP with a finite time horizon. The process ends at time $n = 9$. This means that from the start point $n = 0$ exactly nine decisions are made. When a value iteration algorithm is used to calculate an optimal decision rule, backward induction is used to step-by-step derive the decision rule. First, the cost at the time horizon are determined. The time horizon is the point in time at which the process ends and no decisions can be made anymore. This point is either represented by the time, i.e. $n = 9$, or by the number of decision epochs to the end of the process, i.e. $t = 0$. Then, a strategy (f_1) is calculated for the last decision step in the process, i.e. the situation in which one decision can be made $(t = 1)$. This calculation is based on the expected cost at $t = 0$. Subsequently, the decision rules for $t = 2, 3, ...$ can be determined.*

Let $h_{n-1} = (i_0, a_0, ..., i_{n-1}, a_{n-1})$ denote a particular history of a process in state i_n at time n, which consists of the states visited and decisions taken in the process previously. Then, the decision rule **f** = $(f_0, f_1, ..., f_N)$ is defined, where $f_0, f_1, ..., f_N$ is a sequence of functions, such that:

$$f_n(h_{n-1}, i_n) \in A_{i_n}$$

Thus, decision rule **f** is a function that assigns a strategy (f_n) to each point in time n, $0 \leq n \leq N$. The strategy, f_n, again is a function that prescribes a decision a for each state (i_n) and each history (h_{n-1}). A decision rule is called a *Markov decision rule* if it does not depend on the history of the process (i.e. $f_n(h_{n-1}, i_n) = f_n(i_n)$). A Markov decision rule is called *stationary* if the recommended decisions remain identical over time (i.e. $f_0 = f_1 = ... = f_N$).

A Markov decision rule for an MDP with a finite horizon can be determined using a value-iteration algorithm. In this algorithm, *backward induction* is used to determine the minimal total expected cost and the best decision strategy. Backward induction is the process of reasoning backwards in time, from the end of a problem to the start, such that the optimal course of decisions is determined step-by-step. For such a calculation, the steps often are indexed by the number of decisions to be made until the end of the process, i.e. the number of decision epochs to the time horizon, instead of the time point at which the decision is made. Figure 2.11 shows the relation between these two variables.

In the value iteration algorithm for our MDP problem, first, the cost at the time horizon $(t = 0)$ are calculated. At this time horizon no decisions can be made anymore. Therefore, only the final cost $(q(i)$, for $i \in S)$ are considered and no strategy is given. Then, for $t = 1$ the expected cost for one decision are calculated based on the cost at time $t = 0$, and the best strategy is determined based on the arguments from this calculation. Next, t is again increased by 1 and the expected cost and best strategy are computed. This is repeated until $t = N$.

Proposition 2.7.1. Let $V_0(i) := q(i)$, and $V_t(i), t = 1, 2, ...$ be recursively given by

$$V_t(i) := \min_{a \in A_i} [\, c^a(i) + \sum_{j \in S} p_{ij}(a) \cdot V_{t-1}(j) \,].$$

Then, any decision rule f_t determined by

$$f_t(i) = \arg\min_{a \in A_i} [\, c^a(i) + \sum_{j \in S} p_{ij}(a) \cdot V_{t-1}(j) \,]$$

achieves the minimal cost over t periods to the time horizon.

Note that $V_0(i)$ is the terminal cost in state i and that $V_t(i)$ can be interpreted as the minimal total expected cost when being in the current state i and with t decision epochs left to the time horizon. The `argmin` function gives the argument of the minimum value; in this case it gives the decision that has the minimum cost. For example, $V_1(i)$ represents the minimal total expected cost when there is only one time period left to the time horizon and only one decision can be made. Similarly, $f_t(i)$ denotes the decision that achieves the minimal total expected cost when t decisions can be made before the end of the process.

Example

Consider an MDP with two states (0 and 1), i.e. $S = \{0, 1\}$, and two decisions (a and b) which can both be taken in each state, i.e. $A = \{a, b\}$, $A_0 = \{a, b\}$, and $A_1 = \{a, b\}$. The transition probabilities are given by the following transition matrices:

$$P^a = \begin{pmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \qquad \text{and} \qquad P^b = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{3}{4} & \frac{1}{4} \end{pmatrix},$$

e.g. $p^a(0,1) = \frac{2}{3}$ and $p^b(0,1) = \frac{1}{3}$. Figure 2.12 shows a graphical representation of this MDP. The direct cost function and the final cost function are given by:

$$\begin{aligned} c^a(0) &= 2, \qquad & c^a(1) &= 1, \qquad & q(0) &= 1, \\ c^b(0) &= 2, \qquad & c^b(1) &= 0, \qquad & q(1) &= 2. \end{aligned}$$

We assume a finite planning horizon of two decision epochs, i.e. $T = \{0, 1, 2\}$. Now, we determine the minimal cost and the corresponding optimal strategy per decision epoch. At time $t = 0$ no decision can be made anymore. Thus, the final cost per state are counted:

$$\begin{aligned} V_0(0) &= q(0) = 1 \\ V_0(1) &= q(1) = 2 \end{aligned}$$

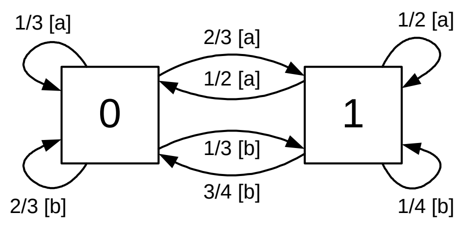


Figure 2.12: *A graphical representation of the example MDP.*

At time $t = 1$, one decision is to be taken. The direct cost for this decision are counted plus the final cost for the state the process is in after taking the decision:

$$\begin{aligned}
V_1(0) &= \min\{c^a(0) + p_{00}(a)\cdot V_0(0) + p_{01}(a)\cdot V_0(1)\ ;\ c^b(0) + p_{00}(b)\cdot V_0(0) + p_{01}(b)\cdot V_0(1)\}\\
&= \min\{2 + \frac{1}{3}\cdot 1 + \frac{2}{3}\cdot 2\ ;\ 2 + \frac{2}{3}\cdot 1 + \frac{1}{3}\cdot 2\} = \min\{2 + \frac{1}{3} + \frac{4}{3}\ ;\ 2 + \frac{2}{3} + \frac{2}{3}\}\\
&= \min\{\frac{11}{3}\ ;\ \frac{10}{3}\} = \frac{10}{3}\\
V_1(1) &= \min\{c^a(1) + p_{10}(a)\cdot V_0(0) + p_{11}(a)\cdot V_0(1)\ ;\ c^b(1) + p_{10}(b)\cdot V_0(0) + p_{11}(b)\cdot V_0(1)\}\\
&= \min\{1 + \frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 2\ ;\ 0 + \frac{3}{4}\cdot 1 + \frac{1}{4}\cdot 2\} = \min\{1 + \frac{1}{2} + 1\ ;\ 0 + \frac{3}{4} + \frac{1}{2}\}\\
&= \min\{\frac{5}{2}\ ;\ \frac{5}{4}\} = \frac{5}{4}\\
f_1(0) &= \arg\min\{c^a(0) + p_{00}(a)\cdot V_0(0) + p_{01}(a)\cdot V_0(1)\ ;\ c^b(0) + p_{00}(b)\cdot V_0(0) + p_{01}(b)\cdot V_0(1)\}\\
&= b\\
f_1(1) &= \arg\min\{c^a(1) + p_{10}(a)\cdot V_0(0) + p_{11}(a)\cdot V_0(1)\ ;\ c^b(1) + p_{10}(b)\cdot V_0(0) + p_{11}(b)\cdot V_0(1)\}\\
&= b
\end{aligned}$$

At time $t = 2$, two decision can be made until the time horizon. The direct cost for each decision are counted plus the expected cost for the state the process is in after taking the decision:

$$\begin{aligned}
V_2(0) &= \min\{c^a(0) + p_{00}(a)\cdot V_1(0) + p_{01}(a)\cdot V_1(1)\ ;\ c^b(0) + p_{00}(b)\cdot V_1(0) + p_{01}(b)\cdot V_1(1)\}\\
&= \min\{2 + \frac{1}{3}\cdot\frac{10}{3} + \frac{2}{3}\cdot\frac{5}{4}\ ;\ 2 + \frac{2}{3}\cdot\frac{10}{3} + \frac{1}{3}\cdot\frac{5}{4}\}\\
&= \min\{\frac{72}{36} + \frac{40}{36} + \frac{30}{36}\ ;\ \frac{72}{36} + \frac{80}{36} + \frac{15}{36}\} = \min\{\frac{142}{36}\ ;\ \frac{167}{36}\} = \frac{142}{36} = 3\frac{17}{18}\\
V_2(1) &= \min\{c^a(1) + p_{10}(a)\cdot V_1(0) + p_{11}(a)\cdot V_1(1)\ ;\ c^b(1) + p_{10}(b)\cdot V_1(0) + p_{11}(b)\cdot V_1(1)\}\\
&= \min\{1 + \frac{1}{2}\cdot\frac{10}{3} + \frac{1}{2}\cdot\frac{5}{4}\ ;\ 0 + \frac{3}{4}\cdot\frac{10}{3} + \frac{1}{4}\cdot\frac{5}{4}\}\\
&= \min\{1 + \frac{5}{3} + \frac{5}{8}\ ;\ 0 + \frac{10}{4} + \frac{5}{16}\} = \min\{\frac{79}{24}\ ;\ \frac{45}{16}\} = \frac{45}{16} = 2\frac{13}{16}\\
f_2(0) &= \arg\min\{c^a(0) + p_{00}(a)\cdot V_1(0) + p_{01}(a)\cdot V_1(1)\ ;\ c^b(0) + p_{00}(b)\cdot V_1(0) + p_{01}(b)\cdot V_1(1)\}\\
&= a\\
f_2(1) &= \arg\min\{c^a(1) + p_{10}(a)\cdot V_1(0) + p_{11}(a)\cdot V_1(1)\ ;\ c^b(1) + p_{10}(b)\cdot V_1(0) + p_{11}(b)\cdot V_1(1)\}\\
&= b
\end{aligned}$$

The optimal strategy f for two periods is given by

$$f = \{(1, f_1), (2, f_2)\} \qquad \text{with} \qquad \begin{aligned} f_1 &= \{(0, f_1(0)), (1, f_1(1))\} = \{(0, b), (1, b)\}\\ f_2 &= \{(0, f_2(0)), (1, f_2(1))\} = \{(0, a), (1, a)\}\end{aligned}$$

Thus, if only one time unit is left to the time horizon the best decision in both states, 0 and 1, is b. When two time units are left, it is best to decide for a in state 0 and b in state 1.

2.8 Technical Infrastructure

The research described in this thesis is supported by the implementation of a number of prototypes. In this section the technical infrastructure is described that was used as a basis for developing these prototypes. The infrastructure consists of two frameworks: the ProM framework and the DECLARE framework.

2.8.1 The ProM Framework

Approximately five years ago, the development of ProM as a framework for *process mining* was initiated [12, 84, 222, 223, 295]. The idea of process mining is to discover, monitor and improve business processes by extracting knowledge from event logs. These event logs contain records of activities that occurred during the execution of cases in the business process in the past. Information on (i) the activity in the process, (ii) the case (i.e. a process instance), (iii) the originator (executing the activity), and (iv) the timestamp is recorded for each log event [84].

By using process mining techniques (see e.g. [19, 24, 31, 34, 83, 294]) one can learn from the recorded behavior of a process. With this knowledge new models can be discovered, the conformance of a model can be determined by checking whether the modeled behavior matches the observed behavior, and existing models can be extended by projecting information extracted from the log onto some initial model [12]. The information derived from the logs can be related to the process perspective (i.e. control-flow of the process), to the organizational perspective (e.g. the originators or performers of activities), or to the case perspective (i.e. case properties such as the value of certain data elements). Moreover, process mining can provide insight in process performance issues such as flow time, execution frequencies or utilization of resources [84].

In the last two years the ProM framework has evolved to more than a process mining tool. It has become a broad and powerful *process analysis* tool supporting all kinds of analyses related to business processes [84]. The architecture of ProM is set up in such a way that it is easy to include newly developed algorithms and implementations without modifying and recompiling the ProM framework. Therefore, it is a truly ‘pluggable’ environment. Each functional part of implementation is called a ‘plugin’ in ProM terms. Currently, there are more than 250 plugins available in the framework and their application is no longer limited to process mining only. For instance, ProM also supports the import, export and conversion of models designed with several well-known modeling languages, such as (Colored) Petri Nets, EPCs, BPEL, YAWL, and WF-nets. Moreover, ProM supports the verification and analysis of process models. The ProM plugins can be grouped into five different categories:

· *Import plugins* - through the import plugins a wide variety of models can be loaded, e.g. ranging from a Petri net to logical formulae.

· *Mining plugins* - the mining plugins deal with the actual process mining of the log, resulting in, for example, the visualization of a process model such as a Petri net or EPC, or in further analysis or conversion. Figure 2.13 shows the results of the application of some of the mining plugins on an example log.

· *Analysis plugins* - by using the analysis plugins a model (possibly obtained via a mining plugin) can be analyzed for specific properties, e.g. the soundness of a Petri net.

· *Conversion plugins* - via the conversion plugins a model can be converted from one modeling language or format to the other, e.g. transforming a Petri net to an EPC.

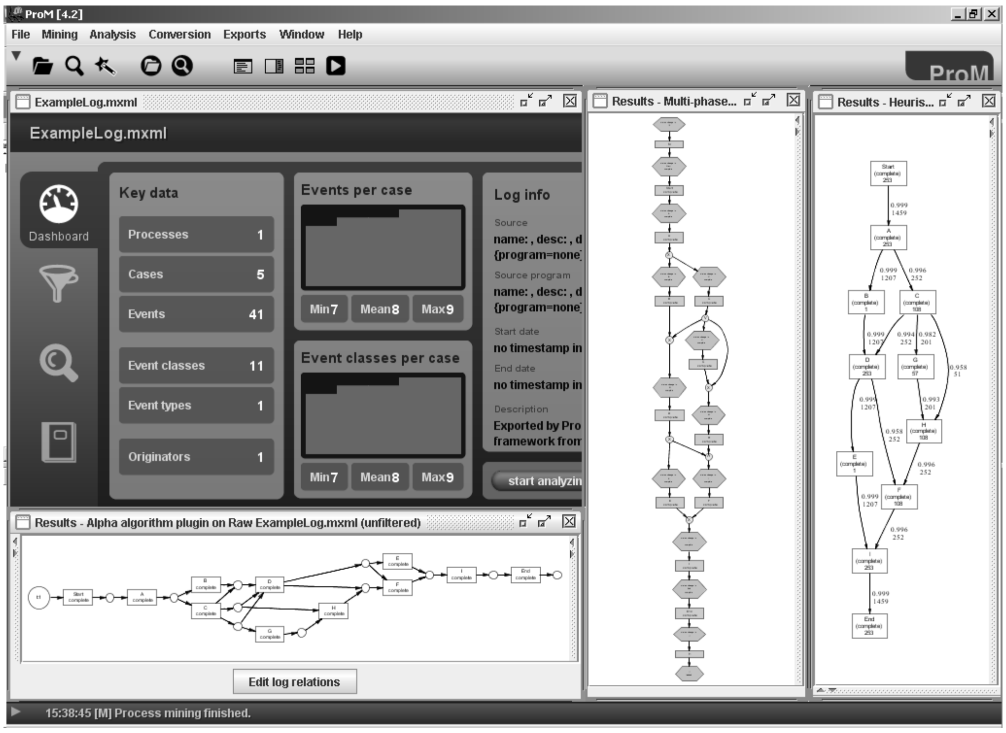


Figure 2.13: *A screenshot of ProM showing the summary of the log loaded and a number of process models resulting from different mining algorithms.*

· *Export plugins* - the export plugins make it possible to write a model to an output file such that it can be loaded by another system, e.g. a YAWL model can be exported to a file that can be loaded by the YAWL editor or by the YAWL engine.

The ProM framework is the basis for the implementation of the prototypes that support the ideas presented in this thesis. The following chapters contain references to ProM and its plugins and further explanation is added where results presented in this thesis are supported by ProM. In Chapter 4 the representation of a PDM in ProM is explained. Chapter 5 presents a number of conversion plugins from a workflow product structure to a process model. One of the analysis plugins, the *recommendation service* [293], is of particular interest for the research described in Chapter 6. Finally, the business process metrics presented in Chapter 7 are implemented as analysis plugins in ProM.

2.8.2 The DECLARE Framework

DECLARE is a special kind of workflow management system [214, 215, 216, 217]. Because DECLARE is based on a *declarative* approach to business process modeling and execution, it provides more flexibility than the conventional workflow management systems. In traditional workflow management systems, rigid modeling languages (e.g. BPMN, EPCs, BPEL, Petri nets) define a process model as a detailed specification of a step-by-step procedure that should be followed during execution [216]. This *imperative* approach leads to highly-structured, and

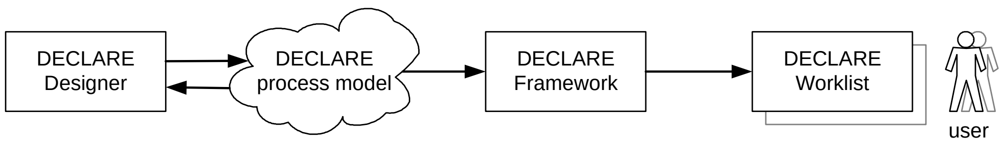


Figure 2.14: *The DECLARE system.*

possibly over-specified, processes. Unlike these conventional systems, DECLARE is developed as a constraint-based system and uses a declarative language based on temporal logic [216]. A process model in DECLARE merely restricts the behavior that is prohibited in the execution of a process by adding constraints to the process model.

Even though it is a declarative system, DECLARE offers most features that traditional workflow management systems have: model development, model verification, automated model execution, changing a model at run-time, analysis of already executed processes, and process decomposition. In addition, DECLARE can be used to overcome the paradox between user support and flexibility by providing the user with history based recommendations during process execution [216].

The DECLARE system consists of three components that are typical to every workflow management system:

· *Designer*: a modeling tool, that is used for system settings and process model development.

· *Framework*: a tool for process enactment, i.e. the process model execution engine of the system.

· *Worklist*: a process execution tool, which is the interface between the user and the framework for executing process instances.

Together, these tools provide build-time and run-time support for a declarative approach to workflow management. In this thesis we will not use the DECLARE constraint-based modeling language, nor the DECLARE Designer tool, but we will illustrate the research described in Chapter 6 by a prototype using the DECLARE Framework and DECLARE Worklist for interactions with the user.

2.9 Summary

In this chapter the preliminaries to the research described in this thesis have been presented. In particular a brief introduction has been given of first-order logic, set theory, graphs, process modeling languages, Markov decision processes, the ProM framework, and the DECLARE framework, some knowledge of which is necessary in reading the foolowing chapters.

Chapter 3

Workflow Product Structures

In this chapter, the main concept of product based workflow design and support, the *Product Data Model (PDM)*, is introduced. The use of a PDM for the design of workflow processes is inspired by similar approaches in field of production and manufacturing. Section 3.1 elaborates on the use of product specifications for the manufacturing process. Next, in Section 3.2, the concept of a PDM is introduced in more detail. An informal definition is given, illustrated by some examples. Section 3.3 deals with the design of workflow processes based on a PDM. This approach is called *Product Based Workflow Design (PBWD)* (see Section 1.4). The application of PBWD in practice has only been a manual process and no tool support is available for this design process. Therefore, Section 3.4 focuses on an evaluation of current workflow technology to support PBWD. Finally, similar data- and product-driven approaches to workflow process design are discussed in Section 3.5, followed by a summary in Section 3.6.

3.1 Product Specifications for Process Design

The use of *product descriptions to structure processes* is common in manufacturing. Based on the notion of a parts list, the so-called *Bill of Material (BoM)* was introduced in the 1970s by Joseph Orlicky [201]. A BoM captures the structure of the product to be produced in the manufacturing process. It describes the dependencies and relationships between the elements of the parts list. The BoM is usually created as a part of the design process of a product and it is used by manufacturing engineers to determine which components should be purchased and which components should be manufactured. Many methods for production planning and inventory control use the BoM in combination with a master production schedule to determine the components for which purchase requisitions and production orders must be released, e.g. Material Requirements Planning (MRP-I) [43, 201, 202] and Manufacturing Resources Planning (MRP-II) [43, 289]. The BoM is also used in accounting to cost or price the manufactured product [99, 170].

In this section, the concept of a BoM is introduced in more detail and the similarities and differences between manufacturing processes and workflow processes are discussed. Similarities between these two kinds of processes justify the application of a product-driven approach to workflow process design, whereas the differences explain why it is hard to straightforwardly adopt the notions from the manufacturing area.

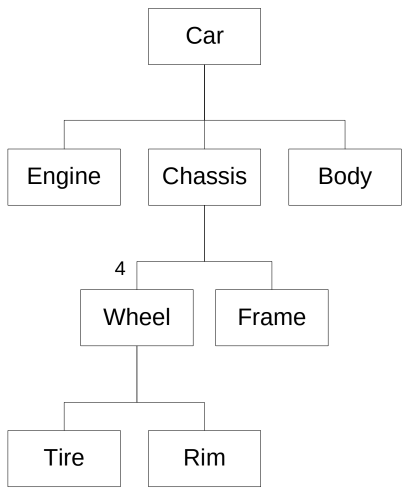


Figure 3.1: *A simple BoM of a car.*

3.1.1 The Bill of Material

A BoM specifies the 'ingredients' needed to make (e.g. assemble, mix or produce) a product. It describes step-by-step how a physical product, such as a car, a bike, or a computer, is manufactured from raw materials until the finished product. The BoM is represented as a tree-like structure with the end product of the process as the root of the tree and raw materials as leafs. Note that raw materials can also be semi-finished products that are purchased from other parties. The intermediate nodes in the tree correspond to sub-assemblies.

Consider for instance the simple BoM of a car in Figure 3.1. The car (end product) is built from an engine, a chassis and a body. The chassis consists of four wheels and a frame. Finally, a wheel is composed of a tire and a rim. The raw materials in this case are all purchased products and can have a BoM of their own: the engine, the body, the tires, the rims, and the frame. The assembly of parts to new parts involves activities such as welding, screwing, and cutting.

The simple form of a BoM can be extended with information on, for instance, manufacturing steps, tooling and resources [43, 170, 303]. In combination with the BoM, a *Bill-of-Processes (BoP)* may be used, which describes the manufacturing steps needed for each of the components in the BoM [43]. Also, many variants of the simple form of a BoM exist [139], e.g. the modular BoM [284], the percentage BoM [170], the super BoM, the variant BoM [283, 284], and the generic BoM [283, 284]. However, for our purposes, it suffices to regard a BoM as a decomposition of the end product into several parts that are related to each other.

3.1.2 Manufacturing vs Workflow Processes

Workflow processes encountered in e.g. banking, insurance and government also generate products. However, the products of these workflow processes are *information items* instead of physical parts, e.g. tax declarations, traffic fines, and decisions on insurance claims. The production of *workflow products* corresponds to the processing of cases. From a logistical point of view, many similarities exist between workflow processes and manufacturing processes [5, 220, 231]:

· Managing the process focuses on the routing of work and the allocation of work to resources.

· There is a common notion of a process as a set of tasks that have to be executed in an order that is fixed to some level and incorporates some degree of flexibility as well.

· The performance of both types of processes is measured in highly similar ways with indicators such as throughput time, waiting time, client satisfaction and utilization.

· Concepts to affect the performance of a process that originate from manufacturing are frequently seen to be applied in workflows as well, e.g. Earliest Due Date (EDD), First In First Out (FIFO).

However, also some subtle differences between workflow processes and manufacturing processes exist [5, 231]:

· In a manufacturing system, the products are physical objects and the principal resources are machines, robots, humans, conveyor belts and trucks. In a workflow process, the products are informational and most of the resources are human.

· In a workflow process, making a copy of the product is straightforward and inexpensive: it is relatively easy to copy a piece of information (especially if it is in electronic form). By contrast, copying the product from a manufacturing process is more difficult and implies executing the whole process again.

· No significant limitations with respect to the in-process inventory exists in a workflow process. Informational products do not require much space and are easy to access, especially if they are stored in a database. In manufacturing processes the products are usually stored in large warehouses and may be subject to special requirements (e.g. perishable products, chemicals).

· A workflow process imposes fewer requirements with respect to the order in which tasks are executed. Human resources are flexible in comparison to machines, and few technical constraints exist regarding the lay-out of the workflow process. In a manufacturing environment machines usually cannot be moved around easily and it is often costly to interrupt and restart a working machine.

· In a manufacturing process clear quality requirements can be formulated and verified for each product. However, the quality of a workflow product is difficult to measure. Criteria

	Manufacturing process	Workflow process
Product	Physical object	Informational
Resources	Machines	People
Inventory	Warehouse (limited capacity)	Digital (almost unlimited capacity)
Quality	Well-defined	Difficult to measure
Quality checks	Random test	Everything is checked at certain points in the process
Routing	Known in advance	Often depends on the outcome of certain activities
Kind of production	Often production-to-stock	Make-to-order

Table 3.1: *Differences between manufacturing and workflow processes.*

to assess the quality of an informational product are usually less explicit than those in a manufacturing environment.

· The quality of workflow products may vary. A manufacturer of goods usually has a minimum number of components that any product should consists of. However, in an administrative process it may be attractive to skip certain checks in producing the informational product to reduce workload or throughput time, e.g. the tax authority may skip checks for persons that have earned a good reputation over the previous years.

· The transportation of parts in the supply chain of a manufacturing process usually forms a substantial part of the total lead time of the process. On the contrary, the transportation of electronic data in a workflow process is instantaneous.

· In a manufacturing process products are often produced to stock[1]. Each product is identical to other products from the same production line. In a workflow process, production to stock is seldom possible. Every workflow case is unique and therefore it is difficult to produce in advance, e.g. an insurance claim can not be processed before it is filed by the client and the client has provided specific details for the case. Workflow processes are make-to-order processes (see also Section 1.3).

Despite these fundamental differences, the workflow domain can benefit from the adoption of the principles to structure manufacturing processes based on product structures. The next section explains how the notion of a BoM can be translated to the workflow domain.

3.2 The Product Data Model

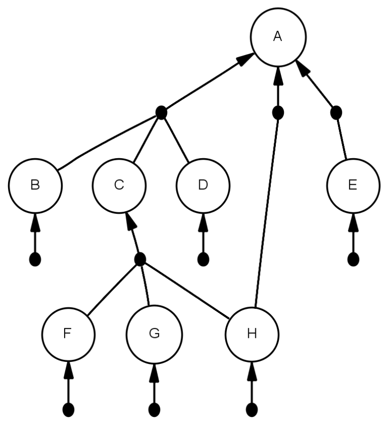
This section elaborates on the product description of a workflow product, the Product Data Model (PDM). First, the notion of a PDM is introduced in Section 3.2.1. Then, a number of running examples are presented, which are used throughout this thesis for explanation. Finally, the differences between a BoM and a PDM are described in Section 3.2.3.

3.2.1 Introduction

The product of a workflow process is an *informational* product, e.g. the decision on an insurance claim, the allocation of a subsidy or the approval of a loan. Based on the input data provided by the client or retrieved from other systems, the process constructs the end product step-by-step. In each step new information is produced based on the specific data present for the case. Below, it is explained how the structure of a workflow product is modeled by a PDM. The PDM is described by a tree-like structure similar to a BoM. However, the building blocks of a PDM are not the physical parts that have to be assembled, but the data elements (e.g. name, date of birth, gross salary) that are used to produce new information.

A small example of a PDM is shown in Figure 3.2. It describes the calculation of the maximum amount of mortgage a client is able to borrow from a bank. The figure shows that the maximum mortgage (element A in Figure 3.2) is dependent either on a previous mortgage offer (E), or on the registration in the central credit register (H), or on the combination of the percentage of interest (B), the annual budget to be spent on the mortgage (C), and the term of the mortgage (D). The annual budget (C) is determined from the gross income of the client per year (G), the credit registration (H), and the percentage of the income the client is allowed to

[1]There are four ways to describe the relation of the manufacturing environment with the customer in a logistic chain: (i) production-to-stock, (ii) assemble-to-order, (iii) make-to-order, and (iv) engineer-to-order (see [43, 90]).



A	Maximum mortgage
B	Percentage of interest
C	Annual budget to be spent on the mortgage
D	Term of mortgage
E	Previous offer (within the period of validity of the offer)
F	Percentage of income to be spent on the mortgage
G	Gross income per year
H	Credit registration

Figure 3.2: *The PDM of the maximum mortgage calculation.*

spend on paying the mortgage (F). The calculation is based on (i) information provided by the client (e.g. the gross income per year, or the term of the mortgage), (ii) on company specific policies (e.g. the interest percentage for the mortgage, or the percentage of the gross salary the client is allowed to spend on housing), and (iii) on other systems (e.g. the credit register).

The bank has three alternative ways to decide on the maximum mortgage. First of all, if the client has a negative registration in the central register for credits (e.g. the client has a history of non-payment), the bank may directly reject this person for a mortgage (leading to a maximum amount of zero). The central credit register keeps track of all loans a person has and helps providers of loans in their assessment of the creditworthiness of such a person.

Secondly, if the client has previously requested a mortgage offer and the term of validity of this offer is not expired yet, this may determine the amount of mortgage. Typically, the percentage of interest changes over time and a mortgage offer is valid for some months. In case the interest has increased since the previous offer, the valid offer might be better than a new one which is based on the higher interest percentage.

Finally, if the credit register shows a positive credit history, the bank needs more information on the client's situation (e.g. gross income, type of mortgage) in order to decide on the maximum mortgage. To a certain extent, a bank defines its own internal rules and policies to decide on how much risk a mortgage applicant is to their business if they allow a mortgage. Therefore, each bank uses a percentage of the gross income of the client to calculate how much money the client is allowed to spend on the house. With this rule a bank ensures that the client can afford the cost of food and recurring expenses and that the probability is high that he will meet the monthly payment liabilities with the bank. However, this percentage is not fixed and can vary based on the bank's current situation as well as the client's.

Data Elements and Operations

The information that is processed in a workflow process is described by so-called *data elements* in a PDM as is shown in Figure 3.2. For each specific case a data element can have a different *value*. Data elements are depicted as circles in the PDM. The actions that are taken on the data element values are called *operations* and are represented by arcs. Each operation can have zero or more *input data elements* and produces exactly one *output data element*. The arcs are ‘knotted’ together when a value for all data elements is needed to execute the particular operation. Compare for instance the arcs from B, C, and D leading to A on the one hand, and the arc leading from E to A on the other hand in Figure 3.2. In the latter case only one data element value is needed to determine the outcome of the process, while in the case of B, C, and D all three data element values are needed to produce A. An operation is *executable* when a value for all of its input elements is available.

Several operations can have the *same* output element while having a different set of input elements. Such a situation represents *alternative ways* to produce a value for that output element. For example, a value for the end product A in Figure 3.2, can be determined in three alternative ways: (i) based on a value for E, (ii) based on a value for H, and (iii) based on values for B, C, and D.

The top element of the PDM, i.e. the end product, is called the *root* of the PDM. The *leaf elements* are the elements that are provided as inputs to the process. They are produced by operations with no input elements (e.g. the operations with output elements B, D, E, F, G, and H). The operations producing values for the leaf elements are denoted as *leaf operations* or input operations. The leaf element values for a particular instance can be retrieved from three parties: (i) internally from the same institution, (ii) from the client, and (iii) from external parties (e.g. the central credit registration). The values can be obtained from databases or other information systems (e.g. credit register), laws and regulations, company policies (e.g. the percentage of the income to be spent on the mortgage), paper or digital forms (e.g. gross income per year), etc.

An operation can be identified by a tuple consisting of the output element and a set of input elements, e.g. $(A, \{B, C, D\})$ for the operation producing a value for A based on data elements B, C, and D. Throughout this thesis operations are also referred to by identifiers, such as $Op01$.

Operation Attributes

The actual action that is taken based on the values of the input data elements of an operation can be e.g. an automatic calculation, an assessment by a human, or a rule-based decision to determine a new data element value. These operations may consume time and money before they are completed. In general, an operation can have a number of attributes associated to it that describe the characteristics of the operation in more detail:

· *Execution cost*: the cost associated with executing the operation, given by a probability distribution and its parameters.

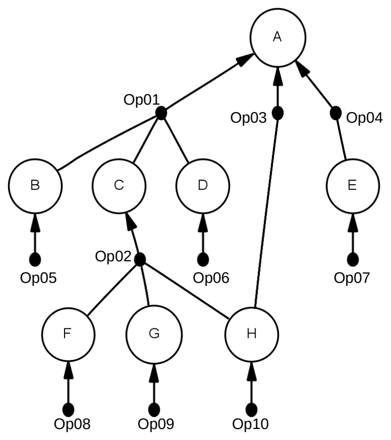
· *Processing time*: the time that is needed to complete the operation, given by a probability distribution and its parameters.

· *Execution conditions*: conditions on the value of the input data elements that restrict the execution of the operation. If the condition is not satisfied, the operation is not executable even if a value for all of its input data elements is available.

· *Failure probability*: the probability that the operation is not performed successfully, i.e. the probability that the output data element is not produced.

· *Resource class*: the resource class or role that is required to perform the operation.

Most of these attributes are optional and not always all are available for a specific process. The theory presented in this thesis assumes certain attributes to be present. For instance, in Chapter 6, decision strategies are presented based on the cost or processing time of an operation.



A	Maximum mortgage
B	Percentage of interest
C	Annual budget to be spent on the mortgage
D	Term of mortgage
E	Previous offer (within the period of validity of the offer)
F	Percentage of income to be spent on the mortgage
G	Gross income per year
H	Credit registration

Figure 3.3: *The PDM of the maximum mortgage calculation including identifiers for the operations.*

We use the mortgage example (see Figure 3.2) again to elaborate on these attributes. Considering the cost of executing an operation, for instance, the bank has to pay for receiving a copy of the client's registration in the credit register. Thus, the cost for this operation are higher than the cost for e.g. determining the gross income, since the information on income is provided by the client. The same holds for the processing time of these operations. Since an external party is involved, it may take more time to retrieve the credit registration than to ask the client for his gross income. For the same reason, the probability of not receiving the client's credit registration from the agency may be higher than the probability of not being able to determine the term of the mortgage. Finally, determining the maximum amount of mortgage based on a negative credit registration is an example of an operation with an execution condition. If the input element does not satisfy the requirement of having a value equal to 'negative' the operation is not executable even though a value for the input element is present.

3.2.2 Running Examples

Throughout this thesis some running examples are used to illustrate the theory presented. This section introduces these running examples. In the remainder, references to these examples are added when appropriate.

Example 1: Mortgage

The first example is the mortgage example which was used to informally explain the notion of a PDM in Section 3.2.1. It describes a simplified version of a mortgage calculation. The PDM has 8 data elements and 10 operations (6 of which are leaf operations i.e. having no input data elements). We have also added (fictive) attributes to the operations of the PDM which will be used in later explanations. These operation attributes are summarized in Table 3.2. For instance, the execution cost for $Op10$ are constant since the the credit registration has a fixed tariff for providing information. The processing time of operation $Op10$ is rather high ($\mu = 10.0$) in comparison with the other operations because it may take some time before the information from the credit registration is received. Moreover, there is a high variance in the processing time, indicated by a large σ. For $Op07$ the execution cost and processing time are

	Output	Input	Cost	Time	Prob.	Conditions
Op01	A	B, C, D	$Normal(\mu = 5.0, \sigma = 0.5)$	$Normal(\mu = 1.0, \sigma = 0.13)$	0.05	
Op02	C	F, G, H	$Normal(\mu = 5.0, \sigma = 0.45)$	$Normal(\mu = 4.0, \sigma = 0.4)$	0.05	
Op03	A	H	$Normal(\mu = 9.0, \sigma = 0.9)$	$Normal(\mu = 3.0, \sigma = 0.1)$	0.05	H = ‘negative’
Op04	A	E	$Normal(\mu = 2.0, \sigma = 0.06)$	$Normal(\mu = 2.0, \sigma = 0.15)$	0.00	
Op05	B	-	$Constant(0.0)$	$Constant(0.0)$	0.00	
Op06	D	-	$Constant(0.0)$	$Constant(0.0)$	0.00	
Op07	E	-	$Normal(\mu = 1.0, \sigma = 0.12)$	$Normal(\mu = 1.0, \sigma = 0.03)$	0.50	
Op08	F	-	$Constant(0.0)$	$Constant(0.0)$	0.00	
Op09	G	-	$Constant(0.0)$	$Normal(\mu = 2.0, \sigma = 0.7)$	0.00	
Op10	H	-	$Constant(3.0)$	$Normal(\mu = 10.0, \sigma = 2.0)$	0.15	

Table 3.2: *Operations and their attributes for the mortgage example.*

considerably smaller and vary less. However, the failure probability is rather high (i.e. 50%) because many customers do not have a valid previous offer.

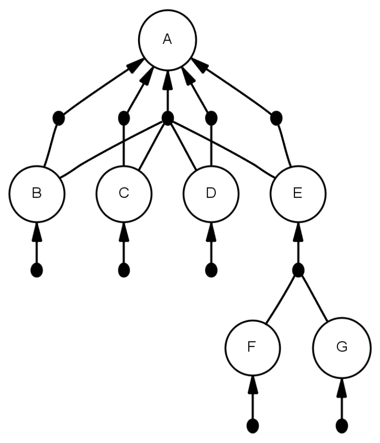
Example 2: Naturalization

The second example deals with the process of naturalization of foreigners to obtain Dutch citizenship in the Netherlands. Such a citizenship is only allowed to persons that meet certain requirements. These requirements are declared by law (see Figure 3.4). Based on the Dutch statute law on Dutch citizenship [241] it is straightforward to derive the PDM for this process as shown in Figure 3.5. For instance, the first requirement in the Law (art 8.1.a) is reflected in the PDM as data element D. The last requirement in the Law has two sub requirements that are translated to data elements F and G in the PDM. In total the PDM has 7 data elements and 11 operations. Note the four requirements at the top of the PDM. If one of these requirements is not met, this may directly determine the outcome of the process. If all of them are met, their combination also leads to the outcome of the process. Thus, there are five operations with the end product as output element. For this example no other operation attributes are available than the execution conditions (see Table 3.3).

Statute law on Dutch citizenship, art. 8

1. For granting of the Dutch citizenship is only qualified the applicant
a. who is of age,
b. against whose permanent stay in the Netherlands, Netherlands Antilles or Aruba no objections exist,
c. who has had admittance and main residence in the Netherlands, Netherlands Antilles or Aruba, for at least five years immediately preceding the request, and
d. who can be seen as adapted to the Dutch, Dutch-Antillean or Aruban society based on the fact that he has a sufficient level of knowledge of the Dutch language [...] as well as sufficient knowledge about the Dutch constitution and society [...].

Figure 3.4: *Fragment of the Dutch statute law on Dutch citizenship [241] (This is a non-legal translation to English, only used for the purpose of illustration).*



A	Applicant is entitled to Dutch citizenship
B	Objections exist against permanent stay of applicant
C	Applicant has admittance and main residence in the Netherlands, Netherlands Antilles or Aruba for at least five years immediately preceding the request
D	Age of applicant
E	Applicant can be seen as naturalized
F	Knowledge of the Dutch language
G	Knowledge of the Dutch constitution and society

Figure 3.5: *The PDM of the decision on awarding Dutch citizenship.*

	Output	Input	Conditions
Op01	A	B	B = ‘no’
Op02	A	C	C = ‘no’
Op03	A	D	D = ‘no’
Op04	A	E	E = ‘no’
Op05	A	B, C, D, E	
Op06	B	-	
Op07	C	-	
Op08	D	-	
Op09	E	F, G	
Op10	F	-	
Op11	G	-	

Table 3.3: *Operations and their attributes for the naturalization example.*

Example 3: Social Benefits

The third example describes the process of awarding unemployment benefits in the Netherlands. When a person becomes unemployed, he may be entitled to an unemployment benefit, which is part of the social security system in the Netherlands. An agency (the Dutch UWV) deals with the assessment of persons that have become unemployed and checks whether these persons meet all requirements to receive an unemployment benefit. The main regulations with regard to the decision to award unemployment benefits are laid down in the Dutch Unemployment Law [296]. The agency also maintains operational interpretations of this law in handbooks. Furthermore, a detailed administration is kept of reasons for denying unemployment benefits to individual cases, as well as other statistical figures on the operations of the agency.

Typical factors that are taken into account in the decision on awarding a unemployment benefit to an applicant are the reason for this person to have become unemployed, the length of the period that the previous job was held, and the coverage regulations. Figure 3.6 shows the PDM of this social benefits process and Table 3.4 contains a description of the data elements. The PDM contains 45 data elements and 50 operations. The details on the operation attributes can be found in Table 3.5.

The unemployment benefits example is derived from an actual workflow process redesign case in industry. More details of this example can be found in earlier descriptions of this case in [231, 233].

The three running examples are used throughout this thesis to clarify concepts and ideas.

3.2.3 Similarities and Differences between the BoM and the PDM

In Section 3.1, we identified a number of similarities and differences between manufacturing and workflow processes. Now the notion of a PDM is clarified we can also compare the BoM and the PDM. Although the BoM and the PDM are quite similar, it may have become clear from the preceding descriptions and examples that there are some substantial differences.

First of all, in a PDM information may be *re-used*. When a data element has been determined it is present for as long as the case lasts and can be used in several operations. In a manufacturing environment this would not be possible, since a part can only be assembled

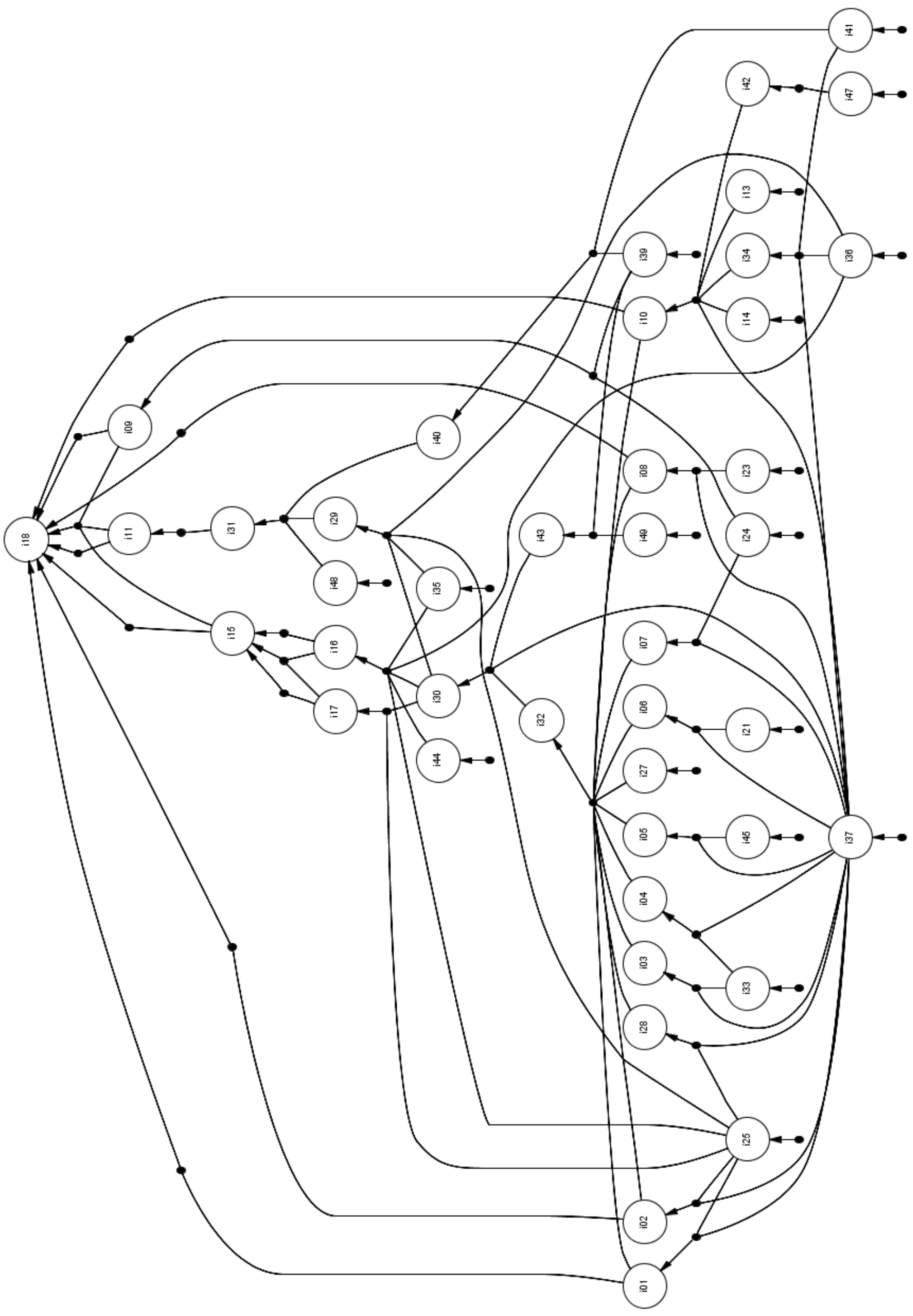


Figure 3.6: *The PDM of the social benefits process. A description of the data elements and the operations is given in tables 3.4 and 3.5.*

Date element	Description
i01	Period in which claimant receives illness benefits
i02	Period in which claimant receives combined social benefits
i03	Period claimant lives/resides outside the Netherlands
i04	Period in which claimant does not rightfully live in the Netherlands
i05	Period in which claimant is detained/imprisoned
i06	Period in which claimant is 65 years or older
i07	Period in which claimant has legal scruples against insurance
i08	Period in which claimant enjoys holiday
i09	Period in which claimant is an employee
i10	Period in which claimant is unemployed
i11	Claimant satisfies refer requirement
i13	Date from which claimant lost the right for payment
i14	Date from which claimant is available to accept labor
i15	Claimant satisfies labor history requirement
i16	Claimants satisfies 4-out-of-5-years requirement
i17	Claim is directly following labor disablement benefits
i18	Claimant is entitled to (pay-related) unemployment benefits
i21	Birth date of claimant
i23	Claimant's holiday administration
i24	Registration of unemployment insurance
i25	Registration of social benefits
i27	Claimants unemployment is caused by strike/work stoppage
i28	Period in which claimant receives re-integration benefits
i29	Refer period for claimant
i30	First day of unemployment of claimant
i31	Number of weeks claimant worked in refer period
i32	First week of unemployment of claimant
i33	Registration of housing
i34	Average number of labor hours per week of claimant
i35	First day of labor history for claimant
i36	Day status survey of claimant's labor history
i37	Loss pattern of labor hours of claimant
i38	Care data on claimant
i39	Employment function of which the claimant has become unemployed
i40	Employment functions that have been followed up by the employment function of which the claimant has become unemployed
i41	Earlier employment function of the claimant
i42	Approved labor courses for unemployed
i43	Common first labor day for claimant
i44	List of claimant's annual worked days
i45	Register of convictions
i47	Claimant's courses that precede or follow on the loss of labor hours
i48	Weeks in refer period already taken into account
i49	Labor pattern of claimant
i50	Register of special classes of employment functions
i51	Claimant has taken care of under-age children

Table 3.4: *Meaning of data elements of the PDM of the unemployment benefits example.*

ID	Output	Input	Cost	Failure probability	Execution condition
Op01	i01	i25, i27	0.0	0.0	
Op02	i02	i25, i37	0.0	0.0	
Op03	i03	i33, i37	0.0	0.0	
Op04	i04	i33, i37	0.0	0.0	
Op05	i05	i37, i45	0.0	0.0	
Op06	i06	i21, i37	0.0	0.0	
Op07	i07	i24, i37	0.0	0.0	
Op08	i08	i23, i37	0.0	0.0	
Op09	i09	i24, i39	0.0	0.0	
Op10	i10	i13, i14, i34, i37, i42	0.0	0.0	
Op11	i11	i31	0.6	0.0	
Op12	i15	i16	0.0	0.003	i16 = ‘true’
Op13	i15	i17	0.0	0.997	i17 = ‘true’
Op14	i15	i16, i17	0.0	0.0	
Op15	i16	i25, i30, i35, i36, i44	5.61	0.0	
Op16	i17	i25, i30	0.0	0.0	
Op17	i18	i01	0.0	0.991	i37 in i01
Op18	i18	i02	0.0	0.987	i37 in i02
Op19	i18	i08	0.0	0.984	i37 in i08
Op20	i18	i09	0.0	0.998	i9 = ‘false’
Op21	i18	i10	0.0	0.932	i10 not defined
Op22	i18	i11	0.0	0.981	i11 = ‘false’
Op23	i18	i15	0.0	0.79	i15 = ‘false’
Op24	i18	i09, i11, i15	0.0	0.0	
Op25	i28	i25, i37	0.0	0.0	
Op26	i29	i25, i30, i35, i36	0.0	0.0	
Op27	i30	i32, i37, i43	0.0	0.0	
Op28	i31	i29, i40, i48	0.0	0.0	
Op29	i32	i01, i02, i03, i04, i05, i06, i07, i08, i10, i27, i28	0.0	0.0	
Op30	i34	i36, i37, i41	4.2	0.0	
Op31	i40	i39, i41	0.3	0.0	
Op32	i42	i47	0.3	0.0	
Op33	i43	i39, i49	0.6	0.0	
Op34	i13	-	0.08	0.0	
Op35	i14	-	0.08	0.0	
Op36	i21	-	0	0.0	
Op37	i23	-	0.67	0.0	
Op38	i24	-	0	0.0	
Op39	i25	-	0	0.0	
Op40	i27	-	0.08	0.0	
Op41	i33	-	0	0.0	
Op42	i35	-	0	0.0	
Op43	i36	-	1.0	0.0	
Op44	i37	-	1.67	0.0	
Op45	i39	-	0.17	0.0	
Op46	i41	-	0	0.0	
Op47	i44	-	0	0.0	
Op48	i45	-	0	0.0	
Op49	i47	-	0.33	0.0	
Op50	i48	-	0	0.0	
Op51	i49	-	0	0.0	

Table 3.5: *Operation attributes for the unemployment benefits example.*

once, e.g. it is not possible to use a screw in two different locations.

Secondly, in workflow processes *alternative paths* to produce the end product exist. Decisions are made based on available information. The same output product can be achieved in different ways. Normally, in a manufacturing process fewer alternative ways of producing the end product exist because of physical constraints. The alternative paths in a BoM usually are related to producing different variants of a product (cf. the variant-BoM mentioned in Section 3.1.1).

3.3 Product-Based Workflow Design

Now we have introduced the notion of a PDM, we briefly look at how such a PDM is constructed and used as a basis for PBWD in practice. The research described further in this thesis assumes, as a starting point for actual applications, that the PDM of a process is available and does not focus on how to obtain such a PDM.

3.3.1 Discovery and Construction of a PDM

In previous applications of PBWD, the construction of a PDM was done manually. Information on data elements and their dependencies was retrieved from rules and regulations, work instructions, forms, information systems and the knowledge on the workflow product that is present with stakeholders. For instance, in the naturalization example (Section 3.2.2), the PDM was completely derived from the information that was found in the Dutch law. Also, for the example of social benefits the Dutch Unemployment Law was used to construct the PDM, complemented with information from handbooks, and historical information on previous cases.

Typically, PDMs for actual cases are much larger than the PDM for the mortgage or naturalization example. For instance, the PDM of the social benefits example, which is based on an actual case from industry, already is comprised of 45 data elements. Other practical examples contain even larger PDMs: e.g. 580 data elements in the credit application process within a Dutch bank [229, 231]. One can imagine that the manual construction of such large product structures is time-consuming and that mistakes are easily made in this process. Especially when the PDM is constructed by several analysts at the same time it is difficult to make a complete and consistent design. For example, the PDM of the credit application process was mapped by 5 persons, each one of them focussing on a different part of the product while also keeping track of the overall picture. The analysts faced issues with respect to the granularity of the model and redundant definitions of the same data element[2]. This example shows that constructing a PDM is non-trivial, although the focus on the product provides a more objective view on the process.

In this thesis we do not focus on supporting the construction of a PDM. We assume that a PDM is available and that it can be obtained in two ways. First of all, the PDM may be constructed *manually* by analysts who unravel the product as described above by looking at e.g. rules, regulations, work instruction, forms. Secondly, data elements and their dependencies may be retrieved from *data logs* of systems that support the current situation of the process via mining techniques. These data logs contain ordered sequences of data field changes for every executed instance of the process. This approach is similar to the discovery of activity dependencies from event logs with process mining techniques [20, 21, 22, 222]. A first step towards retrieving knowledge from data logs is the mining of activity clusters described in

[2]Note that ontologies (e.g. [272, 211]) can be used to overcome the problem of redundant definitions.

[118]. In this approach recurring patterns of data modifications are extracted from the log. Such a cluster indicates a common activity on a higher level of abstraction.

Although the discovery of data elements via the mining of data logs has not been used in practice yet, we think it may provide an overview of all data that is used in a process. Therefore, it may overcome the problems of granularity and completeness in the manual approach.

3.3.2 Design of a Workflow Process Model based on a PDM

The PDM can be used as a starting point to develop support for the workflow process in the form of a process model. In previous practical applications, the design of process models based on a PDM has also been a manual process. The operations and their data elements are *grouped into activities* that somehow form a logical piece of work. Based on the data dependencies from the PDM, a process model is built of these clusters.

At the moment, no design guidelines or tool support are available for this process and therefore it is usually done in an unstructured way. For instance, the redesign for the credit application process [229, 231] was achieved during brain storm sessions in which process analists physically grouped data elements onto post-it notes, discussed their design and changed it a number of times. Designing the first version of the process model took two months. Below, the manual design process and the resulting process model for the example of the social benefits case are briefly described as an illustration.

Process Design for the Social Benefits Example

The example of the social benefits process (Section 3.2.2) is based on an actual redesign project in practice in which PBWD has been applied. At the start of the redesign project, a number of design objectives were formulated which should be met by the final process model: (i) the expected average effort in terms of human labor hours for the process should be reduced, (ii) the case worker principle should be applied, i.e. for each particular case the same resource should execute all steps within the process, and (iii) the number of contacts with the client should be minimized.

The final design of the process model was achieved in five steps. These five steps start at the bottom of the PDM, i.e. with retrieving a value for the leaf elements either from internal systems or from the client, and determine which operations are executable and in which order they should be executed. The process model resulting from these design steps is shown in Figure 3.8. The activities in the process model correspond to parts of the PDM as indicated in Figure 3.7. Each activity is a group of operations and its corresponding data elements. Activity t_G, for instance, is a grouping of operations $Op30$, $Op32$, and $Op10$ and data elements $i10$, $i13$, $i14$, $i34$, $i36$, $i37$, $i41$, $i42$, and $i47$. A more detailed description of the steps taken to derive this process model from the PDM can be found in [231, 233].

The manual design of a process model based on a PDM is a time-consuming and error-prone process. Moreover, reproducing this process and its results is only possible if process designers document their design choices properly. In practice, often less effort is put into reporting on the design process (e.g. the design process with post-it notes). Therefore, it is difficult to justify the design choices and convince stake holders of the power or usefulness of the resulting design.

In this thesis, we solve the problem of designing a process model based on a PDM and focus on automatic support for PBWD. In Chapter 5, we present a number of algorithms to automatically derive a process model based on a PDM. But before we elaborate on these algorithms,

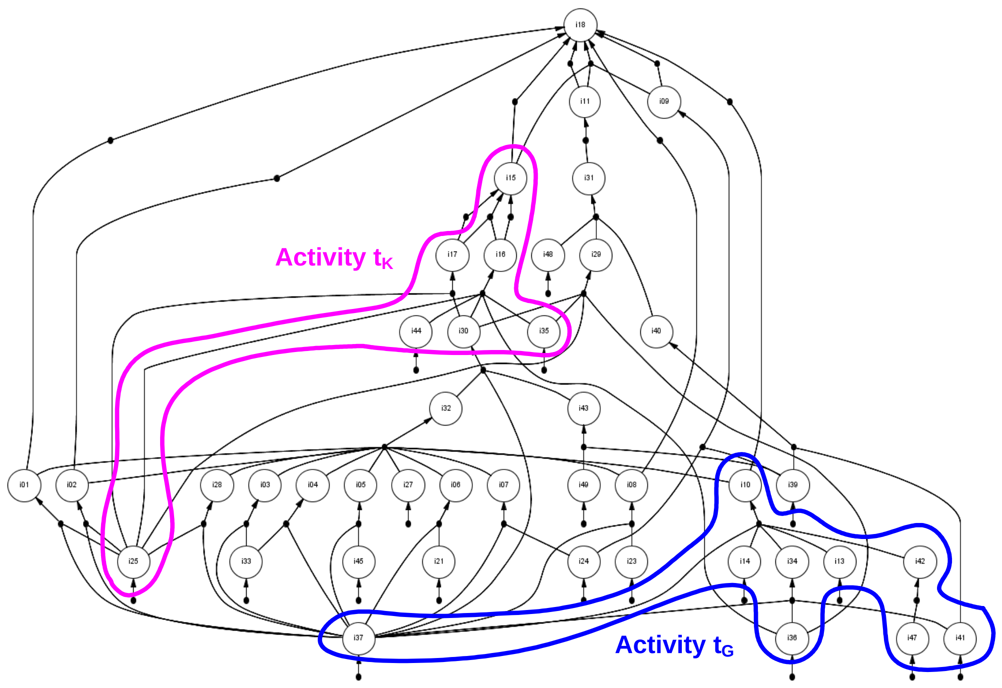


Figure 3.7: *Activities t_G and t_K from the process design are indicated in the PDM of the unemployment benefits example. Activity t_G is a group of three operations and activity t_K consists of two operations.*

it is important to determine whether current workflow management tools provide support for designing process models based on a PDM. The next section discusses the assessment of two contemporary workflow management systems with respect to their ability to support PBWD.

3.4 Tool Evaluation for PBWD

As was explained above, designing a process model based on a PDM is a manual process in which data elements and operations from the PDM are grouped together to form activities. Since this may be a time-consuming process, we have investigated to which degree current workflow technology is able to support the design of workflow process models based on a PDM. From all workflow management systems available in the market, case handling systems fit best with our investigation because they are *data-driven* (see Section 1.3.2) and therefore potentially provide good support for the product-driven design method. Therefore, we specifically focus on case handling systems in our assessment of current workflow technology for PBWD.

The objectives of our assessment can be formulated as follows: (i) to determine whether the concepts of PBWD can be translated to the concepts of current case handling systems, and (ii) to establish to what extent build-time features of case handling systems support the design of workflow models based on PBWD. To investigate these objectives, we list some features that in our view should be present in a system that supports the product-based design approach in a proper way:

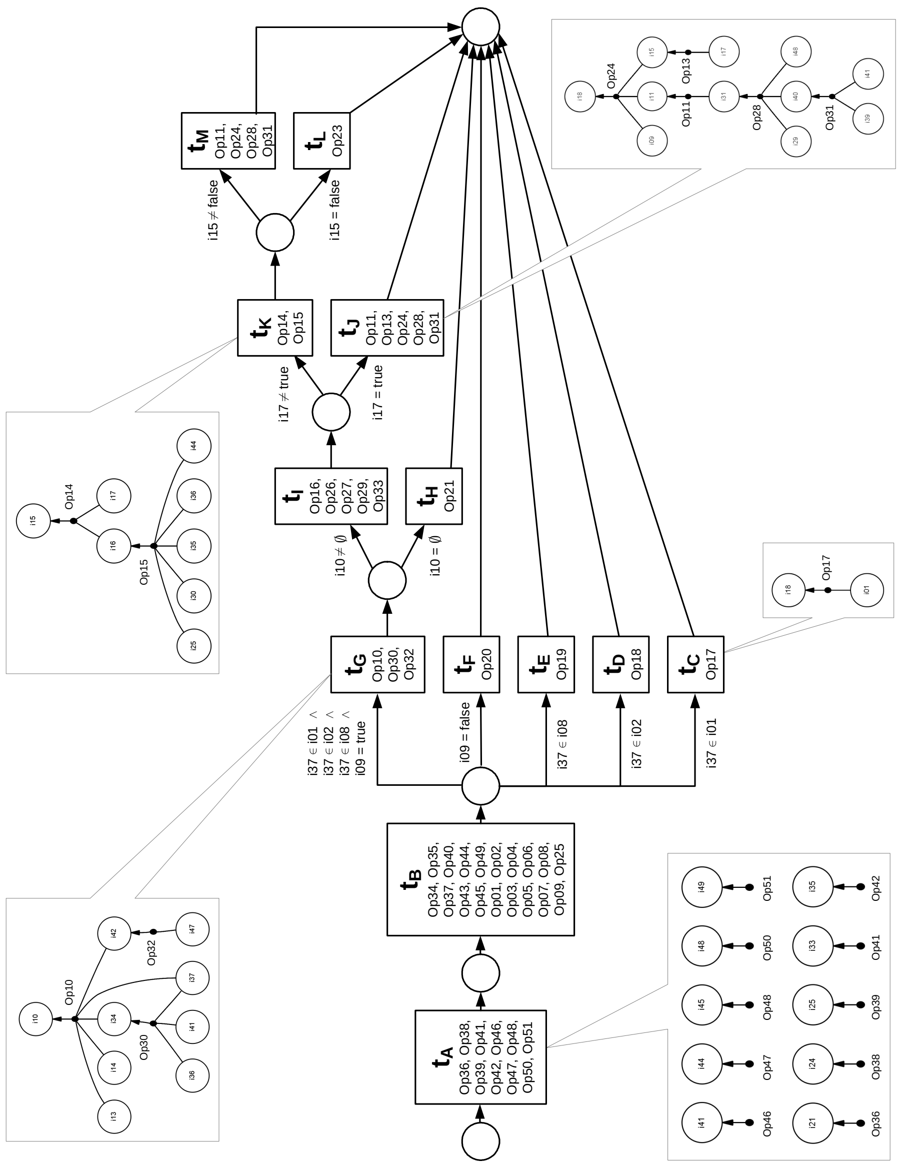

tA Op36, Op38, Op39, Op41, Op42, Op46, Op47, Op48, Op50, Op51
tB Op34, Op35, Op37, Op40, Op43, Op44, Op45, Op49, Op01, Op02, Op03, Op04, Op05, Op06, Op07, Op08, Op09, Op25
i37 ∈ i01 ∧ i37 ∈ i02 ∧ i37 ∈ i08 ∧ i09 = true
i09 = false
i37 ∈ i08
i37 ∈ i02
i37 ∈ i01
tG Op10, Op30, Op32
tF Op20
tE Op19
tD Op18
tC Op17
i10 ≠ ∅
i10 = ∅
tI Op16, Op26, Op27, Op29, Op33
tH Op21
i17 ≠ true
i17 = true
tK Op14, Op15
tJ Op11, Op13, Op24, Op28, Op31
i15 ≠ false
i15 = false
tM Op11, Op24, Op28, Op31
tL Op23

Figure 3.8: The final workflow process design of the unemployment benefits example. The content of each activity is given in terms of operations from the PDM. See also figures 3.6 and 3.7.

1. A means to define and view the product structure,
2. A way to define and view the content of each activity (in terms of data elements and operations), and
3. Proper support for the process of designing a process model based on the PDM (for example, it should give the designer some freedom to create and experiment with different designs and groupings of operations and data elements).

The two case handling systems we have selected for this assessment are FLOWer [206] and Activity Manager [140]. FLOWer is developed by Pallas Athena and consists of a number of components, of which FLOWer Studio is used at build-time to define case definitions consisting of activities, precedences, data objects, roles and forms. Activity Manager by BPi is an 'add-on' that can be used in combination with a workflow management system, such as COSA and Staffware. Activity Manager combines the structure and control of a workflow management system with the flexibility of case handling. It imports the process model from the workflow management system via a database and provides the means to further define the activities in this model by adding the operations.

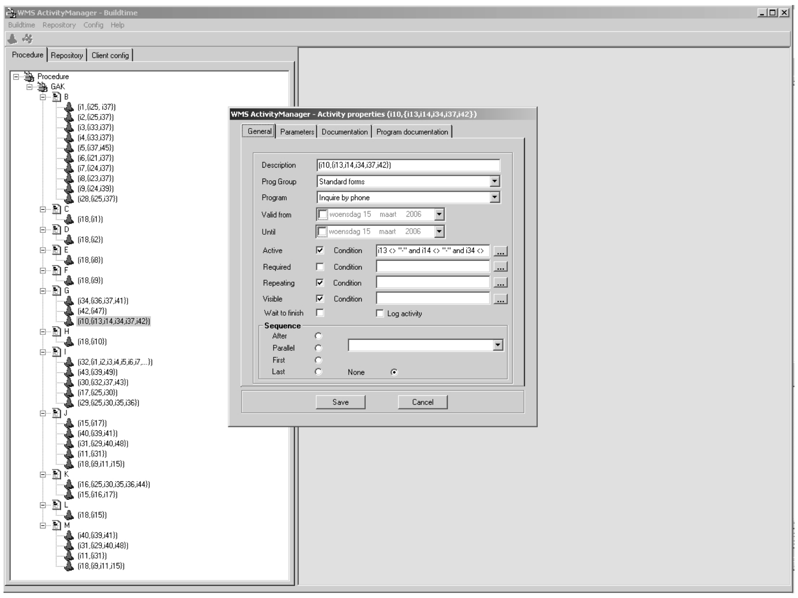
The assessment of the two systems is done by elaborating the design process of the social benefits example from the previous section. The focus in both assessments is on the process of designing and defining the process model based on the PDM. A detailed description of all steps taken in the assessment of both systems can be found in [277, 278].

Looking at the evaluation of case handling systems for PBWD from a conceptual viewpoint, we conclude that both systems do not (yet) provide a facility to display the PDM as a hierarchical structure. Although all concepts of the PDM and PBWD (e.g. data elements, operations, activities as clusters of data elements and operations) could be mapped to concepts in Activity Manager, the system does not provide an overview of the full PDM. Operations are explicitly shown per activity, but data elements are only implicitly defined within an operation (see Figure 3.9). FLOWer is able to represent all concepts except for the operations. A complete overview of data elements is given for the process as well as for each activity, but it is not clear how these data elements are linked together. Therefore, we conclude that the first requirement is not met in both systems.

Both systems meet the second requirement to some degree. The content of each activity is shown by a set of operations in Activity Manager (see Figure 3.9) and FLOWer clearly presents the content of an activity by forms containing all data elements belonging to the activity (see Figure 3.10).

From this assessment we also conclude that both systems still put quite some emphasis on the control-flow of the process, despite of their innovative focus on data. BPi's Activity Manager is considerably more process-driven than data-driven, as it starts from the definition of a process model. Of course, this follows from the fact that Activity Manager is 'added on' to a workflow system, which only allows Activity Manager to further specify the process structure already given. Because of this, it is not possible to directly design a process model which is purely based on a PDM. The user needs to have a good understanding of how the activities are organized and what the content of each activity should be. This means that the process of designing a process model based on the PDM should then be done outside the tool, in such a way that the result (i.e. the activities including their operations) can be implemented in the system. This violates our third requirement, i.e. that the tool itself should provide some support in the design process.

In contrast to BPi's Activity Manager, in FLOWer we can start reasoning from the PDM

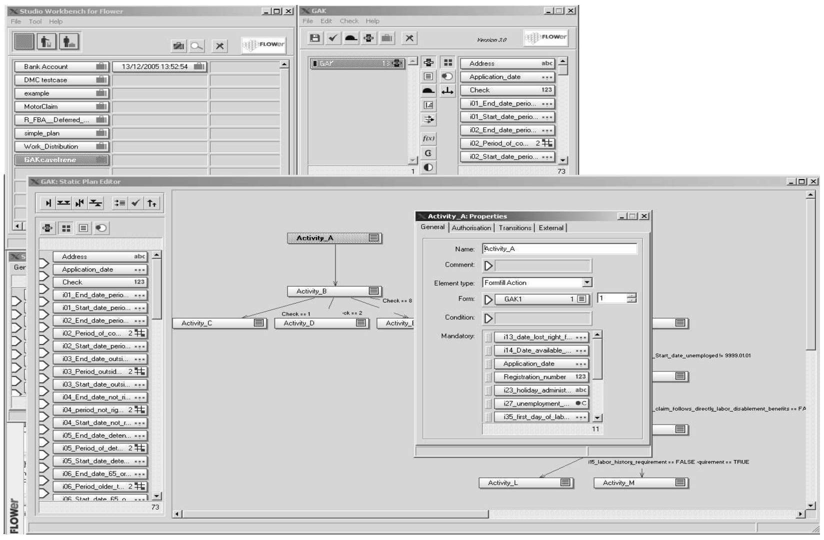


Procedure
- GAK
  - B
    - (i1,{i25, i37})
    - (i2,{i25,i37})
    - (i3,{i33,i37})
    - (i4,{i33,i37})
    - (i5,{i37,i45})
    - (i6,{i21,i37})
    - (i7,{i24,i37})
    - (i8,{i23,i37})
    - (i9,{i24,i39})
    - (i28,{i25,i37})
  - C
    - (i18,{i1})
  - D
    - (i18,{i2})
  - E
    - (i18,{i8})
  - F
    - (i18,{i9})
  - G
    - (i34,{i36,i37,i41})
    - (i42,{i47})
    - (i10,{i13,i14,i34,i37,i42})
  - H
    - (i18,{i10})
  - I
    - (i32,{i1,i2,i3,i4,i5,i6,i7,...})
    - (i43,{i39,i49})
    - (i30,{i32,i37,i43})
    - (i17,{i25,i30})
    - (i29,{i25,i30,i35,i36})
  - J
    - (i15,{i17})
    - (i40,{i39,i41})
    - (i31,{i29,i40,i48})
    - (i11,{i31})
    - (i18,{i9,i11,i15})
  - K
    - (i16,{i25,i30,i35,i36,i44})
    - (i15,{i16,i17})
  - L
    - (i18,{i15})
  - M
    - (i40,{i39,i41})
    - (i31,{i29,i40,i48})
    - (i11,{i31})
    - (i18,{i9,i11,i15})

WMS ActivityManager - Activity properties (i10,{i13,i14,i34,i37,i42})

Description: (i10,{i13,i14,i34,i37,i42})
Prog Group: Standard forms
Program: Inquire by phone
Valid from: ☐ woensdag 15 maart 2006
Until: ☐ woensdag 15 maart 2006
Active ☑ Condition: i13 <> "-" and i14 <> "-" and i34 <>
Required ☐ Condition:
Repeating ☑ Condition:
Visible ☑ Condition:
Wait to finish ☐ Log activity ☐
Sequence: After ○ Parallel ○ First ○ Last ○ [✓ None]

Figure 3.9: A screenshot of the buildtime environment of Activity Manager. On the left hand side a list of activities and their underlying operations is presented. The small window shows the properties which can be defined for each operation. Note that an operation is called an activity in Activity Manager. Data elements are defined as parameters in the second tab.



Bank Account 13/12/2005 13:52:54
DMC testcase
example
MotorClaim
R_FBA__Deferred_...
simple_plan
Work_Distribution
GAKcaseIrene

GAK 1

Address	abc
Application_date	•••
Check	123
i01_End_date_perio...	•••
i01_Start_date_perio...	•••
i02_End_date_perio...	•••
i02_Period_of_co...	2
i02_Start_date_perio...	•••
i03_End_date_outsi...	•••
i03_Period_outsid...	2
i03_Start_date_outsi...	•••
i04_End_date_not_ri...	•••
i04_period_not_rig...	2
i04_Start_date_not_r...	•••
i05_End_date_deten...	•••
i05_Period_of_det...	2
i05_Start_date_dete...	•••
i06_End_date_65_or...	•••
i06_Period_older_t...	2
i06_Start_date_65_o...	•••

73

Name: Activity_A
Comment:
Element type: Formfill Action
Form: GAK1 1 1
Condition:
Mandatory:

i13_date_lost_right_f...	•••
i14_Date_available_...	•••
Application_date	•••
Registration_number	123
i23_holiday_administ...	abc
i27_unemployment_...	●C
i35_first_day_of_lab...	•••

11

Figure 3.10: A screenshot of FLOWer Studio. On the left hand side an overview of all data elements in the process is given. The window in the middle of the screenshot shows the definition of an activity: a form containing a number of mandatory data elements is specified.

(i.e. by starting with the definition of data elements and their dependencies). This provides the opportunity to really focus on the grouping of data elements instead of on the definition of activities, although activities still have to be defined separately. By putting groups of data elements on one form and playing around with these combinations it is possible to compose activities based on the data and operations of the PDM instead of first defining the activities and afterwards determining what should be done in these activities. Taking this design perspective we can remark that FLOWer partly supports the third requirement by providing some freedom to experiment with different groupings of data elements and operations.

The discussion above shows that current case handling systems, and thus current workflow technology in general, are *not yet ready* to support the design of process models based on product structures. A research challenge is to develop better support for applying this design approach in practice. The remainder of this thesis deals with this issue: Chapter 4 introduces a basic correctness notion for a process model with respect to the corresponding PDM. Chapter 5 focuses on the automatic derivation of process models from a PDM. Chapter 6 elaborates on the support for direct execution of PDMs, and Chapter 7 presents business process metrics to evaluate a created process model.

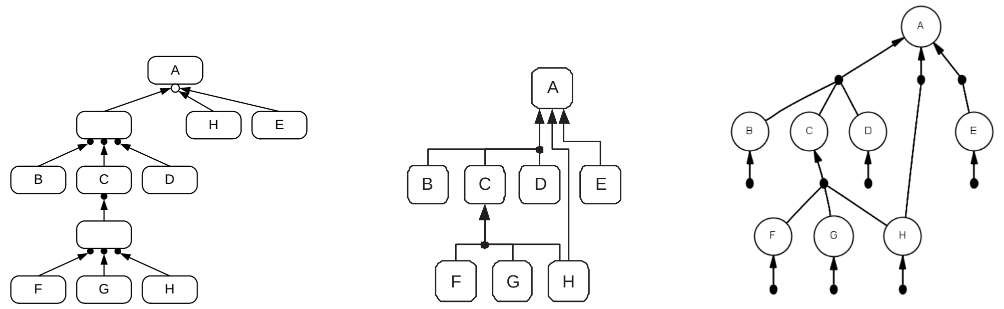
3.5 Related Work

This section focuses on the related work on product-driven and data-driven approaches in the workflow management domain. We briefly summarize the literature on the BoM and PDM. Next, we discuss other approaches to process design which do not focus on the control-flow in isolation. Note that, although these approaches have different names, they all focus on the objects that are processed instead of on the activities that are performed in the process.

3.5.1 Bill-of-Material for Manufacturing Processes

The notion of a Bill-of-Material (BoM) was introduced by Orlicky [201] in the 1970s. A BoM describes how a (physical) product can be decomposed into smaller parts. Over the years, many extensions and variants of the simple BoM have been introduced: e.g. the modular BoM [284], the percentage BoM [170], the super BoM, the variant BoM [284, 283], the generic BoM [283, 284], and the augmented BoM [263]. The BoM is used in e.g. Material Requirements Planning (MRP-I) [43, 201, 202], Manufacturing Resources Planning (MRP-II) [43, 289], but also in accounting to cost or price the manufactured product [99, 170].

Besides the BoM, also the notion of a *Bill-of-Processes (BoP)* exists in the manufacturing field [43]. Such a BoP is a separate model which describes the manufacturing steps that are needed for each of the components, sub assemblies and final assemblies in the product [43]. A difference with our PDM is that each production step in the BoM is a process on its own, i.e. it may consist of more than one manufacturing step or operation. For example, the assembly of a table from four legs and a tabletop may involve some welding, cooling down, cleaning, painting, and finishing. Sometimes the information on the manufacturing steps is subordinate to the information in the BoM, e.g. in an MRP system [303]. Also, the BoM and BoP may be integrated, resulting in a *P-graph* which represents the primary flow of both activities and materials in the manufacturing process [303]. Similar to the BoP also a resource model can be coupled to the BoM [43].



(a) The notation used by Van der Aalst [5].

(b) The notation used by Reijers et al. [233].

(c) The current notation as explained in this chapter.

Figure 3.11: *The models of the product structure for the mortgage example using different definitions and notations. (a) shows the notation used by Van der Aalst [5]. The black dots indicate mandatory components. Optional components are represented by a simple arrow and a circle indicates that a choice is made between several components. (b) shows the notation used by Reijers et al. [233]. This notation is rather similar to the notation we use in this thesis (c).*

3.5.2 Product Data Models for Workflow Processes

The idea to use product structures to design workflow processes was introduced by Van der Aalst [5]. The notion of the BoM in manufacturing is translated to administrative products and extended with options and choices. A notation to represent such a BoM for workflow products includes mandatory components, optional components and choice components (see Figure 3.11(a)). This BoM for workflow products is a tree.

Reijers et al. [233] take the definition of Van der Aalst as a basis for their product/data model and slightly change its notation. They introduce the notion of *production rules* or *operations*. Optional components are implicitly modeled in these production rules and choice components (i.e. alternative production rules) are modeled by separate arrows. Their models still have a tree-like structure, but actually are networks since a data element value may be used in several production rules. The differences between the two notations are illustrated in Figure 3.11. In this thesis, we have adopted the latter notion of product data models and extended it with production rules for leaf data elements (see Figure 3.11(c)).

Both earlier notions of product data models did not consider *leaf operations*, i.e. those operations that produce a value for the leaf elements of the PDM. In many cases the value for such a leaf element has to be filled in on a form, retrieved from an information system, requested from another party, which also requires some effort in time and money, and is part of the process to produce the end product. We have added the notion of leaf operations to the definition of a PDM to account for this time and cost.

Finally, in [233] some relations are added to the formal definition of the earlier notion of a PDM by van der Aalst [5]. These relations add constraints, execution time, execution cost, and the execution probability to each of the operations. We have extended this list of relations by adding the attribute of resources to it. The resource relation indicates which resources from the organizational model are allowed to execute the operation. Moreover, we have generalized the notion of execution times and execution cost to a distribution instead of a constant value.

Besides the work that is directly related to PBWD and PDMs, other researchers have also developed different approaches to business process design which neither focus on the control-flow of the process, but on e.g. the documents, artifacts, and objects in the process. Note that these approaches have some similarities.

3.5.3 Document-Centric Models

Dourish et al. [85, 86, 159] provide an approach for document-centric collaboration based on *active documents*. A document may have a number of properties associated to it, describing meta data on the document, e.g. giving information on whether the document is a paper, draft, or final version, or stating the history of actions done on the document. Besides these static properties, there are also *active properties* which may influence the behavior of a document by runnable code. These active properties may for instance be used to handle reading and writing of the document and to send notifications of document changes to interested parties. The authors claim that, with this approach, the coordination functionality is moved out of the workflow system and onto the documents themselves.

Although the use of (active) document properties may support collaborative work, and coordinate a process, it is unclear how several documents in one process are linked. Another difference is that the document-centric approach is more high level with respect to data elements. A document may e.g. contain several data elements that are specified in the PDM.

Wang and Kumar [292] present a framework for document-driven workflow systems. The main difference with conventional workflow systems is that in a document-driven workflow system no predefined control flow exists. This makes the process more flexible. All activities in the process are executed based on the availability of their input documents. The dependencies between activities are discovered by an analysis of the document-flow between activities. These dependencies then provide a partial ordering of the activities. In [156], the authors extend their document-driven approach to a resource-driven approach in which data resources, human resources, physical resources, and equipment resources are considered.

This approach by Wang and Kumar is very similar to the way we look at process models designed based on a PDM. In these process models activities also have a number of input data elements and one output data element, like the activities in the document-driven workflow system by Wang and Kumar have a number of input documents and a number output documents. However, we do derive process models with a specified control flow from the data dependencies in the PDM. Moreover, in a document-driven workflow system the relationship between the different documents is not made explicit, while the links between the data elements in a PDM are explicitly modeled. Finally, this document-centric approach is more high level with respect to data.

3.5.4 Proclets

In [8, 9], Van der Aalst et al. introduce *proclets* to solve problems of workflow processes that are not entirely case-oriented. In such processes it is unclear what a case exactly is. Consider for instance the process of hiring new employees. After the advertisement of an open position, several candidates may apply. Each of these candidates must be assessed individually, while they are also ranked. Moreover, other candidates may send open applications that are not related to an advertisement. The question now is on which level the process model should be described. Some activities in the process may be related to one candidate or one single job application, while other activities are on a higher level of aggregation, e.g. the advertisement of an open position is related to several job applications.

Proclets are lightweight workflow processes [9]. They represent only one aspect or one element of the whole workflow process and can interact with other elements. One can think of proclets as objects or entities (e.g. a single job application, a job position, and an advertisement campagne) which are equipped with an explicit life cycle or active documents (i.e. documents aware of activities and processes). Thus, by using proclets, complex workflow definitions are broken up into smaller interacting parts.

Although the objects or entities of the process are central in the definition of proclets, the proclets themselves are still process-oriented. This is different from our approach in which the product is the central object. Moreover, in a PDM the relation between different parts is explicitly modeled. In contrast, proclets are related to each other via the messages they can exchange.

3.5.5 Artifact-Centric Business Process Models

Bhattacharya et al. [44, 45, 108, 109, 166] take the *business artifacts* of a process (e.g. purchase order, or insurance claim) to be central in their approach, instead of the actions that are taken to achieve a goal. Only when the business artifacts are identified, it is considered how these artifacts are processed to achieve a certain business goal. An artifact-centric business process model consists of three different constructs: business artifacts, business tasks and repositories. A business artifact is an identifiable, self-describing unit of information. Business tasks describe the work acting upon an artifact by which value is added to this artifact. A repository describes a buffer for an artifact. Business tasks can push an artifact into a repository or pull it out of a repository. Artifact-centric models are designed manually, but there is a formal basis to these models which enables the analysis of the correctness of a model [45, 108, 166].

Business artifacts may be related to each other if they share data [108]. However, the artifact-centric approach does not provide a way to visualize these dependencies or to check for the constraints they impose on activity executions.

3.5.6 Object Lifecycles and Business Process Modeling

Müller et al. [186, 187, 188] have developed an approach for data-driven modeling and coordination of large process structures with workflow technology. Large process structures are usually comprised of several sub processes with many interdependencies, e.g. production and development processes in the automotive industry. Their execution must be coordinated and synchronized. Typically, the sub processes are related to different (data) objects (e.g. product components) enacted by different organizational units (e.g. dealing with the testing or releasing of single components) and controlled by different IT systems. Changes in the product description also affect the process design.

At the basis of the so-called COREPRO approach are two models: the *(product) data structure*, describing all components of the product, and the Life Cycle Coordination Structure, which describes the life cycles of the components. The relationships between the components in the product data structure indicate process dependencies. For instance, a car can only be tested when all of its sub systems (e.g. the engine and the navigation system) are properly tested. The structure of the process is then described by the data on the life cycles of the components in the product structure and is therefore called a *data-driven process structure*.

Küster et al. [157, 249, 250] also propose an approach in which business process models are combined with data on the life cycles of the objects which are manipulated in the business process. Object life cycles model the behavior of an object and the states an object can be in. The activities in the process model should manipulate the object in such a way that only the allowed state transitions can be a result of the activity. Two notions of consistency between a business process model and the object life cycles are proposed [250]. First of all, a given process model should be *compliant* with a given life cycle of a particular object. Secondly, the whole life cycle of an object should be *covered* during the course of the process. Moreover, in [157], a technique is presented for the generation of business process models based on object life cycles.

In contrast to the approach by Küster et al. [157, 249, 250], who do not relate the different object life cycles, Müller et al. [186, 187, 188] use the dependencies between the parts in the (product) data structure to link the object life cycles and to derive the structure of the large process model from the life cycles of the parts. This is similar to our PBWD approach. However, the difference with the our approach is that the research of Müller et al. is focused on physical products instead of informational products. The data used to link sub processes is data on the status of a physical product within its life cycle (e.g. designed, tested, released). Therefore, their notion of the development of information is more extensive than in the PBWD approach.

Another difference between our approach and the approaches of Küster et al. and Müller et al. is that we do not consider the life cycles of the data elements in the PDM. We assume a data element value is either available or unavailable.

3.5.7 Goal Graphs for Workflow Processes

Browne et al. [55, 56, 57] define a two-tier, goal-driven model for workflow processes in the healthcare domain. This two-tier model clearly splits high-level business goals (e.g. lower blood pressure) from lower-level workflow processes to achieve these goals (e.g. administer blood pressure lowering drug, or prescribe exercise).

Any healthcare regime may have a set of goals to achieve. These goals can be organized in a decomposition and represented graphically by a directed acyclic graph. This *goal graph* is similar to the PDM (cf. the example of a goal-graph in Figure 3.12). A mapping is defined from the goal-graph to (sub) processes and activities; each of the (sub) processes is designed to achieve one of the goals. This approach leads to a hierarchy of process models with on the top level of the workflow a number of goals being implemented through sub processes to achieve these goals.

Although the goal-graph has a PDM-like structure and it is used to derive a process model to support the process, there are some differences between these two approaches. First of all, the elements in the goal-graph are not always data elements, they can also represent actions that should be taken (indicated by verbs, e.g. quit smoking). Moreover, the process models derived from this approach are highly hierarchical. Each goal is related to a sub process in the process model, while the derivation of process models based on a PDM leads to less hierarchical models.

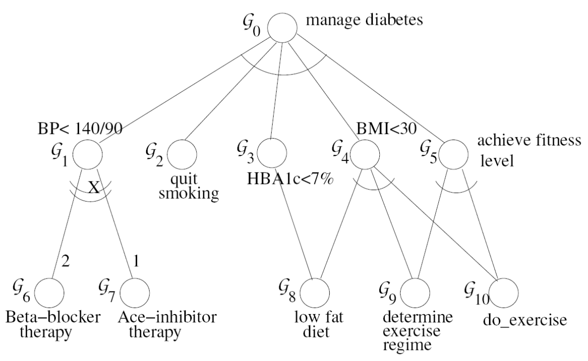

$\mathcal{G}_0$ manage diabetes
BP< 140/90
$\mathcal{G}_1$
X
$\mathcal{G}_2$ quit smoking
$\mathcal{G}_3$ HBA1c<7%
BMI<30
$\mathcal{G}_4$
$\mathcal{G}_5$ achieve fitness level
2
1
$\mathcal{G}_6$ Beta−blocker therapy
$\mathcal{G}_7$ Ace−inhibitor therapy
$\mathcal{G}_8$ low fat diet
$\mathcal{G}_9$ determine exercise regime
$\mathcal{G}_{10}$ do_exercise

Figure 3.12: *A goal-graph for diabetes management [55].*

3.6 Summary

In this chapter, the notion of the PDM as a workflow product description has been introduced. A PDM is similar to a BoM and describes the information processed in a workflow process using data elements and operations. The PDM is used to design process models to support workflow processes. Applications of PBWD in practice require the manual construction of a process model from the PDM. This is a time-consuming process in which mistakes are easily made, especially for large PDMs. We have shown that there is a need for tool support for PBWD. Current workflow tools, which originally have not been developed for this purpose, are not suitable for supporting the design of workflow processes based on product structures. Moreover, we did not find a solution for PBWD support in related work. This thesis contributes to the development of tool support for PBWD. Chapter 5 elaborates on the derivation of a process model from a PDM with 7 different algorithms. The direct execution of a PDM is discussed in Chapter 6. Finally, Chapter 7 presents business process metrics that help to evaluate the quality of a process model given by a grouping of data elements and operations.

Chapter 4

Formal Definition

In the previous chapter the notion of a PDM was introduced and illustrated with some examples. This chapter deals with the formal definition of a PDM. The formal definition is the basis for the development of a number of algorithms to derive a process model from a PDM (Chapter 5), and is also used to describe the direct execution of a PDM (Chapter 6). Finally, it is used to define a number of business process metrics which allow one to evaluate a process model and compare different process models for the same process (Chapter 7). In Section 4.1, the formal definition of a PDM is given, followed by the formal description of the running examples. Next, we focus on a formal approach to verify the consistency of a process model with the corresponding PDM. In Section 4.3 a number of assumptions that are made related to the semantics of the PDM are mentioned explicitly. This chapter is concluded with a section on related work and a summary.

4.1 Formal Definition of a PDM

After describing the notion of a PDM in the previous chapter, we can now define it formally. A PDM is represented by a 10-tuple consisting of the set of data elements, the set of operations, the set of resource classes, a special root element, and a number of functions that specify properties such as the execution cost, the processing time, the execution conditions, the failure probability and the authorized resource class for each operation. This formal definition is based on the formal definition in [233] and it is used to represent PDMs in the remainder of this thesis.

Definition 4.1.1 (PDM)**.** A PDM is a tuple $(D, O, C, W, root, cost, time, cond, fprob, res)$ with:

· D: the set of data elements,

· $O \subseteq D \times \mathcal{P}(D)$: the set of operations on the data elements. Each operation, $o = (d, ds)$, has one output element d and a set of zero or more input elements ds,

· D and O form a hypergraph $H = (D, O)$ such that its structure graph is connected and acyclic (cf. definitions 2.3.1, 2.3.5, 2.3.8, and 2.3.11).

· C: the set of conditions. A condition is a logic expression (see Section 2.1) consisting of e.g. constants, variables, logical operators, and data element values.

· W^1: the set of resource classes (or roles) that are available to the process. A partial order relation $\preceq$ is defined on these resource classes: '$v \preceq w$' means that a person with role w is allowed to do all the work v is allowed to do (and potentially more).

· $root \in D$: the root element of the PDM,

· $cost : O \to \mathbb{R}^{+\ 2}$: the function that specifies the execution cost for each operation,

· $time : O \to \mathbb{R}^{+\ 2}$: the function that specifies the processing time for each operation,

· $cond : O \not\rightarrow C$: the partial function (see Definition 2.2.7) that specifies the condition (if any) for each operation,

· $fprob : O \to [0.0, 1.0]$: the function that specifies the failure probability of each operation,

· $res : O \to W^{\ 1}$: the function that specifies the authorized resource class for each operation.

Note that, because the structure graph of hypergraph $H = (D, O)$ is *connected* and *acyclic*, the PDM contains no 'dangling' data elements and that no value of a data element depends on itself. Also note that the requirement that an operation may not have more than one output element does not lead to a loss of generality. Two output elements that are produced based on exactly the same set of input elements can be represented by two separate operations.

Each PDM should at least contain the set of data elements, the set of operations and a root element. Some of the operation attributes (e.g. cost, failure probability, resource class) are sometimes unknown for a specific process. For example, in the social benefits example no information is available on the processing time of operations. In such a case the respective function does not exist, which is represented by the empty set symbol. Below, the formal definitions for the running examples are elaborated.

4.1.1 Running Examples

In this section, the running examples which have been presented in Section 3.2.2 are defined using the formal definition of a PDM.

Example 1: Mortgage

The mortgage example of Section 3.2.2 contains 8 data elements and 10 operations (see Figure 4.1). For each operation information is available on the cost of executing the operation, the processing time, failure probability and execution conditions (see also Table 3.2). No information

[1]Although we have identified the resource perspective as an important part of the PDM we have not further investigated resource related issues in this thesis.
[2]Note that for simplicity we use constant values instead of probability distributions. However, it is straightforward to extend this to a probability distribution function and its parameters.

on resources is available. The PDM for the mortgage example is defined as follows:

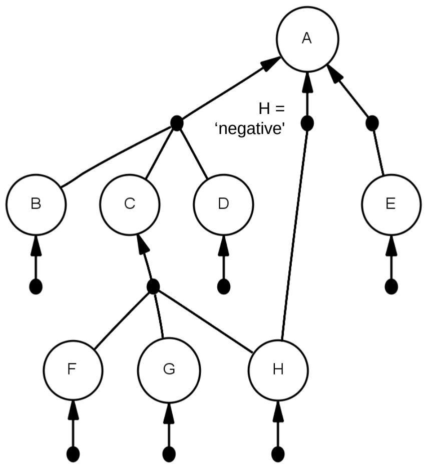
$$\begin{array}{rcl}
D & = & \{A,B,C,D,E,F,G,H\},\\
O & = & \{(A,\{B,C,D\}),(C,\{F,G,H\}),(A,\{H\}),(A,\{E\}),(B,\emptyset),(D,\emptyset),(E,\emptyset),\\
 & & (F,\emptyset),(G,\emptyset),(H,\emptyset)\},\\
C & = & \{H = \text{‘negative’}\},\\
W & = & \emptyset,\\
root & = & A,\\
cost & = & \{((A,\{B,C,D\}),5.0),((C,\{F,G,H\}),5.0),((A,\{H\}),9.0),((A,\{E\}),2.0),\\
 & & ((B,\emptyset),0.0),((D,\emptyset),0.0),((E,\emptyset),1.0),((F,\emptyset),0.0),((G,\emptyset),0.0),((H,\emptyset),3.0)\},\\
time & = & \{((A,\{B,C,D\}),1.0),((C,\{F,G,H\}),4.0),((A,\{H\}),3.0),((A,\{E\}),2.0),\\
 & & ((B,\emptyset),0.0),((D,\emptyset),0.0),((E,\emptyset),1.0),((F,\emptyset),0.0),((G,\emptyset),2.0),((H,\emptyset),10.0)\},\\
cond & = & \{((A,\{H\}),H = \text{‘negative’})\},\\
fprob & = & \{((A,\{B,C,D\}),0.05),((C,\{F,G,H\}),0.05),((A,\{H\}),0.05),\\
 & & ((A,\{E\}),0.00),((B,\emptyset),0.00),((D,\emptyset),0.00),((E,\emptyset),0.50),((F,\emptyset),0.00),\\
 & & ((G,\emptyset),0.00),((H,\emptyset),0.15)\},\\
res & = & \emptyset.
\end{array}$$

Note that the graphical representation of this PDM (see Figure 4.1) indeed is a hypergraph. Its structure graph is connected and acyclic.

Example 2: Naturalization

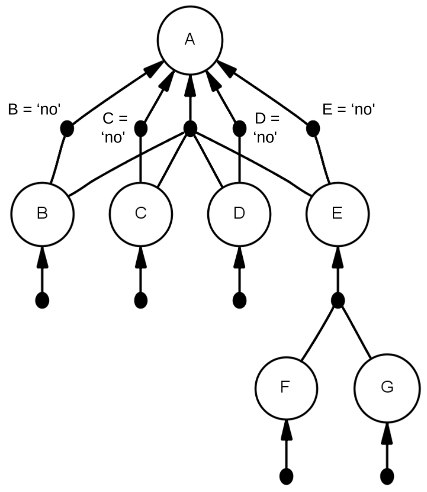
The naturalization example (see Section 3.2.2) is completely based on the text of the Law. Since this text does not provide any information on e.g. authorized resources, and execution cost, no information on the operation attributes is available except for the execution conditions. The formal definition of the naturalization example is given by:

$$\begin{array}{rcl}
D & = & \{A,B,C,D,E,F,G\},\\
O & = & \{(A,\{B\}),(A,\{C\}),(A,\{D\}),(A,\{E\}),(A,\{B,C,D,E\}),(B,\emptyset),(C,\emptyset),\\
 & & (D,\emptyset),(E,\{F,G\}),(F,\emptyset),(G,\emptyset)\},\\
C & = & \{B = \text{‘no’},C = \text{‘no’},D = \text{‘no’},E = \text{‘no’}\},\\
W & = & \emptyset,\\
root & = & A,\\
cost & = & \emptyset,\\
time & = & \emptyset,\\
cond & = & \{((A,\{B\}),B = \text{‘no’}),((A,\{C\}),C = \text{‘no’}),((A,\{D\}),D = \text{‘no’}),\\
 & & ((A,\{E\}),E = \text{‘no’})\},\\
fprob & = & \emptyset,\\
res & = & \emptyset.
\end{array}$$



A	Maximum mortgage
B	Percentage of interest
C	Annual budget to be spent on the mortgage
D	Term of mortgage
E	Previous offer (within the period of validity of the offer)
F	Percentage of income to be spent on the mortgage
G	Gross income per year
H	Credit registration

Figure 4.1: *The PDM of the mortgage example (see also figures 3.2 and 3.3).*



A	Applicant is entitled to Dutch citizenship
B	Objections exist against permanent stay of applicant
C	Applicant has admittance and main residence in the Netherlands, Netherlands Antilles or Aruba for at least five years immediately preceding the request
D	Age of applicant
E	Applicant can be seen as naturalized
F	Knowledge of the Dutch language
G	Knowledge of the Dutch constitution and society

Figure 4.2: *The PDM of the naturalization example (see also Figure 3.5).*

Example 3: Social Benefits

In the social benefits example, that was introduced in Section 3.2.2, information on execution conditions, execution cost, and failure probability is available for every operation. Also, for each operation it is known whether it can be executed automatically (by a computer), manually (by a human), or partly automatically (i.e. by a combination of automatic and manual execution). This information is too high-level to define resource classes and their partial order relation, since the case description in [231, 233] does not provide information on which human resources or which resource classes are authorized to perform the manual and partly automatic operations. Neither are the processing times mentioned in these publications. The formal definition of the PDM is as follows:

$$
\begin{array}{rcl}
D & = & \{i01, i02, i03, i04, i05, i06, i07, i08, i09, i10, i11, i13, i14, i15, i16, i17, i18, i21, i23, i24, i25, \\
 & & i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i47, i48, \\
 & & i49, i50, i51\}, \\
O & = & \{(i01, \{i25, i27\}), (i02, \{i25, i27\}), (i03, \{i33, i37\}), (i04, \{i33, i37\}), (i05, \{i37, i45\}), \\
 & & (i06, \{i21, i37\}), (i07, \{i24, i37\}), (i08, \{i23, i37\}), (i09, \{i24, i39\}), \\
 & & (i10, \{i13, i14, i34, i37, i42\}), (i11, \{i31\}), (i15, \{i16\}), (i15, \{i17\}), (i15, \{i16, i17\}), \\
 & & (i16, \{i25, i30, i35, i36, i44\}), (i17, \{i25, i30\}), (i18, \{i01\}), (i18, \{i02\}), (i18, \{i08\}), \\
 & & (i18, \{i09\}), (i18, \{i10\}), (i18, \{i11\}), (i18, \{i15\}), (i18, \{i09, i11, i15\}), (i28, \{i25, i37\}), \\
 & & (i29, \{i25, i30, i35, i36\}), (i30, \{i32, i37, i43\}), (i31, \{i29, i40, i48\}), \\
 & & (i32, \{i01, i02, i03, i04, i05, i06, i07, i08, i10, i27, i28\}), (i34, \{i36, i37, i41\}), (i40, \{i39, i41\}), \\
 & & (i42, \{i47\}), (i43, \{i39, i49\}), (i13, \emptyset), (i14, \emptyset), (i21, \emptyset), (i23, \emptyset), (i24, \emptyset), (i25, \emptyset), (i27, \emptyset), \\
 & & (i33, \emptyset), (i35, \emptyset), (i36, \emptyset), (i37, \emptyset), (i39, \emptyset), (i41, \emptyset), (i44, \emptyset), (i45, \emptyset), (i47, \emptyset), (i48, \emptyset), (i49, \emptyset)\}, \\
C & = & \{i16 = \text{true}, i17 = \text{true}, i37 \in i01, i37 \in i02, i37 \in i08, i9 = \text{false}, i10 = \emptyset, i11 = \text{false}, i15 = \text{false}\ \}, \\
W & = & \emptyset, \\
root & = & i18, \\
cost & = & \{((i01, \{i25, i27\}), 0.0), ((i02, \{i25, i27\}), 0.0), ((i03, \{i33, i37\}), 0.0), ((i04, \{i33, i37\}), 0.0), \\
 & & ((i05, \{i37, i45\}), 0.0), ((i06, \{i21, i37\}), 0.0), ((i07, \{i24, i37\}), 0.0), ((i08, \{i23, i37\}), 0.0), \\
 & & ((i09, \{i24, i39\}), 0.0), ((i10, \{i13, i14, i34, i37, i42\}), 0.0), ((i11, \{i31\}), 0.6), \\
 & & ((i15, \{i16\}), 0.0), ((i15, \{i17\}), 0.0), ((i15, \{i16, i17\}), 0.0), \\
 & & ((i16, \{i25, i30, i35, i36, i44\}), 5.61), ((i17, \{i25, i30\}), 0.0), ((i18, \{i01\}), 0.0), \\
 & & ((i18, \{i02\}), 0.0), ((i18, \{i08\}), 0.0), ((i18, \{i09\}), 0.0), ((i18, \{i10\}), 0.0), \\
 & & ((i18, \{i11\}), 0.0), ((i18, \{i15\}), 0.0), ((i18, \{i09, i11, i15\}), 0.0), \\
 & & ((i29, \{i25, i30, i35, i36\}), 0.0), ((i30, \{i32, i37, i43\}), 0.0), ((i28, \{i25, i37\}), 0.0), \\
 & & ((i31, \{i29, i40, i48\}), 0.0), ((i32, \{i01, i02, i03, i04, i05, i06, i07, i08, i10, i27, i28\}), 0.0), \\
 & & ((i34, \{i36, i37, i41\}), 4.2), ((i40, \{i39, i41\}), 0.3), ((i42, \{i47\}), 0.3), ((i43, \{i39, i49\}), 0.6), \\
 & & ((i13, \emptyset), 0.08), ((i14, \emptyset), 0.08), ((i21, \emptyset), 0.0), ((i23, \emptyset), 0.67), ((i24, \emptyset), 0.0), ((i25, \emptyset), 0.0), \\
 & & ((i27, \emptyset), 0.08), ((i33, \emptyset), 0.0), ((i35, \emptyset), 0.0), ((i36, \emptyset), 1.0), ((i37, \emptyset), 1.67), ((i39, \emptyset), 0.17), \\
 & & ((i41, \emptyset), 0.0), ((i44, \emptyset), 0.0), ((i45, \emptyset), 0.0), ((i47, \emptyset), 0.33), ((i48, \emptyset), 0.0), ((i49, \emptyset), 0.0)\}, \\
time & = & \emptyset, \\
cond & = & \{((i15, \{i16\}), i16 = true), ((i15, \{i17\}), i17 = true), ((i18, \{i01\}), i37 \in i01), \\
 & & ((i18, \{i02\}), i37 \in i02), ((i18, \{i08\}), i37 \in i08), ((i18, \{i09\}), i9 = false), \\
 & & ((i18, \{i10\}), i10 = \emptyset), ((i18, \{i11\}), i11 = false), ((i18, \{i15\}), i15 = false)\},
\end{array}
$$

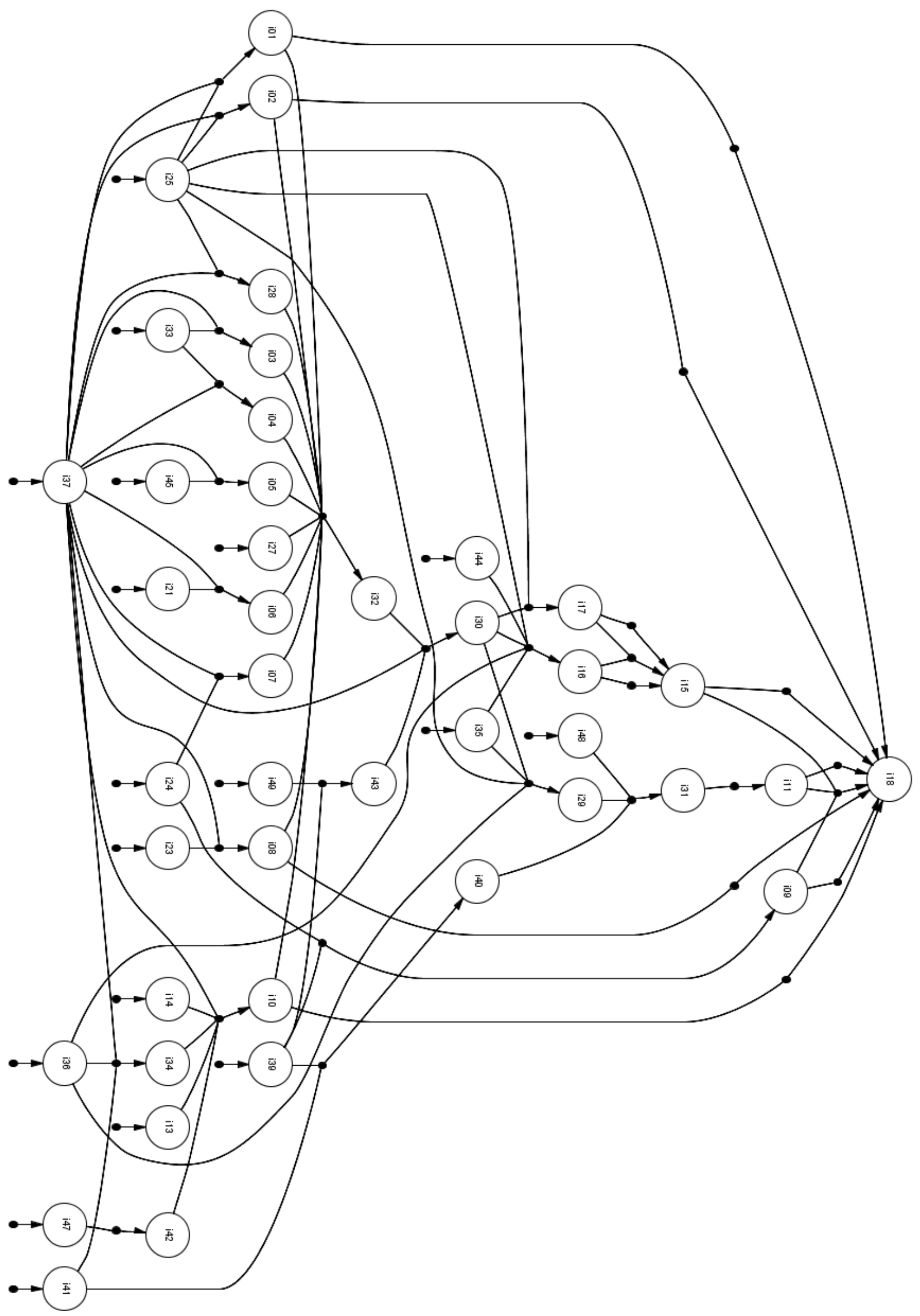


Figure 4.3: *The PDM of the social benefits process (cf. Figure 3.6).*

ID	Output	Input	Cost	Failure prob.	Execution condition
Op01	i01	{i25,i27}	0.0	0.0	
Op02	i02	{i25,i37}	0.0	0.0	
Op03	i03	{i33,i37}	0.0	0.0	
Op04	i04	{i33,i37}	0.0	0.0	
Op05	i05	{i37,i45}	0.0	0.0	
Op06	i06	{i21,i37}	0.0	0.0	
Op07	i07	{i24,i37}	0.0	0.0	
Op08	i08	{i23,i37}	0.0	0.0	
Op09	i09	{i24,i39}	0.0	0.0	
Op10	i10	{i13,i14,i34,i37,i42}	0.0	0.0	
Op11	i11	{i31}	0.6	0.0	
Op12	i15	{i16}	0.0	0.003	$i16$ = true
Op13	i15	{i17}	0.0	0.997	$i17$ = true
Op14	i15	{i16,i17}	0.0	0.0	
Op15	i16	{i25,i30,i35,i36,i44}	5.61	0.0	
Op16	i17	{i25,i30}	0.0	0.0	
Op17	i18	{i01}	0.0	0.991	$i37 \in i01$
Op18	i18	{i02}	0.0	0.987	$i37 \in i02$
Op19	i18	{i08}	0.0	0.984	$i37 \in i08$
Op20	i18	{i09}	0.0	0.998	$i9$ = false
Op21	i18	{i10}	0.0	0.932	$i10 = \emptyset$
Op22	i18	{i11}	0.0	0.981	$i11$ = false
Op23	i18	{i15}	0.0	0.79	$i15$ = false
Op24	i18	{i09,i11,i15}	0.0	0.0	
Op25	i28	{i25,i37}	0.0	0.0	
Op26	i29	{i25,i30,i35,i36}	0.0	0.0	
Op27	i30	{i32,i37,i43}	0.0	0.0	
Op28	i31	{i29,i40,i48}	0.0	0.0	
Op29	i32	{i01,i02,i03,i04,i05,i06, i07,i08,i10,i27,i28}	0.0	0.0	
Op30	i34	{i36,i37,i41}	4.2	0.0	
Op31	i40	{i39,i41}	0.3	0.0	
Op32	i42	{i47}	0.3	0.0	
Op33	i43	{i39,i49}	0.6	0.0	
Op34	i13	∅	0.08	0.0	
Op35	i14	∅	0.08	0.0	
Op36	i21	∅	0	0.0	
Op37	i23	∅	0.67	0.0	
Op38	i24	∅	0	0.0	
Op39	i25	∅	0	0.0	
Op40	i27	∅	0.08	0.0	
Op41	i33	∅	0	0.0	
Op42	i35	∅	0	0.0	
Op43	i36	∅	1.0	0.0	
Op44	i37	∅	1.67	0.0	
Op45	i39	∅	0.17	0.0	
Op46	i41	∅	0	0.0	
Op47	i44	∅	0	0.0	
Op48	i45	∅	0	0.0	
Op49	i47	∅	0.33	0.0	
Op50	i48	∅	0	0.0	
Op51	i49	∅	0	0.0	

Table 4.1: *Operation attributes for the unemployment benefits example.*

$$\begin{aligned}
\mathit{fprob} \;=\; & \{((i01,\{i25,i27\}),0.0),((i02,\{i25,i27\}),0.0),((i03,\{i33,i37\}),0.0),((i04,\{i33,i37\}),0.0),\\
& ((i05,\{i37,i45\}),0.0),((i06,\{i21,i37\}),0.0),((i07,\{i24,i37\}),0.0),((i08,\{i23,i37\}),0.0),\\
& ((i09,\{i24,i39\}),0.0),((i10,\{i13,i14,i34,i37,i42\}),0.0),((i11,\{i31\}),0.6),\\
& ((i15,\{i16\}),0.003),((i15,\{i17\}),0.997),((i15,\{i16,i17\}),0.0),\\
& ((i16,\{i25,i30,i35,i36,i44\}),5.61),((i17,\{i25,i30\}),0.0),((i18,\{i01\}),0.991),\\
& ((i18,\{i02\}),0.987),((i18,\{i08\}),0.984),((i18,\{i09\}),0.998),((i18,\{i10\}),0.932),\\
& ((i18,\{i11\}),0.981),((i18,\{i15\}),0.79),((i18,\{i09,i11,i15\}),0.0),((i28,\{i25,i37\}),0.0),\\
& ((i29,\{i25,i30,i35,i36\}),0.0),((i30,\{i32,i37,i43\}),0.0),((i31,\{i29,i40,i48\}),0.0),\\
& ((i32,\{i01,i02,i03,i04,i05,i06,i07,i08,i10,i27,i28\}),0.0),((i34,\{i36,i37,i41\}),4.2),\\
& ((i40,\{i39,i41\}),0.3),((i42,\{i47\}),0.3),((i43,\{i39,i49\}),0.6),((i13,\emptyset),0.08),\\
& ((i14,\emptyset),0.08),((i21,\emptyset),0.0),((i23,\emptyset),0.67),((i24,\emptyset),0.0),((i25,\emptyset),0.0),((i27,\emptyset),0.08),\\
& ((i33,\emptyset),0.0),((i35,\emptyset),0.0),((i36,\emptyset),1.0),((i37,\emptyset),1.67),((i39,\emptyset),0.17),((i41,\emptyset),0.0),\\
& ((i44,\emptyset),0.0),((i45,\emptyset),0.0),((i47,\emptyset),0.33),((i48,\emptyset),0.0),((i49,\emptyset),0.0)\},\\
res \;=\; & \emptyset.
\end{aligned}$$

Because the representation of this formal definition is not very readable the values are also summarized in Table 4.1, such that they can be looked up easily.

The formal definition of a PDM forms the basis for the formal verification of a process model with respect to the data dependencies given by the PDM. The next section introduces an approach to verify whether a process model is consistent with a PDM.

4.2 Correctness of Process Models

In Section 3.3.2 we have seen that a process model can be designed based on a PDM by clustering its operations. Each activity in the process model consists of a set of operations and their data elements. But when is such a process model a correct realization of the PDM? In this section, we elaborate on this question and introduce a *basic notion of correctness* for a process model with respect to a PDM. The correctness notion is partly based on *data anomalies* [267] in the data flow of a process model (see also Section 4.5.3).

A process model is consistent with a PDM if it respects the data dependencies given by this PDM. For instance, it should not be possible to use a data element as input before its value is determined. Therefore, the process model should (i) contain a way to produce each data element in the PDM, (ii) respect the data dependencies in the PDM in its control flow, and (iii) never have a possibility to execute an operation more than once in a path from the start to the end of the process.

This section describes a formal approach to check for these three correctness criteria. First, we formalize the notion of execution traces in a process model. An execution trace describes a path of activities in the process model from start to end. We also make a formal link between the activities in the process model and the data elements and operations in the PDM. Each activity in the process model is a set of operations and therefore has input and output data sets. Next, we propose a generic and formal method to verify the correctness of a process model with respect to the PDM.

In general, a process model consists of a set of *activities* and a *flow relation* on these activities. Each specific process modeling language has its own formalization to construct the flow relation. For instance, in a Petri net or WF-net the activities (transitions in Petri net terms) are related through places (cf. Section 2.4), while in the EPC language functions and events are alternating and can also be related to each other via connectors (cf. Section 2.6).

Executing a process model for a specific case means that the activities are executed in a specific order respecting the constraints in the flow relation. Such an order is called an *execution trace*.

Definition 4.2.1 (Execution trace)**.** An *execution trace* of a process model is a sequence of activities, $\sigma = \langle t_0, t_1, ..., t_n \rangle \in T^*$ where T is the set of activities, describing the execution of the process from the beginning to the end. The set of all execution traces Γ describes all possible executions of the process model.

The execution traces for a WF-net are given by its set of *firing sequences* (cf. Section 2.4). For example, the execution traces of the process model of Figure 3.8 for the social benefits example are:

$$\begin{aligned} \sigma_1 &= \langle t_A, t_B, t_G, t_I, t_K, t_M \rangle & \sigma_5 &= \langle t_A, t_B, t_F \rangle \\ \sigma_2 &= \langle t_A, t_B, t_G, t_I, t_K, t_L \rangle & \sigma_6 &= \langle t_A, t_B, t_E \rangle \\ \sigma_3 &= \langle t_A, t_B, t_G, t_I, t_J \rangle & \sigma_7 &= \langle t_A, t_B, t_D \rangle \\ \sigma_4 &= \langle t_A, t_B, t_G, t_H \rangle & \sigma_8 &= \langle t_A, t_B, t_C \rangle \end{aligned}$$

Note that e.g. $\sigma_9 = \langle t_A, t_C, t_B, t_J \rangle$ is not an execution trace for this process model. The execution traces for process models represented in other modeling languages can be determined in a similar way.

The content of each activity in a process model is defined by the PDM, since an activity is a group of operations and their data elements. Each activity can also be characterized by two sets of data elements: the input data set and the output data set. The input data set contains all data elements of which the value is provided by another activity and of which the value is only used and not produced in the activity. The output data set contains the output elements of all operations in the activity.

Definition 4.2.2 (Activity)**.** An *activity*, t, on a PDM is a set of operations, i.e. $t \subseteq O$. Each activity is associated with two sets of data, since each operation has a set of input data elements and one output data element:

- The output data set of activity t, denoted as Ω_t, contains the root element and all intermediate data elements of the activity. Ω_t is defined as

$$\Omega_t = \bigcup_{(d,ds) \in t} \{d\}$$

- The input data set of activity t, denoted as I_t, contains the input elements of the activity. I_t is defined as

$$I_t = \bigcup_{(d,ds) \in t} ds \; - \; \Omega_t$$

The data elements that are output elements of one operation in the activity and input to another operation in the same activity are called the *intermediate* data elements. Intermediate data elements are added to the output data set since these elements are a product of the activity and

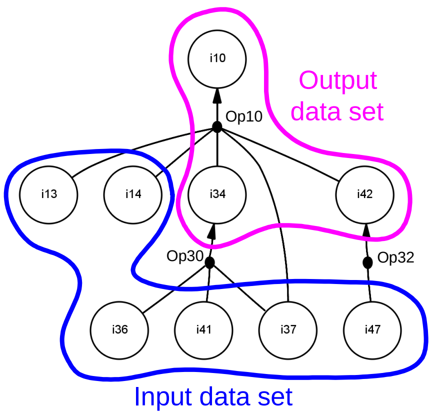


Figure 4.4: *The content of activity t_G (cf. Figure 3.8) consists of operations $Op30$, $Op32$, and $Op10$ and their data elements. Thus, the input data set of activity t_G is $I_{t_G} = \{i13, i14, i36, i37, i41, i47\}$ and the output data set is $\Omega_{t_G} = \{i10, i34, i42\}$.*

can be used as input to another activity. Figure 4.4 shows the input and output data sets for activity t_G of the social benefits example. In Figure 4.5 the input and output data sets for all activities are indicated in the process model of this example.

Based on the input and output data sets for each activity and the execution traces of the process model, we can perform a data flow analysis to verify the correctness of the process model with respect to the PDM. This approach is described in the next section.

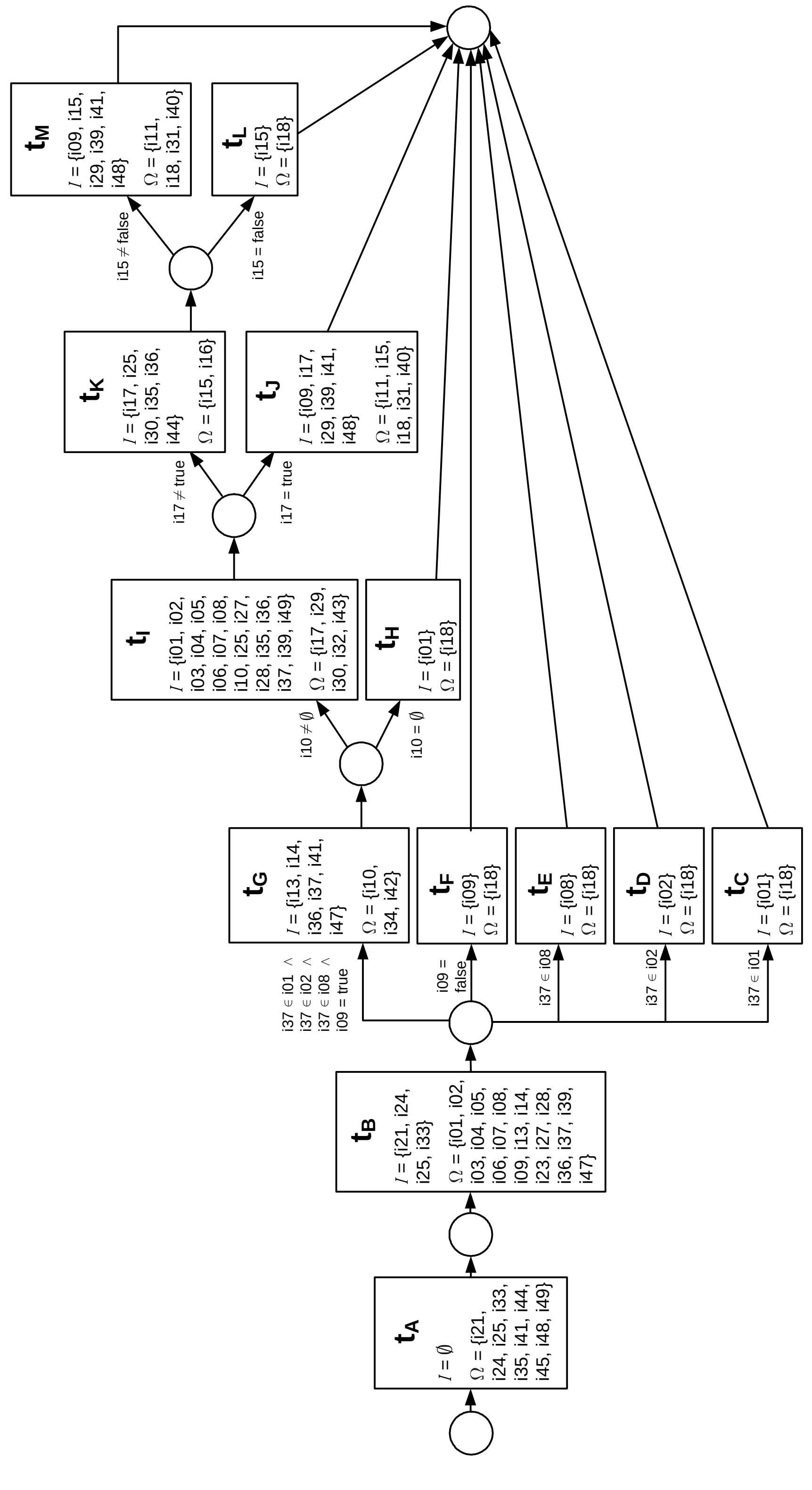
4.2.1 Verification of Correctness

Given a PDM and a process model with the set of execution traces and input and output data sets for each activity, we can check whether the process model is consistent with the PDM. The process model should satisfy the following three basic correctness requirements, which were introduced earlier in this section:

1. The process model should contain a way to produce each data element in the PDM,
2. The process model should respect the data dependencies in the PDM in its control flow, and
3. The process model should never have a possibility to execute an operation more than once in a path from the start to the end of the process.

To verify these requirements, we use a simplified version of the data flow verification method by Sun et al. [267] and we adopt their notion of *missing data anomalies* (see Section 4.5.3). Based on the input/output data relations between activities, a number of activity dependencies is derived[3]. Next, it is checked whether the process model respects these dependencies for all cases by comparing the execution traces of the process model with the derived activity dependencies.

[3]Note that these activity dependencies are purely derived from the input and output data sets of the activities. The control flow specified by the flow relation in the abstract process model is not taken into account when deriving the activity dependencies.



t_A
$I = \emptyset$
Ω = {i21, i24, i25, i33, i35, i41, i44, i45, i48, i49}
t_B
I = {i21, i24, i25, i33}
Ω = {i01, i02, i03, i04, i05, i06, i07, i08, i09, i13, i14, i23, i27, i28, i36, i37, i39, i47}
i37 ∈ i01 ∧ i37 ∈ i02 ∧ i37 ∈ i08 ∧ i09 = true
i09 = false
i37 ∈ i08
i37 ∈ i02
i37 ∈ i01
t_G
I = {i13, i14, i36, i37, i41, i47}
Ω = {i10, i34, i42}
t_F
I = {i09}
Ω = {i18}
t_E
I = {i08}
Ω = {i18}
t_D
I = {i02}
Ω = {i18}
t_C
I = {i01}
Ω = {i18}
i10 ≠ ∅
i10 = ∅
t_I
I = {i01, i02, i03, i04, i05, i06, i07, i08, i10, i25, i27, i28, i35, i36, i37, i39, i49}
Ω = {i17, i29, i30, i32, i43}
t_H
I = {i01}
Ω = {i18}
i17 ≠ true
i17 = true
t_K
I = {i17, i25, i30, i35, i36, i44}
Ω = {i15, i16}
t_J
I = {i09, i17, i29, i39, i41, i48}
Ω = {i11, i15, i18, i31, i40}
i15 ≠ false
i15 = false
t_M
I = {i09, i15, i29, i39, i41, i48}
Ω = {i11, i18, i31, i40}
t_L
I = {i15}
Ω = {i18}

***Figure 4.5:** The data dependencies for the social benefits example (cf. Figure 3.8).*

Activity (t)	Activity Dependencies (Δ_t)	Activity Dependency Expression (ξ_t)
t_A	–	–
t_B	$\{t_A \mapsto^{i21} t_B, t_A \mapsto^{i24} t_B, t_A \mapsto^{i25} t_B, t_A \mapsto^{i33} t_B\}$	$t_A \mapsto t_B$
t_C	$\{t_B \mapsto^{i01} t_C\}$	$t_B \mapsto t_C$
t_D	$\{t_B \mapsto^{i02} t_D\}$	$t_B \mapsto t_D$
t_E	$\{t_B \mapsto^{i08} t_E\}$	$t_B \mapsto t_E$
t_F	$\{t_B \mapsto^{i09} t_F\}$	$t_B \mapsto t_F$
t_G	$\{t_A \mapsto^{i41} t_G, t_B \mapsto^{i13} t_G, t_B \mapsto^{i14} t_G, t_B \mapsto^{i36} t_G, t_B \mapsto^{i37} t_G,$ $t_B \mapsto^{i47} t_G\}$	$(t_A \mapsto t_G) \wedge (t_B \mapsto t_G)$
t_H	$\{t_G \mapsto^{i10} t_H\}$	$t_G \mapsto t_H$
t_I	$\{t_A \mapsto^{i25} t_I, t_A \mapsto^{i35} t_I, t_A \mapsto^{i49} t_I, t_B \mapsto^{i01} t_I, t_B \mapsto^{i02} t_I, t_B \mapsto^{i03} t_I,$ $t_B \mapsto^{i04} t_I, t_B \mapsto^{i05} t_I, t_B \mapsto^{i06} t_I, t_B \mapsto^{i07} t_I, t_B \mapsto^{i08} t_I, t_B \mapsto^{i27} t_I,$ $t_B \mapsto^{i28} t_I, t_B \mapsto^{i36} t_I, t_B \mapsto^{i37} t_I, t_B \mapsto^{i39} t_I, t_G \mapsto^{i10} t_I\}$	$(t_A \mapsto t_I) \wedge (t_B \mapsto t_I) \wedge (t_G \mapsto t_I)$
t_J	$\{t_A \mapsto^{i41} t_J, t_A \mapsto^{i48} t_J, t_B \mapsto^{i09} t_J, t_B \mapsto^{i39} t_J, t_I \mapsto^{i17} t_J, t_I \mapsto^{i29} t_J\}$	$(t_A \mapsto t_J) \wedge (t_B \mapsto t_J) \wedge (t_I \mapsto t_J)$
t_K	$\{t_A \mapsto^{i25} t_K, t_A \mapsto^{i35} t_K, t_A \mapsto^{i44} t_K, t_B \mapsto^{i36} t_K, t_I \mapsto^{i17} t_K, t_I \mapsto^{i30} t_K\}$	$(t_A \mapsto t_K) \wedge (t_B \mapsto t_K) \wedge (t_I \mapsto t_K)$
t_L	$\{t_K \mapsto^{i15} t_L, t_J \mapsto^{i15} t_L\}$	$((t_K \mapsto t_L) \vee (t_J \mapsto t_L))$
t_M	$\{t_A \mapsto^{i41} t_M, t_A \mapsto^{i48} t_M, t_B \mapsto^{i09} t_M, t_B \mapsto^{i39} t_M, t_I \mapsto^{i29} t_M,$ $t_K \mapsto^{i15} t_M, t_J \mapsto^{i15} t_M\}$	$(t_A \mapsto t_M) \wedge (t_B \mapsto t_M) \wedge (t_I \mapsto t_M) \wedge$ $((t_K \mapsto t_M) \vee (t_J \mapsto t_M))$

Table 4.2: *The activity dependencies and the activity dependency expressions for the activities from the social benefits example.*

Definition 4.2.3 (Activity Dependency)**.** Given two activities v and t, i.e. $v \in T \wedge t \in T$, t is dependent on v (denoted as $v \mapsto t$), if:

$$\Omega_v \cap I_t \neq \emptyset.$$

We use $v \mapsto^d t$ to indicate the dependency of t on v via data element d, i.e. $d \in (\Omega_v \cap I_t)$. Moreover, Δ_t is the set of all activity dependencies in which an arbitrary activity $v \in T$ via data element d influences activity t, i.e. $\Delta_t = \{v \mapsto^d t \mid v \in T \ \wedge \ d \in (\Omega_v \cap I_t)\}$. Note that $v \mapsto t$ and $v \mapsto^d t$ are not logical expressions with a truth value, but they denote the relationship between v, t, and d.

Informally, we say that activity t is dependent on activity v if v provides input data for t. The activity dependencies for the social benefits example are derived from the input and output data sets indicated in Figure 4.5 and are shown in Table 4.2. Note that activity t_L is dependent on activity t_K and activity t_J through the same data element: $i15$. This means that there are two alternative ways to produce data element $i15$. For such a situation we do not require that both activities should be executed before activity t_L is executed. Only one of them is enough. Thus, $t_K \mapsto t_L$ and $t_J \mapsto t_L$ do not have to be satisfied at the same time. To express this, we introduce the notion of activity dependency expressions, which describe the logical relationship between the activity dependencies of an activity.

Definition 4.2.4 (Activity Dependency Expression)**.** The activity dependency expression ξ_t describes the activity dependencies for activity t. The expression is built from several components. Each component is related to an input data element of activity t:

$$\xi_t = e_{d_1} \wedge e_{d_2} \wedge e_{d_3} \wedge ...$$

for $d_1, d_2, d_3, ... \in I_t$. Component e_d contains the activity dependencies for activity t based on data element d. If there is only one v such that $v \mapsto^d t$ then $e_d = (v \mapsto t)$. When there are more activities producing $d \in I_t$, i.e. $v_1 \mapsto^d t, v_2 \mapsto^d t, ...$, then $e_d = ((v_1 \mapsto t) \vee (v_2 \mapsto t) \vee ...)$.

Note that activity dependency expression ξ_t always is in *conjunctive normal form* (see Section 2.1.2) since the main formula consists of a conjunction of components and each component is a disjunction of activity dependencies: $\xi_t = (((v_{11} \mapsto t) \vee (v_{12} \mapsto t) \vee ...) \wedge ((v_{21} \mapsto t) \vee (v_{22} \mapsto t) \vee ...) \wedge ... \wedge ((v_{k1} \mapsto t) \vee (v_{k2} \mapsto t) \vee ...))$. In Table 4.2 the activity dependency expressions for the social benefits example are shown. For instance, the activity dependency expression for activity t_G is:

$$\xi_{t_G} = (t_A \mapsto t_G) \wedge (t_B \mapsto t_G)$$

and the activity dependency expression for activity M is:

$$\xi_{t_M} = (t_A \mapsto t_M) \wedge (t_B \mapsto t_M) \wedge (t_I \mapsto t_M) \wedge ((t_K \mapsto t_M) \vee (t_J \mapsto t_M)).$$

To evaluate an activity dependency expression, all components of the expression should be evaluated.

Definition 4.2.5 (Evaluation of Activity Dependency Expression)**.** An activity dependency expression for activity t, $\xi_t = (((v_{11} \mapsto t) \vee (v_{12} \mapsto t) \vee ...) \wedge ((v_{21} \mapsto t) \vee (v_{22} \mapsto t) \vee ...) \wedge ... \wedge ((v_{k1} \mapsto t) \vee (v_{k2} \mapsto t) \vee ...))$, is evaluated to true for a set of execution traces Γ if [4]:

$$
\begin{array}{l}
\forall_{\sigma \in \Gamma}[\, (t \in \sigma) \Rightarrow \\
\\
\quad (\, (\exists_i[\, \sigma(i) = v_{11} \;\wedge\; \forall_{j<i}[\, \sigma(j) \neq t\,]\,]) \quad \vee \quad (\exists_i[\, \sigma(i) = v_{12} \;\wedge\; \forall_{j<i}[\, \sigma(j) \neq t\,]\,]) \quad \vee \quad ... \;) \\
\quad \wedge \\
\quad (\, (\exists_i[\, \sigma(i) = v_{21} \;\wedge\; \forall_{j<i}[\, \sigma(j) \neq t\,]\,]) \quad \vee \quad (\exists_i[\, \sigma(i) = v_{22} \;\wedge\; \forall_{j<i}[\, \sigma(j) \neq t\,]\,]) \quad \vee \quad ... \;) \\
\quad \wedge \\
\quad ... \\
\quad \wedge \\
\quad (\, (\exists_i[\, \sigma(i) = v_{k1} \;\wedge\; \forall_{j<i}[\, (\sigma(j) \neq t)\,]\,]) \quad \vee \quad (\exists_i[\, \sigma(i) = v_{k2} \;\wedge\; \forall_{j<i}[\, \sigma(j) \neq t\,]\,]) \quad \vee \quad ... \;) \\
\\
]
\end{array}
$$

Thus, given a component $(v_{11} \mapsto t \vee v_{12} \mapsto t)$ of an activity dependency expression, it should hold, for all execution traces $\sigma \in \Gamma$, that if t is an element of the execution trace σ, then either v_{11} or v_{12} or both should precede (all occurrences of) t in the execution trace σ. Next, all components of an activity dependency expression should be true for the activity dependency expression to evaluate to true as a whole. For example, to evaluate the activity dependency expression for activity t_G ($\xi_{t_G} = (t_A \mapsto t_G) \wedge (t_B \mapsto t_G)$) we should check the execution traces of the process model for the following:

$$
\begin{array}{l}
\forall_{\sigma \in \Gamma}[(t_G \in \sigma) \;\Rightarrow \\
\qquad\qquad (\, \exists_i[\, \sigma(i) = t_A \;\wedge\; \forall_{j<i}[\, \sigma(j) \neq t_G\,]\,]) \;\wedge\; (\exists_i[\, \sigma(i) = t_B \;\wedge\; \forall_{j<i}[\, \sigma(j) \neq t_G\,]\,])\,]
\end{array}
$$

This is true for σ_1 since t_G is an element of σ_1 and t_A precedes t_G in σ_1 and t_B precedes t_G in σ_1. Similarly, this is also true for σ_2, σ_3, and σ_4. For σ_5, σ_6, σ_7, and σ_8 the formula is also true because t_G is not an element of σ_5, σ_6, σ_7, and σ_8. Thus, the properties hold for all $\sigma \in \Gamma$ and therefore the activity dependency expression for activity t_G of the process model for the social benefits example evaluates to true.

At the beginning of this section, three criteria were introduced for a process model to be consistent with a PDM. First of all, every data element in the PDM that is used as an input element by an activity or that is the root element needs to be initialized. Secondly, if an activity t produces a data item that is used as input by another activity v, then t must already be executed when v is executed for the first time to guarantee that the data element needed by v has been produced (the activity dependencies are respected in the control flow of the process model). And finally, an operation may never be executed more than once in any execution trace. These criteria are formalized in the following proposition.

Proposition 4.2.1 (Correctness Verification)**.** A process model is consistent with a PDM if and only if the following basic correctness conditions are satisfied:

[4]Note that $\sigma(i)$ denotes the element at index i in sequence σ (cf. Definition 2.2.9).

1. All data elements which are either an input of some activity or the root element are produced as an output element by another activity:

$$(\bigcup_{t \in T} I_t \cup \{root\}) \subseteq \bigcup_{t \in T} \Omega_t.$$

2. The activity dependencies derived from the input/output data relations are respected by the process model:

$$\Xi = \bigwedge_{t \in T} \xi_t \text{ evaluates to true.}$$

3. An operation may never be executed more than once in any execution trace:

$$\forall_{\sigma \in \Gamma} \forall_{1 \leq i < j \leq |\sigma|} \, [\, \sigma(i) \cap \sigma(j) = \emptyset \,].$$

The third requirement is based on an assumption we make related to the semantics of a PDM. We assume that an operation can be uniquely identified by its input and output elements. In the next section, a number of these assumptions is listed and explained. Therefore, we do not elaborate on them here.

If the three basic correctness criteria are satisfied, the process model is consistent with the PDM. The process model for the social benefits example in Figure 3.8 is a correct realization of the PDM, because (i) all data elements from the PDM are produced by some activity, (ii) all activity dependencies are respected by the process model, and (iii) none of the operations can be executed more than once in any execution trace of the process model.

4.3 Assumptions Related to the Semantics of a PDM

In this section we explicitly list a number of assumptions that are made about the semantics of the PDM. These assumptions have been mentioned implicitly in Chapter 3, but they are important for the understanding of the semantics of a PDM in further explanations in this thesis. Therefore, they are explicitly stated here.

The first assumption we make is that an operation can have only one output element.

Assumption 4.3.1 (One output element per operation)**.** Every operation has exactly one output element.

The core of the product based workflow design approach is that a process has exactly one end product which can be described by a PDM. Although the determination of intermediate product values is possible, the whole process focuses on the final product. Similarly, each operation should have one output. If we would allow multiple outputs for one operation then we should also allow for more end products in one process since the last operation may be producing more than one output element. This is an undesirable situation since it would lead to unclear product descriptions. The assumption that an operation has exactly one output element is reflected in the formal definition of a PDM since $O \subseteq D \times \mathcal{P}(D)$.

This assumption does not lead to a loss of generality since two output elements that are produced based on exactly the same input elements can be represented by two separate operations with the same input elements and a different output element, e.g. an activity with input elements C and D and output elements A and B can be represented by the operations $(A, \{C, D\})$ and $(B, \{C, D\})$ as is shown in Figure 4.6.

The second assumption states that operations can be uniquely identified by their set of input elements and the output element.

Assumption 4.3.2 (Operations are unique)**.** Operations can be uniquely identified by their set of input elements and the output element. There cannot be two operations with exactly the same input elements and the same output element.

In the formal definition of a PDM, no unique identifiers for operations are included. Operations are uniquely identified by their content, i.e. by their output element and their set of input elements. The existence of two operations with exactly the same input and output elements is not very probable since that would mean that there are two ways to do exactly the same thing. However, if one would like to allow for the occurrence of operations with exactly the same input and output elements, the formal definition can easily be extended with a function that relates a unique identifier to an operation. The identification of several operations that produce the same output element based on different input elements is no problem, since the operations can be distinguished based on their different input elements. Also, two operations with the same input elements and different output elements (e.g. as shown in Figure 4.6) do not cause problems in this approach, since they can be distinguished on the difference in their output.

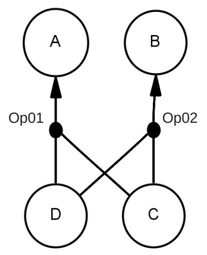


Figure 4.6: *Two operations with the same input elements and different output elements:* $Op01 = (A, \{C, D\})$ *and* $Op02 = (B, \{C, D\})$.

Our third assumption focuses on the number of times an operation can be executed. We assume that an operation can be executed only once for a case.

Assumption 4.3.3 (Execution of operations)**.** Each operation in the PDM can be executed only once for a specific case. The execution is either successful or unsuccessful. In the first case a value for the output data element of the operation is produced. In the latter case, no value for the output data element is produced. Afterwards, the operation can never be executed again.

If we would allow for an operation to be executed multiple times for the same case, this would introduce a number of problems. First of all, it may be so that multiple executions of the same operation lead to different output values. If that is the case, it is unclear which data element value should be used in further steps for the case. Secondly, if a new value is produced for an already existing data element value, all steps that have been executed after the time that the value was produced for the first time should be executed again. This means that there should be a kind of 'roll-back' mechanism to support the cancellation of steps made. These two consequences of multiple operation executions make supporting product based workflows complicated. Our assumption circumvents such problems.

Note that when the execution of an operation fails and no value for the output element is produced, this does not mean that the value for the output element will never be produced. The unsuccessfully executed operation cannot be repeated anymore, but there may still be another operation that can produce a value for the same data element.

The fourth assumption states that once the value of a data element has been determined for a case this value will be available for as long as the case is running.

Assumption 4.3.4 (Availability of data element values)**.** Whenever the value of a data element is determined for a case, this value will be available for the entire duration of the case. It is not possible to produce a new value for the same data element.

Once a data element value is produced it may be used in several other operations as an input element. It may be used immediately after it has been determined, but it may also be required in further operations. Therefore, it should be accessible for the total duration of the case. Because a value for the data element is available there is no need to execute another operation producing a value for the same data element.

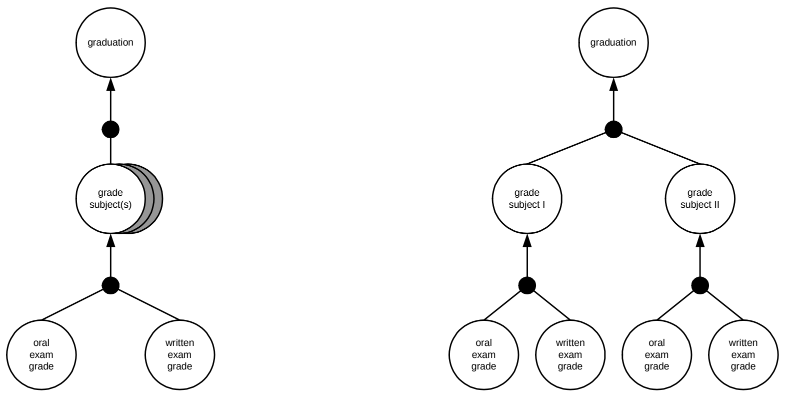
This assumption implies that an operation which potentially produces an output data element that is already available will always be skipped. Appendix A in [231] describes how the skipping of an operation works if the output element is already available. If we would allow for the possibility to still execute this operation, we would run into to same problem as above: having different values for the same data element.

Finally, the last assumption deals with multiple instances of a data element in the PDM. This is a construct that is similar to the notion of multiple instances of activities in workflow processes as described by the multiple instance workflow patterns [17, 302].

Assumption 4.3.5 (Single instance of data elements)**.** A data element can only have a single instance for a case; multiple instances of one data element in the same case are not allowed.

Multiple instances of the same data element may be used to model the existence of several distinct occurrences of a data element for a single case. Each of these occurences may have a different value. For example, the decision whether a student may receive his diploma is dependent on his grades for a number of subjects. The grade for a subject is determined by the grade for an oral exam and a written exam. A student may only graduate if he passes all subjects with a sufficient grade. Sometimes the number of instances of a data element is known at design time. In that case, each of the instances can be modeled as a separate data element as is shown in Figure 4.7(b). However, if this number is not known, it is unclear how many instances of the data elements should be dealt with (see Figure 4.7(a)). Therefore, we assume that these multiple instances of a data element do not occur in a PDM.

Note that the last two assumptions imply that loops are not allowed in the PDM or the process model. The occurrence of a loop may either indicate incorrect executions (see e.g. [29, 30]), or multiple instances of the same element. In case of incorrect executions, a loop exists to repeat an activity until the execution of the activity is correct. Then, the value for the same data element may be determined multiple times through multiple executions of the same activity.



(a) Unknown number of instances of data element 'grade subject'.

(b) Two instances of data element 'grade subject'.

Figure 4.7: *Illustration of multiple instances for the same data element in the PDM. In (a) it is undefined how many instances of 'grade subject' occur for a specific case. In (b) the existence of two sepatate instances of the data element 'grade subject'.*

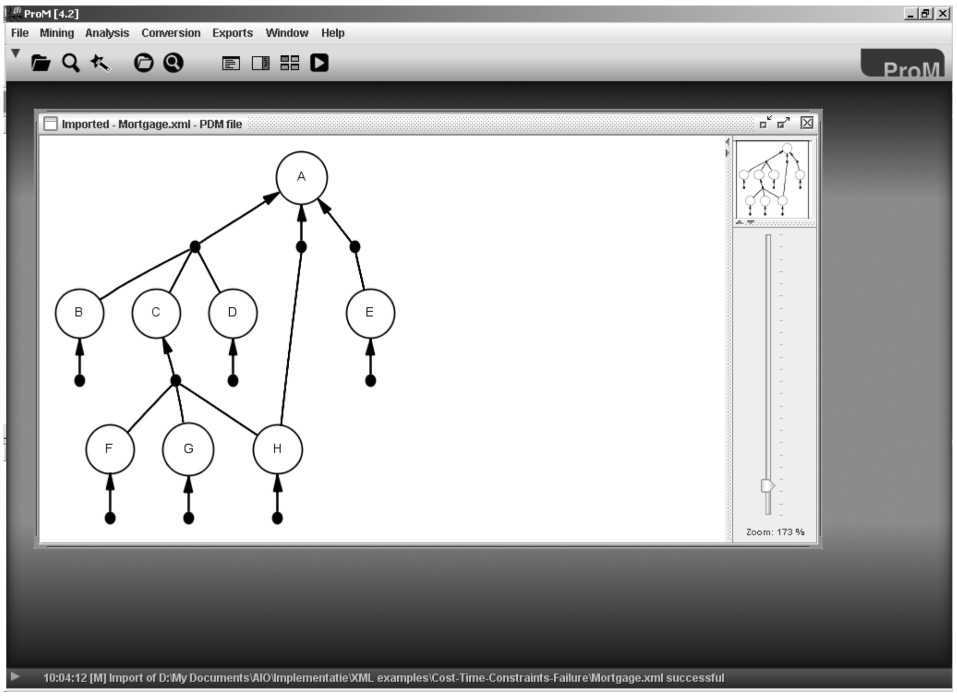
Under Assumption 4.3.4, this is not allowed. Loops related to multiple instances can neither occur, since we assume that multiple instances of the same data element for a single case are not allowed (Assumption 4.3.5).

The ideas presented throughout the remainder of this thesis are based on the assumptions that have been explained in this section. Based on the formal definition of a PDM and the above assumptions we have developed tool support for the PDM in the ProM Framework. This tool support is explained in the next section.

4.4 Tool Support

To represent a PDM in ProM, we have developed a data structure for the PDM. With this data structure it is possible to define a specific PDM, to load it in the ProM Framework, and to represent it in a graphical way. Moreover, several analysis and conversion plugins are available for a PDM. The functionality of these plugins is described in the next chapters. Figure 4.8 shows a screenshot of the visualization of a PDM object in the ProM Framework.

The format of the data structure for a PDM object is defined by an XML schema definition. Figure 4.9 shows a graphical representation of this format. The complete schema definition is included in Appendix A.1. A PDM always contains a list of data elements, a list of resources, a list of operations, and a root element. The root element is one of the elements from the list of data elements and is therefore implemented as a reference to this specific data element. Each operation has zero or more input data elements (also implemented as references to the list of data elements) and exactly one output data element (implemented as a reference). Moreover, an operation can have several optional attributes associated to it, e.g. execution cost, processing time, and failure probability. This list of attributes is extensible by the field 'Data'.



```
10:04:12 [M] Import of D:\My Documents\AIO\Implementatie\XML examples\Cost-Time-Constraints-Failure\Mortgage.xml successful
```

Figure 4.8: *The visual representation of the PDM for the mortgage example, defined by an XML file (see Appendix A.2) based on the XML schema definition presented in Figure 4.9 and Appendix A.1.*

4.5 Related Work

In this section, we focus on related work on formal definitions of workflow product structures and on the verification of process models. Several approaches exist to verify whether a process model is a correct model. The first set of approaches we discuss verify the structural correctness of a model. The second set of approaches focus on the verification of the data flow in the process model.

4.5.1 Formal Definitions of Workflow Product Structures

Van der Aalst [5] gives a formal definition of a workflow product structure based on the notion of a Bill-of-Material (cf. Section 3.5.2). His formal definition consists of a set of *components*, a special root component, and for each component three sets of related components, i.e. a set of mandatory components, a set of optional components, and a set of choice components. The product structure is represented by a tree.

Reijers et al. [233] formally define a *product/data model* (cf. Section 3.5.2). This model contains a set of data elements, a set of constraints, a function pre that gives the set of input elements for a data element, a set of production rules based on the pre function, and a number of functions that associate constraints ($constr$), flow time ($flow$), cost (cst), and probability ($prob$) to each production rule. The model is not represented by a tree, but by a network.

The definition of Reijers et al. has been the basis of the formal definition of the PDM in this thesis. We have extended it with the notion of leaf operations and resources.

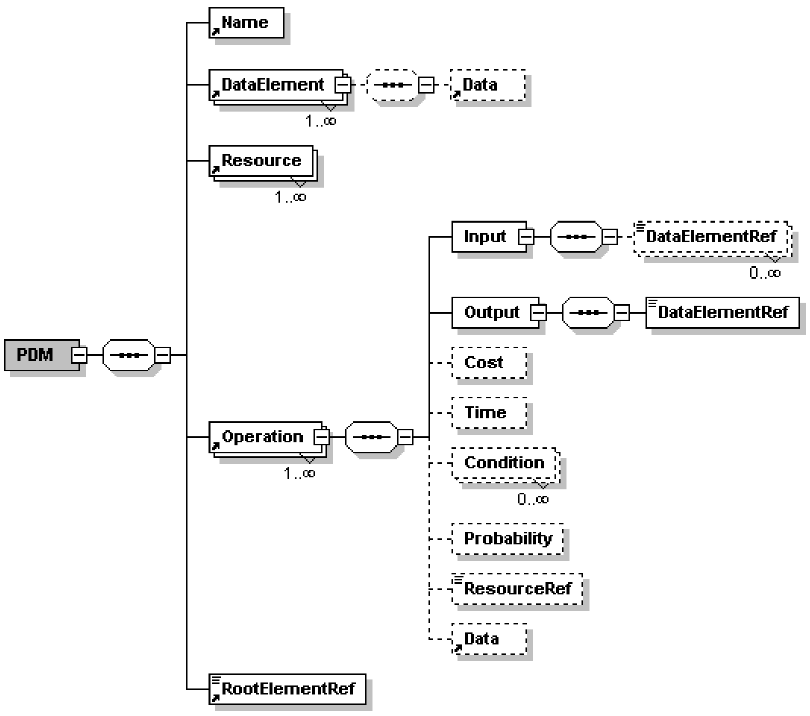


Figure 4.9: *A graphical representation of the XSD schema for a PDM XML file. The schema describes the structure of the XML file containing the PDM. Note that an operation contains exactly one output element and zero or more input elements. The attributes specifying the execution cost, execution time, etc. of the operation are optional.*

4.5.2 Soundness Verification

Soundness is a property of a process model that indicates whether the structure of the process model is correct. It does not require domain knowledge of the process and it guarantees the absence of a number of structural problems in the process model such as livelocks and deadlocks. A process model can either be sound or unsound; there is nothing in between. For process models represented by WF-nets, soundness is the main correctness criterion. A number of variants of the classical notion of soundness for WF-nets have been developed, e.g. k-soundness [124, 125], weak soundness [169], relaxed soundness [80, 81, 95], and lazy soundness [224, 225] (see also Section 2.4.2). An overview of these soundness notions for workflow nets can be found in [14]. The soundness notion also plays an important role in the verification of models designed in other modeling languages, such as EPCs [83] and YAWL models [304, 305].

Our correctness notion does not focus on the structural properties, but on the data dependencies in the process model. These two perspectives are orthogonal. Therefore, a sound process model can still contain problems with respect to the data flow, while a correct process model with respect to the data dependencies may not be sound.

4.5.3 Data Flow Verification

Sun et al. [267] provide a data flow perspective for systematically discovering errors in a workflow process model. They first present a method for specifying the data flow in a process model based on the *write* and *read* operations on data elements in each activity. Next, a formal approach to analyse the data flow in a process model based on the so-called "dependency analysis" is proposed. Three kinds of *data flow anomalies* can be detected with the framework: (i) missing data, (ii) redundant data, and (iii) potential data conflicts. Missing data anomalies occur when a data item is accessed before it is initialized. A redundant data anomaly exists when an activity produces data items that do not contribute to the production of the final output data. Finally, conflicting data anomalies occur if different versions of the same data element exist in the same instance of the process. A process model has a correct data flow specification if it is free of these data flow anomalies.

We have adopted the input/output data perspective and the notion of missing data anomalies to verify the correctness of a process model for a PDM. We have simplified the approach and abstracted from e.g. conditional input data (i.e. data that is not necessarily needed for the execution of an operation). In our case, we consider redundant data as unproblematic. In a process model based on a PDM, an intermediate data element may be used to produce the final output of an activity. At the same time, it may also be an input to another activity. Therefore, intermediate data elements are counted as output, but are not necessarily used as input to a different activity, since they can just be an input element to an operation in the same activity. Conflicting data anomalies can never occur in our approach since we assume that operations producing a value for a data element that already exist are skipped (cf. Assumption 4.3.4).

Finally, Sun et al. require that if two activities produce the same output element, the activity using this element as an input should be dependent on both activities producing it. In our approach it suffices if there is one of both activity dependencies for each execution trace.

4.6 Summary

In this chapter, a formal definition specifying a PDM has been presented. It consists of four sets (data elements, operations, conditions, and resource classes), a special root element, and five functions that map the cost, processing time, conditions, failure probability and resource class to each operation. Moreover, we have introduced a formal approach to verify the correctness of a process model based on a PDM, using the data-dependencies from the PDM and a method for data flow verification. The formal basis of a PDM is used to develop an XML data structure to store a PDM and to provide tool support for product based workflow design in Chapter 5 and the direct execution of a PDM in Chapter 6.

Chapter 5

Derivation of Process Models

In Chapter 3 we have seen how a process model can be manually derived from a PDM by grouping operations and their data elements together in activities. We have also seen that current workflow technology is not (yet) capable of supporting this product-based *design* process. In this chapter, we focus on the automatic generation of process models from a PDM. We have developed seven different algorithms to derive a process model from a PDM. Such a process model can be used to deploy a PDM using current workflow technology.

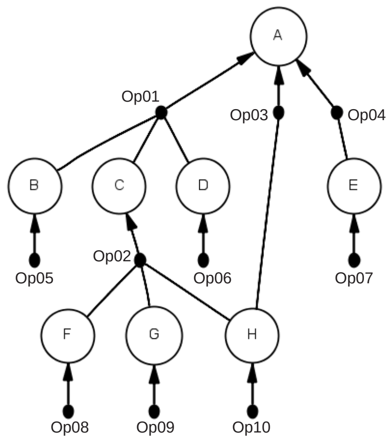
First of all, a classification framework for algorithms which generate process models from PDMs is presented in Section 5.1, followed by a description of these algorithms in Section 5.2, and a comparison and an evaluation of the algorithms in Section 5.3. Section 5.4 discusses the applicability of this approach. Next, in Section 5.5, an explanation of our tool support for the automatic generation of process models is given. The chapter concludes with a section on related work and a summary.

5.1 Classification Framework

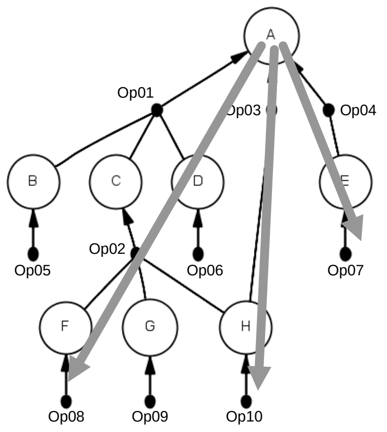
The seven algorithms we have developed can be classified based on their characteristics. In the framework that we use to classify the algorithms, two perspectives can be distinguished: the *construction* perspective and the *process execution* perspective. The former perspective focuses on the method of constructing a process model from a PDM (i.e. the point of view of a process designer), while the latter describes the runtime behavior of the process model that is generated (i.e. the point of view of a user). Below, these two perspectives are further explained.

5.1.1 Construction Perspective

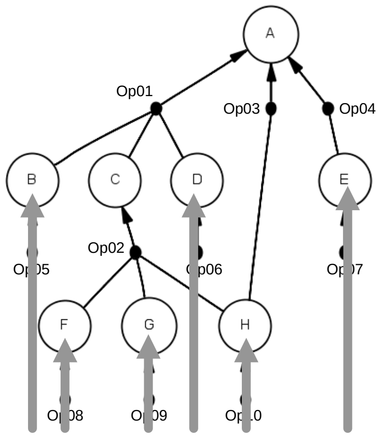
In the construction perspective we look at how a process model can be constructed based on a PDM. We take the viewpoint of a process designer and focus on the translation of the concepts of a PDM to a process model. Within this perspective three dimensions are distinguished: representation, order and focus.



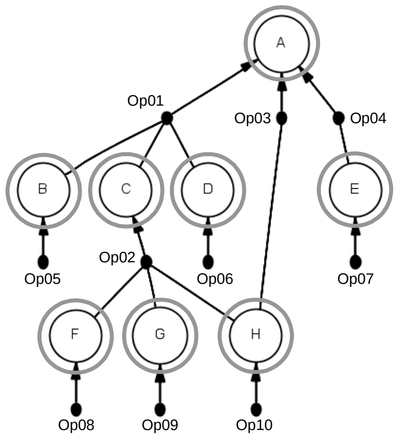
(a) The PDM of the mortgage example. See also Figure 3.3.



(b) A top-down order of walking through the PDM: starting from the root element walking to the leaf elements. In this PDM three alternative branches can be recognized to produce data element A.



(c) A bottom-up order of walking through the PDM: starting from the leaf elements walking to the root element.



(d) A middle-out order of walking through the PDM: the data elements and operations are selected in an arbitrary order.

Figure 5.1: *A visualization of the three different orders (see (b), (c), and (d)) in which an algorithm can translate the PDM of (a) to a process model.*

Representation

The first dimension is the modeling language in which the process model is *represented*. Each modeling language has its own features and limitations. Sometimes the workflow system to be used for enacting the process model restricts the process designer to use certain constructs or does not provide support for specific workflow patterns [16], e.g. OR-splits/OR-joins and cancellation regions. For instance, if Petri nets are used as the modeling language of the system, it is laborious to design a model in which parts are cancelled by the execution of a particular activity. In the development of our algorithms we only used Petri nets and YAWL since they have a formal basis, including a notion of soundness [2, 3, 6, 14, 304, 305]. The resulting models can be easily translated to other modeling languages such as BPMN or EPCs. For instance, a Petri net can be converted to an EPC [286], and the EPC can be loaded into a workflow system such as ARIS.

Order

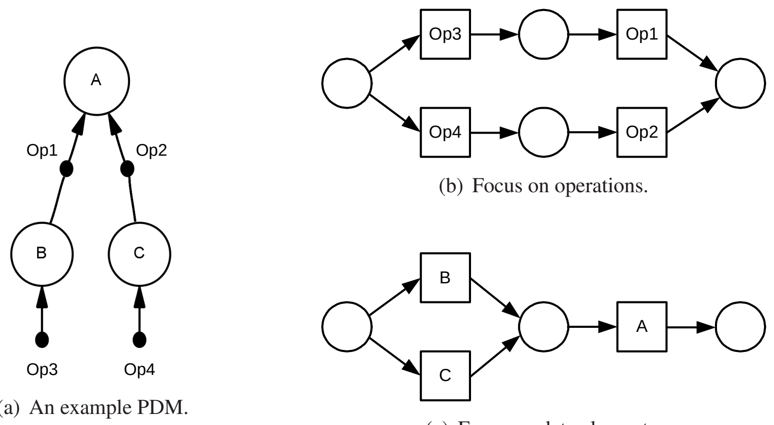
The second dimension in the construction perspective is the *order* in which the algorithm translates the PDM to a process model. A process model is designed based on the dependencies between data elements and operations in the PDM. To build the process model each algorithm 'walks' through the PDM in one of the following ways (see also Figure 5.1 for a graphical explanation):

· *Top-down* - We start at the top of the PDM, i.e. with the root element, and walk towards the leaf elements by following the dependencies in the PDM (see Figure 5.1(b) for an illustration of this top-down order). This 'backward chaining' leads to the determination of all minimal execution paths of the PDM which contain no superfluous operations. A *minimal execution path* is a path in the PDM in which all operations contribute to the determination of the end product, i.e. if one of the operations would be left out the end product cannot be produced anymore based on the given path. For instance, walking through the example PDM of Figure 5.2(a) in a top-down order gives two minimal execution paths based on the two alternative operations that produce a value for the root element A: $Op1$, and $Op2$. Thus, the set of minimal execution paths for the example PDM is:

$$S = \{\langle Op3, Op1\rangle, \langle Op4, Op2\rangle\}.$$

Note that other execution paths, such as $\langle Op3, Op4, Op1\rangle$ and $\langle Op4, Op3, Op1\rangle$, are possible in this example PDM. However, these execution paths are not minimal, since they contain superfluous operations, e.g. in $\langle Op3, Op4, Op1\rangle$ the execution of $Op4$ is superfluous since the output data element of the operation is never used in any further operation. Thus, with a top-down order we 'climb down the tree (or graph)' by looking at the output element and following the links via operations to its input elements, until we reach the leaf elements.

· *Bottom-up* - We start with the leaf elements and walk towards the root element (see Figure 5.1(c)). Based on the data elements for which a value is available, it is determined which operations are executable and which new data element values can be determined. For example, in the PDM of Figure 5.2(a), we can start from a situation in which no data element values are available. Then, all leaf operations are executable, i.e. $Op3$ and $Op4$. Now suppose we first select $Op3$ and produce a value for data element B. The other leaf operation $Op4$ is still executable and now also operation $Op1$ has become executable. Suppose we select $Op4$. Then $Op1$ and $Op2$ are executable, etc. Using this approach, we can 'climb up



(a) An example PDM.

(b) Focus on operations.

(c) Focus on data elements.

Figure 5.2: *This figure shows two process models (b) and (c) for an example PDM (a). In the construction of the process model in (b) the focus has been on operations, i.e. each transition represents the execution of one operation in the PDM. In the construction of the process model in (c) the focus is on data elements, i.e. each transition represents the determination of a value for a specific data element. Thus, transition A contains two operations: $Op1$ and $Op2$ which both produce a value for A. During execution of the process model only one of these two alternative operations is performed, based on the available input data element, i.e. B or C.*

the tree (or graph)' from the input elements via the operations to their output elements. New steps are based on the available data elements. In general, the bottom-up order of walking through the PDM gives all possible execution paths of the PDM, i.e. all orders in which the operations in the PDM can be executed and data element values can be produced, starting from no data element values at all.

· *Middle-out* - Data elements and/or operations are translated to transitions in the process model in an arbitrary order, not following the dependencies in a top-down or bottom-up manner (see Figure 5.1(d)). The links between data elements and operations are only added to the model later. In Figure 5.2(a), for instance, the order in which the data elements are translated to transitions in the process model is not dependent on how they are related to each other through input/output data relations of the operations. They can be e.g. translated in the order C, A, and B as well as in the order A, B, and C.

Focus

The last dimension of the construction perspective is the *focus* of the translation. There are two important concepts in a PDM: data elements and operations. If the focus of the translation is on the data elements, the activities in the process model represent the data elements of the PDM. If the focus is on the operations, the activities represent the execution of an operation. In Figure 5.2 a graphical explanation of the focus dimension is shown.

5.1.2 Process Execution Perspective

The second perspective in the classification framework is the process execution perspective. This perspective focuses on the runtime behavior of the process model that is derived from the PDM. This behavior is a direct result of the structure of the process model. Typically, this is the perspective of the user of the process model. We distinguish three dimensions that describe the runtime interpretation of the process model: moment of choice, eagerness, and concurrency.

Moment of choice

The first dimension of the process execution perspective is the *moment of choice* in the process model. At some point in time during the execution of a process model, it has to be decided which of the alternative operations (if any) is executed to produce the value for a specific data element, e.g. in the mortgage example the value for A can be determined in three ways (see Figure 5.1(a)). Possibly, the choice for an alternative route is already made at the start of the process before any data element values are determined. If that is the case, we say that the moment of choice is *early*. In contrast, it is also possible to defer choices (as long as possible). Then, we say the moment of choice is *late*. Figure 5.3 shows an example of a process model with an early choice between two alternative activities and one with a late choice.

Eagerness

The second dimension in the process execution perspective is the *eagerness* of the execution. During execution the values for several data elements are determined. The process model can have a structure in which it is encouraged to produce as many data element values as possible (cf. a breadth first search strategy), but it can also be structured to only produce values that are strictly needed to determine the value of the end product, i.e. none of the data element values can be left out (cf. a depth first search strategy). We distinguish three degrees of eagerness:

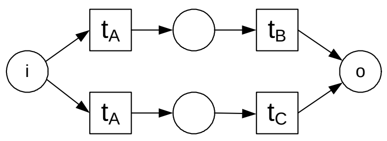
· *Strict* - Only those data elements are determined that directly contribute to the determination of the end product. Strict eagerness is related to minimal execution paths of the PDM because the set of data elements to be determined in a process model with strict eagerness is based on the data elements in a minimal execution path.

· *Overcomplete, but restricted* - More data element values may be determined than strictly needed for the determination of a value for the end product. However, when a value for the end product is produced, the execution of the case ends and no more data element values are determined[1].

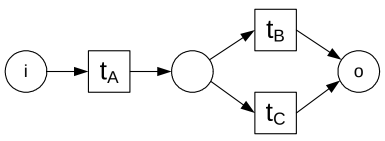
· *Overcomplete and unrestricted* - More data element values than strictly needed for the determination of the end product are produced. Even after the end product is produced, it cannot be excluded that the execution of the case may continue to produce more data element values[1].

Figure 5.4 explains the differences between these three degrees of eagerness. In a situation with strict eagerness, a minimal execution path of the PDM is chosen. All operations that are

[1]Note that an overcomplete degree of eagerness may also lead to multiple productions of the same data element. Since we have assumed that the value of a data element will be available for the entire duration of a case and cannot be overwritten or updated (cf. Assumption 4.3.4), the execution of another operation leading to a value for the same data element is superfluous and is therefore skipped. Thus, executing such an operation may lead to extra execution cost and processing time, but actually does not add new information.

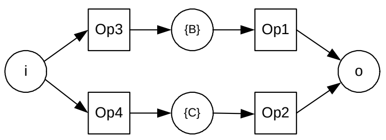


(a) Early choice.

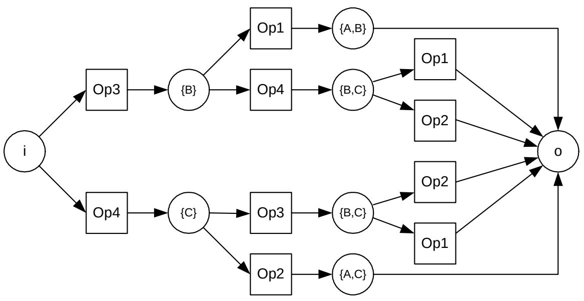


(b) Late choice.

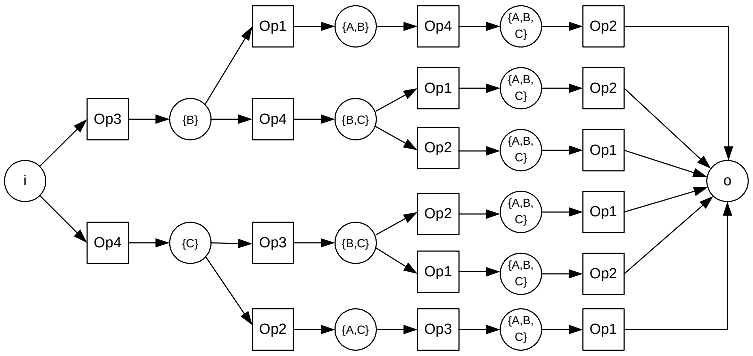
Figure 5.3: *Both process models in (a) and (b) have the same set of firing sequences, i.e.* $S = \{\langle t_A, t_B\rangle, \langle t_A, t_C\rangle\}$*. However, in the first example (a) it has to be decided before* t_A *is executed whether* t_B *or* t_C *will follow on the execution of* t_A*. In the second example (b), this choice is deferred to the moment when* t_A *has been executed. We say that the moment of choice in the first model is* early*, and in the second model is* late.



(a) Eagerness: Strict.



(b) Eagerness: Overcomplete but restricted.



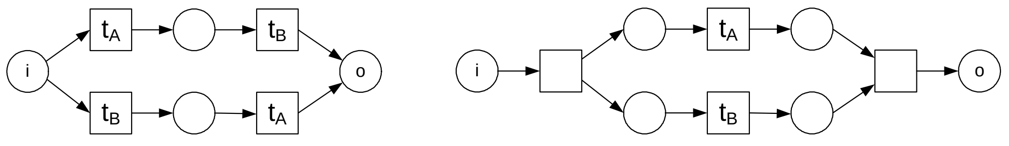
(c) Eagerness: Overcomplete and unrestricted.

Figure 5.4: *Example process models based on the example PDM in Figure 5.2(a). In the process model of (a) either a value for data elements* A *and* B*, or* A *and* C *is determined. The eagerness is strict because no superfluous data element values are produced. The process model of (b) has an overcomplete but restricted eagerness. For instance, when a value for* B *has been determined, it is still possible to also determine a value for* C *although this value for* C *is not strictly needed to produce a value for the root element* A*. However, once a value for the root element* A *is determined no more data element values can be produced. Finally, the process model in (c) is overcomplete and unrestricted. Like in the process model of (b) it is possible to determine data element values that are superfluous, but in this case it is also possible to continue producing data element values after a value for the root element* A *has been determined until all operations are executed once.*

executed and all data elements that are produced are necessary steps in this execution path. With the production of more data element values than necessary in the other two degrees of eagerness, also superfluous data element values may be determined that are not used in further operation executions[2]. The production of these values then does not contribute to the determination of the end product. This may, for instance, lead to extra execution cost because of the extra operations that are executed. However, producing more values than strictly necessary may be desirable if it is not known a priori which path through the PDM is to be followed to produce the end product, or if it is expected that the planned path fails and an alternative path should be followed.

Concurrency

The last dimension in the process execution perspective is the *concurrency* of activities in the process model. If there is concurrency in a process model, there is a possibility to simultaneously execute several activities of the process model for the same case. Concurrency is usually modeled by parallel branches after an AND-split in the process model. Based on the structure of a process model, concurrent activity executions either are possible or impossible. Figure 5.5 shows the difference between a process model with and without concurrency[3].



(a) No concurrent executions possible. (b) Concurrent executions possible.

Figure 5.5: *The process model in (a) does not allow for concurrent execution of activities for one case. Before any of the activities is executed it has to be decided in which order they are executed: either t_A is executed before t_B or t_B is executed before t_A. In the second process model in (b) concurrent execution of activities for the same case is possible. Activities t_A and t_B are in parallel branches and can be enabled at the same time. It is not specified which of the two operations should be executed first. They can even be executed simultaneously.*

In Figure 5.6 a graphical representation is shown of the two perspectives and the resulting six dimensions used by the classification framework. The different dimensions in this classification framework are partly dependent on each other. For instance, choices in the execution perspective may influence the options left in the construction perspective. If a top-down order is chosen in the construction perspective, the minimal execution paths are immediately distinguished (see Figure 5.1(b)). Since these minimal paths describe the alternative routes to determine a value for the end product without producing superfluous data element values, one

[2]Note that an overcomplete degree of eagerness may also lead to multiple productions of the same data element. Since we have assumed that the value of a data element will be available for the entire duration of a case and cannot be overwritten or updated (cf. Assumption 4.3.4), the execution of another operation leading to a value for the same data element is superfluous and is therefore skipped. Thus, executing such an operation may lead to extra execution cost and processing time, but actually does not add new information.

[3]Note that we assume that the process model has an initial marking in which one token is present in the start place, i.e. $[i]$ (cf. Section 2.4). Then, the parallel branches in the process model indicate the possibility of concurrent execution of several activities for the same case.

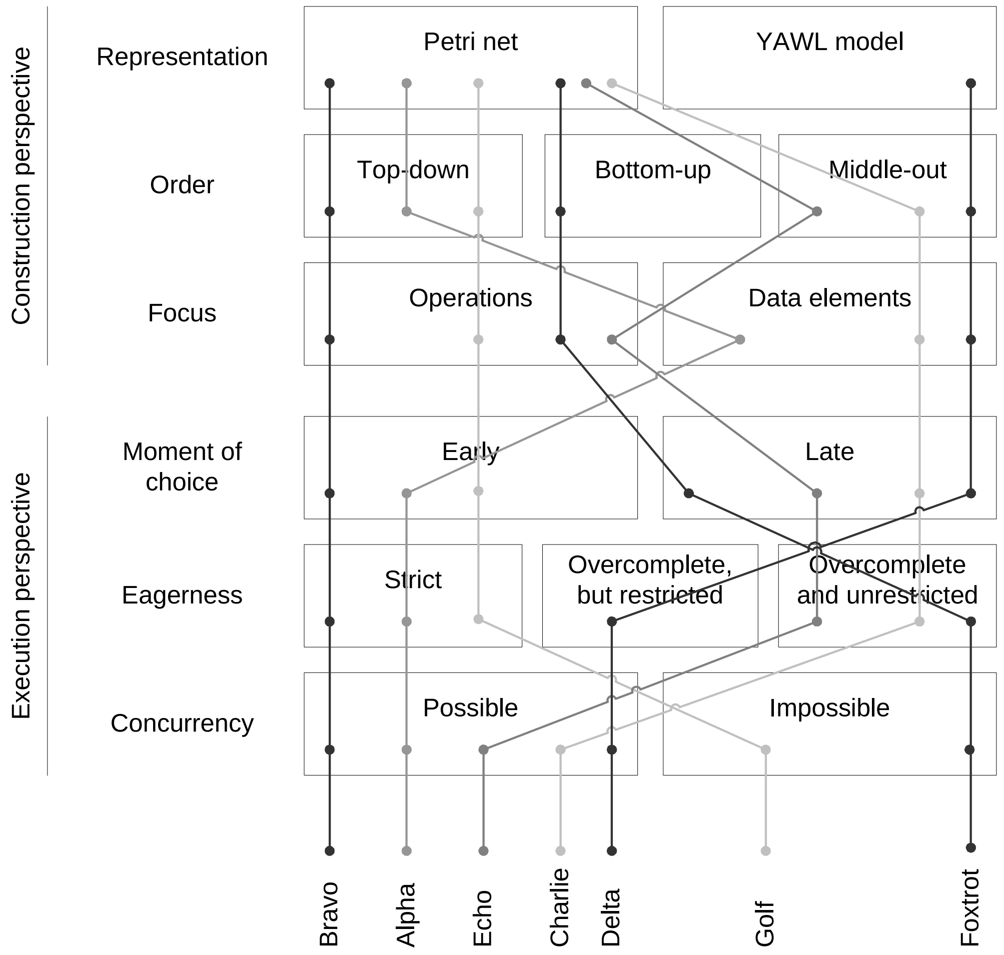


Figure 5.6: *Classification of the seven algorithms based on the six dimension in the classification framework: representation, order, focus, moment of choice, eagerness, and concurrency.*

path has to be chosen at the start of the process. This implies that the moment of choice in the execution perspective is early, and that the eagerness of execution is strict.

Also, some of the choices within each of the perspectives are related. When the moment of choice is set to be late, for example, it is unavoidable to choose the eagerness of execution to be overcomplete (i.e. either overcomplete, but restricted, or overcomplete and unrestricted). The determination of superfluous data element values cannot be ruled out because it is not known at the start of execution which alternative path is chosen.

The presented framework is a basis for the descriptions of the algorithms in the next section. It describes the dimensions on which the algorithms vary. However, we do not claim it is complete. For instance, on the dimension of the representation there are clearly more possible values besides Petri nets and YAWL models, e.g. EPCs, BPEL, BPMN. Also, with the development of new algorithms some extra dimensions (or even an extra perspective) may emerge. Nevertheless, the current framework covers all differences and perspectives we could think of when developing the algorithms. This development has been an incremental process, as is explained in the next section. The issues that we found in one algorithm are solved in the next algorithm that was developed. In the next section, the seven algorithms are explained in more detail.

5.2 The Algorithms

We have developed seven algorithms to automatically derive a process model from a PDM. These algorithms focus on the direct translation of data elements and operations to a process model. In contrast to the manual derivation of a process model described in Section 3.3.2, an activity is only related to one operation or one data element and does not represent a group of operations and their data elements. However, this does not mean that we loose generality with these algorithms. After generating a process model it is still possible to group activities to sub processes in the process model. This would result in a similar structure compared to the process model obtained by first grouping operations and their data elements.

In this section, a short description of each algorithm is given, illustrated with a diagram of the resulting process model for the mortgage example PDM of Figure 5.1(a). Next to that, its place in the framework is explained (cf. Figure 5.6). An explanation of the incremental development process of the seven algorithms is given after the descriptions, because the reader needs to have some knowledge of each of the algorithms to understand how one algorithm has led to the development of another. More information on the algorithms and their classification can be found in [142, 143] and in Appendix B.

5.2.1 Algorithm Alpha

Algorithm Alpha is based on the algorithm introduced in [5]. It has been adapted to the changed notation of a PDM (see Section 3.5.2) and it has been extended to support alternative ways to produce a data element value. The algorithm builds the process model with a top-down order and a focus on data elements. The process model is represented as a Petri net.

Algorithm Alpha starts with the root element of the PDM. It creates a start and an end place and a transition for the root element. The start place is connected to the transition and the transition is connected to the end place. Then, this transition is substituted with a *sub process* describing the production of the root element. Each input data element of an operation producing the root element is translated to a new transition. Alternative operations to determine the root element are translated to implicit choices in the Petri net. If several input elements are needed for the same operation, then they are translated to parallel branches in the sub process. Next, the transitions for the input elements of operations producing the root element are substituted by sub processes in the same way. This substitution process is repeated until the leaf data elements are translated to transitions. Figure 5.7 shows the process model for the mortgage example generated by algorithm Alpha. A step-by-step explanation of the algorithm using a simple example is presented in Appendix B.1.

As can be seen from the structure of the process model in Figure 5.7, the moment of choice is early, since all minimal execution paths are elaborated separately (see also Section 5.1.1). The eagerness is strict, because no superfluous data element values can be produced in a minimal execution path. Finally, concurrent execution is possible since e.g. transitions F, G, and H are in parallel branches.

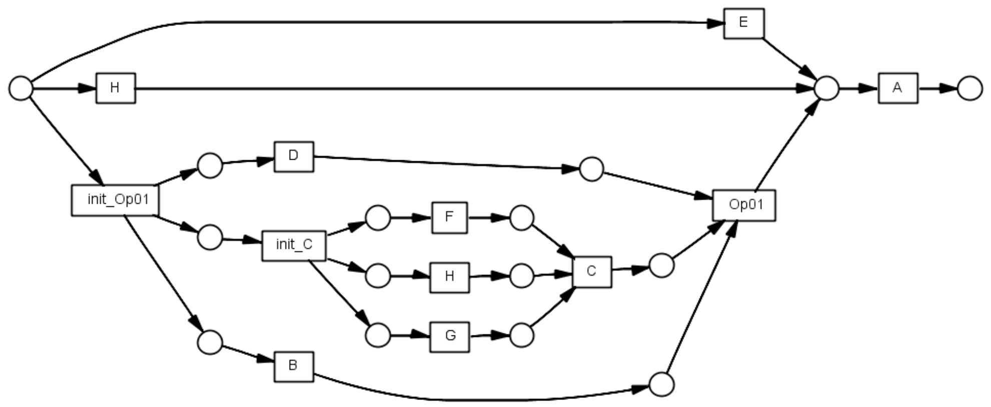


Figure 5.7: *The process model for the mortgage example (of figures 3.3 and 5.1(a)) generated by algorithm Alpha. In the first step transition A is added. Then, the transitions for the input elements to A are created, i.e. E and H and $init\_Op01$ and $Op01$. In the next step, the transitions B, D, and $init\_C$ and C are added. Finally, transitions F, G, and H, which are needed to produce C, are created.*

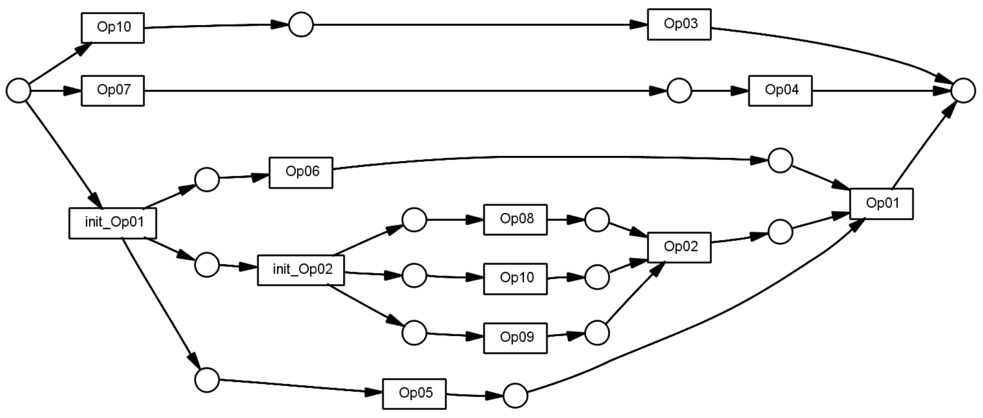


Figure 5.8: *The process model for the mortgage example generated by algorithm Bravo. First, one of the operations producing the root element is selected, e.g. $Op03$, and a transition is created for this operation. Then, the operation producing the input element of $Op03$, i.e. $Op10$ is selected and a transition is created. Now, one top-down execution path is finished and another operation producing the root element is selected, e.g. $Op04$. Transitions $Op04$ and $Op07$ are added. Then, the third path is elaborated top-down. First transitions $Op01$ and $init\_Op01$ are added. Then $Op06$, $init\_Op02$, $Op02$, $Op08$, $Op10$, $Op09$, and finally $Op05$.*

5.2.2 Algorithm Bravo

Algorithm Bravo also generates a Petri net and has a top-down order of constructing the process model from the PDM, like algorithm Alpha. However, it focuses on the operations instead of on the data elements.

The algorithm starts with only a start and an end place. Then, a transition is created for one of the operations that produce a value for the root element. This transition is connected to the end place. Then, the algorithm looks for an operation that produces an input element to this transition. If this operation can be found, a new transition is added to the model and this procedure is repeated until the leaf operations are reached. Then, a next operation producing the root element is chosen. This top-down execution path is also elaborated until the leaf operations, and so on. Thus, the different execution paths are elaborated separately starting from the root element taking a depth-first approach instead of a breadth-first approach to select operations, in contrast to algorithm Alpha. In Figure 5.8 the process model for the mortgage example is shown. A stepwise explanation of this algorithm can be found in Appendix B.2.

Similar to the result of algorithm Alpha, in the process model generated by algorithm Bravo choices are made early, the eagerness is strict, and parallel execution is possible.

Since algorithms Alpha and Bravo do not necessarily provide sound and correct process models (see sections 5.3.2 and 5.3.3), we started looking for a different (and simpler) translation of the PDM to a process model in which the operation and data dependencies play a less prominent role. In the next three algorithms therefore a middle-out order of constructing the process model is used.

5.2.3 Algorithm Charlie

Algorithm Charlie constructs the process model in a middle-out order with a focus on data elements. It creates a Petri net.

As a first step, all data elements are translated to separate components of the process model, consisting of a transition named after the data element with one input and one output place. Next, these components are connected to each other by transitions corresponding to operations. The input places for each transition that is related to an operation are the output places of the transitions that represent the input data elements of the operation. The output place of the transition related to an operation is the input place of the transition representing the output element of the operation. Finally, an initialization transition is added and connected to all input places of transitions representing the leaf elements of the PDM. Figure 5.9 shows the process model generated by algorithm Charlie. A step-by-step elaboration of a simple example using this algorithm is presented in Appendix B.3.

The resulting process model allows for late choices, since it e.g. only has to be decided just before the execution of A which input is used to produce A. The eagerness is overcomplete and unrestricted because it is possible to e.g. determine the value for F, G, and H and then execute $Op03$ and A, which makes the values for F and G superfluous. Finally, the concurrent execution of transitions is possible, e.g. transitions B, D, E, F, G, and H can be enabled at the same time.

Unfortunately, the process models generated by algorithm Charlie may have some problems related to their behavior. In some situations, a deadlock may occur or tokens may be left in the model after the determination of a value for the end product. Section 5.3.2 elaborates on these issues. To overcome these problems, algorithm Delta was developed.

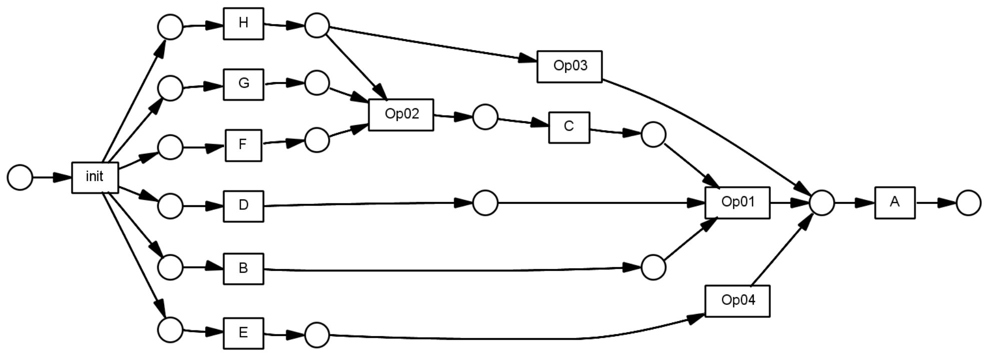


Figure 5.9: *The process model for the mortgage example generated by algorithm Charlie. First, all transitions related to data elements (A, B, C, D, E, F, G, and H) are created in an arbitrary order, including their input and output places. Then, transitions $Op01$, $Op02$, $Op03$, and $Op04$ are added. The output places of transitions B, C, and D are connected to transition $Op01$ such that they become an input place. Similarly, the output places of E, H, and F, G, and H are connected to $Op04$, $Op03$, and $Op02$ respectively. Then, the transitions related to operations are connected to the input place of the transition representing the output element of the operation: $Op02$ is connected to the input place of C and $Op01$, $Op03$, and $Op04$ are connected to the input place of A. Finally, the $init$ transition is created and connected to the input places of B, D, E, F, G, and H.*

5.2.4 Algorithm Delta

The basis for algorithm Delta is the same as for algorithm Charlie. It is a top-down algorithm with a focus on data elements. However, algorithm Delta creates a YAWL model. Since the YAWL modeling language allows for cancellation regions, the problem of tokens that are left in the model could be solved elegantly.

First a component for each data element is added to the YAWL model. Like in algorithm Charlie, this component consists of a transition named after a data element with one input and one output place. Then, the components are connected through transitions representing the operations from the PDM. In addition to algorithm Charlie, some extra arcs are added to the model to prevent deadlock situations: the input places of a transition related to an operation are also output places for this transition, i.e. input data is *used but not consumed* by the transitions. Finally, the initializing transition is added, control places are added for each transition in the model and a cancellation region consisting of almost all places and transitions is added to the transition representing the root element (cf. Section 2.5). The process model for the mortgage example is shown in Figure 5.10. A step-by-step explanation can be found in Appendix B.4.

Similar to algorithm Charlie, the eagerness of the process model generated by algorithm Delta is overcomplete but restricted, because no actions can be taken anymore after the determination of a value for the root element. The cancellation region removes all tokens present in the net when the transition representing the root element fires. Concurrent execution is possible.

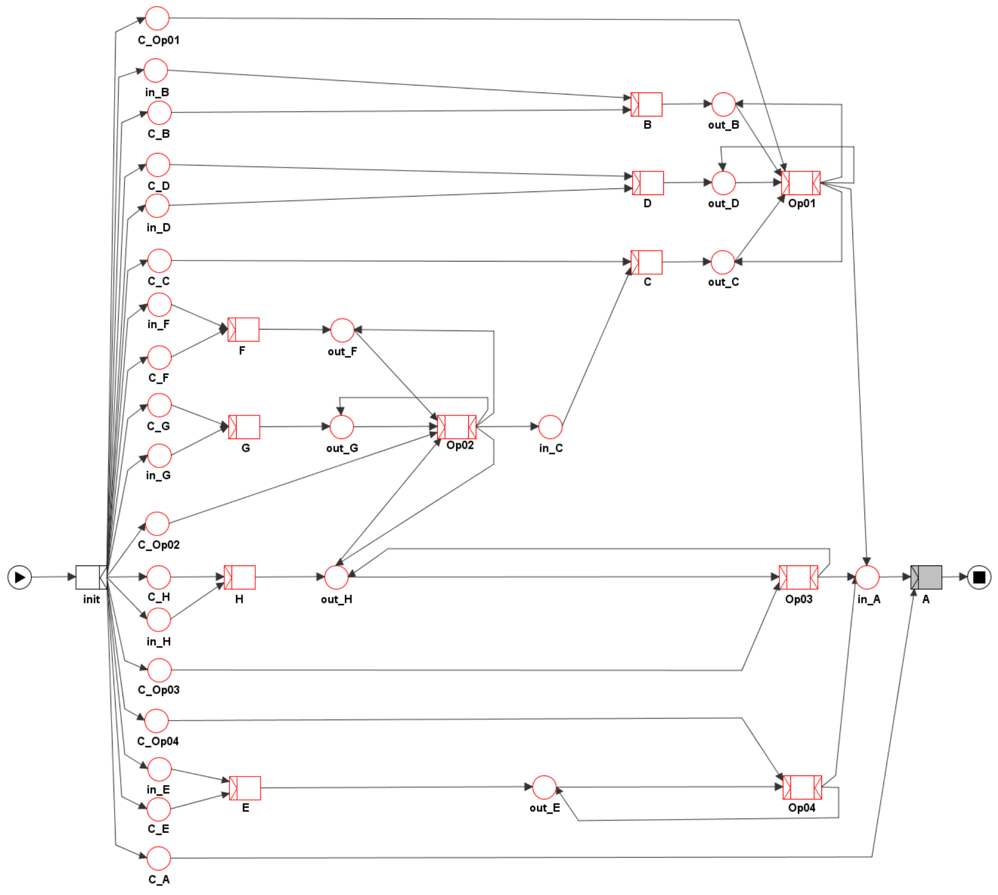


Figure 5.10: *The process model for the mortgage example generated by algorithm Delta. The YAWL Editor that we used to visualize the YAWL model does not show the cancellation region by a dashed line. Instead, the cancellation region (i.e. places and activities) of activity A is shown in red. Like in algorithm Charlie, first all transitions related to data elements (A, B, C, D, E, F, G, and H) are created, including their input and output places. Then, transitions $Op01$, $Op02$, $Op03$, and $Op04$ are added. The output places of transitions B, C, and D are connected to transition $Op01$ such that they become an input place. Similarly, the output places of E, H, and F, G, and H are connected to $Op04$, $Op03$, and $Op02$ respectively. Then, the transitions related to operations are connected to the input place of the transition representing the output element of the operation: $Op02$ is connected to the input place of C and $Op01$, $Op03$, and $Op04$ are connected to the input place of A. Also, an extra arc to its input place is added for each transition corresponding to an operation, such that the transition does not consume the tokens in its input places. Next, the $init$ transition is created and connected to the input places of B, D, E, F, G, and H. Finally, a control place (e.g. C_B, C_C, C_Op03) is added for each transition in the model and the cancellation region for transition A is set by creating cancellation arcs from A to all places and transitions (e.g. $Op02$, H, in_E, C_D), except for the start place, the end place, and the transitions $init$ and A.*

5.2.5 Algorithm Echo

Algorithm Echo is the final algorithm with a middle-out order of constructing the process model. It provides a Petri net, but in contrast to algorithm Charlie and Delta it mainly focuses on operations instead of data elements.

Algorithm Echo starts with the creation of an output place, and an input place connected to an initialization transition. Next, for each data element of the PDM, a transition with a single input place is added to the model. Then, also for each operation in the PDM a component is added. Such a component consists of a transition together with an input place for each input data element of the operation. If the operation does not have input elements, then one input place is added. Compared to the process models generated by algorithm Charlie, the introduction of such an input place for each input element of an operation leads to more places in the process model. Next, these input places represent the availability of a data element. They are connected as output places to the transition that produces the data element. Also, transitions which correspond to an operation are connected to the input place of the transition representing the output data element of the operation. Finally, the initialization transition is connected to the input places of all transitions that correspond to a leaf element, a control place to the transition that represents the root element is added and the transition corresponding to the root element is connected with the end place. Figure 5.11 shows the process model generated by algorithm Echo. A step-by-step explanation of the algorithm is available in Appendix B.5.

The moment of choice is late in the process model, the eagerness is overcomplete and unrestricted, and concurrent execution is possible in the process model. Like the process models generated by algorithm Charlie, process models generated by algorithm Echo also have some issues related to their soundness (see Section 5.3.2). Tokens may be left in the model after the determination of a value for the end product. However, the deadlock situations that may occur in the process models of algorithm Charlie cannot occur in process models generated by algorithm Echo.

The last two algorithms use a bottom-up approach to build the process model. Because these algorithms decide in a stepwise fashion which operation can be executed next, they have a strictly sequential structure.

5.2.6 Algorithm Foxtrot

The order of constructing the process model with algorithm Foxtrot is bottom-up. A Petri net is generated. The focus is on the operations in the PDM. Therefore, each transition corresponds to one operation. A place in the Petri net represents the data elements for which a value is available.

Algorithm Foxtrot starts at the bottom of the PDM with an empty set of data elements. The algorithm computes all executable operations, i.e. the leaf operations. Each executable operation is translated to a transition and an output place. The output place again represents the set of available data element values and the executable operations from this place are computed and translated to transitions. Places which represent the same set of data elements are mapped to each other. Finally, each place containing the root element is connected to the end place of the Petri net via a *silent transition*, i.e. a transition that is only used for a correct routing and does not have any content in the form of operations or data elements. In Figure 5.12 a fragment of the process model for the mortgage example generated using algorithm Foxtrot is shown. A step-by-step elaboration of the algorithm for a simple example is available in Appendix B.6.

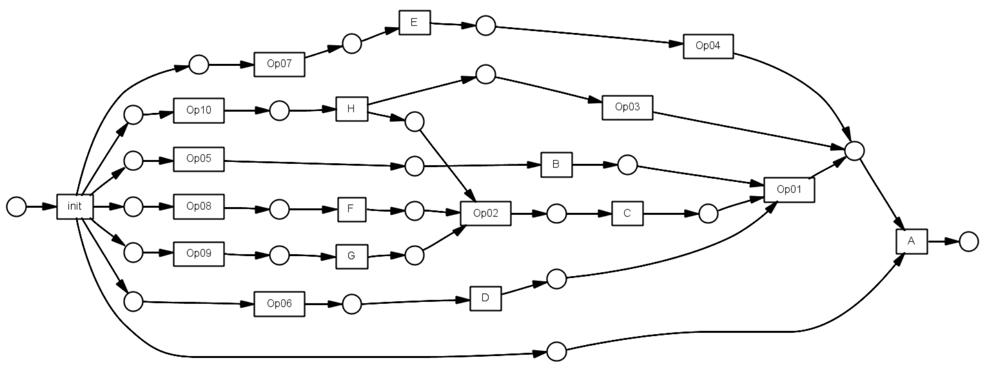


Figure 5.11: *The process model for the mortgage example generated by algorithm Echo. The input place, output place and initialization transition are created first. Then the transitions A, B, C, D, E, F, G, and H are created together with one input place each. Next, transitions $Op03$, $Op04$, $Op05$, $Op06$, $Op07$, $Op08$, $Op09$, and $Op10$ are added, each with one input place. Transitions $Op01$ and $Op02$ get three input places each since they both have three input data elements. Then, these input places become output places of the transitions that correspond to data elements, e.g. transition H is connected to the input place of transition $Op03$, transition C is connected to one of the input places of transition $Op01$, and transition B is connected to another input place of transition $Op01$. Next, each transition corresponding to an operation is connected to the input place of the transition that represents the output element of the operation, e.g. $Op10$ is connected to the input place of H, $Op02$ is connected to C, and $Op01$ is connected to A. Finally, the initialization transition is connected to $Op05$, $Op06$, $Op07$, $Op08$, $Op09$, and $Op10$. An additional place is added between $init$ and A, to make sure that A is occurring only once.*

Because all possible next steps are added from each place in the Petri net, choices are made late and more data element values than needed can be produced even after a value for the root element is determined already (the eagerness is overcomplete and unrestricted). However, we assume that an operation is only executed if its output element does not exist yet (cf. Assumption 4.3.4). Thus, a value for the root element cannot be updated or determined twice, even if a transition is executed which produces a data element value that already exists. Note that the result produced by algorithm Foxtrot is a completely sequential process model, since concurrent executions are not possible. The process model only contains XOR-splits and no AND-splits.

5.2.7 Algorithm Golf

Algorithm Golf first determines all minimal execution paths in the PDM in a top-down manner (see Section 5.1.1). Next, each execution path is constructed in the process model in a bottom-up order similar to algorithm Foxtrot. The difference with algorithm Foxtrot is that building the minimal execution paths substantially restricts the set of possible next steps from each place in the process model. The order of construction of the process model is top-down because the minimal execution paths are determined in a top-down manner first, before the construction of the process model is done bottom-up per path[4]. Like algorithm Foxtrot, the focus of construction is on the operations. Also, a Petri net is generated.

[4]Note that the Theory of Regions [92] could also be used to build a process model from the set of all minimal execution paths in the PDM [83].

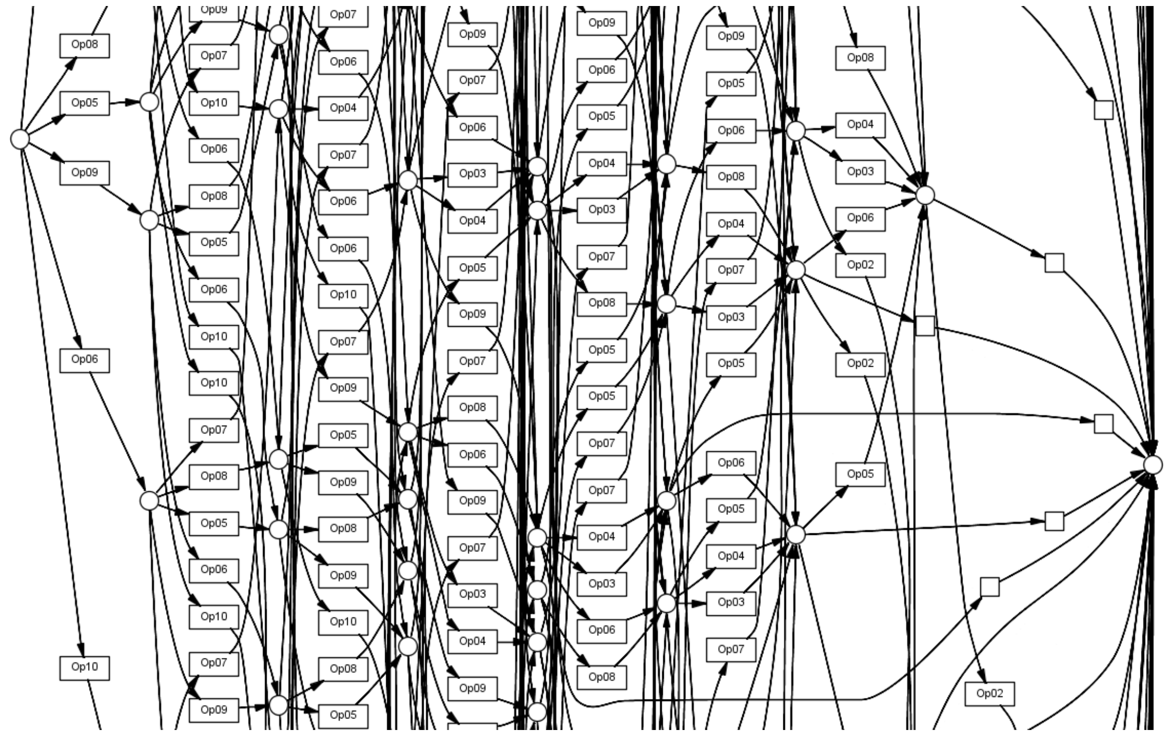


Figure 5.12: *A fragment of the process model for the mortgage example generated by algorithm Foxtrot. The complete process model contains 438 transitions. The algorithm starts from the situation in which no data element values are available, i.e. the start place. Then, the leaf operations, i.e. operations $Op05$, $Op06$, $Op07$, $Op08$, $Op09$, and $Op10$ are executable. A transition is created for these operations together with an output place, and the start place is connected to these transitions. Then, one of the transitions is selected, e.g. $Op06$. The output place of transition $Op06$ represents the situation in which a value for data element D is available. From this situation, operations $Op05$, $Op07$, $Op08$, $Op09$, and $Op10$ are executable. Therefore, the output place of $Op06$ is connected to new transitions $Op05$, $Op07$, $Op08$, $Op09$, and $Op10$ with their own output place. The output place of transition $Op09$ represents the situation in which data element values D and G are available. From this situation, operations $Op05$, $Op07$, $Op08$, and $Op10$ are executable, etc. This way, all possible execution paths are elaborated one by one in the model. Finally, the silent transitions are added to connect the places that represent the situation in which a value for the root element is available with the end place. Note that the execution of $Op09$ followed by $Op06$ leads to the same place as the execution of $Op06$ followed by $Op09$.*

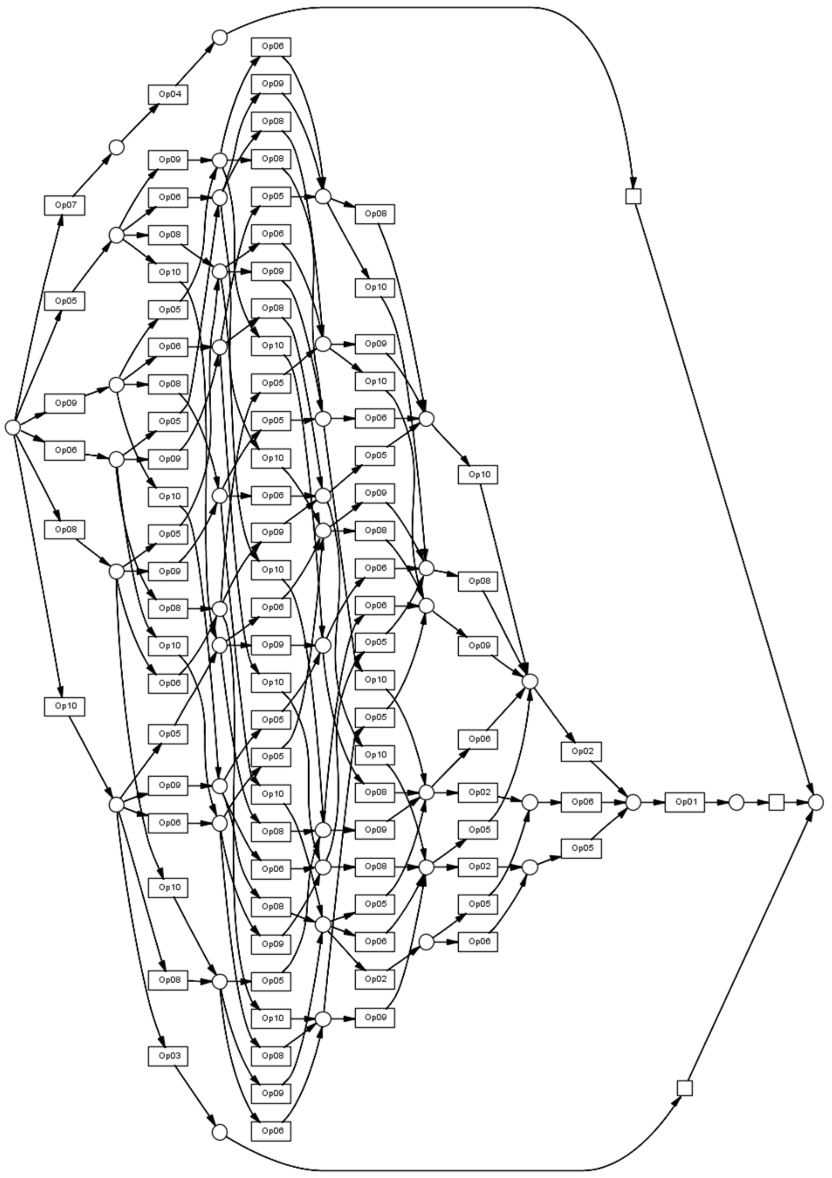


Figure 5.13: *The process model for the mortgage example generated by algorithm Golf. First, the minimal execution paths are determined in a top-down way. Then, one of these paths is selected, e.g.* $\langle Op07, Op04\rangle$*, and constructed in a bottom-up manner. First, a transition* $Op07$ *is created together with its output place. The start place is also connected to this transition. Then, a transition* $Op04$ *with an output place is created. The output place of* $Op07$ *is connected to* $Op04$*. And, since there are no other operations in the path, the output place of* $Op04$ *is connected to the end place with a silent transition. Then, another minimal execution path is selected, e.g.* $\langle Op09, Op08, Op10, Op06, Op02, Op05, Op01\rangle$ *and constructed bottom-up starting with a transition for* $Op09$*,* $Op08$*, etc. Then, another minimal execution path is selected and elaborated, and so on, until all minimal execution paths are in the model.*

In the first step, all minimal execution paths are determined top-down. Then these execution paths are constructed in the process model one by one in a bottom-up way. Again, each place represents the set of data elements for which a value is available. The transitions to which this place is linked as an input place represent the operations that are executable based on the set of available data elements. The process model for the mortgage example generated by algorithm Golf is shown in Figure 5.13. A step-by-step explanation of the algorithm can be found in Appendix B.7.

Note that because of the use of the minimal execution paths that are determined top-down, choices are made rather early in the process model, e.g. if $Op07$ is chosen as a first step, the only next step that is possible is $Op04$. However, when $Op10$ is chosen as a first step, there are more possibilities for a next step, since $Op10$ is needed in two of the three alternative branches to produce A. Because the alternative paths are determined in the first step of the algorithm no more data element values than strictly needed are produced. The bottom-up construction of each alternative path leads to a process model in which concurrent execution is not possible. Also note, that the process model generated by algorithm Golf is very similar to an elaboration of the parallel behavior in the process model generated by algorithm Bravo.

The development of these seven algorithms has been a creative trial-and-error process. Starting from the algorithm described in [5], algorithm Alpha was developed first, adapting its predecessor to the different notation of a PDM and extending it to deal with alternative operations. From a number of tests with simple PDMs, we have concluded that the process models generated by algorithm Alpha are not necessarily sound and correct (see sections 5.3.2 and 5.3.3), while these are important properties in our opinion. Also, our aim is to provide an overview of different algorithms that is as complete as possible with respect to the classification framework. Therefore, we started looking for different algorithms and thought of an algorithm that takes the structure of a PDM as the starting point. We started reasoning from putting a transition on each of the operations, letting the circles for the data elements be the input and output places of each transition. Eventually, this idea evolved to algorithm Charlie. However, when analyzing the process models resulting from algorithm Charlie, we found that a number of problems exist in these models. Often, algorithm Charlie does not provide sound models (see Section 5.3 for more details). Algorithm Delta was developed to solve these problems. Since it uses a construct that is not available in Petri nets, i.e. the cancellation region, the output of algorithm Delta cannot be represented as an ordinary Petri net. The YAWL language provides for this construct and is chosen to present the process model resulting from algorithm Delta.

Having these three algorithms, we realized that the construction of all algorithms focused on the data elements in the PDM, which led to a number of issues related to soundness and correctness (see also Section 5.3). Since the actual 'work' to obtain values for data elements is done by the operations of the PDM, we decided to focus on the translation of operations to transitions in the process model. This has resulted in algorithm Bravo, which is similar to algorithm Alpha, and algorithm Echo, which is similar to algorithm Charlie.

In the process models resulting from algorithms Alpha and Bravo, the *moment of choice* is very early. This may be undesirable because it is not always known at the start of a case which is the best path, e.g. conditions may not be satisfied which lead to different execution choices. Therefore, we started thinking of ways to defer these choices and leave execution options open as long as possible. This has led to algorithm Foxtrot. The *moment of choice* in this algorithm is as late as possible, since all possible operation executions are specified. The process model describes the complete state space of possible operation executions (cf. the state space in Section 6.4). Since algorithm Foxtrot generates very large process models we have thought of a way to limit the size of the process model in algorithm Golf while keeping some freedom in the order of executing operations at the same time. Before the process model is constructed, the minimal execution paths of the PDM are determined top-down. This leads to early choices on a high level, but within each of the branches the order of execution of operations is free. Compare for instance the moment at which choices are made in the process model generated by algorithm Bravo in Figure 5.8 and the process model generated by algorithm Golf in Figure 5.13. After the execution of $Op10$ in the process model generated by algorithm Golf it is still possible to choose for $Op03$ on the one hand, or for $Op05$, $Op06$, $Op08$, or $Op09$ on the other hand, while in the process model generated by algorithm Bravo this choice has to be made before the execution of $Op10$ already.

With the seven algorithms and the development process in mind, we analyzed the differences between the algorithms. This has resulted in the classification framework presented in Section 5.1.

5.3 Evaluation of the Algorithms

In this section, we evaluate the seven algorithms. First of all, similar algorithms are compared pair-wise to show their differences. Next, we examine the soundness property of the models derived with the algorithms. Finally, the correctness and size of the process models generated by each of the algorithms are discussed.

5.3.1 Comparison of the Algorithms

All algorithms presented above are different from each other. However, some of the algorithms are rather similar and only differ in one of the dimensions of the framework. To show the impact of these small differences we compare those algorithms with each other and indicate how these differences are reflected in the resulting process models.

Algorithms Alpha & Bravo

Algorithms Alpha and Bravo differ in their main *focus*. Algorithm Alpha focuses on the data elements in the PDM, while algorithm Bravo takes the operations as a starting point. Because of its focus on data elements, algorithm Alpha does not distinguish between the operations that produce a value for the same output element. For example, the PDM of the mortgage example of Figure 5.1(a) has three alternative ways to produce a value for output element A. In algorithm Alpha, these three ways are distinguishable, but the production of a value for A is represented by only one transition (A) in the Petri net of Figure 5.7. This transition does not contain information on which input elements should be used to produce a value for A. Algorithm Bravo clearly distinguishes between the three operations that produce a value for A. Each operation is translated to a specific transition in the Petri net: $Op01$, $Op03$, and $Op04$ (see Figure 5.8).

Algorithms Charlie & Echo

Algorithms Charlie and Echo vary with respect to the *focus* dimension in the construction perspective. In algorithm Charlie the main focus is on the data elements, while algorithm Echo starts from the operations in the PDM. Because of this difference in focus, the algorithms deal differently with multiple uses of the same data element. In the mortgage example, for instance, data element H is used in two different operations, i.e. $Op02$ and $Op03$. In algorithm Charlie this is translated to a construct in which eventually only one of the two can be executed (see Figure 5.9: transition $Op02$ and $Op03$ share the input place connected with transition H), while algorithm Echo generates a process model in which both operations can be executed (see Figure 5.11). Moreover, the extra control place for the transition representing the root element (and the production of sufficient tokens) in the process models generated by algorithm Echo ensure the lazy soundness of the model (cf. Definition 2.4.13). The process models generated by algorithm Charlie are not necessarily lazy sound; deadlocks can still occur even if the extra control place would be added.

Algorithms Bravo & Golf

Algorithms Bravo and Golf score differently on *concurrent execution*. In the process model generated by algorithm Bravo, parallel execution of activities is possible, while the process model derived using algorithm Golf does not allow for concurrent execution. This difference in concurrency can be clearly seen from the process models in Figures 5.8 and 5.13. In the process model of algorithm Golf all possible execution orders of transitions $Op05$, $Op06$, $Op08$, $Op09$, and $Op10$ are explicitly modeled, while they are just summed up as parallel branches in the process model generated by algorithm Bravo.

Algorithms Charlie & Delta

The difference between algorithms Charlie and Delta lies in two dimensions: the *representation* dimension and the *eagerness* dimension (see Figure 5.6). Essentially, both construction algorithms are the identical. However, the process model generated by algorithm Charlie is represented by a Petri net while algorithm Delta delivers a YAWL model. The YAWL language provides for a construct that cannot be used in the Petri net language (i.e. cancellation arcs). With this extra construct some of the problems with the process models generated by algorithm Charlie can be solved (see also Section 5.3.2 for more details on the soundness property for process models generated by these algorithms).

The second difference between algorithm Charlie and algorithm Delta is related to their eagerness. Because of the extra control place in the model generated by algorithm Delta, the model is overcomplete but *restricted*, while the process model generated by algorithm Charlie does not have such a control place and therefore is overcomplete and *unrestricted*.

By this comparison we can see that choosing different values on the six dimensions in the classification framework leads to differences in the resulting process models. Which choices are best for a certain situation is dependent on the characteristics of this situation, the system, the environment, etc. Section 8.3 elaborates on a practical application of the classification framework to choose an algorithm for the generation of a process model based on a PDM. In the next two sections, we evaluate the algorithms and the resulting process models with respect to criteria such as soundness, correctness, and size.

5.3.2 Soundness of the Process Models

Soundness is a desirable property for any process model. If a process model is sound, it does not contain behavioral errors such as livelocks and deadlocks (cf. Definition 2.4.9). The

	Alpha	Bravo	Charlie	Delta	Echo	Foxtrot	Golf
Soundness	+	+	-	+	-	+	+
1. Option to complete	+	+	-	+	-	+	+
2. Proper completion	+	+	-	+	-	+	+
3. No dead transitions	+	+	+	+	+	+	+
Correctness	-	-	-	-	-	+	+
1. All data elements produced	+	+	+	+	+	+	+
2. Data dependencies respected	-	+	-	-	+	+	+
3. No duplicate operations	-	-	-	+	-	+	+

Table 5.1: *This table indicates for each of the algorithms whether the process models generated by the algorithm satisfy the soundness and correctness criteria.*

process models generated by algorithm Alpha, Bravo, Delta, Foxtrot and Golf are sound by definition.

Algorithms Charlie and Echo also generate a Petri net, but these Petri nets are not necessarily sound because the initialization transition enables all branches leading to the final transition, one of which is enough to enable this transition, i.e. the AND-split at the start is mapped with an XOR-join at the end. Because of this, tokens may be left in the model after the final transition has fired and put a token in the end place. However, if we add an extra control place to the model (like in algorithm Echo) that ensures that the final transition producing a value for the root element is only executed once, and sufficient tokens are produced, the process model is lazy sound (cf. Definition 2.4.13).

Moreover, algorithm Charlie may also contain deadlocks when a specific situation occurs in the PDM. If the value of the same data element is used in more than one operation, e.g. data element E in Figure 5.14, this may lead to a deadlock in the process model if all of these operations need to be executed. An example of such a deadlock situation is shown in the process model of Figure 5.15. After the execution of e.g. the transitions $init$, E, F, and $Op3$, transition $Op2$ will never be enabled and therefore the process will never finish.

Algorithm Delta is based on algorithm Charlie but it generates a YAWL model instead of a Petri net. As we have seen before, the process model generated by algorithm Charlie has some problems related to soundness, i.e. tokens are left behind in the model because all branches are initially enabled and deadlocks can occur. However, these problems are solved in algorithm Delta by adding extra arcs, control places and a cancellation region[5]. Thus, algorithm Delta provides a sound YAWL model.

Besides the soundness property of the resulting process model, the algorithms can also be evaluated based on other desirable properties. First, the correctness of the process models with respect to the PDM is elaborated. Next, the size of the process models that are generated is assessed.

5.3.3 Correctness of the Process Model with Respect to the PDM

In Section 4.2 we have introduced a number of basic correctness requirements for a process model that is (manually) derived from a PDM by grouping operations and their data elements into activities:

[5]Recall that the notion of soundness in a YAWL model comprises four desirable properties: (i) soundness, (ii) weak soundness, (iii) irreducible cancellation regions, and (iv) immutable OR-joins (see Section 2.5).

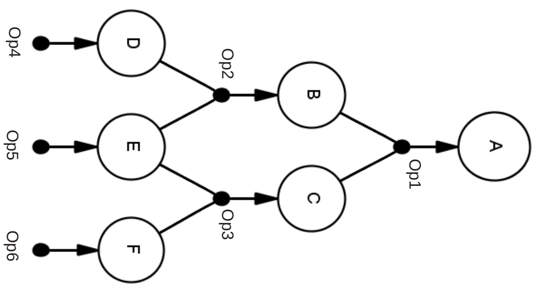


Figure 5.14: *An example PDM with six data elements.*

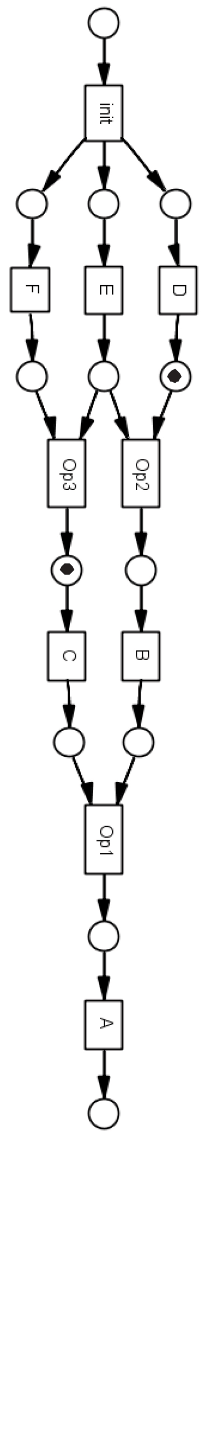


Figure 5.15: *The process model for the PDM of Figure 5.14, generated by algorithm Charlie. The tokens indicate a deadlock situation.*

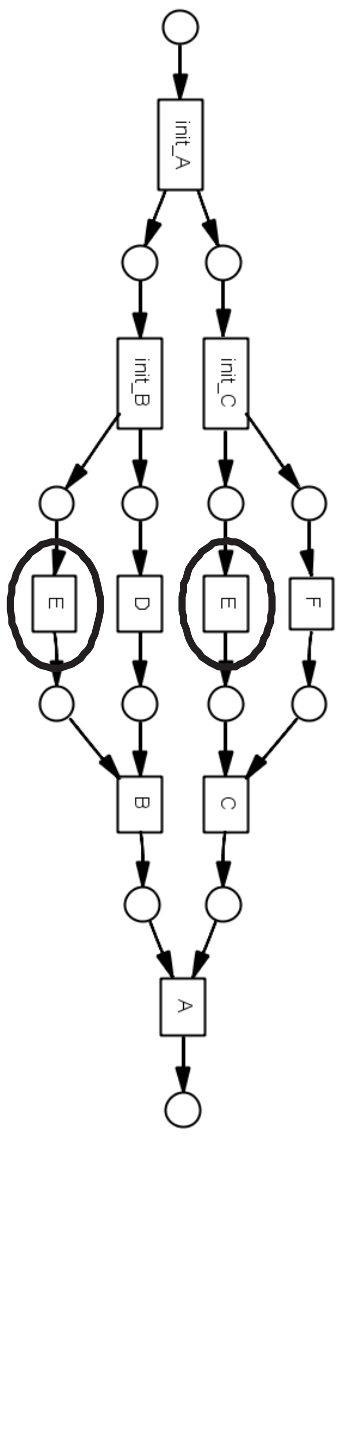


Figure 5.16: *The process model for the PDM of Figure 5.14, generated by algorithm Alpha. The ellipses show the duplicate occurrence of the same transition.*

1. The process model should contain a way to produce each data element in the PDM,
2. The process model should respect the data dependencies in the PDM in its control flow, and
3. The process model should never have a possibility to execute an operation more than once in an execution trace of the process model.

The seven algorithms presented in this chapter take the data elements and/or operations of the PDM as a starting point to derive a process model. Because this does not lead to groupings of operations and their data elements, it may not be clear what the content of each activity in the process model is and therefore it may not be possible to distinguish input and output data sets for each operation. Therefore, the correctness requirements from Section 4.2 may not be directly applicable to the generated process models. However, if we assume that the focus dimension in the construction perspective determines which activities contain operations, it is possible to perform a basic correctness verification based on these requirements.

For process models generated by an algorithm with a focus on *data elements*, we then assume that only the activities that refer to a data element in the PDM contain operations. Specifically, these activities contain all operations that produce the data element as an output element, e.g. transition B in Figure 5.18 contains operations $Op2$ and $Op3$ because they both produce a value for data element B. Activities which correspond to operations in the PDM do not have content in the form of operations and data elements and are only used for routing purposes, e.g. transition $Op2$ in Figure 5.18 is only used for routing purposes.

For process models generated by algorithms with a focus on *operations* in the construction perspective, we assume that only the activities which correspond to operations have content. Then, activities related to data elements in the PDM are empty and are only used for routing purposes. For example, transitions $Op6$ and $Op2$ in the process model shown in Figure 5.19 contain operations $Op6$ and $Op2$ from the PDM in Figure 5.17, respectively, while e.g. transition B and A are empty.

Based on these assumptions, we can now evaluate the algorithms for the correctness of the process models they generate. Let us first consider algorithm Charlie. Algorithm Charlie has a focus on data elements. Thus, we assume that only activities which correspond to data elements in the PDM have content. Then, many of the models generated by algorithm Charlie will satisfy the three correctness requirements. However, this is not true in general. From the evaluation with a number of test PDMs we have seen that some issues may occur when alternative operations to produce one data element exist in the PDM.

The first correctness requirement is always satisfied since all data elements are translated to an activity and each of these activities contains all operations that produce the data element. Thus, there is a way to produce each of the data elements in the PDM. However, the second requirement is not satisfied in general. The process model shown in Figure 5.18 illustrates this. Because activity B contains two alternative operations $Op2$ and $Op3$, it can be already executed when one of the two is executable, i.e. when either transitions C and D, or transitions D and E are executed. The activity dependency expression for this process model requires that transitions C, D and E are always executed before B is executed (i.e. $C \mapsto B \wedge D \mapsto B \wedge E \mapsto B$). However, the execution traces of this process model do not satisfy these activity dependencies, e.g. transition E may also be executed after the execution of $Op2$ or may even not be executed at all. Finally, the third correctness requirement may also be violated by process models derived with algorithm Charlie. A single operation can never be in two or more different activities because it can have only one output element and each data element

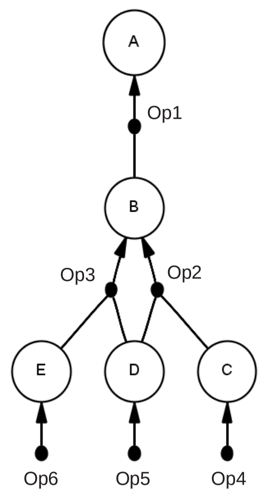


Figure 5.17: *An example PDM with six data elements.*

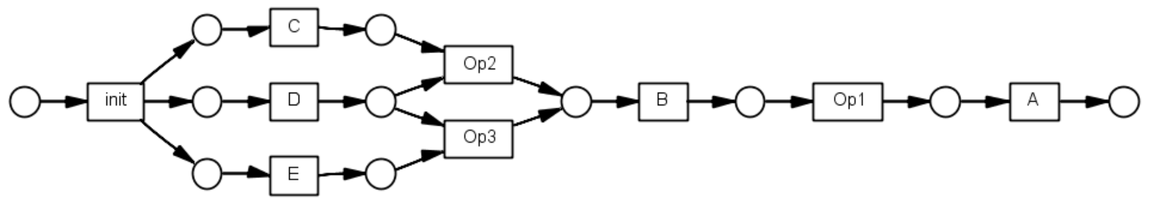


Figure 5.18: *The process model for the PDM of Figure 5.17, generated by algorithm Charlie.*

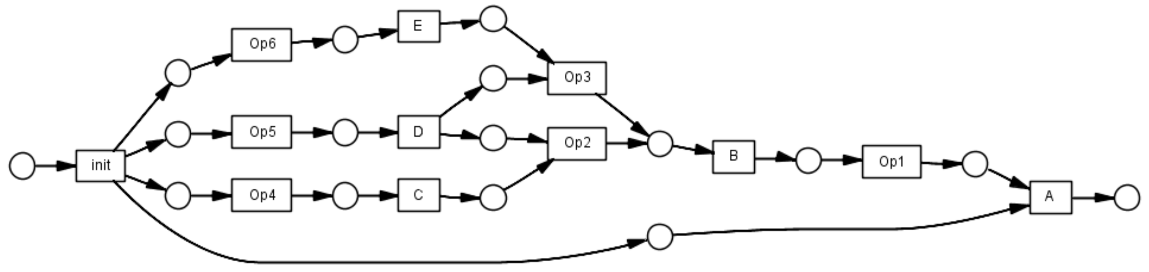


Figure 5.19: *The process model for the PDM of Figure 5.17, generated by algorithm Echo. Transition $Op1$ will be executed twice.*

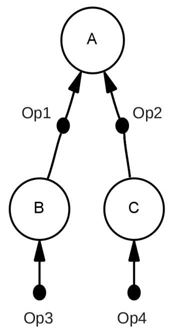


Figure 5.20: An example PDM with six data elements.

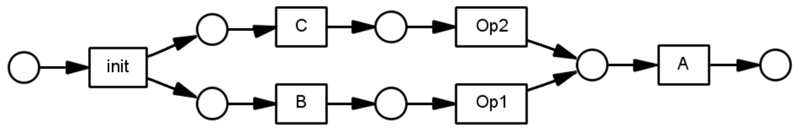


Figure 5.22: *The process model for the PDM of Figure 5.20, generated by algorithm Charlie. Transition A will be executed twice.*

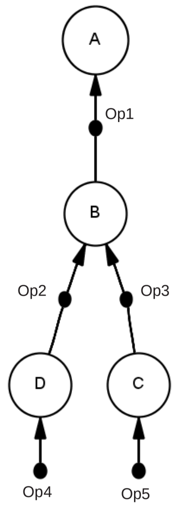


Figure 5.21: *An example PDM with six data elements.*

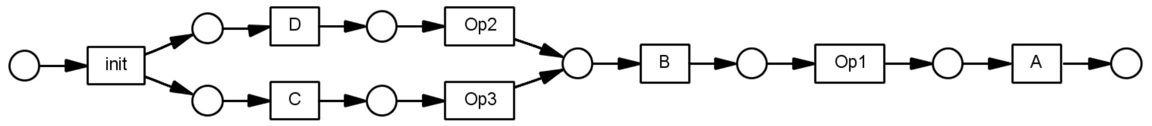


Figure 5.23: *The process model for the PDM of Figure 5.21, generated by algorithm Charlie. Transition A will be executed twice.*

is translated to exactly one activity. However, when the process model is not sound, the same activity may be executed twice in the same execution trace (see e.g. the process model in figures 5.22 and 5.23). Thus, process models generated by algorithm Charlie do not necessarily satisfy the third correctness requirement. Because we have assumed that a value for a data element can only be produced once (cf. Assumption 4.3.4), this is no problem: the second execution of an operation producing a value for the data element is skipped.

Since algorithm Delta has the same basis of constructing a process model as algorithm Charlie, a process model generated by algorithm Delta does satisfy the first basic correctness requirement but does not need to satisfy the second requirement. We thus need to check the third requirement. An activity can never be executed multiple times because of the extra control places that are added to the model. There are no execution traces in which the same operation can be executed more than once. Thus, the process models by algorithm Delta satisfy the first and third requirement, but not the second.

Also the process models generated by algorithm Alpha can be verified in this way. Algorithm Alpha has a focus on data elements and only contains activities related to operations when they are needed for a correct routing. Thus, the operations of the PDM can be found in activities that are named after the output data element of the operation. Besides the problem with alternative operations that produce the same output element as described above in the discussion on correctness of process models generated with algorithm Charlie, the application of the basic correctness verification to a number of test PDMs generated with algorithm Alpha has shown another interesting issue. If a data element value is used multiple times in the process model, this construct also leads to issues related to correctness in the process models generated by algorithm Alpha. Figure 5.16 contains an example to illustrate this. Different transitions in the Petri net may refer to the same operation(s), cf. the two occurrences of transition E in the figure. Thus, process models generated by algorithm Alpha do not automatically satisfy the third condition in the basic correctness requirements. This also means that transition E may be executed more than once in the same execution trace. However, this is not a problem if we assume that the two transitions E are actually the same activities (because they have the same content), and the system supporting this process model can detect whether a value for E has been established already and can then skip the second execution of E (cf. Assumption 4.3.4).

Because algorithm Echo has a focus on operations when it constructs a process model based on a PDM, we assume that only those activities that are named after operations in the PDM are related to these operations. Then, all process models generated by algorithm Echo contain a way to produce a value for all data elements, since all operations are translated to activities in the process model. Thus, the first correctness requirement is satisfied by definition. Also, the activity dependencies are respected by the control flow of the process model. Thus, the second requirement is satisfied too. Finally, the third requirement is not always satisfied. If the process model is not sound (e.g. the process model in Figure 5.19), some activities may be executed multiple times. This may also lead to duplicate executions of the operation(s) in the activity. However, this is not necessarily a problem, as we have seen above in the evaluation of algorithm Charlie, since we have assumed that a value for a data element can only be produced once (cf. Assumption 4.3.4). The extra execution of an operation producing a value for the same data element is then skipped.

The problem with duplicate occurrences of the same transition, as explained in the above evaluation of algorithm Alpha, also occurs with algorithm Bravo. Thus, a process model generated by algorithm Bravo does not satisfy the third correctness requirement by default. However, because of its focus on operations, the first two requirements are satisfied.

For the other algorithms, i.e. Foxtrot and Golf, it is also possible to perform a basic correctness verification. These algorithms generate models that are per definition correct with respect to the PDM. For instance, in the process model of Figure 5.13, all transitions contain exactly one operation, except for the silent transitions that are only used for correct routing. The first requirement, that all data elements of the PDM are produced in the process model, is satisfied. Also, the activity dependencies are respected by the control flow of the process model. Thus, the second requirement is satisfied as well. The third condition requires that an operation may never be executed more than once in any execution trace. Almost all operations from the PDM are represented multiple times in the process model (e.g. $Op10$), but they can never occur twice or more in the same execution trace, because there are no loops in the model and all occurrences of transitions containing $Op10$ are in different branches of an XOR-split. Thus, all three basic correctness conditions are satisfied by the process model for the mortgage example generated by algorithm Golf. Because algorithm Golf considers all top-down execution paths separately, the process models generated by algorithm Golf will satisfy the basic correctness conditions in general. The same conclusion holds for the process models generated with algorithm Foxtrot.

From this evaluation, we conclude that only algorithms Foxtrot and Golf always provide correct models, while process models generated by the other algorithms may contain some problems, related to soundness, correctness or both.

5.3.4 Size of the Process Models

Another point of evaluation for the algorithms is the size of the process models they generate. This size is measured in terms of the number of transitions, places, and arcs as a function of the number of data elements and operations in the PDM.

The smallest process models are generated by algorithm Charlie. This algorithm adds an initializing transition to the model plus one transition for each data element and one for each operation. Thus, the number of transitions for algorithm Charlie is: $1 + |D| + |O|$. With each transition that corresponds to a data element in the PDM, also one input place and one output place are added. Additionally, a start place for the process model is added as input place to the initializing transition. The total number of places is therefore $2 \cdot |D| + 1$. Finally, the number of arcs in the model is dependent among others on the number of leaf data elements in the PDM. There are at most $|D|$ leaf data elements in the PDM. First, there is 1 arc from the input place of the model to the initializing transition. Then, there are a maximum of $|D|$ arcs from the initializing transition to all input places of transitions that correspond to leaf elements in the PDM. Next, two arcs exist for each transition that corresponds to a data element, i.e. one from the input place to the transition and one from the transition to the output place. In total, these count for $2 \cdot |D|$. Then, the number of outgoing arcs from a transition that corresponds with an operation is $|O|$. Finally, the ingoing arcs of transitions that correspond to operations have to be counted. Each transition has a maximum of $|D|$ input places, thus the maximum number of arcs to these transitions is $|D| \cdot |O|$. Thus, the maximum number of arcs is: $1 + |D| + 2 \cdot |D| + |O| + |D| \cdot |O| = |D| \cdot |O| + 3 \cdot |D| + |O| + 1$.

In a similar way also the size of the models generated by algorithm Echo and Delta can be determined (see Table 5.2). Note that the process models generated by algorithm Delta contain substantially more places and arcs because of the introduction of control places and cancellation arcs. The cancellation arcs in process models generated by algorithm Delta, for instance, are counted by the number of all places and transitions, i.e. $|D| + |O| + 1 + 3 \cdot |D| + |O| + 1 =$

$4 \cdot |D| + 2 \cdot |O| + 2$, minus the start and the end place and minus the $init$ transition and the transition for the root element. Thus, the total number of cancellation arcs is: $4 \cdot |D| + 2 \cdot |O| - 2$.

Algorithm Foxtrot generates considerably more complex models. In principle, all combinations of operations are generated, i.e. all possible ways in which $|O|$ operations can be ordered. That is $2^{|O|}$ combinations. Thus, there are a maximum of $2^{|O|}$ transitions that correspond to operations. In addition, silent transitions are added for each combination, at maximum: $2^{|O|}$. Thus, the upper bound for the number of transitions is $2^{|O|+1}$. In a similar way, the number of places can be determined. Each transition that corresponds to an operation from the PDM has an output place, i.e. $2^{|O|}$. Additionally, the process model has an initial and an end place. Thus, the total number of places is $2^{|O|} + 2$. The number of arcs is at most $2 \cdot 2^{|O|+1} = 2^{|O|+2}$ because each transition has one ingoing and one outgoing arc and there are $2^{|O|+1}$ transitions. However, for many models these upper bounds will not be reached since some simplifications are made. For example, in Figure 5.12, all combinations that start with the execution of $Op09$ are covered by only one transition $Op09$ and there are several transitions that may consume the token from this transition. The model of Figure 5.12 has 438 transitions, while the upper bound for this model is $2^{|O|+1} = 2048$.

Because of the recursive steps in algorithms Alpha and Bravo it is more difficult to give a precise upper bound for the size of the models. An upper bound for the number of transitions for algorithm Bravo can be determined as follows. The output element can be produced by at most $|O|$ operations. Each of these operations has at most $|D|$ input elements. Each of these input elements may again be produced by at most $|O|$ operations. Each of these operations again has at most $|D|$ input elements, etc. Figure 5.24 shows a graphical explanation of this recursive tree. The tree can be at most $|O|$ levels deep. Thus, an upper bound for the number of transitions then is $(|D| \cdot |O|)^{|O|}$. However, all transitions also need an initializing transition at the beginning of the process model; therefore we have to multiply the number transitions by 2, leading to a total number of transitions of $2 \cdot (|D| \cdot |O|)^{|O|}$.

A lower bound for the models generated by algorithm Bravo is given by $|O|$ for the number of transitions, $|O| + 1$ for the number of places, and $2 \cdot |O|$ for the number of arcs. This is the size of the model if the PDM does not have alternative operations, if data elements are not used in more than one operation, and if all operations have at most one input element. In most cases, the process models generated by algorithm Bravo will have a size that is closer to the lower bound than to the upper bound. Compare for instance the size of the process model of Figure 5.8, i.e. 13 transitions, 16 places, and 34 arcs, with the outcomes of the upper bounds (e.g. the number of transitions: $2 \cdot (|D| \cdot |O|)^{|O|} = 2 \cdot (8 \cdot 10)^{10} = 2.14 \cdot 10^{19}$) and lower bounds (i.e. 10 transitions, 11 places, 20 arcs).

The process models generated by algorithm Alpha are a little larger than the process models generated by algorithm Bravo because they also include transitions for data elements. Table 5.2 summarizes the maximum size of the process models that are generated by the seven algorithms.

If the process models obtained with one of the algorithms are used to communicate with stakeholders in e.g. a redesign project, a first requirement is that the model should be readable and not too complex. The process models resulting from the seven different algorithms may be quite large as we have seen above, because they are at the same level of detail as the typically rather fine-grained PDM model. However, the sizes of models generated by algorithms Alpha, Bravo, Charlie, and Echo are in general similar to the size of the corresponding PDMs.

	Alpha	Bravo	Charlie
Size			
Number of transitions	$4 \cdot (\lvert D\rvert \cdot \lvert O\rvert)^{\lvert O\rvert}$	$2 \cdot (\lvert D\rvert \cdot \lvert O\rvert)^{\lvert O\rvert}$	$\lvert D\rvert + \lvert O\rvert + 1$
Number of places	$4 \cdot (\lvert D\rvert \cdot \lvert O\rvert)^{\lvert O\rvert}$	$2 \cdot (\lvert D\rvert \cdot \lvert O\rvert)^{\lvert O\rvert}$	$2 \cdot \lvert D\rvert + 1$
Number of arcs	$8 \cdot (\lvert D\rvert \cdot \lvert O\rvert)^{\lvert O\rvert}$	$4 \cdot (\lvert D\rvert \cdot \lvert O\rvert)^{\lvert O\rvert}$	$\lvert D\rvert \cdot \lvert O\rvert + 3 \cdot \lvert D\rvert + \lvert O\rvert + 1$
	Delta	Echo	Foxtrot
Size			
Number of transitions	$\lvert D\rvert + \lvert O\rvert + 1$	$\lvert D\rvert + \lvert O\rvert + 1$	$2^{\lvert O\rvert+1}$
Number of places	$3 \cdot \lvert D\rvert + \lvert O\rvert + 1$	$\lvert D\rvert \cdot \lvert O\rvert + \lvert D\rvert + 2$	$2^{\lvert O\rvert} + 2$
Number of arcs	$2 \cdot \lvert D\rvert \cdot \lvert O\rvert + 9 \cdot \lvert D\rvert + 5 \cdot \lvert O\rvert - 1$	$2 \cdot \lvert D\rvert \cdot \lvert O\rvert + 2 \cdot \lvert O\rvert + \lvert D\rvert + 3$	$2^{\lvert O\rvert+2}$
	Golf		
Size			
Number of transitions	$2^{\lvert O\rvert} + \lvert O\rvert$		
Number of places	$2^{\lvert O\rvert} + 2$		
Number of arcs	$2^{\lvert O\rvert+2}$		

Table 5.2: *Some upper bounds to the size of the process models generated by the different algorithms, given by the number of transitions, places and arcs in the process model as a function of the number of data elements, i.e. $\lvert D\rvert$, and operations, i.e. $\lvert O\rvert$, in the PDM.*

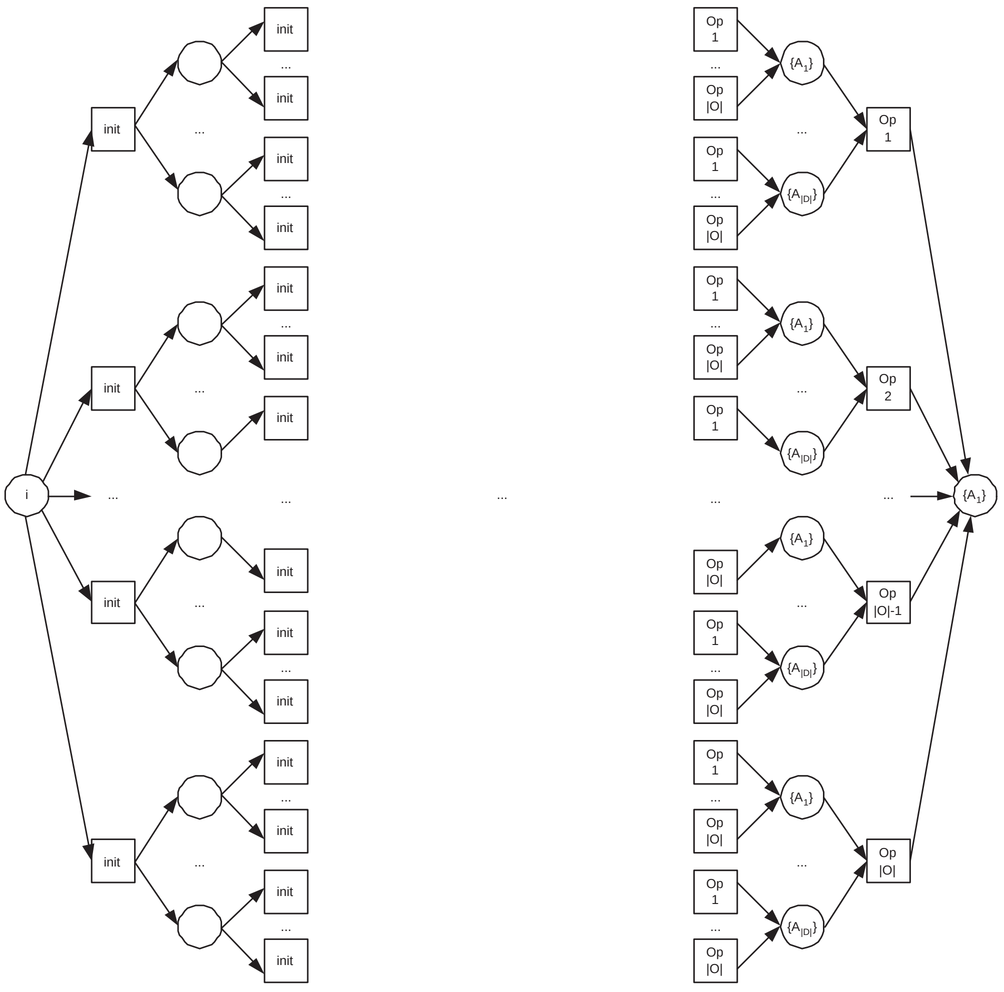


Figure 5.24: *A graphical representation of the maximum size of a process model generated by algorithm Bravo. The output element, e.g. A, can be produced by at most $|O|$ operations. Each of these operations has at most $|D|$ input elements. Each of these input elements may again be produced by at most $|O|$ operations. Each of these operations again has at most $|D|$ input elements, etc.*

Therefore, these models are still readable. The process models derived with algorithm Delta easily become too large and too complex because of the extra control places and cancellation regions. The readability of the process models generated by algorithm Foxtrot and algorithm Golf is generally poor because no parallelism is used in the models and all possible paths to produce the end product are explicitly modeled.

From this evaluation of the algorithms it may be clear that each of the algorithms has its advantages and disadvantages with respect to the characteristics of the generated process model. These issues should be carefully considered when using the algorithms to develop a process model, but they do not have to be insurmountable in practice, as is shown by a practical application in Section 8.3. The next section contains a discussion on how to use these algorithms.

5.4 Discussion

In the previous sections, we have seen a number straightforward ways to translate a PDM to a process model, purely based on the structure of the PDM. The generation of such process models based on a PDM fits with current workflow systems which need a process model to support the business process. Although we have seen that some of the presented algorithms have issues related to the structure of the process model they generate, the development of these seven algorithms is an important step towards the support for designing workflow process models based on product structures. The different process models and the classification framework provide a basis to consider the best representation of the process described by the PDM. Even though the pocess model may not be sound or correct per se, it may still be a good starting point for communication about the process with stakeholders or for further refinement.

At this point in time, there is no concrete guidance yet to help users to choose one of the algorithms for their specific situation, but the classification framework can be used as a starting point. Users should try to identify which perspective and/or which dimensions are the most important for their application. For instance, the workflow management system selected for supporting the process may have some limitations in the process model language that it can support. This may force a choice for an algorithm which generates a process model in the right modeling language, such that it can be directly supported by the system. Also, the policy of the company may require that no extra work is allowed for a case to keep the cost for executing the process as low as possible. Then, the selection of an algorithm with a strict eagerness is best. Or, if a short throughput time is required, a process model with concurrent executions may be preferable. In Section 8.3, a practical application of the classification framework and the algorithms is elaborated on.

Another issue with the seven algorithms presented here is that they do not provide a way to group operations and data elements into activities. All algorithms derive process models directly from the PDM and they retain the high level of detail of the PDM. However, after a process model has been derived, grouping of the activities of the process model is still possible, e.g. by creating a number of sub processes in the generated process model. Then, the end result is similar to a process model in which the grouping has been done before. Note that in both cases there is no automatic support yet to identify groups of operations and their data elements. The practical case in Section 8.3 also contains an illustration of the grouping of activities in the generated process model.

Finally, the algorithms presented here are purely based on the hierarchical dependencies between data elements in the PDM. To refine these process models, additional information, e.g. the conditions on the execution of operations and the value of data elements, can be used. For instance, the process model of Figure 5.25 shows a different process model for the mortgage example. Additional information on conditions for execution of operations and failure probability has been used to construct this model. It would be worthwhile for future work to capture decisions based on this kind of additional information in the algorithms.

Section 5.3 has given a deeper insight in the shortcomings of the current algorithms. The various issues related to the structure and behavior of the process models and the grouping of operations in the process models have motivated and inspired us to develop a more flexible approach to support product-based workflow processes. The principle behind this approach is a direct execution of the PDM, without the generation of a process model as an intermediate step. The next chapter explains this approach. First, however, the next two sections elaborate

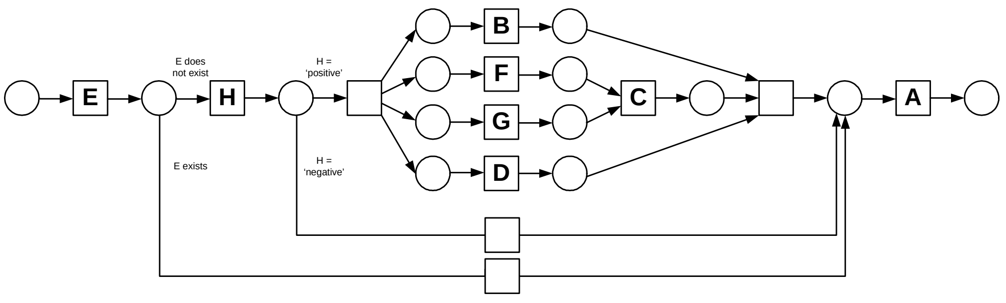


Figure 5.25: *An alternative process model for the mortgage example of Figure 5.1(a). Additional information on execution conditions and failure probability for operations is used to construct this model.*

on the tool support in ProM for the seven algorithms presented in this chapter and the related work on deriving process models from product structures.

5.5 Tool Support with ProM

The seven algorithms which we have presented in this chapter are implemented as *conversion plugins* in ProM. Each plugin takes a PDM as input and produces the process model (represented by a Petri net or by a YAWL model) as output. Figure 5.26 shows a screenshot of the ProM environment with an example PDM, the list of conversion plugins that can be applied to this PDM, and a number of process models generated by the different algorithms. Using the basic functionality which is already present in ProM, these process models can be converted to other languages, e.g. a Petri net can be converted to an EPC, or exported to a file that can be loaded into another system, e.g. the YAWL model in ProM can be exported to a YAWL engine file, which can then be edited in the YAWL Editor and executed with the YAWL Engine.

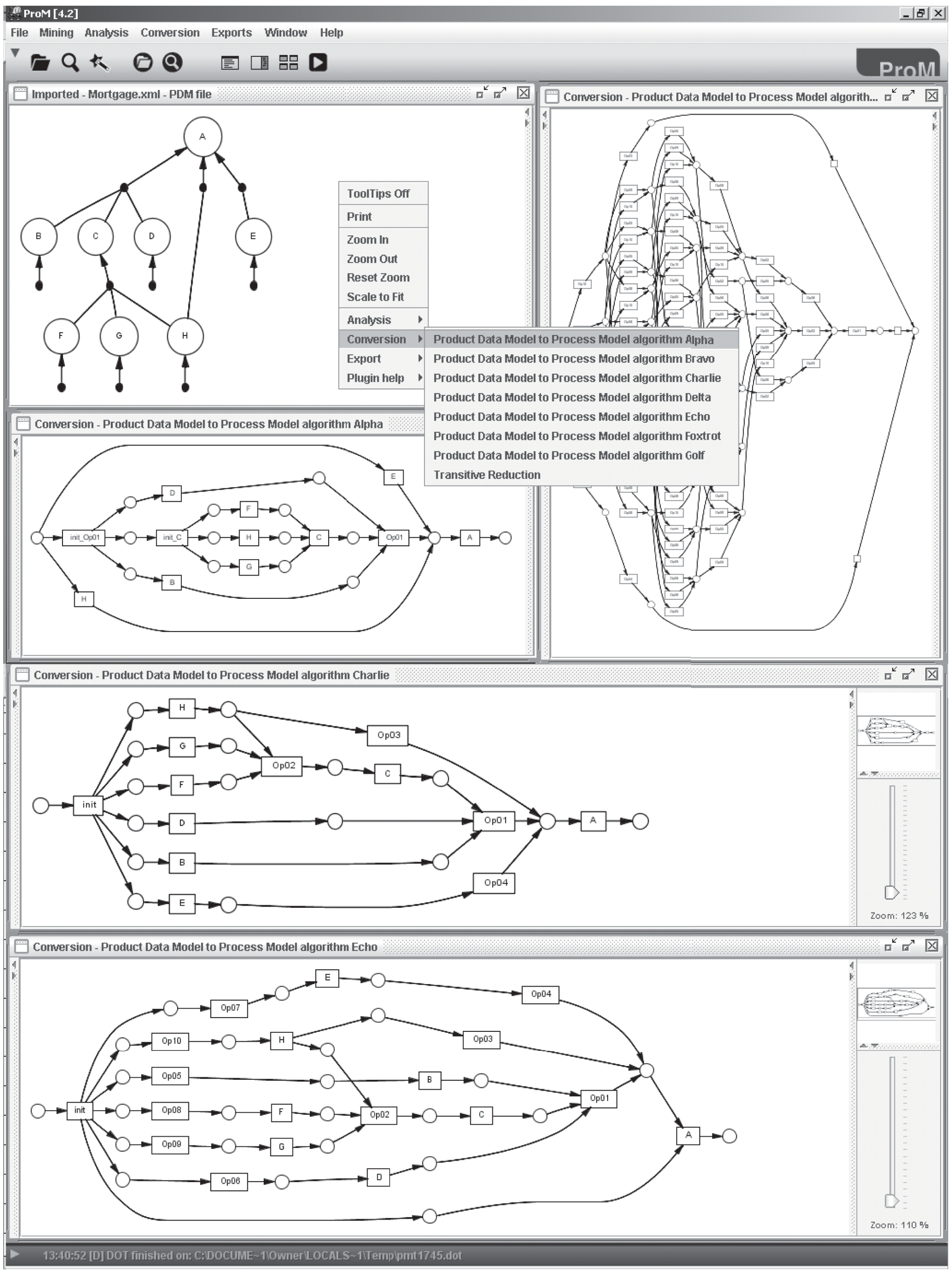
Depending on the representation of the process model, some process model analysis and verification can be done within the ProM Framework. If the model is represented as a Petri net, it can for instance be verified for soundness using the Woflan analysis tool [285, 301] incorporated in ProM. Figure 5.27 shows the results of a Woflan analysis of the process model of Figure 5.8. Also, a YAWL engine file (obtained by exporting the YAWL model generated in ProM) can be analyzed in the YAWL Editor. In Figure 5.28, the outcome of the analysis of the YAWL model of Figure 5.10 is shown.

5.6 Related Work

The related work of this section focuses on the generation of process models based on product structures. We also briefly touch upon the design of assembly lines in a production environment and we give an overview of 'intelligent' BPR methods that guide the design of process models.

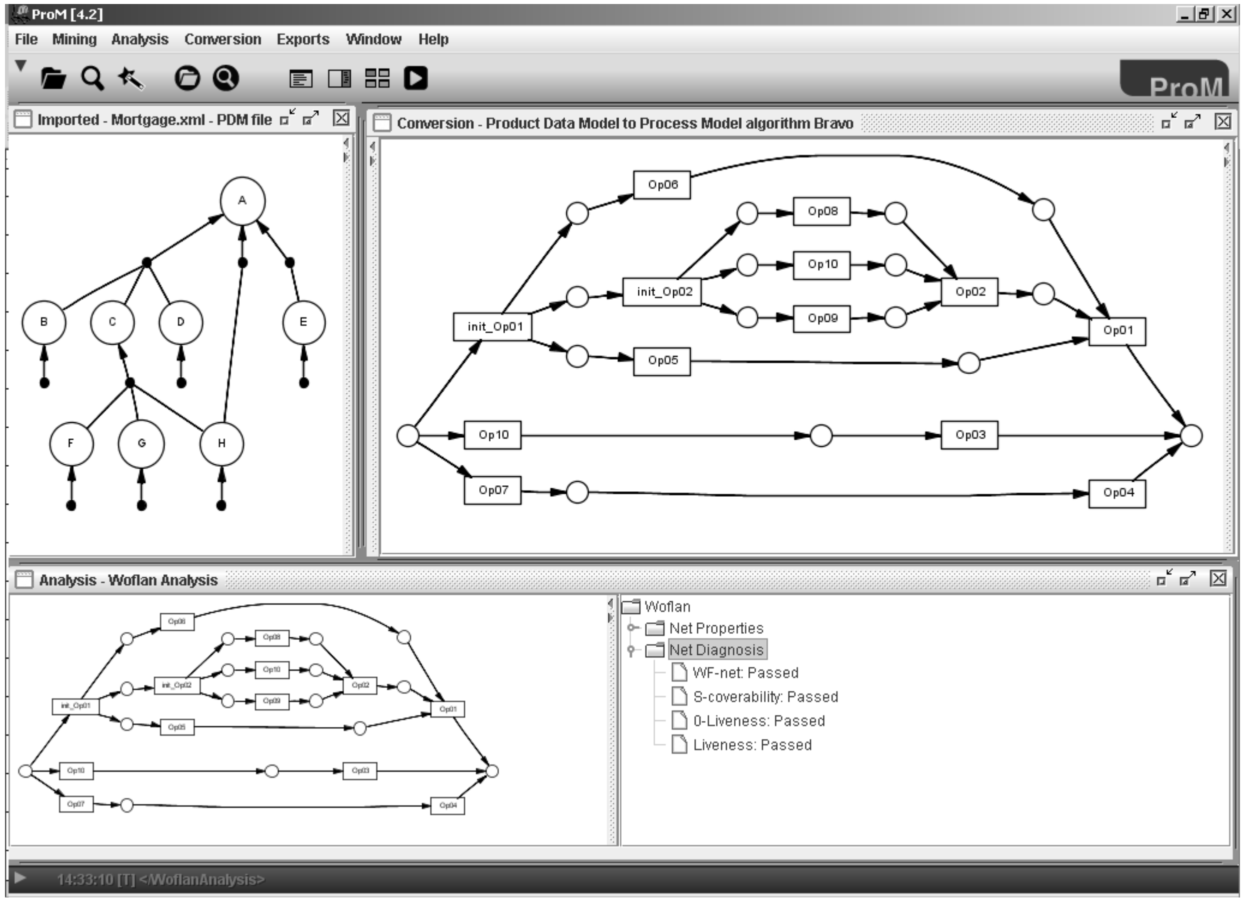
5.6.1 Deriving Process Models from Product Structures

Sørensen introduces the *augmented bill of materials* (ABOM) in [263], which supports the integration between mechanical design, process planning, and production. The ABOM extends


ProM [4.2]
File Mining Analysis Conversion Exports Window Help
ProM
Imported - Mortgage.xml - PDM file
Conversion - Product Data Model to Process Model algorith...
ToolTips Off
Print
Zoom In
Zoom Out
Reset Zoom
Scale to Fit
Analysis
Conversion
Export
Plugin help
Product Data Model to Process Model algorithm Alpha
Product Data Model to Process Model algorithm Bravo
Product Data Model to Process Model algorithm Charlie
Product Data Model to Process Model algorithm Delta
Product Data Model to Process Model algorithm Echo
Product Data Model to Process Model algorithm Foxtrot
Product Data Model to Process Model algorithm Golf
Transitive Reduction
Conversion - Product Data Model to Process Model algorithm Alpha
Conversion - Product Data Model to Process Model algorithm Charlie
Zoom: 123 %
Conversion - Product Data Model to Process Model algorithm Echo
Zoom: 110 %

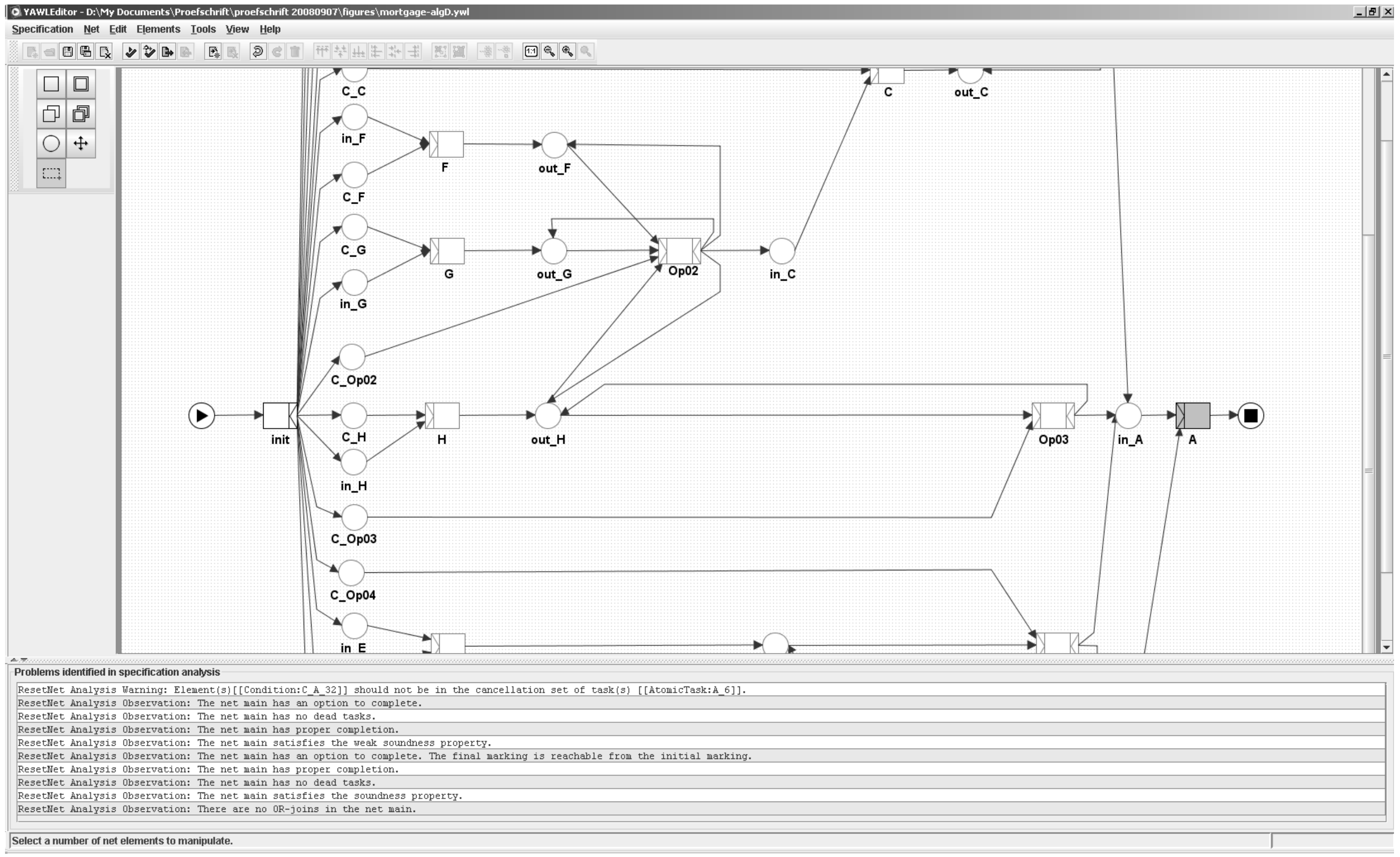
13:40:52 [D] DOT finished on: C:\DOCUME~1\Owner\LOCALS~1\Temp\pmt1745.dot

Figure 5.26: *A screenshot of ProM showing the conversion plugins available for a PDM. All seven algorithms are listed. The eighth conversion plugin in the list (transitive reduction) is a standard functionality in ProM. Also, some process models generated by the conversion plugins are shown.*


ProM [4.2]
File Mining Analysis Conversion Exports Window Help
ProM
Imported - Mortgage.xml - PDM file
A
B
C
D
E
F
G
H
Conversion - Product Data Model to Process Model algorithm Bravo
init_Op01
init_Op02
Op06
Op08
Op10
Op09
Op05
Op02
Op01
Op10
Op03
Op07
Op04
Analysis - Woflan Analysis
Woflan
Net Properties
Net Diagnosis
14:33:10 [T] <WoflanAnalysis>

WF-net: Passed
S-coverability: Passed
0-Liveness: Passed
Liveness: Passed

Figure 5.27: *A screenshot of ProM showing the output of the Woflan analysis plugin for the process model generated with algorithm Bravo. The process model has passed all tests and therefore is a sound process model.*


YAWLEditor - D:\My Documents\Proefschrift\proefschrift 20080907\figures\mortgage-algD.ywl
Specification Net Edit Elements Tools View Help
C_C
in_F
C_F
C_G
in_G
C_Op02
C_H
in_H
C_Op03
C_Op04
in_E
init
F
G
H
out_F
out_G
out_H
Op02
in_C
C
out_C
Op03
in_A
A

Problems identified in specification analysis

```
ResetNet Analysis Warning: Element(s)[[Condition:C_A_32]] should not be in the cancellation set of task(s) [[AtomicTask:A_6]].
ResetNet Analysis Observation: The net main has an option to complete.
ResetNet Analysis Observation: The net main has no dead tasks.
ResetNet Analysis Observation: The net main has proper completion.
ResetNet Analysis Observation: The net main satisfies the weak soundness property.
ResetNet Analysis Observation: The net main has an option to complete. The final marking is reachable from the initial marking.
ResetNet Analysis Observation: The net main has proper completion.
ResetNet Analysis Observation: The net main has no dead tasks.
ResetNet Analysis Observation: The net main satisfies the soundness property.
ResetNet Analysis Observation: There are no OR-joins in the net main.
```

Select a number of net elements to manipulate.

Figure 5.28: *A screenshot of the YAWL Editor showing the results of the soundness analysis of the YAWL model which was generated by algorithm Delta (cf. Figure 5.10).*

the regular BoM with information on the execution of the manufacturing process, e.g. the employee responsible for the production of a component, the machine that is scheduled to produce the component from its parts, and the scheduled delivery week of the component. The ABOM is accompanied by a process model of the organizational procedure to add the information related to the execution of the process to the regular BoM. Although the ABOM contains much information on the process, the process model is not derived from it. The links between the BoM and the process model are made manually.

In [5], Van der Aalst introduces the idea to generate a process model based on the product structure for workflow processes. He defines a formalization for the product structure of an administrative product with mandatory, optional, and choice components, followed by an algorithm to map this product structure on a Petri net. This algorithm has been at the basis of Algorithm Alpha (Section 5.2.1). Algorithm Alpha extends the algorithm from [5] with the translation of alternative paths.

Browne et al. [55, 56, 57] propose to use *goal graphs* to structure workflow processes in the healthcare domain (cf. Section 3.5). A goal graph describes the goals to be achieved to cure a patient with a certain disease. Each higher level goal can be achieved by a number of lower level goals (see Section 3.5.7). They define a mapping from the goal graph to a workflow process model in which every goal is translated to a sub process that is designed to achieve the specific goal. In contrast to our approach, the process model derived from the goal graph is a hierarchical model with many subprocesses, which makes the overall model less surveyable. To build this model, a top-down approach is used with a focus on the goals (cf. data elements). This is similar to the approach in algorithm Alpha. However, Browne et al. do not discuss the issues that may be related to this approach and the choices that are made with respect to e.g. representation, moment of choice, and eagerness.

Müller et al. [186, 187, 188] derive *large process models* based on the life cycle information of the elements in the *(product) data structure* (cf. Section 3.5). Each element in the (product) data structure has a subprocess related to it that describes the life cycle of this element, e.g. designed, tested, released. These life cycles are represented in the *Life Cycle Coordination Model*. The relationships between the elements in the (product) data structure indicate the dependencies between the sub processes. The structure of the large process model is derived from these dependencies. The resulting large process model is called a *data-driven process structure* since it is derived from the life cycle information of all elements. Although Müller et al. derive process models based on data dependencies, the application of their research has mainly focused on development processes of *physical* products (such as cars) instead of on informational products[6]. The data used to link sub processes is data on the status of a physical product within its life cycle (e.g. designed, tested, released). Although no life cycle information of data elements in the PDM is used in our approach, we can use the values of a data element to link parts of the process model to each other by possibly using conditions on the arcs or guards on activities in the process model. The approach by Müller et al. does not fit in the classification framework we use for our seven algorithms, because the large data-driven process models are built from existing subprocesses that are linked to each other based on the life cycle information and the product structure.

Also Küster et al. [157] present a technique to automatically derive compliant process models from given life cycles of the objects in the process (cf. Section 3.5). To do so, they first generate a set of actions based on the state transitions of the objects. Then, the order in which these actions should appear in the process model is determined. Next, actions are combined into process fragments, where they are additionally connected to decision and merge nodes.

[6]However, note that they do not restrict their approach to physical products.

Finally, the proces fragments are connected to produce the process model. This approach is similar to our middle-out order of constructing the process model, since process fragments are constructed per state transition and these fragments are later connected to each other to form a process model. However, Küster et al. do not focus on the other dimensions we have identified in the framework.

In contrast to the implementations of our algorithms for the automatic derivation of process models from the PDM which are presented in Section 5.5, none of the related approaches described above are actually realized in a tool yet. The COREPRO tool by Müller et al. [189, 190] provides support for the modeling and simulation of a data-driven process structure, but is not able yet to generate process structures based on a product structure and the object life cycles.

5.6.2 Structuring Assembly Lines

As we have seen in Chapter 3, the BoM provides a way to describe the product in a manufacturing environment. A BoM of a physical product is used for several processes, e.g. production planning, inventory control, accounting, and pricing. Oddly, the BoM is not directly used to structure the assembly lines of the product. Issues in the literature that are related to the structure and design of assembly lines mainly deal with *line balancing*, which is the problem of allocating work to work stations [300]. Most methods for line balancing date from the 1950s and mainly rely on dynamic programming techniques. A survey of line balancing techniques is available in [136]. The order in which work should be allocated to work stations is described by an *assembly plan*. Theoretically, there are $n!$ different plans to assemble a product consisting of n parts. However, various constraints exist that reduce the number of valid assembly plans for a product. One kind of constraints consists in the *precedence constraints* on the assembly operations on the parts [300], e.g. it can be necessary to first put parts a and b together before they can be assembled to part c. These precedence constraints can be recognized in the BoM and are similar to the input/output dependencies in the PDM. Moreover, the dependencies between parts that are stated in a BoM are also used in MRP to decide when to order which parts. Finally, in the BoP (see sections 3.1.1 and 3.5.1) the routing of operations and activities within one assembly step is described. Such an assembly step may be a process on its own and may involve a number of operations on the input material to produce the (sub) assembly product. However, the order of these operations is determined manually, e.g. by the engineers that define the technical specifications and design of the product [43, 303].

5.6.3 'Intelligent' BPR Methods

As we have mentioned in the introduction of this thesis, many tools, techniques and methodologies exist to support the design of process models. Although they are often presented as 'intelligent' most of them are not providing any guidance to derive (alternative) process models. A few exceptions exist that do provide some design guidance. Below a summary of these approaches is given, which is based on [195].

Case based reasoning (CBR) systems are presented in [154, 181]. They enable an efficient search and retrieval of earlier redesign solutions that hopefully fit the aims of a new BPR effort. However, it is the human designer who must still weigh their applicability and perform the adaptations to the situation at hand. Another drawback is that the cases are typically restricted to a certain business domain, e.g. banking.

A first attempt to generalize CBR systems for process redesign is presented in [165]. In this approach, the cases are linked to more general redesign best practices (see [232]), to give the designer insight into the drivers for the case solution and the particular impact of these best practices.

An entirely different approach claiming to support the process redesign is presented in [308]. It is suggested that systematic design generation can be supported by using a newly proposed type of modeling that focuses on the relationships between strategic actors. Still, it seems that it is the designer who must think of these alternatives - a process at best simplified by more insight into the actor relationships.

More promising seems the approach on the basis of the MIT Process Handbook as presented in [167]. The Process Handbook allows for navigation through families of similar processes. However, a process in the Process Handbook only consist of a set of activities. Besides a simple notion of routing by sequential dependencies between activities, there is no notion of control-flow. The *process recombinator* tool is implementing this approach [42]. Through the notions of (i) process specialization and (ii) coordination mechanisms, new designs can be systematically generated on the basis of an identified list of core activities. It is the end user who then can select the most satisfactory process. In contrast to the earlier CBR approaches, the existing design knowledge extends over multiple business domains and the end user is supported in a meaningful way to generate alternatives.

In research that is associated to the MIT Process Handbook, a quite different, yet promising approach is presented in [162, 212]. It attempts to capture the *grammar* underlying a business process. Just like natural language consists of words and rules to combine these words, business processes are seen as consisting of activities that can be combined using rewrite rules. A clear advantage of this approach would be that different process variants can be systematically generated and explored. However, it seems difficult to identify the rules and constraints that apply for certain categories of processes and to represent these in a grammatical framework.

Another approach worth mentioning here is the KOPeR tool described in [199, 197]. The idea pursued in it is that a limited set of process measures (e.g. process length, process handoffs, etc.) can be used to identify *process pathologies* in a given process (e.g. a problematic process structure, fragmented process flows, etc.). These process pathologies can then be matched to redesign transformations known to effectively deal with them. Although the tool does not generate new designs itself, e.g. by visualizing the effect of a suggested transformation on an existing design, experiments suggest that the tool 'performs redesign activities at an overall level of effectiveness exceeding that of the reengineering novice' [198]. Note that the KOPeR tool has a much broader scope than the assessment of the process structure. It also focuses on e.g. the organizational structure and the IT infrastructure of the process to be redesigned.

Finally, with a process mining approach [12, 84, 222, 223, 295] process models may be derived from event logs. These event logs contain records of activities that occurred during the execution of cases in the business process in the past. From the order in which events occur in this event log, the dependencies between activities in the process model are derived. The ProM tool [223] supports the automatic discovery of process models by a number of different techniques (see e.g. [19, 24, 31, 34, 83, 294]).

5.7 Summary

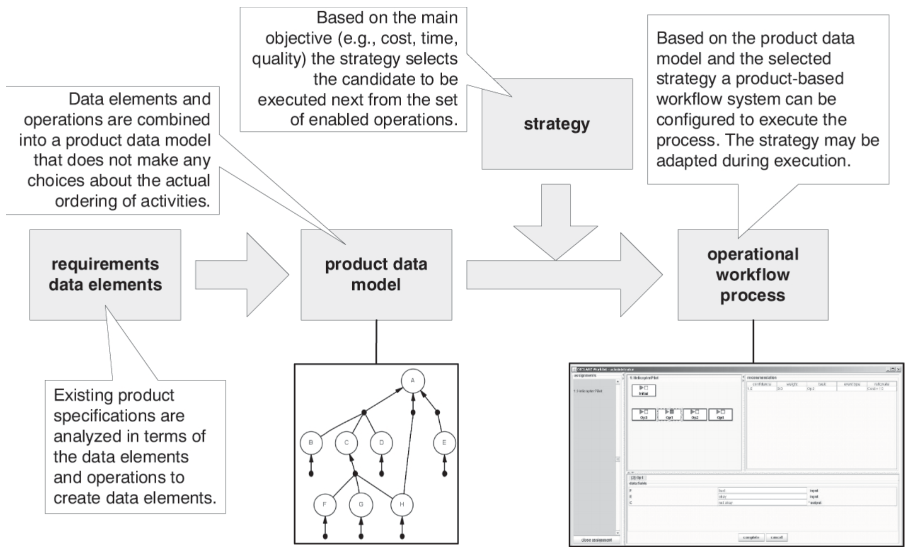
In this chapter we have introduced seven algorithms to automatically generate a process model from a PDM. With such a process model it is possible to deploy current workflow technology for product based workflow support, even though this technology is not suitable to support the product-based design process. The characteristics of the seven algorithms have been discussed, as well as the properties (e.g. soundness, readability, correctness) of the resulting process models. We conclude that the direct translation of a PDM to a (correct and sound) process model is not straightforward, because a number of problems related to the structure and behavior of the process model are introduced depending on the specific algorithm. To overcome these difficulties of translating a PDM into a process model we focus on a different support for product-based workflow processes in the next chapter by looking at the direct execution of a PDM.

Chapter 6

Direct Execution of a PDM

This chapter deals with the direct execution of a PDM. In contrast to the approach in the previous chapter, no process model is derived to support the runtime execution of the process described by the PDM, but the PDM itself is executed step-by-step based on the information available for a particular case at a certain point in time. This approach provides more flexible and dynamic support for the execution of a workflow process. We will refer to this approach as *Product Based Workflow Support (PBWS)*. An overview of PBWS is given in Figure 6.1. This figure shows that a product-based workflow system supports the execution of a workflow process based on (i) a PDM which is derived from product specifications, and (ii) a selected optimization strategy. The product-based workflow system guides the user step-by-step through all operations of the PDM that should be executed to produce the end product by means of recommending the next step the user should execute for a case. The enabled steps are calculated based on the PDM and the available information for the case. From this set of enabled steps, one step is recommended based on the performance goal of the process, e.g. minimization of cost.

In the first section of this chapter the *step-by-step execution* of a PDM is introduced by elaborating an example. In Section 6.2, the *functional design* of a tool supporting the direct execution of a PDM is presented in the form of a colored Petri net. This functional design ensures a correct execution of the PDM, without focusing on the performance objective of the process. Next, a number of *strategies* for an optimal execution of the PDM with respect to the performance goals, e.g. minimize cost or minimize processing time, are introduced. In each step of the execution of a PDM several operations may be executable. Therefore, a decision has to be made for which step is best to proceed with. A number of simple strategies to select this best next step from the set of enabled steps is presented in Section 6.3. These strategies only focus on local, i.e. short-term, optimization. Since this may lead to a sub-optimal execution of the entire case, an approach to determine a strategy to select the overall best step is introduced in Section 6.4. Some of the strategies are compared with each other in Section 6.5. We have also implemented these simple strategies and the global optimization approach as a prototype of a recommendation tool in the ProM and DECLARE frameworks. The support for the direct execution of a PDM provided by this tool is described in Section 6.6. The chapter ends with a section on related work and a summary.



Based on the main objective (e.g., cost, time, quality) the strategy selects the candidate to be executed next from the set of enabled operations.
Based on the product data model and the selected strategy a product-based workflow system can be configured to execute the process. The strategy may be adapted during execution.
Data elements and operations are combined into a product data model that does not make any choices about the actual ordering of activities.
strategy
requirements data elements
product data model
operational workflow process
Existing product specifications are analyzed in terms of the data elements and operations to create data elements.
A
B
C
D
E
F
G
H

Figure 6.1: An illustration of Product Based Workflow Support (PBWS).

6.1 Execution of a PDM

In this section, the direct execution of a PDM is explained. When a PDM is directly executed, we consider the operations that are executable based on the information available for a case at a certain point in time. We recall from Section 3.2.1 that an operation is executable if a value for all of its input elements is available. Typically, more than one operation can be enabled by the same set of available data element values. Therefore, we make an assumption on the concurrent execution of operations.

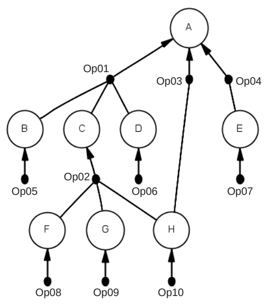
Assumption 6.1.1 (No concurrent execution of operations)**.** During the direct execution of a PDM no concurrent execution of operations is possible. Thus, only one operation at a time is executed.

This assumption is made because the problem of managing the execution of a PDM becomes much more complex if we allow for the concurrent execution of operations. For instance, if two operations producing a value for the same output element are executed in parallel, we do not know what happens if they both produce a different value (cf. Assumption 4.3.4). Also, the parallel execution of operations may lead to multiple executions of the same operation, which we ruled out earlier (cf. Assumption 4.3.3).

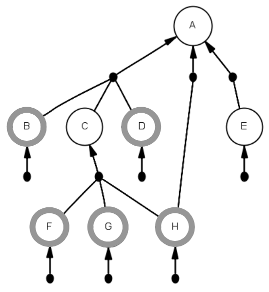
Figure 6.2 illustrates how the runtime (step-by-step) execution of a PDM works for the mortgage example which was introduced in Section 3.2.2[1]. Suppose that the values for the leaf elements B, D, F, G, and H (i.e. the interest percentage, term of mortgage, percentage of income to be paid in rent) are available at the start of the process (see Figure 6.2(b))[2]. The operations that are now enabled for execution are $Op02$ and $Op03$, since a value for all of their

[1]For reasons of simplicity we abstract here from the execution conditions on the operations. Moreover, we suppose that all leaf operations are already executed, i.e. a value is available for the output data elements of all successfully executed leaf operations.

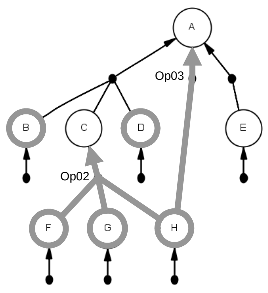
[2]Note that in this case not all leaf operations have been executed successfully. Because the execution of $Op07$ has failed, there is no value for data element E available.



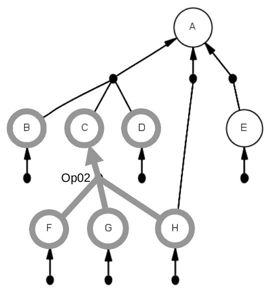
(a) The PDM for the mortgage example.



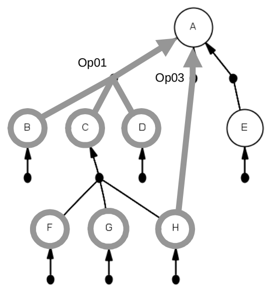
(b) The values for some of the leaf data elements (B, D, F, G, H) are available (indicated by bold circles). We refer to this situation as the initial state.



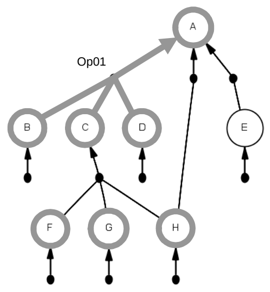
(c) Executable operations in the first step: $Op02$ and $Op03$.



(d) The value for data element C is produced by operation $Op02$.



(e) Executable operations in step two: $Op01$ and $Op03$.



(f) The value for the end product (A) is determined by executing operation $Op01$.

Figure 6.2: *The step-by-step execution of a product data model. Bold circles represent available data element values for the case under consideration; bold arrows indicate executable operations.*

input elements is available (Figure 6.2(c)). Operation $Op01$ is not executable because a value for data element C is not available yet and $Op4$ is not executable since there is no value for E present. Now, we have to choose between the two executable operations ($Op02$, $Op03$). Suppose we select $Op02$. Then, a value for data element C is produced (Figure 6.2(d)). The executable operations are calculated again, i.e. $Op01$ and $Op03$, and one of those operations is to be selected next. Suppose we select $Op01$ (Figure 6.2(f)). Then, the value for the end product A is determined and the process ends.

In many situations, more than one operation is executable, e.g. in the first step of this example we could have chosen for $Op03$ instead of $Op02$. This would have led to the end product immediately, but it generally also could have resulted in a different execution of the process with e.g. different cost, throughput time, etc. For example, if we focus on the cost of execution it seems best to choose $Op02$ since the execution cost for $Op02$ are lower than the execution cost for $Op03$ (5.0 vs 9.0, see Table 3.2). However, if we consider the processing times for the operations, the selection of $Op03$ as a next step would perhaps be a better decision (4.0 for $Op02$ vs 3.0 for $Op03$, see Table 3.2).

Now the question arises how to select the best operation to proceed with from a set of executable operations. We define ‘best’ with respect to the single case, i.e. the performance goal (e.g. total cost, total processing time) of the case in isolation is optimized. It is essential to see that *functional requirements and performance considerations are implicitly mixed in conventional approaches, while here they are made explicit*. A process model is typically based on both functional requirements (“this information is needed to make this decision”) and performance goals (“to minimize flow time it is better to do things in parallel”). Hence, if the performance goal changes (e.g. from ‘minimize flow time’ to ‘minimize execution cost’), the model needs to be revised. In this chapter, it is shown how these functional and performance requirements can be separated from each other by providing support for directly executing a PDM. As will be shown, this direct execution of a PDM leads to a much more dynamic and flexible workflow support. In the next section, the functional design of such a system is discussed.

6.2 Functional Design

The functional design of a tool supporting the direct execution of a PDM takes care of a *correct* execution of the PDM. This means that the functional design ensures that only executable operations are executed, that no duplicate data element values are produced, and that only one operation at a time is performed. It does not deal with performance and the selection of the best operation for the next step.

The functional design is described as a simulation model which can execute a PDM step-by-step. Its structure is generic since the model does not need to be changed for a different PDM or a different selection strategy. The PDM is added to the model as a variable and the selection strategy as a function. The simulation model for the functional design is represented by a colored Petri net (CPN, [137, 138]) and has been developed in CPN Tools [74]. This section describes the CPN model on a high level. For more details we refer to [274] and Appendix C.

The main level of the CPN model is depicted in Figure 6.3. It contains eight places and three transitions. All three transitions are sub processes in which the actions to be taken are described in detail. The main execution stream in the model is indicated by thick lines: First the executable operations are determined, then one of the executable operations is selected and

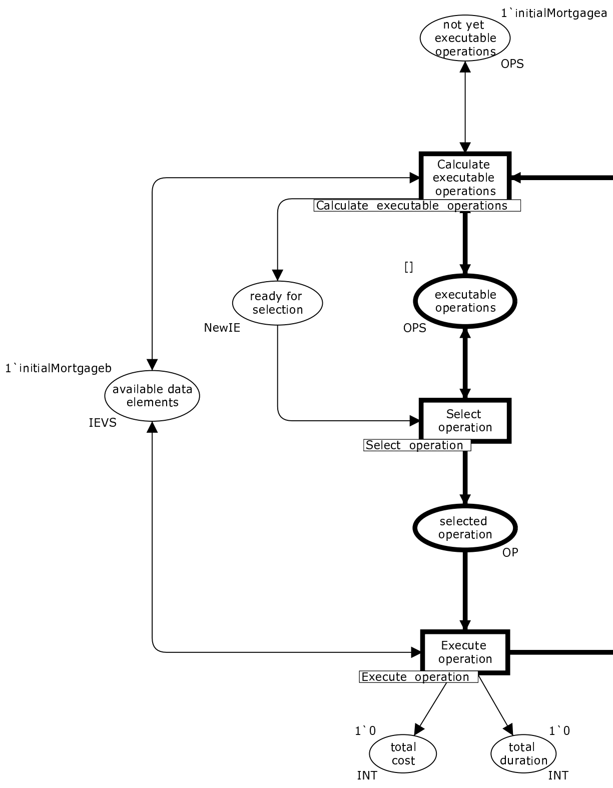


Figure 6.3: *The main level of the CPN model. A specific PDM is represented by its list of operations (e.g.* initial1a = $[(Op01, A, \{B, C, D\}, (5.0, 1.0, 0.05)), (Op02, C, \{F, G, H\}, (5.0, 4.0, 0.05)), ...]$ *for the mortgage example), which is initially stored in the place* not yet executable operations*. The available data element values for a specific case are represented by the token in available data elements (i.e.* initial1b*). The main execution stream is indicated by thick lines and contains three sub processes. In the first sub process* calculate executable operations *it is determined for each of the operations in the lists* not yet executable operations *and* executable operations *whether a value for all of its input elements is available (i.e. all input elements should be in the list of* available data elements*), whether its execution condition is satisfied, and whether its output element is not already available in* available data elements*. Secondly, the* select operation *sub process contains a function that implements a specific selection strategy (e.g. select the first operation in the list, select the operation with the lowest cost, or randomly select one). The sub process takes the list of* executable operations *and uses the selection strategy to select one of these operations from the list. Finally, the* execute operation *sub process takes the selected operation for execution. It introduces a time delay for the execution time of the operation. It also updates the value of the tokens in the places* total cost *and* total duration *with the execution cost and execution time of the executed operation, and it randomly decides whether the execution was successful or unsuccessful. In case of a successful execution, the value of the output data element of the operation is added to the* available data elements*. In case of an unsuccessful execution, the value for the output data element is not produced, but the execution cost and execution time are still added.*

executed and this procedure is repeated until the end product is produced or no executable operations remain.

The operations of the PDM are initially stored in the place *not yet executable operations*. An operation is represented by a tuple containing the identifier, the output element and the input elements, and a list of cost, time, and failure probability attributes, e.g. $(Op01, A, \{B, C, D\}, (5.0, 1.0, 0.05))$. The place *available data elements* contains all data elements of which the value is determined. Also, places *ready for calculation*, *total cost* and *total duration* each contain a token. The token in the former place indicates that a calculation for the case can be started, while the latter two places are used to monitor the cumulative cost and processing time for the case.

When a case is started the executable operations are determined first, i.e. transition *calculate executable operations* fires. This results in a division of the list of operations: the operations that are executable are put in the list in place *executable operations*, the other non-executable operations are put back in the *not yet executable operations* place. Also, a token is put in the place *ready for selection*. Then, transition *select operation* can fire. In the sub process related to this transition, a selection strategy (e.g. random, lowest cost, shortest processing time, first in list) is specified which prescribes which of the operations in *executable operations* should be selected. The selected operation is put in the place *selected operation* and the other operations are left in *executable operations*. Then, the selected operation is executed by the *execute operation* transition. A time delay is introduced for the processing time of the operation. The total cost as well as the total duration monitors of the case are increased with the execution cost and the processing time of the operation respectively. Finally, the output data element is produced with a certain probability. If the execution of the operation was successful, the data element and its value are stored in the place *available data elements*. If not, no data element is added to the place *available data elements*.

As soon as one of the operations has been completed, we are ready to determine the executable operations again. This is done based on the list of operations in *not yet executable operations* as well as on the list of operations in *executable operations*. Note that some of the operations in *not yet executable operations* may now have become executable because of new information produced in the previous step and that some of the operations in *executable operations* may no longer be executable since a value for their output data element was produced by another operation. Again, one operation is selected from the list of executable operations and the operation is executed, either successfully or unsuccessfully. This cycle is repeated until the end product of the PDM is produced or no executable operations remain.

Note that the structure of this model is generic, i.e. no changes need to be made to the structure in order to execute a different PDM. The information for a particular PDM is completely stored in the initial marking of the net (by the tokens in *not yet executable operations* and *available data elements*) and not in its structure. The CPN model describes the functional design of a PBWS tool to support the direct execution of a PDM. In Section 6.6 the prototype we have developed for PBWS based on this functional design is presented, but first the next two sections deal with two different approaches to select an operation from the set of executable operations. These selection strategies focus on optimizing the performance of a case within the boundaries of a correct execution ensured by the functional design.

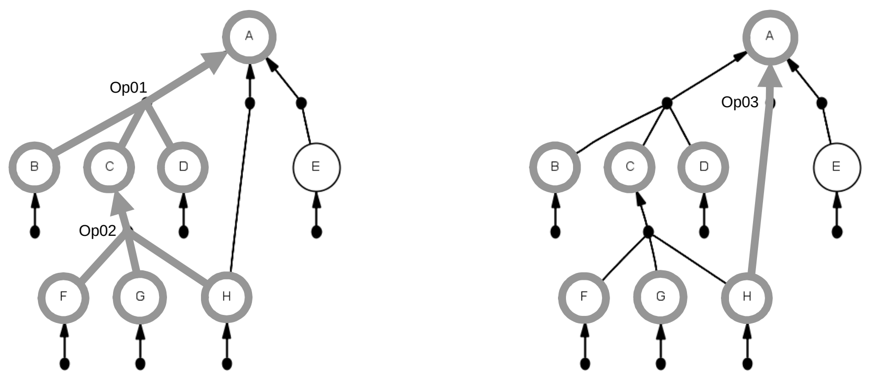
6.3 Simple Selection Strategies

During the runtime execution of a PDM, a number of alternative operations is available for execution, as was explained in Section 6.1. Only one of these alternatives can be chosen as the best next step to be performed (cf. Assumption 6.1.1). Choosing the best candidate depends on the performance goal(s) of the process. For instance, when it is important to produce the end product at low cost or rather at great speed one should choose the operation with the lowest cost or shortest processing time respectively.

In the functional design presented in the previous section, we mentioned some basic selection strategies based on the data structure of a list (e.g. first in list, random, lowest cost). In this section we extend these basic selection strategies with more realistic ones. For that we have drawn inspiration from *sequencing and scheduling rules* in the field of logistics and production planning [210, 261]. Short-range production planning is a method to decide, a priori, which resource is going to perform which jobs in which order for production systems with a shop structure (i.e. flow shop, job shop, or open shop) [261]. Typically, the solution to this planning problem is a dispatch list (also called *sequence* or *schedule*) for each resource, containing the order in which the jobs should be handled. The most favorable schedule is determined based on the performance objectives for the process. Well-known strategies are for instance First Come First Served (FCFS) or Earliest Due Date (EDD) [261]. Note that, although there are many similarities between job scheduling and our problem of selecting the best operation from the set of executable operations, we are not looking for a schedule but only for the first element in this sequence. Moreover, production planning schedules jobs for different cases on one machine, while we are ranking the operations (cf. jobs) of one single case according to a certain performance goal.

Production planning problems are usually too large to be solved mathematically and researchers therefore have developed pragmatic ways (e.g. heuristics) that approximate the desired solution to a scheduling problem. Many different strategies or rules exist to schedule jobs that have to be performed by a machine [210, 261]. The following strategies can also be translated to our selection problem:

- *Random* - The operation is randomly selected from the set of executable operations (cf. [210]).
- *Lowest cost* - The operation with the lowest cost is selected.
- *Shortest processing time* - The operation with the shortest duration is chosen (cf. SPT defined in [210]).
- *Lowest failure probability* - The operation with the lowest probability of failure of execution is chosen.
- *Distance to root element* - The operation with the shortest distance to the root element (measured in the total number of operations) is selected. The distance of an operation to the root element is the 'shortest path' from this operation to an operation that produces the root element. The distance to the root element can be measured as the total number of operations to the root element (cf. FOPNR defined in [210]).
- *Shortest remaining processing time* - The operation with the shortest remaining processing time is chosen. The shortest remaining processing time is another form of the distance to the root element. In this case the processing times of the operations on the path to the root element are added up (cf. SR defined in [210]).



(a) Total cost: 5.0 + 5.0 = 10.0.   (b) Total cost: 9.0.

Figure 6.4: *Two different execution paths leading to the end product of the PDM.*

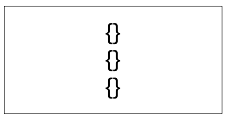
The above strategies present ways to select the next step based on a single strategy, i.e. only one performance goal is considered.

A more advanced way to use these simple selection strategies is by combining them. With a single strategy there may be several best candidates with the same value for the performance goal. To distinguish between the operations that all have the optimal value based on the first strategy, one can use another strategy to complement the first one. For instance, if the lowest cost strategy leads to three operations with the same minimal execution cost, another strategy (e.g. lowest failure probability) may be used to rank these three operations and find the best one among the three.

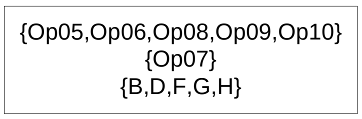
Additionally, weighted criteria can be used. For instance, if the processing time and the execution cost are equally important for the selection of an operation, the selection strategy can be based on an equal weight (0.5 vs 0.5) for both performance criteria.

6.3.1 Local Optimization

Using the strategies presented above, the selection of the next candidate for execution is only optimized locally (i.e. within the set of currently executable operations); the effect of the selected operation on future steps is not taken into account. This may lead to a less than desirable situation if we consider the overall performance of the case. Consider for instance the execution steps for the mortgage example described in Section 6.1 with respect to the total execution cost. The first execution sequence contains $Op02$ followed by $Op01$. The total cost of this execution are: $5.0 + 5.0 = 10.0$. The total cost of the alternative execution path containing only $Op03$ are 9.0. So, the cost of the second execution path are lower. Selecting the best candidate based on the lowest cost selection strategy in this case does not lead to the best overall decision for the case considering the total cost of execution (see Figure 6.4). Hence, we will use a global optimization approach to overcome this problem. Our approach is based on a Markov Decision Process and is discussed in the next section.



(a) The state in which no data element values are available yet, cf. Figure 6.2(a).



(b) The state in which the values for data elements B, D, F, G, and H are available, cf. Figure 6.2(b).

Figure 6.5: *A state in the state space is described by three sets: (i) the operations that have been executed successfully so far, (ii) the operations that have been executed unsuccessfully, and (iii) the data elements for which a value is available.*

6.4 Selection Strategies based on a Markov Decision Process

The technique we use to determine globally optimized selection strategies is based on the theory of Markov Decision Processes (MDPs, introduced in Section 2.7.2). This section first presents a mapping from the execution process of a PDM to an MDP. This mapping is illustrated by an example. A drawback of this method is that problems easily become too large to be computed. Therefore, also some attention is paid to restricting the state space of the MDP problem in such a way that a near-optimal decision strategy can still be calculated for large PDMs.

6.4.1 Formulation of the Markov Decision Process

In this section we describe the mapping from a PDM to an MDP. An MDP is defined by a number of components: the state space, the time space, the decision space, the transition function, and the cost functions (see Section 2.7.2). For each component of the MDP we describe the corresponding part in a PDM, showing that the selection problem in a PDM can be translated to an MDP and solved by the standard algorithms for MDPs. A simple example, based on the mortgage process, is presented to clarify this mapping.

State Space

The state space (S) describes the states the process can be in. The states of the execution of a PDM can be described by the operations that have been executed (either successfully or unsuccessfully, see Assumption 4.3.3) together with the data elements for which a value is available. A state in the state space is therefore represented by a tuple of three sets: (i) the successfully executed operations, (ii) the unsuccessfully executed operations, and (iii) the data elements for which a value is available[3]. In Figure 6.5, for example, the state in the mortgage example in which a value for each of the leaf elements B, D, F, G, and H is available (cf. Figure 6.2(a)) is denoted by $(\{Op05, Op06, Op08, Op09, Op10\}, \{Op07\}, \{B, D, F, G, H\})$. Thus, $S \subseteq \mathcal{P}(O) \times \mathcal{P}(O) \times \mathcal{P}(D)$. The state space is finite, since the number of operations is finite.

[3]Note that the set of available data elements in a state actually gives redundant information, since the set of available data elements can also be determined based on the set of successfully executed operations. However, this information is added for reasons of clarity.

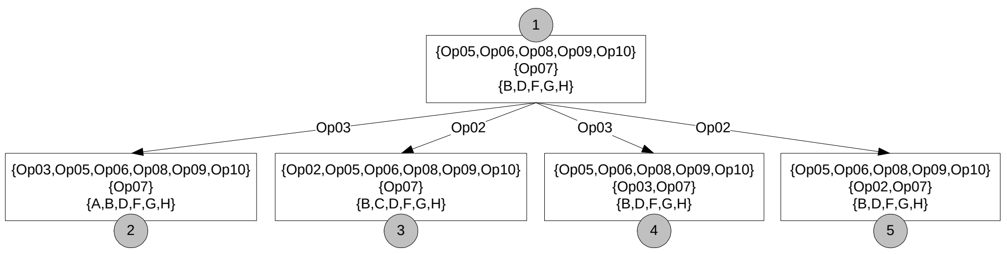

1
{Op05,Op06,Op08,Op09,Op10}
{Op07}
{B,D,F,G,H}
Op03
Op02
Op03
Op02
{Op03,Op05,Op06,Op08,Op09,Op10}
{Op07}
{A,B,D,F,G,H}
2
{Op02,Op05,Op06,Op08,Op09,Op10}
{Op07}
{B,C,D,F,G,H}
3
{Op05,Op06,Op08,Op09,Op10}
{Op03,Op07}
{B,D,F,G,H}
4
{Op05,Op06,Op08,Op09,Op10}
{Op02,Op07}
{B,D,F,G,H}
5

Figure 6.6: *From the state in which all leaf operations have been executed and the values for B, D, F, G, and H are available (state 1), the process can move to four possible states. The decision to execute $Op02$ leads to state 5 if the execution of the operation fails and to state 3 if the execution of the operation is successful. Similarly, decision $Op03$ leads to either state 4 or state 2.*

Time Space

The time space describes the time points at which a decision is taken and a state transition occurs (also called *decision epochs*). The set of times (T) is deterministic and can be represented by the number of executed operations ($T = \{0, 1, 2, 3, ..., |O|\}$). The times are not necessarily equidistant, but since there is a finite number of operations in a product data model and no concurrency, the time space is bounded by this number, i.e. the MDP problem has a finite time horizon. Thus, the set of times can be used as a counter.

Decision Space

In each state a number of decisions can be made. For the execution of a PDM these decisions are described by the set of operations that are executable in the current state of execution (i.e. those operations of which the input elements are available and that have not yet been executed)[4]. Moreover, if there are no executable operations for a certain state there is only one decision possible, i.e. to stop. Thus, the decision space A is equal to the set of operations plus the decision to stop, i.e. $A = (O \cup \{stop\})$. Furthermore, the decision space in a particular state A_i is a subset of $(O \cup \{stop\})$. Note that the decision space in a certain state is time-independent, i.e. given a state, at any point in time, the same decisions can be made.

Transition Probabilities

The transition probabilities are given by a matrix P that describes the probabilities that the system moves from the current state to any of the other states in the system. These transition probabilities are dependent on the decision that was made. For our application, a decision a in state i can lead to two new states: j_1 (for a successful execution of the operation given in decision a) and j_2 (for a failed execution of the operation given in decision a), each with their own probabilities. The two transition probabilities under decision a always add up to 1. For example, recall the execution of the mortgage example in Figure 6.2. If we start in the state with available values for data elements B, D, F, G, and H, there are two decisions which can be taken, i.e. $\{Op02, Op03\} \subseteq O$. Each of these two operations can either fail or be successfully executed. This leads to four new states (Figure 6.6). The transition probabilities correspond to the probabilities of failure or successful execution of an operation:

[4]Note that the decisions are not dependent on the value of a data element since in general data elements may have infinitely many values, which would lead to an infinitely large state space.

$$p_{1,3}(Op02) = 0.95 \qquad p_{1,2}(Op03) = 0.95$$
$$p_{1,5}(Op02) = 0.05 \qquad p_{1,4}(Op03) = 0.05$$

Thus, the probability of moving from state 1 to state 3 under decision $Op02$ is 0.95.

Immediate and Final Cost

The immediate cost of a transition from state i to state j under decision a are the cost of executing operation a. In our model there are no cost for staying in a certain state. Depending on the performance objective of the process, e.g. minimization of cost or processing time, the immediate cost are defined as the cost or the processing time of the operation. Suppose we want to minimize the execution cost of the process, then the immediate cost in state 1 are:

$$c^{Op02}(1) = 5.0 \qquad c^{Op03}(1) = 9.0$$

The final cost incurred at time $|O|$, when no decisions can be made anymore, in state i are zero, i.e. $q(i) = 0$.

By using the mapping presented above, the process of executing a PDM can be translated to a time-homogenous MDP with a finite horizon. We have used a small part of the mortgage example to illustrate the mapping. Below, it is demonstrated how a stationary Markov decision rule is determined for the complete mortgage example.

Note that this mapping of a PDM to an MDP results in a Markov chain with some special properties. First of all, the state space of the Markov chain is *finite*, i.e. there is a finite number of states. Secondly, each operation can be executed at most once. Thus, each decision can also be taken at most once during any execution. Moreover, at each point in time an operation is selected until there are no executable operations anymore. The only decision that can be made then is to stop and the system stays in the same state. These end states are absorbing states. The Markov chain is a *transient* chain (see Section 2.7.1) since each closed class has exactly one absorbing state. Besides the absorbing end states there are *no cycles* in the Markov chain, i.e. it is not possible to return to a state that was previously visited.

6.4.2 The Mortgage Example as an MDP

In this section, the MDP problem for the mortgage example is detailed. The initial situation in which there is already a value for data elements B, D, F, G, and H (described in Section 6.1) is used again because in this case the decision space is rather small. Therefore the state space for this MDP problem stays small and readable. The state space is depicted in Figure 6.7 and describes all possible execution steps from the initial state. The decision space per state can also be derived from the figure, e.g. in state 3 two alternatives exist: $Op01$ and $Op03$. Because the execution of both operations can either be successful or unsuccessful, state 3 has 4 outgoing arcs to four new states. If operation $Op01$ is executed successfully, the system moves from state 3 to state 6. Confronted with an unsuccessful execution of $Op01$, the system moves to state 7. Similarly, upon execution of $Op03$ the system moves to state 8 or 9. The probabilities for these transitions are summarized in Table 6.1. Finally, for this example we focus on minimizing the total cost for a case. The cost for executing an operation can be found in Table 3.2.
The best strategy for decision making can be calculated using the value iteration algorithm already described in Section 2.7.2. Below, the first steps are detailed, starting from the situation in which no decision can be made anymore ($n = 0$). Recall that the value iteration algorithm

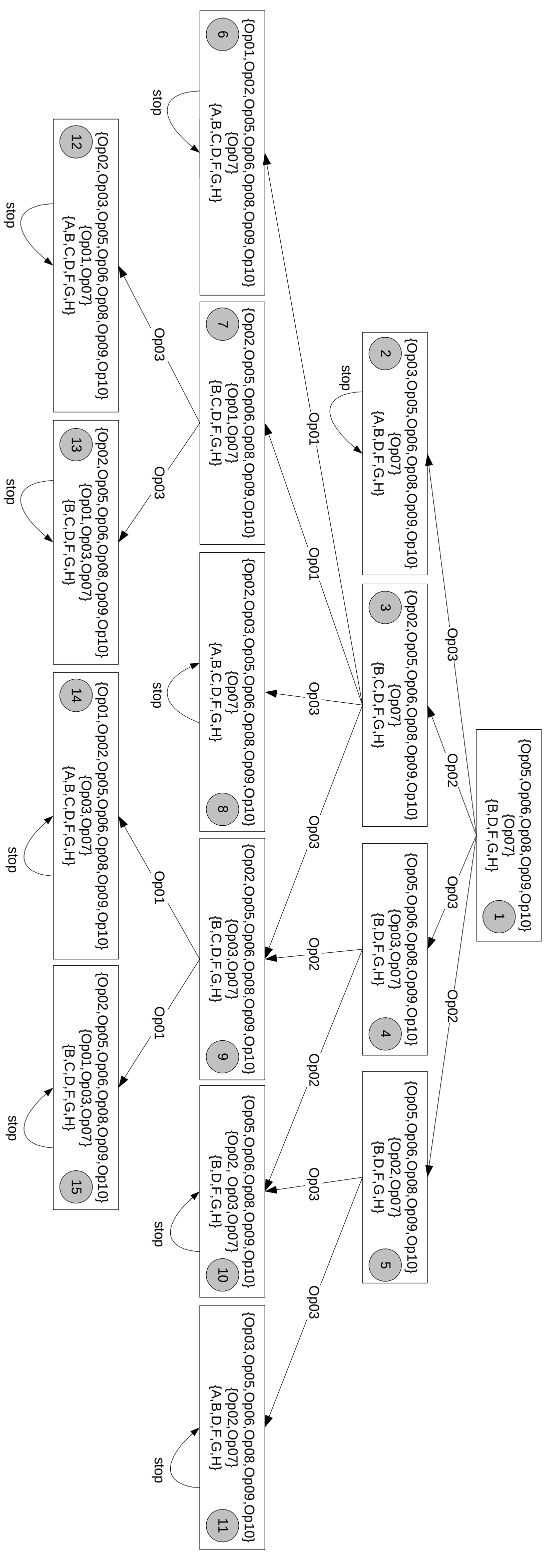

1 {Op05,Op06,Op08,Op09,Op10} {Op07} {B,D,F,G,H}
2 {Op03,Op05,Op06,Op08,Op09,Op10} {Op07} {A,B,D,F,G,H}
3 {Op02,Op05,Op06,Op08,Op09,Op10} {Op07} {B,C,D,F,G,H}
4 {Op05,Op06,Op08,Op09,Op10} {Op03,Op07} {B,D,F,G,H}
5 {Op05,Op06,Op08,Op09,Op10} {Op02,Op07} {B,D,F,G,H}
6 {Op01,Op02,Op05,Op06,Op08,Op09,Op10} {Op07} {A,B,C,D,F,G,H}
7 {Op02,Op05,Op06,Op08,Op09,Op10} {Op01,Op07} {B,C,D,F,G,H}
8 {Op02,Op03,Op05,Op06,Op08,Op09,Op10} {Op07} {A,B,C,D,F,G,H}
9 {Op02,Op05,Op06,Op08,Op09,Op10} {Op03,Op07} {B,C,D,F,G,H}
10 {Op05,Op06,Op08,Op09,Op10} {Op02, Op03,Op07} {B,D,F,G,H}
11 {Op03,Op05,Op06,Op08,Op09,Op10} {Op02,Op07} {A,B,D,F,G,H}
12 {Op02,Op03,Op05,Op06,Op08,Op09,Op10} {Op01,Op07} {A,B,C,D,F,G,H}
13 {Op02,Op05,Op06,Op08,Op09,Op10} {Op01,Op03,Op07} {B,C,D,F,G,H}
14 {Op01,Op02,Op05,Op06,Op08,Op09,Op10} {Op03,Op07} {A,B,C,D,F,G,H}
15 {Op02,Op05,Op06,Op08,Op09,Op10} {Op01,Op03,Op07} {B,C,D,F,G,H}
Op01
Op02
Op03
stop

Figure 6.7: *The state space of the mortgage example. Note that there are eight end states:* {2, 6, 8, 10, 11, 12, 13, 14, 15}.

state	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0.0	0.95	0.95	0.05	0.05	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.95	0.05	0.95	0.05	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.95	0.05	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.05	0.95	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.95	0.05	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.95	0.05
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Table 6.1: *Transition probabilities for the state space of the mortgage example in Figure 6.7.*

uses backward tracking (see Section 2.7.2) and therefore starts in the situation in which no decisions are left to be made ($n = 0$). $V_n(i)$ denotes the total expected cost at decision epoch n and state i. The cost at decision epoch $n = 0$ are equal to the final cost, i.e. $V_0(i) = 0$, $\forall i \in S$, cf. Section 2.7.2.

$$V_0(i) = 0, \forall i \in S.$$

Next, the values for $n = 1$ can be calculated based on all $V_0(i)$ values. For instance, two decisions can be made in state 3: $Op01$ and $Op03$. The expected cost for choosing $Op01$ are dependent on the immediate cost for choosing $Op01$ in state 3, and the expected cost and transition probabilities for the states the process can move to under decision $Op01$:

$$c^{Op01} + p_{3,6}(Op01) \cdot V_0(6) + p_{3,7}(Op01) \cdot V_0(7) = 5.0 + 0.95 \cdot 0.0 + 0.05 \cdot 0.0 = 5.0.$$

Similarly, the expected cost in state 3 for decision $Op03$ can be calculated (9.0). Since decision $Op01$ has the lowest expected total cost it is the best decision. Thus, if the process is in state 3 and there is only one decision to be taken until the end of the process, the best decision is $Op01$, i.e. $f_1(3) = Op01$, with expected total cost of 5.0, i.e. $V_1(3) = 5.0$. In the same way the expected cost for all other states can be calculated:

$$\begin{aligned}
V_1(1) &= \min\{Op02; Op03\} \\
&= \min\{c^{Op02} + p_{1,3}(Op02) \cdot V_0(3) + p_{1,5}(Op02) \cdot V_0(5); \\
&\quad c^{Op03} + p_{1,2}(Op03) \cdot V_0(2) + p_{1,4}(Op03) \cdot V_0(4)\} \\
&= \min\{5.0 + 0.95 \cdot 0.0 + 0.05 \cdot 0.0; 9.0 + 0.95 \cdot 0.0 + 0.05 \cdot 0.0\} \\
&= \min\{5.0; 9.0\} = 5.0 \\
V_1(2) &= \min\{stop\} \\
&= \min\{c^{stop} + p_{2,2}(stop) \cdot V_0(2)\} \\
&= \min 0.0 + 1.0 \cdot 0.0 = 0.0 \\
&= V_1(6) = V_1(8) = V_1(10) = V_1(11) = V_1(12) = V_1(13) = V_1(14) = V_1(15) \\
V_1(3) &= \min\{Op01; Op03\} \\
&= \min\{c^{Op01} + p_{3,6}(Op01) \cdot V_0(6) + p_{3,7}(Op01) \cdot V_0(7); \\
&\quad c^{Op03} + p_{3,8}(Op03) \cdot V_0(8) + p_{3,9}(Op03) \cdot V_0(9)\} \\
&= \min\{5.0 + 0.95 \cdot 0.0 + 0.05 \cdot 0.0; 9.0 + 0.95 \cdot 0.0 + 0.05 \cdot 0.0\} \\
&= \min\{5.0; 9.0\} = 5.0 \\
V_1(4) &= \min\{Op02\} \\
&= \min\{c^{Op02} + p_{4,9}(Op02) \cdot V_0(9) + p_{4,10}(Op02) \cdot V_0(10)\} \\
&= \min\{5.0 + 0.95 \cdot 0.0 + 0.05 \cdot 0.0\} \\
&= \min\{5.0\} = 5.0 \\
V_1(5) &= \min\{Op03\} \\
&= \min\{c^{Op03} + p_{5,11}(Op03) \cdot V_0(11) + p_{5,10}(Op030) \cdot V_0(10)\} \\
&= \min\{9.0 + 0.95 \cdot 0.0 + 0.05 \cdot 0.0\} \\
&= \min\{9.0\} = 9.0
\end{aligned}$$

$$\begin{aligned}
V_1(7) &= \min\{Op03\} \\
&= \min\{c^{Op03} + p_{7,12}(Op03)\cdot V_0(12) + p_{7,13}(Op03)\cdot V_0(13)\} \\
&= \min\{9.0 + 0.95\cdot 0.0 + 0.05\cdot 0.0\} \\
&= \min\{9.0\} = 9.0 \\
V_1(9) &= \min\{Op01\} \\
&= \min\{c^{Op01} + p_{9,14}(Op01)\cdot V_0(14) + p_{9,15}(Op01)\cdot V_0(15); \\
&= \min\{5.0 + 0.95\cdot 0.0 + 0.05\cdot 0.0\} \\
&= \min\{5.0\} = 5.0 \\
\ldots & \quad \ldots
\end{aligned}$$

From the above calculation we can see that if there is only one time period left, it is best to choose $Op02$ in state 1 since $V_1(1)$ has the minimal value of 5.0 for its argument $Op02$. Also, $Op02$ should be chosen in state 4 ($V_1(4) = 5.0$ for $Op02$), etc. In states 2, 6, 8, 10, 11, 12, 13, 14, and 15 the only decision that can be made is to stop since these states are end states in the state space. Thus, the decision strategy f_1 for time 1 is:

$$\begin{aligned}
f_1 \;=\; & \{(1, Op02), (2, \text{stop}), (3, Op01), (4, Op02), (5, Op03), (6, \text{stop}), (7, Op03), (8, \text{stop}), \\
& (9, Op01), (10, \text{stop}), (11, \text{stop}), (12, \text{stop}), (13, \text{stop}), (14, \text{stop}), (15, \text{stop})\}.
\end{aligned}$$

In a similar way, the values and strategy for all time points can be calculated. The details can be found in tables 6.2 and 6.3. The tables show that the decision rule becomes stationary for sufficiently large n (i.e. $n = 3$ for this case), since the Markov chain does not contain any cycles other than the absorbing end states:

$$\begin{aligned}
f \;=\; & \{(1, Op03), (2, \text{stop}), (3, Op01), (4, Op02), (5, Op03), (6, \text{stop}), (7, Op03), (8, \text{stop}), \\
& (9, Op01), (10, \text{stop}), (11, \text{stop}), (12, \text{stop}), (13, \text{stop}), (14, \text{stop}), (15, \text{stop}))\}.
\end{aligned}$$

Note that the best decision for the first state has changed from $Op02$ at $n = 1$ to $Op03$ at $n \geq 2$. The situation with $n = 1$ is comparable to the simple strategies discussed in Section 6.3, while the algorithm also considers future effects of a decision when $n \geq 2$.

6.4.3 The Size of the State Space

The state space of our MDP problem is finite, since the number of operations is finite and the state space only contains all possible combinations of successfully and unsuccessfully executed operations at a maximum. Nevertheless, the state space can become extremely large. If we use the initial situation in the mortgage example in which none of the operations have been executed yet and no value is available for any of the data elements, i.e. the initial state with $(\{\}, \{\}, \{\})$, the state space contains 2218 states and, as such, is much larger than in our example of Figure 6.7. This is because the six leaf operations can be executed in an arbitrary order. All possible combinations are then explicitly represented in the state space.

An upper bound to the size of the state space can be given in terms of the number of operations of the PDM. Each operation can be either (i) successfully executed, (ii) unsuccessfully executed (failed), or (iii) not yet executed. Thus, there are $3^{|O|}$ possible combinations of operations, which is an upper bound to the size of the state space. However, most state spaces will be smaller since not all combinations of executed operations are possible. For instance, the state space for the mortgage example will never contain the state ({Op02,Op07}{Op09}{C,G}), since $Op02$ can never be executed before both $Op08$ and $Op10$ are executed.

n/i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	5.0	0.0	5.0	5.0	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
2	9.25	0.0	9.25	9.75	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
3	9.49	0.0	9.25	9.75	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
4	9.49	0.0	9.25	9.75	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
5	9.49	0.0	9.25	9.75	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
6	9.49	0.0	9.25	9.75	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
7	9.49	0.0	9.25	9.75	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
8	9.49	0.0	9.25	9.75	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0
9	9.49	0.0	9.25	9.75	9.0	0.0	9.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 6.2: *The minimal cost values obtained by the value iteration algorithm, i.e. $V_n(i)$. Note that the values become stable, i.e. they do not change anymore, for $n \geq 3$.*

n/i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	**Op02**	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
2	**Op03**	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
3	Op03	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
4	Op03	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
5	Op03	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
6	Op03	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
7	Op03	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
8	Op03	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
9	Op03	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-
10	Op03	-	Op01	Op02	Op03	-	Op03	-	Op01	-	-	-	-	-	-

Table 6.3: *The rows in this table denote the decision strategy (f_n) for each point in time (n). In the cells of the table the arguments for the minimal cost in Table 6.2 can be found. Per row the best decision is given for each state i. If a '-' is shown, this means that the only decision to be taken is to stop since the state is an end state in the state space. Note that the strategy differs if there is only one decision epoch left to the time horizon ($n = 1$) from the situation in which more decisions can follow each other. The first row is comparable to the local optimization with a simple selection strategy described in Section 6.3.1.*

If an MDP problem becomes too large due to the number of states in the state space, it cannot be solved mathematically any more. The social benefits example, for instance, is much larger than the mortgage example. During our experiments it turned out that a normal PC[5] was not able to process the complete state space of the unemployment benefits example. However, this does not necessarily mean that no solution can be found based on the global optimization method. By restricting the state space it cannot be guaranteed that the solution which is found is optimal, but it is still possible to find a good (i.e. nearly optimal) solution. Below, some ideas to limit the state space are discussed.

Restricting the Size of the State Space

The first opportunity to limit the size of the state space is to *limit the number of possibilities from the initial state*. If the initial state is defined to be a situation without any of the operations (including the leaf operations) executed, the possibilities to proceed from that state are very diverse, as we explained above. All leaf operations can then be executed in an arbitrary order, leading to a large number of combinations and thus to a large number of states. A way to

[5]A normal PC at the time of writing this thesis is e.g. an Intel Pentium 4 with a 3.8GHz processor and 3GB of RAM.

overcome this problem is to start in a situation that assumes that a number of leaf elements (or all of them) are available already, i.e. all leaf operations have been executed[6]. The illustration with the mortgage example for instance uses an initial situation in which all leaf operation have been executed. It may take some extra effort to be sure that the input information to a case is complete, but it certainly reduces the size of the state space by a large number.

Secondly, we can *abstract from the failure probability* of an operation. By assuming that the execution of an operation always leads to successful determination of its output product, the number of states in the state space can be halved at least, since the decision for one operation in a certain state only leads to one new state instead of two new states.

The number of states in the state space can also be *bounded by a predefined limit*, i.e. the construction of the state space just stops when this limit on the number of states has been reached. Depending on the way of constructing the state space this approach may lead to a very incomplete state space. For instance, if the decision space per state is very large and the state space is built in a breadth-first manner, it may well be that the state space is only constructed for one or two levels and does not reach a proper end state. However, also a depth-first strategy may be used to build the state space.

Moreover, the *breadth of the state space* can be limited when many decisions can be made in each state. The number of possible outgoing arcs from a state can be limited for instance, or the number of decisions can be restricted.

Note that these four options to restrict the size of the state space do not change the PDM or the dependencies between data elements. By using these options the state space for the MDP is built only partially.

We have experimented by calculating the state space for the social benefits example using these ideas to limit the state space. One of the main design objectives for the redesign project of the social benefits case was a minimum number of contacts with the client (see the description of the project in [231]). This requirement was realized in the design by requesting all input information from the claimant at once at the start of the case. Therefore, the restriction of using an initial state in which all leaf operations have been (successfully) executed is not unrealistic. Furthermore, it also seemed realistic to assume that all operations are executed successfully, i.e. the execution of an operation cannot fail, since the failure probabilities for the operations in the PDM are typically very low (cf. Table 3.5). With these two restrictions, the state space for the social benefits case was calculated in 2.5 days on a Pentium 4 PC with 3.8GHz processor and 3GB of RAM. The partial state space contains 66.581 states.

6.5 Comparison of Strategies

As an illustration of the difference in performance between the simple selection strategies and the optimal strategy which is calculated based on an MDP, we have executed a simulation study with the CPN model of the functional design presented in Section 6.2. More information on the implementation of the simple selection strategies in the CPN model can be found in Appendix C. For each of the *lowest cost, shortest processing time, lowest failure probability*, and *random* strategies we have executed several simulation runs for the execution of the mortgage example (see Section 3.2.2). A simulation run corresponds to one complete execution of the PDM. For each execution the total execution cost were determined. Per strategy we collected 60

[6]This assumption is even not that unrealistic. For example, one of the requirements for the redesign of the social benefits case was that all input data values should be requested with the client in one consultation [231].

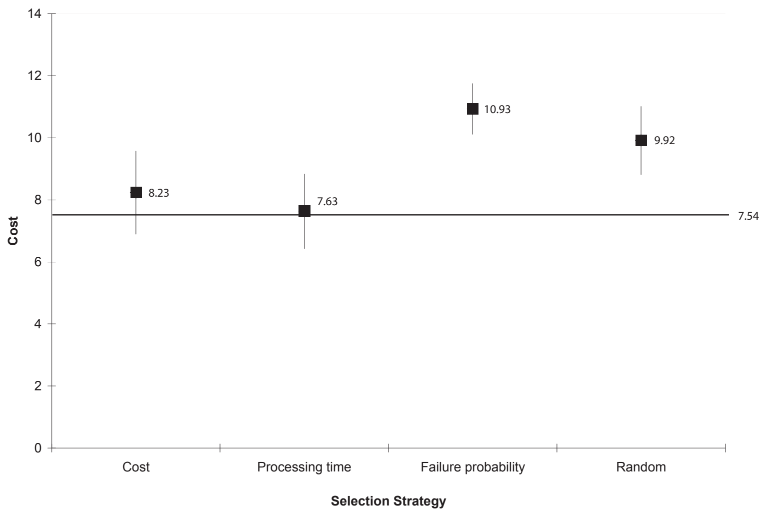


Figure 6.8: *The 90%-confidence intervals for the total cost for the four simple selection strategies, based on a simulation with 60 runs. The* lowest cost *strategy has a mean of* 8.23*, a lower limit of* 6.90 *and an upper limit of* 9.57*. The* shortest processing time *strategy has a mean of* 7.63*, a lower limit of* 6.44 *and an upper limit of* 8.83*. The* lowest failure probability *strategy has a mean of* 10.93*, a lower limit of* 10.11 *and an upper limit of* 11.75*. The* random *strategy has a mean of* 9.92*, a lower limit of* 8.82 *and an upper limit of* 11.01*. Also, the expected total cost related to the decision strategy of the MDP are indicated in the figure:* 7.54*.*

samples (i.e. $n = 60$) and constructed 90%-confidence intervals according to the formula for a t-distribution[7] [182].

The results of this simulation study are shown in Figure 6.8. Note that because the confidence intervals of the different strategies overlap it is not possible to draw reliable conclusions. However, it is striking that, for this specific example, the average total cost under a *shortest processing time* strategy are lower than those for the *lowest cost* strategy. Also, the *lowest failure probability* strategy has a significantly higher expectation of the total cost. In the figure we have also included the total expected cost for the decision strategy calculated for the MDP. The total cost are then 7.54, which indeed is lower than all averages under the four simple strategies.

6.6 Tool Support in ProM and DECLARE

In this section we describe the tool we have developed for supporting the direct execution of a PDM by guiding the user through all steps to the end product. The prototype, developed in the context of the ProM and DECLARE frameworks, presents execution recommendations to the user based on the selected strategy and the current state of the case. Section 6.6.1 explains the architecture of this *PDM recommendation tool*. Next, it is explained how the simple strategies as well as the Markov decision strategies are implemented in the recommendation tool.

[7]We assume that the population has a normal distribution.

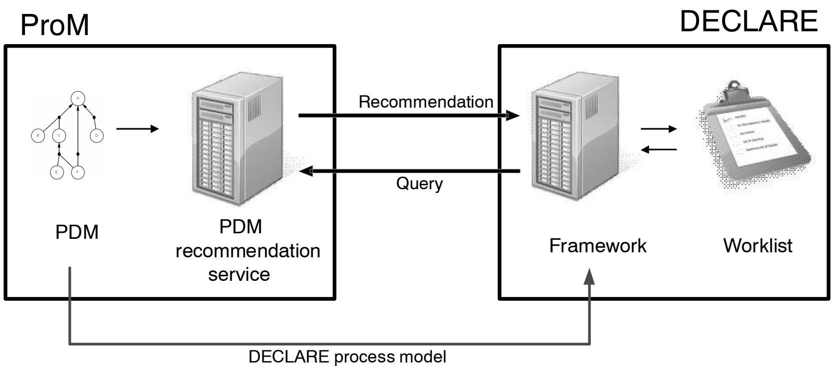


Figure 6.9: *An overview of the architecture of the PDM recommendation tool.*

6.6.1 The PDM Recommendation Tool

The PDM recommendation tool builds on two frameworks: (i) the ProM framework, and (ii) the DECLARE framework. Within each of the frameworks part of the solution is realized. The ProM part is responsible for calculating the recommendations via the *recommendation service*, while the DECLARE part is used to communicate with the user of the recommendation service via the worklist. ProM already incorporated an analysis plug-in that provides process execution recommendations based on the *event log* of a process [293]. The *PDM recommendation service* uses the infrastructure of this log based recommendation service in a different way. Instead of the event log, the PDM recommendation service uses the PDM model and the selected strategy to calculate the recommended activities. An overview of the architecture of the PDM recommendation tool is shown in Figure 6.9.

As a first step in the use of the PDM recommendation tool, a PDM is loaded into the ProM framework and exported to a DECLARE model. This DECLARE model contains solely the operations of the PDM and their input and output data elements. Thus, no control flow is added to the DECLARE model, i.e. the DECLARE activities are not related to each other in any way. In Figure 6.10 the exported DECLARE model for the mortgage example is shown.

Then, the ProM PDM recommendation service is started and a strategy is selected by the user. Actual cases can be handled after the DECLARE model has been loaded into the DECLARE framework and the DECLARE worklist has been started. The DECLARE framework communicates with the ProM recommendation service by sending *queries*. Such a query contains the current state of the case in terms of executed activities and available data element values. Based on this information, the PDM recommendation service calculates which operations are executable and which one of those is the best candidate to be actually executed for the case. The result of this calculation is sent back to the DECLARE framework as a *recommendation*. Finally, this recommendation is shown to the user in the worklist of DECLARE.

Two different implementations of the PDM recommendation service have been developed. The first one uses the simple selection strategies as described in Section 6.3. The second implementation is based on the Markov decision strategies introduced in Section 6.4. Both implementations are described in more detail in the next sections.

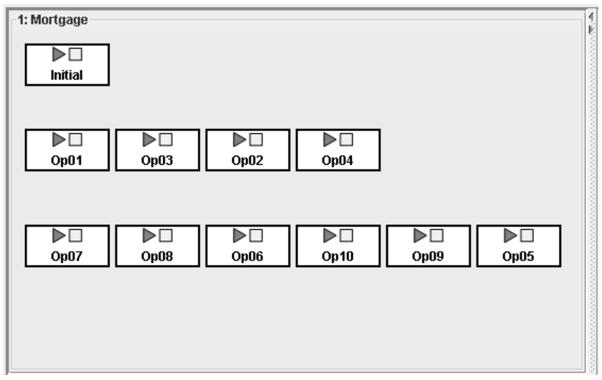


Figure 6.10: *The DECLARE model for the mortgage example. Each operation is translated to a DECLARE activity. At the bottom, all leaf operations are represented. The middle row contains all other operations of the PDM. At the top of the model an 'Initial' activity is added. Through this activity it is possible to fill values for all leaf data elements at once such that the user does not need to go through all leaf operations $Op05$, $Op06$, $Op07$, $Op08$, $Op09$, and $Op10$ separately. Thus, it provides an (optional) shortcut to define the starting situation of a case. Note that there are no dependencies or constraints between the activities in this model.*

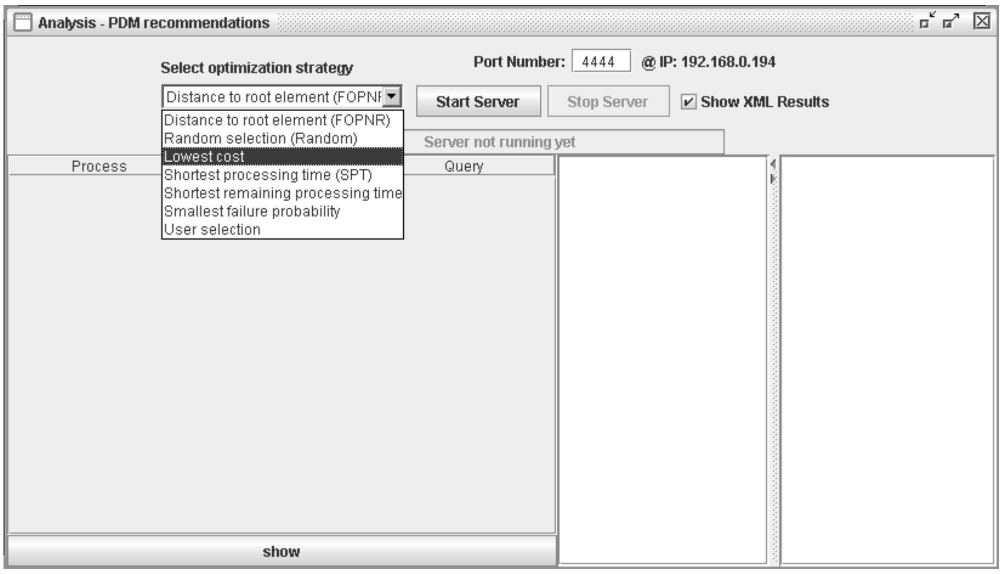
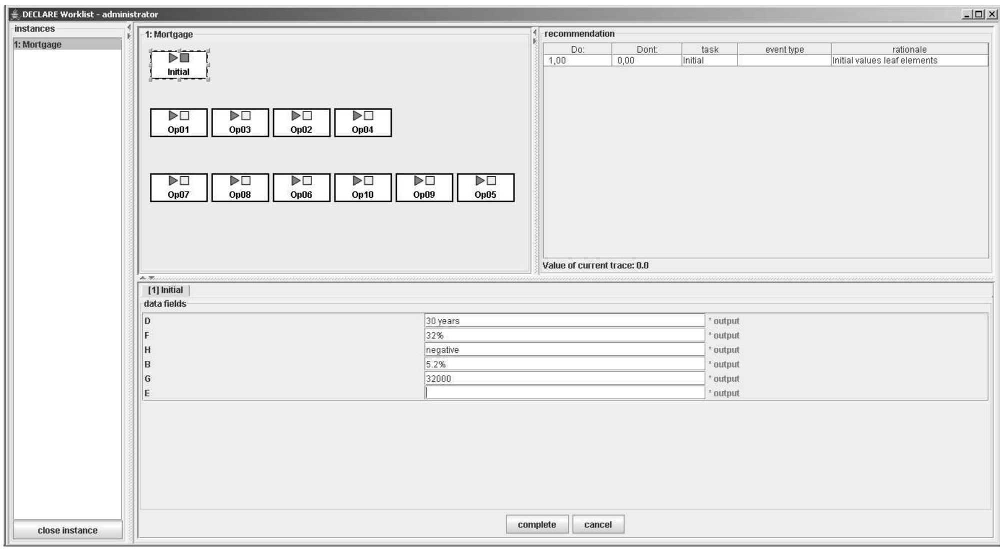

Analysis - PDM recommendations
Select optimization strategy
Distance to root element (FOPNR
Distance to root element (FOPNR)
Random selection (Random)
Lowest cost
Shortest processing time (SPT)
Shortest remaining processing time
Smallest failure probability
User selection
Port Number: 4444 @ IP: 192.168.0.194
Start Server
Stop Server
Show XML Results
Server not running yet
Process
Query
show

Figure 6.11: *The graphical user interface of the PDM recommendation service with simple strategies. One of the strategies can be selected from the drop-down menu and the recommendation service can be started by clicking the 'start server' button.*



recommendation

Do:	Dont:	task	event type	rationale
1,00	0,00	Initial		Initial values leaf elements

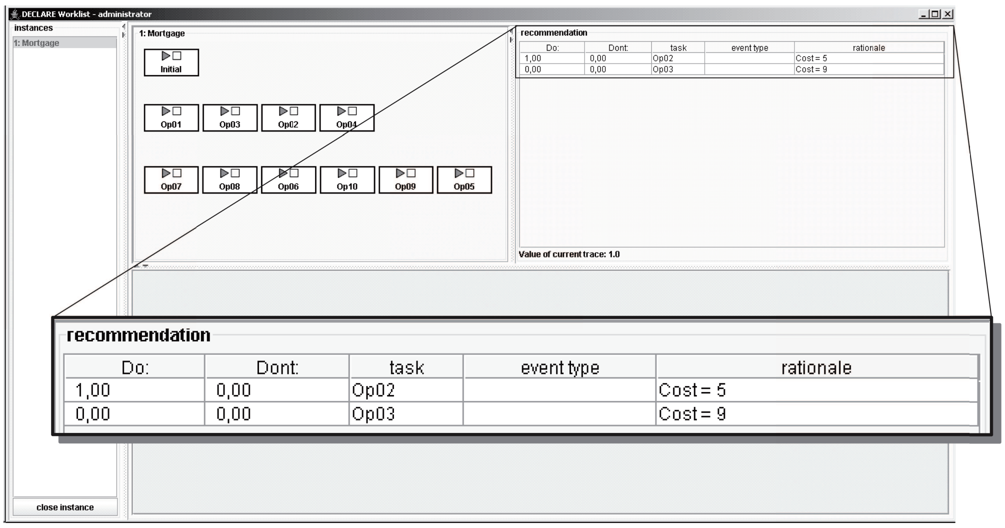
Value of current trace: 0.0

[1] Initial

data fields

D	30 years	* output
F	32%	* output
H	negative	* output
B	5.2%	* output
G	32000	* output
E		* output

Figure 6.12: *The DECLARE worklist with the list of recommendations. In the top left corner the DECLARE process model is shown. In the top right corner the recommendation(s) are presented. In the lower frame the user fills out the data element values to execute an operation. In this case, the recommended operation is the initial operation to fill out all leaf element values and the user has filled out values for data elements B, D, F, G, and H.*

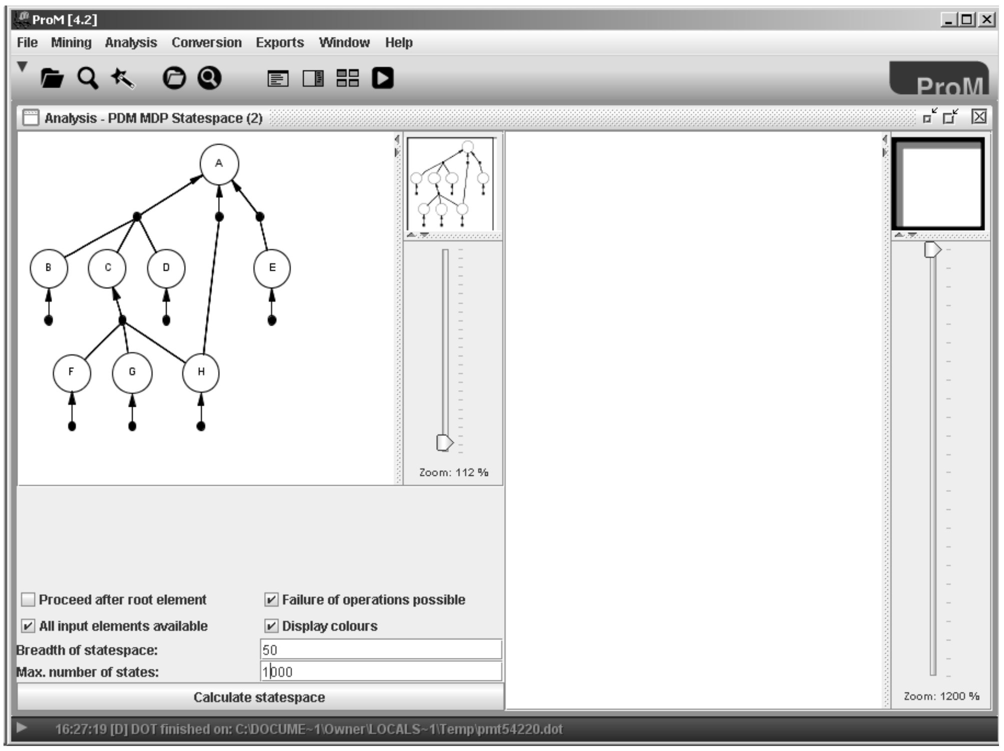


recommendation

Do:	Dont:	task	event type	rationale
1,00	0,00	Op02		Cost= 5
0,00	0,00	Op03		Cost= 9

Value of current trace: 1.0

Figure 6.13: *Two recommended activities are shown for the process which is in the state where all leaf operations have been executed and the values for data elements B, D, F, G, and H are available. There is no value present for E. $Op02$ is the best decision (i.e. the 'do' value is 1.0), since its cost are lower than the cost of the other recommended operation $Op03$. Note that a lowest cost strategy was selected (cf. Figure 6.11).*


ProM [4.2]
File Mining Analysis Conversion Exports Window Help
ProM
Analysis - PDM MDP Statespace (2)
A
B
C
D
E
F
G
H
Zoom: 112 %
Zoom: 1200 %

Proceed after root element
✓ Failure of operations possible
✓ All input elements available
✓ Display colours
Breadth of statespace: 50
Max. number of states: 1000
Calculate statespace

16:27:19 [D] DOT finished on: C:\DOCUME~1\Owner\LOCALS~1\Temp\pmt54220.dot

Figure 6.14: *The user interface of the PDM recommendation tool based on the calculation of a Markov decision strategy. A number of options should be selected to build the state space for the MDP problem: (i) the possibility to proceed the execution process after the root element has been determined, (ii) the possibility to have all leaf data elements available at the start of the process, (iii) the possibility of unsuccessful execution of operations (failure), (iv) the possibility to show colors in the state space, (v) a number to limit the breadth of the number of states following each state, and (vi) a number to limit the total number of states in the state space. When the appropriate options have been checked, the state space can be calculated by clicking on the button in the lower left corner.*

6.6.2 Simple Selection Strategies

In the PDM recommendation service based on simple strategies, the user can select one of the strategies of Section 6.3 via a drop-down menu (see Figure 6.11). Once the strategy has been selected, the recommendation service can be started immediately by clicking the ‘start server’ button. Also, the DECLARE framework and the DECLARE worklist are started. The user fills out the values for the leaf data elements that are available via the ‘initial’ task (see Figure 6.12). Suppose we take the same initial situation for the mortgage example as before (i.e. values for B, D, F, G, and H are available and E is not available). Then, the recommendation service determines which operations are executable and which one of these is the best choice. Figure 6.13 shows the recommended next steps based on the lowest cost strategy. Note that a list of all executable operations is presented with the best next step listed first. The user can now decide whether or not to follow the recommendation and execute a next step. In any case, the recommendation service will again calculate a recommendation for the new state of the process until the end product is produced.

A detailed description of a PDM execution example using the PDM recommendation tool with a simple strategy can be found in [280]. Note that it is possible to change the selection strategy during the execution of a case. This is useful, for example, if a company decides to switch from a low cost strategy to a strategy focusing on a short throughput time to be able to deal with extra case arrivals as quickly as possible.

6.6.3 Selection Strategies based on a Markov Decision Process

In case a Markov decison strategy is chosen the user cannot immediately start the recommendation service in ProM. First, the state space for the PDM has to be computed. Then, the optimal strategy can be determined based on the selected performance objective and the recommendation service can be started. These steps are explained in more detail below.

Calculation of State Space

The calculation of the state space is implemented as an analysis plug-in in ProM. When the plug-in is started, a number of options are available through checkboxes and parameter fields (see Figure 6.14). These options mainly provide a means to restrict the state space as discussed in Section 6.4.3:

· *'All input elements available'*. If this checkbox is checked the state space is built from the situation that all leaf operations have been executed successfully and that a value is available for all leaf data elements. In the default setting this checkbox is not checked and the ({}{}{})-state is used as the initial state.

· *'Failure of operations possible'*. By checking this checkbox, two states are added to the state space for each decision: the selected operation can either be executed successfully or unsuccessfully. By default this option is selected.

· *'Breadth of the state space'*. With this option the user can limit the number of outgoing arcs from each state. Using this option the state space is restricted in its breadth. However, the user should be careful using this option since this may also lead to an incomplete state space with respect to the decision space. By default the breadth of the state space is limited to a rather high number such that in the default setting no information is lost.

· *'Maximum number of states'*. With this option the total number of states in the state space is maximized by the specified number. The state space is built breadth-first. By restricting the number of states, the state space may become incomplete and there may even be no proper end states if the maximum is too restrictive. By default the maximum number of states is set to a reasonably high limit of 1000 states such that any contemporary PC is able to calculate the partial state space in a few minutes. Thus, the program will not crash or enter a time-consuming calculation for the default settings.

Besides these four options related to the construction and restriction of the state space, two extra options are available:

· *'Proceed after root element'*. If this checkbox is checked the state space is continued after a state in which the end product of the process is determined. This means that the process only finishes when all operations have been executed. By default this checkbox is not checked, since we want to finish the process as soon as a value for the end product is produced.

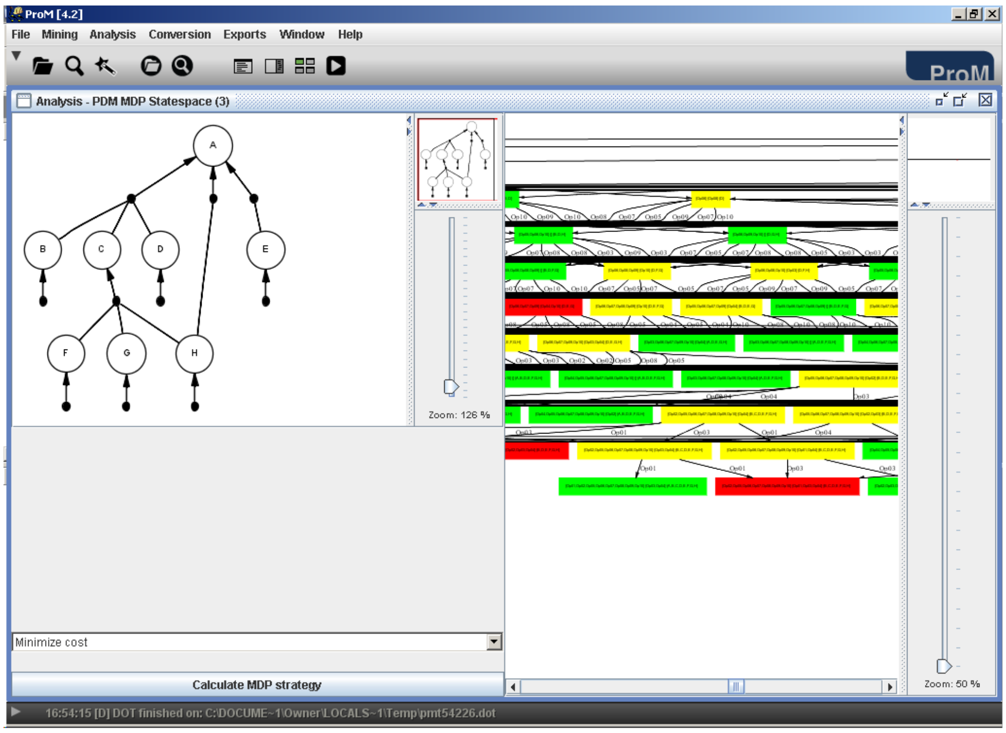
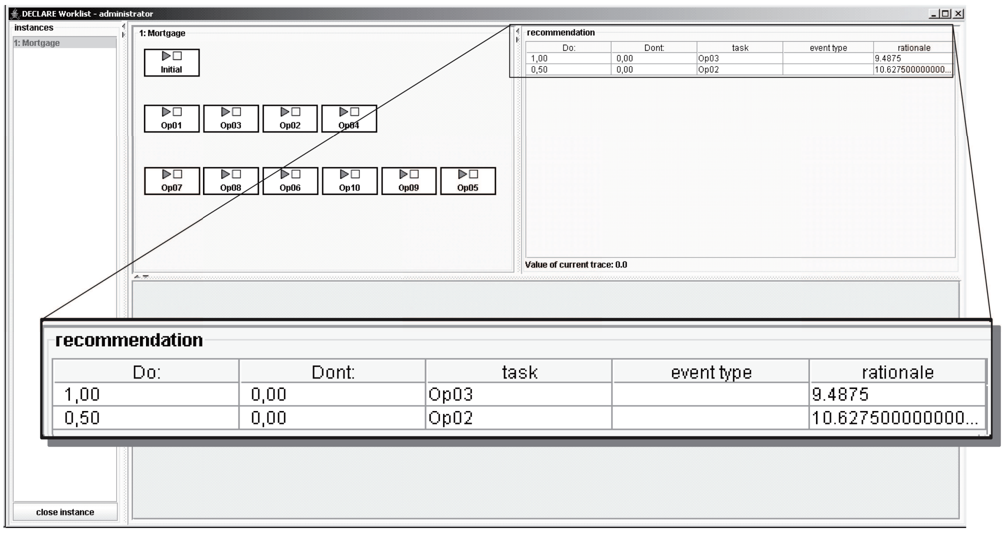

ProM [4.2]
File Mining Analysis Conversion Exports Window Help
Analysis - PDM MDP Statespace (3)
A
B
C
D
E
F
G
H
Zoom: 126 %
Zoom: 50 %
Minimize cost
Calculate MDP strategy
16:54:15 [D] DOT finished on: C:\DOCUME~1\Owner\LOCALS~1\Temp\pmt54226.dot

Figure 6.15: *This screenshot shows the state space that was computed on the right hand side. In the lower left corner the user can select the performance objective to be optimized (e.g. minimize cost, or minimize processing time) via a drop-down menu. Next, the decision strategy can be calculated.*

· *'Display colors'*. If this option is checked the states in the state space are colored. Each state can have one of the colors green, yellow and red. In a green state either nothing has gone wrong yet, i.e. the operation executions that were performed until this state was reached have all been successful, or the state is an end state in which the end product was produced successfully. In a yellow state some of the operation executions have gone wrong, but there is still a possibility to reach a proper end state, i.e. a state in which the end product is successfully produced. Finally, from a red state it is not possible to finish the process in a proper way anymore. In the default setting this feature is switched on.

For the mortgage example the state space can be computed using the default settings (see Figure 6.14). However, for larger PDMs the computation of the complete state space may not be possible. In that case using one or more measures to restrict the state space may help, provided that one is willing to accept that the resulting state space may not be complete anymore.

When the state space has been computed, the optimal strategy can be calculated as a next step towards the PDM based recommendations. The performance objective of the process (i.e. optimization of either cost or processing time) can be selected from a drop-down menu, as is shown in Figure 6.15. Based on the selected objective, the decision strategy is calculated and then the recommendation service can be started. The execution of the process based on these recommendations is similar to the execution based on the simple strategies described above. In the DECLARE worklist a number of recommendations are presented (see Figure 6.16).



recommendation

Do:	Dont:	task	event type	rationale
1,00	0,00	Op03		9.4875
0,50	0,00	Op02		10.627500000000...

Value of current trace: 0.0

Figure 6.16: *The two recommended activities are shown for the process which is in the state where all leaf operations have been executed and the values for data elements B, D, F, G, and H are available. There is no value present for E. Based on the Markov decision strategy $Op03$ is recommended as the next operation to be executed (cf. the $V_n(3)$ and $f_n(3)$, for $n \geq 2$, in Table 6.2 and 6.3). Note that all executable operations are ranked with respect to the performance objective: the second best next step is $Op02$.*

The first one is the recommended activity according to the calculated strategy. The others are second best options. Each recommendation also has a field called 'rationale' in which the total expected cost or processing time to finish the process is shown.

In contrast to the simple selection strategies (cf. Section 6.6.2), changing a strategy during the execution of a case is not allowed, since this would force the system to recalculate the complete strategy. This is not acceptable at run-time.

6.7 Related Work

Although the approach to dynamically execute a PDM is rather new, two related approaches exist. First of all, the research on executable process models by Kress, Melcher and Seese [152, 153] focuses on optimizing the workload of the whole product-based workflow system. Secondly, we drew inspiration from the research by Schonenberg, Weber et al. [255, 293] on execution recommendations based on the history of cases kept in event logs. Finally, we give an overview of other approaches to provide flexibility in workflow management systems.

6.7.1 Executable Process Models

Kress, Melcher and Seese [152, 153] introduce executable product models (EPMs) for the service industry (see also Section 3.5.2). The basis of their approach is the product data model as defined by Reijers et al. in [233]. In their approach the product model is also directly executed instead of deriving a process model first. A Multi-Agent system (MAS) is used for the EPM approach. The MAS comprises two kinds of agents:

- Intelligent business object (IBO) agents - A new IBO agent is created for each instance of the product model. The main objective of such an intelligent business object agent is to assure the successful execution of the product model instance.
- Resource agents (RA) - A resource agent models a human or technical resource. Its main objective is to achieve an optimal workload which corresponds to a minimum idle time.

The IBOs and RAs communicate with each other by exchanging messages. The main part of the communication consists in negotiation. This negotiation is required for the scheduling of executable operations. The IBO calculates the executable tasks based on the data element values available for its case. Next, the IBO starts the negotiation with the RAs by sending an acknowledgement. The RAs that are able to perform the operation send a reply with an estimated finishing time. According to a certain selection protocol, the IBO chooses one of the offers and the task is executed by the assigned RA.

The EPM approach is similar to the PDM recommendation tool described in this chapter. Nevertheless, some differences can be distinguished. First of all, the EPM approach assumes that all operations in the product model have to be executed, i.e. it does not use the notion of alternative paths, while these alternative routes play a prominent role in our approach. Secondly, the MAS has less sophisticated optimization algorithms than the PDM recommendation tool, especially with respect to the execution of a single case. In contrast, the MAS has a more advanced resource perspective and tries to optimize the whole system of agents by scheduling tasks based on simple protocols and machine learning techniques.

6.7.2 Execution Recommendations based on the History of Cases

In [255, 293], Schonenberg, Weber et al. propose a process execution recommendation service which guides users through the process by giving recommendations based on the history of the process. Recommendations for a current case are generated based on similar executions of the process and on specific optimization goals (e.g. the duration of a case, or the business value of a case). The information on similar cases is extracted from the event log of the process in which the history is kept of all actions taken for all cases in the past. To decide on the next step in a partially executed case, this case is compared to the many completed cases in the event log. Whether a previously executed case is similar to the current partial case is decided based on several abstraction mechanisms with respect to e.g. the order and frequency of activities. A prototype of the history based recommendation service is available in ProM. The main idea behind the history and PDM based approaches is similar: to provide process execution recommendations without using a process model. Therefore, on the one hand, some similarities between the two approaches exist. For instance, both recommendation services focus on guiding the user in a flexible way and supporting the execution of a single case. On the other hand, the basis for providing recommendations is completely different. The history based recommendation approach looks at the way the process has been executed many times before and tries to map the active case to one of the previous executions without knowing whether these previous executions are optimal or successful with respect to e.g. cost, throughput times or other performance goals. The PDM recommendations, in contrast, are based on the workflow product description and focus on the optimization of the case execution in isolation, i.e. without considering earlier process executions directly. Note, however, that the PDM may be developed based on knowledge that was gathered from earlier executions of the process.

Besides this work on execution recommendations in the field of BPM, there is a substantial amount of literature on *recommender systems* in the area of artificial intelligence, e.g. [49, 58, 59]. An overview of existing recommender systems can be found in [28, 240].

6.7.3 Flexible Workflow Management Systems

Traditional workflow management systems have well-known limitations with respect to their flexibility and adaptability. As was mentioned in the introduction in Chapter 1, there are at least three ways to provide more flexibility: dynamic change, worklets, and case handling [277, 278].

The basic idea of *dynamic change* [93, 227, 242] is to allow changes at run-time, i.e., while work is being performed processes may be adapted [18, 93, 227, 242]. Clearly, dynamic change mechanisms can be used to support flexibility and adaptability. A dynamic change may refer to a single case (i.e., process instance) or multiple cases (e.g., all running instances of a process). Both changes at the instance level and the type level may introduce inconsistencies, e.g., data may be missing or activities may be unintentionally skipped or executed multiple times. A well-known problem is the 'dynamic change bug' which occurs when the ordering of activities changes or the process is made more sequential [93]. These issues have been addressed by systems such as ADEPT [76, 227, 228, 242]. Such a system can safeguard the consistency of a process. However, an additional complication is that the persons changing the processes should be able to modify process models and truly understand the effects of a change on the whole process. In real-life applications, with hundreds of tasks, few people are qualified to make such changes.

Worklets [27] allow for flexibility and adaptability by the late binding of process fragments (cf. Proclets in Section 3.5). Activities in a process are not bound to a concrete application or subprocess and only when they need to be executed a concrete application or subprocess is selected. YAWL [15] is an example of a system that implements this idea. In YAWL, activities may be handled by a worklet handler. This handler uses an extensible set of ripple-down rules to select the right worklet (i.e., a concrete application or subprocess). Similar ideas have been proposed by other authors (e.g., [297]) and have been implemented in commercial systems (cf. the Staffware extension that allows for process fragments [265]). Although the worklets mechanism is easier to use for end-users than most dynamic change mechanisms, the scope is limited and only particular forms of flexibility and adaptability can be supported.

Case handling [10, 25] is another paradigm for supporting flexible and knowledge intensive business processes. The concept of case handling offers a solution to the lack of flexibility in traditional workflow systems [25]. Case handling focuses on what can be done instead of on what should be done. To support this, a case handling system is much more data-driven than a workflow system. The central concept for case handling is the *case* and not the routing of work or the activities. The case is the product that is manufactured in the process based on the data that is processed. The core features of case handling are [10, 25]: (i) to avoid context tunneling by providing all information available (i.e., present the case as a whole rather than showing just fragments), (ii) to decide which activities are enabled on the basis of the information available rather than the activities already executed, (iii) to separate work distribution from authorization and allow for additional types of roles, not just the execute role, and (iv) to allow workers to view and add/modify data before or after the corresponding activities have been executed (e.g., information can be registered the moment it becomes available).

6.8 Summary

In this chapter we have presented a means for more dynamic and flexible execution of a workflow process based on the PDM. The PDM is executed directly, without deriving a process model as an intermediate step. Based on the information available for a case it is determined which operation is the best step to proceed with. The ‘best’ step is defined with respect to the single case and can be determined by simple strategies or obtained by a global optimization approach based on Markov theory.

Chapter 7

Business Process Metrics

This chapter focuses on business process metrics. These metrics provide a means to evaluate a process model and to compare different process models that essentially capture the same process. Internal attributes of the process model, such as size or cohesion, are related to external attributes, such as understandability and maintainability of the process model, or the occurrence of errors in the model. For instance, it has been shown that size correlates with the number of errors in a process model: the larger the process model (in terms of for instance the number of activities), the more likely it is that errors occur in the model [175, 177]. The development of metrics for business processes has been inspired by existing metrics for software programs. These software metrics have proven themselves as a means to improve maintainability of the software design. Section 7.1 focuses on the similarities between workflow processes and software programs to justify the adoption of these metrics to the business process domain.

In this chapter, we introduce two sets of business process metrics. The *cohesion and coupling* metrics, presented in Section 7.2, assess the size (granularity) of the activities in the process model by focusing on the cohesiveness of the operations within the activity and the coupling between activities. These metrics take the PDM as a basis and focus on the data elements and operations within each activity. The main goal is that an activity should neither be too small nor too large. Next, in Section 7.3, a different metric is introduced: the *Cross-Connectivity* metric. This metric focuses on the process model as a whole, instead of on the content of the activities, and measures the degree of connectivity of the model. The metric considers the cognitive effort a user has to make to understand a process model. It considers the connections between activities in the process model and assigns a weight to each connection. The Cross-Connectivity metric does not concentrate on data elements and operations but calculates the tightness of the links between activities in the process model. Section 7.4 presents our tool support for calculating the above metrics for a given process model in ProM. An overview of related work is given in Section 7.5. The chapter concludes with a summary (Section 7.6).

7.1 Business Processes vs Software Programs

The use of business process metrics to asses the quality of a process model is inspired by software quality metrics. The main purpose of software quality metrics is to obtain program designs that are better structured. Some of the most important advantages of a structured design, pointed out in [71], are that (i) the overall program logic is *easier to understand* for

Software programs	Workflow processes
Program	Workflow process
Module/Class	Activity
Statement/Method	Operation
Variable/Constant	Data element

Table 7.1: *Similarities between software programs and workflow processes.*

both the programmers and the users and (ii) the identification of the modules is easier, since different functions are performed by different modules, which simplifies the *maintenance* of the software program. We distinguish a number of similarities between software programs and workflow processes:

- They both focus on *information processing*. Within each step, one or more outputs are produced on the basis of one or more inputs.
- They have a similar *compositional structure*. A program - procedural or object-oriented - can be split into modules or classes. Every module consists of a number of statements, and every statement contains a number of variables and constants. Likewise, a workflow process has activities. Each activity is composed of elementary operations and each operation uses one or more pieces of information to produce new information.[1] See also Table 7.1.
- Their *dynamic execution* is derived from a static structure. When instantiating either a software program or a workflow process, an execution flow of their elements takes place in accordance with their static representation. This flow may involve consecutive executions, concurrency, conditional routings, etc.

Because of these similarities - which have partly been noted before, e.g. in [115] - it is conceivable that metrics used in software engineering are applicable in the domain of business process modeling as well. There are clear implications for a better maintainability of programs by using this approach and there is also considerable empirical evidence that the resulting computer programs contain fewer run-time errors [61, 257]. Yet, none of the existing software engineering metrics we studied seems to be directly applicable to workflow processes [236].

Before we introduce the business process metrics that we have developed, we first give an overview and classification of the existing approaches in software engineering. The quality of a software program design is generally considered to be related to five design principles [71, 259, 271]:

- *Cohesion* - Cohesion is a measure of the relationships of the elements within a module and is sometimes referred to as *module strength*. It is hypothesized that modules with low cohesion will contain more errors than modules with higher cohesion.
- *Coupling* - Coupling is measured by the number of interconnections between modules. Coupling is a measure for the strength of association established by the interconnections from one module of a design to another. It is hypothesized that programs with a high coupling will contain more errors than programs with lower coupling.
- *Complexity* - A design should be as simple as possible. Design complexity grows as the number of control constructs grows, and also as the size - in terms of the number of modules

[1]Note that the resources executing a program or process are different; the steps of a software program are executed automatically on a computer, while the activities of a business process often are performed by humans.

- grows. The hypothesis is that the higher the design complexity, the more errors the design will contain.

· *Modularity* - The degree of modularization affects the quality of a design. The hypothesis is that low modularity generally corresponds with more errors than high modularity.

· *Size* - A design that exhibits large modules or a deep nesting is considered undesirable. It is hypothesized that programs of large size will contain more errors than smaller programs.

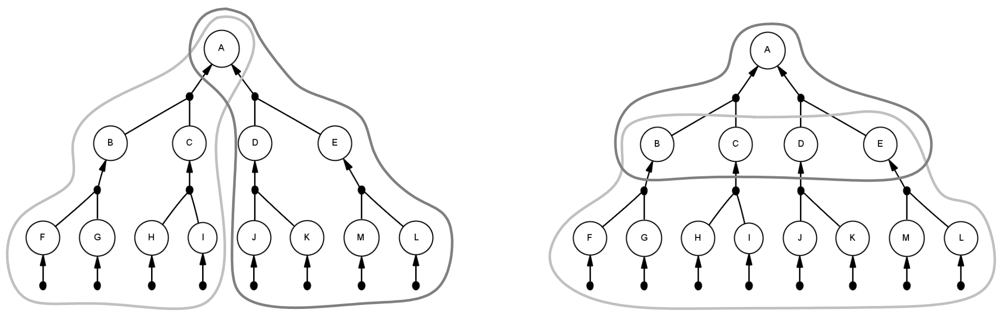
Note that these definitions of complexity and size partly overlap. The term *complexity* is often used as a general characteristic of a software program that can be measured by e.g. size, cohesion, and coupling metrics.

Researchers do not seem to agree on the relative importance of these five principles. In [259, 271], analyses are presented that indicate that coupling is the most influential of the design principles under consideration. However, in [192] cohesion and coupling are considered as equally important. In this thesis, we adopt the latter view and introduce a combined notion of workflow process cohesion and coupling in Section 7.2. Moreover, we propose a metric in Section 7.3 which measures the connectedness of process model elements and which is inspired by cognitive theory.

7.2 Cohesion and Coupling Metrics

This section introduces cohesion and coupling metrics for workflow process models that have been designed based on a PDM. As we have seen in Chapter 3, designing process models does not only involve the design of the control-flow of the process but it also involves the composition of activities as pieces of work. When a PDM is used as a basis for the development of a process model the composition of activities is done by *grouping operations and their data elements* to form logical pieces of work. We have also seen that a process modeler has a considerable amount of freedom in making such a design. To cope with this design freedom may be problematic. Badly chosen sizes of activities in a process may negatively affect its performance when being executed or may enlarge the maintenance burden of the process model in case of updates. Small activities, on the one hand, may increase the number of hand-offs between activities, leading to an increase of the number of errors [231, 256]. Large activities, on the other hand, may become unworkable for humans [75, 231]. This design choice is known as the issue of *process granularity* [75].

The cohesion and coupling metrics we present in this section focus on designing process models with properly sized activities. By using the proposed set of metrics, it can be quantitatively expressed to what extent the operations within one activity 'belong' to each other or, in other words, how *cohesive* such an activity is. In addition, it is important to measure to what extent the various activities are dependent on each other or, in other words, how much they are *coupled*. The inspiration for the proposed metrics comes from software engineering, where an old design aphorism is to strive for *strong cohesion, and loose coupling*. Based on the strong similarities between software programs and workflow processes, we consider this as a valuable design maxim in the workflow domain as well. This is illustrated by Figure 7.1. Moreover, the adoption of this rule of thumb has been suggested previously by other researchers [38, 39]. A *loose coupling* of activities will result in few data element values that need to be exchanged between activities in a workflow process, reducing the probability of run-time mistakes. *Highly cohesive* activities, in turn, are likely to be understood and performed better by humans than large chunks of unrelated work being grouped together. Because the creation of large activities



(a) Good grouping. (b) Bad grouping.

Figure 7.1: *An example of a good and a bad grouping of operations and data elements of a PDM into activities.*

will decrease the degree of coupling and the creation of small activities will increase cohesion, it seems that high cohesion and loose coupling is providing the right balance to improve both the execution quality and maintainability of a workflow process model.

In the remainder of this section definitions of the cohesion and coupling metrics which we have developed are presented. We mainly build on the work by Selby and Basili [257] and Xenos et al. [306], who defined similar coupling and cohesion metrics for software programs. The main objective in this formalization is to support the task of *process design*. This task can be understood as defining on a PDM a number of activities that partition that PDM (cf. the grouping of operations into activities for the social benefits example, Section 3.3.2). The cohesion and coupling metrics provide a means to compare different designs (based on the same PDM) with each other.

As a starting point to compute the cohesion and coupling metrics, we take a process model with a set of activities. Each activity consists of a group of operations. For the explanation of the definitions presented in this section we again use the process model shown in Figure 7.2 (i.e. the social benefits example). We extend the earlier given definition of an activity (see Definition 4.2.2) with a number of notations:

Definition 7.2.1 (Activity)**.** Let PDM be a PDM with its set of data elements D and its set of operations O. An activity t on the PDM is a set of operations, i.e. $t \subseteq O$. The set of all activities that are defined on the PDM is denoted by T. As a shorthand we introduce the following notations:

- $\tilde{t} = \{(d, d') | \exists_{(d,ds)\in t}(d' \in ds)\}$, for the input-output relations in an activity, and
- $\hat{t} = \bigcup_{(d,ds)\in t}(\{d\} \cup ds)$, for the information elements processed in an activity.

Thus, $\hat{t} \subseteq D$. Essentially, any group of operations may be clustered together into an activity. For the social benefits example, we could for instance define an activity t which consists of operations $Op26$ and $Op28$, i.e. $t = \{(i29, \{i25, i30, i35, i36\}), (i31, \{i29, i40, i48\})\}$. For this activity t, the accompanying notations have the following elements:

$$\begin{aligned}
\tilde{t} &= \{(i29, i25), (i29, i30), (i29, i35), (i29, i36), (i31, i29), (i31, i40), (i31, i48)\}, \\
\hat{t} &= \{i25, i29, i30, i31, i35, i36, i40, i48\}.
\end{aligned}$$

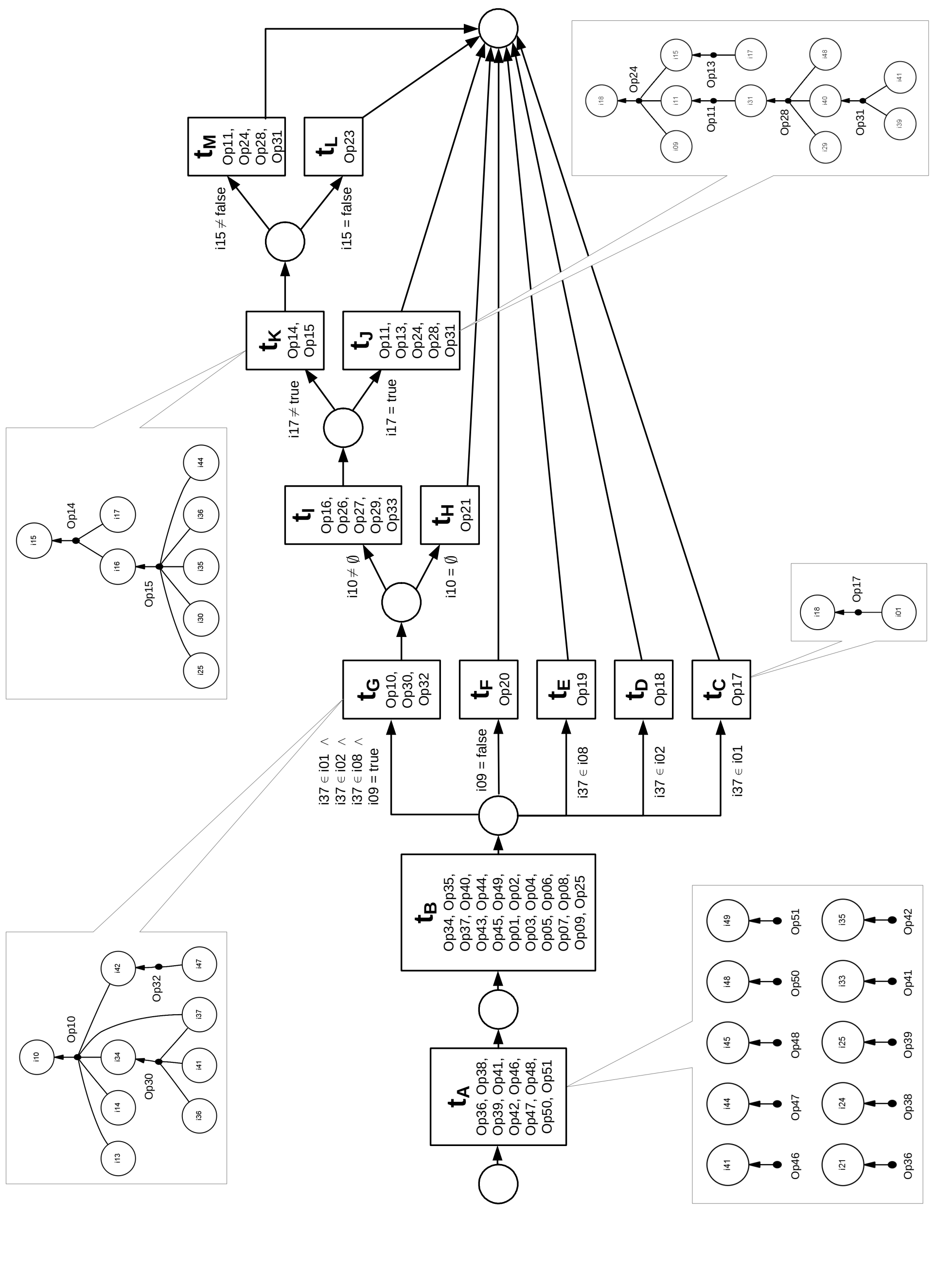

tA Op36, Op38, Op39, Op41, Op42, Op46, Op47, Op48, Op50, Op51
tB Op34, Op35, Op37, Op40, Op43, Op44, Op45, Op49, Op01, Op02, Op03, Op04, Op05, Op06, Op07, Op08, Op09, Op25
i37 ∈ i01 ∧ i37 ∈ i02 ∧ i37 ∈ i08 ∧ i09 = true
i09 = false
i37 ∈ i08
i37 ∈ i02
i37 ∈ i01
tG Op10, Op30, Op32
tF Op20
tE Op19
tD Op18
tC Op17
i10 ≠ ∅
i10 = ∅
tI Op16, Op26, Op27, Op29, Op33
tH Op21
i17 ≠ true
i17 = true
tK Op14, Op15
tJ Op11, Op13, Op24, Op28, Op31
i15 ≠ false
i15 = false
tM Op11, Op24, Op28, Op31
tL Op23
Op10
Op30
Op32
Op14
Op15
Op17
Op24
Op13
Op11
Op28
Op31
Op46
Op47
Op48
Op50
Op51
Op36
Op38
Op39
Op41
Op42

Figure 7.2: *The process model for the social benefits example (cf. Figure 3.8).*

Note that activity t is not part of the process design of this example. The process design for the social benefits example shown in Figure 7.2 consists of 13 activities:

$$\begin{aligned}
t_A &= \{(i21,\emptyset),(i24,\emptyset),(i25,\emptyset),(i33,\emptyset),(i35,\emptyset),(i41,\emptyset),(i44,\emptyset),(i45,\emptyset),(i48,\emptyset),(i49,\emptyset)\}\\
t_B &= \{(i01,\{i25,i27\}),(i02,\{i25,i27\}),(i03,\{i33,i37\}),(i04,\{i33,i37\}),(i05,\{i37,i45\}),\\
&\quad (i06,\{i21,i37\}),(i07,\{i24,i37\}),(i08,\{i23,i37\}),(i09,\{i24,i39\}),(i13,\emptyset),(i14,\emptyset),\\
&\quad (i23,\emptyset),(i27,\emptyset),(i28,\{i25,i37\}),(i36,\emptyset),(i37,\emptyset),(i39,\emptyset),(i47,\emptyset)\}\\
t_C &= \{(i18,\{i01\})\}\\
t_D &= \{(i18,\{i02\})\}\\
t_E &= \{(i18,\{i08\})\}\\
t_F &= \{(i18,\{i09\})\}\\
t_G &= \{(i10,\{i13,i14,i34,i37,i42\}),(i34,\{i36,i37,i41\}),(i42,\{i47\})\}\\
t_H &= \{(i18,\{i10\})\}\\
t_I &= \{(i17,\{i25,i30\}),(i29,\{i25,i30,i35,i36\}),(i30,\{i32,i37,i43\}),\\
&\quad (i32,\{i01,i02,i03,i04,i05,i06,i07,i08,i10,i27,i28\}),(i43,\{i39,i49\})\}\\
t_J &= \{(i11,\{i31\}),(i15,\{i17\}),(i18,\{i09,i11,i15\}),(i31,\{i29,i40,i48\}),(i40,\{i39,i41\})\}\\
t_K &= \{(i15,\{i16,i17\}),(i16,\{i25,i30,i35,i36,i44\})\}\\
t_L &= \{(i18,\{i15\})\}\\
t_M &= \{(i11,\{i31\}),(i18,\{i09,i11,i15\}),(i31,\{i29,i40,i48\}),(i40,\{i39,i41\})\}
\end{aligned}$$

Building on the simple notion of a process as a set of activities which are composed of a group of operations, a *cohesion* metric can now be defined as follows. Its first component, the *activity relation cohesion*, quantifies how much the different operations within one activity are related. It does so by determining for each operation of an activity with how many other operations it overlaps by sharing an input or output, i.e. a non-empty intersection.

Definition 7.2.2 (Activity Relation Cohesion)**.** For an activity t on a PDM, the activity relation cohesion $\lambda(t)$ is defined as follows:

$$\lambda(t) = \begin{cases} \frac{|\{((d_1,ds_1),(d_2,ds_2))\in t\times t \mid ((\{d_1\}\cup ds_1)\cap(\{d_2\}\cup ds_2))\neq\emptyset \ \wedge \ d_1\neq d_2\}|}{|t|\cdot(|t|-1)} & , \text{ for } |t|>1\\ 0 & , \text{ for } |t|\leq 1 \end{cases}$$

Note that in this definition alternative operations to produce the same data element value are treated separately, because we generally assume that in an instantiation of the process model only one of the alternatives will be executed. The overlap between these operations (i.e. the same output element) is therefore not considered. This explains the expression $d_1 \neq d_2$ in the formula.

Next, the average overlap per operation is computed by dividing the total number of overlaps by the number of operations. Finally, note that all overlaps are counted twice, because we consider all pairs of operations separately (for example distinguishing $((d_1,ds_1),(d_2,ds_2))$ and $((d_2,ds_2),(d_1,ds_1))$ as different pairs). Therefore, to get a relative metric score between 0 and 1, the average overlap per operation over all operations within an activity is again divided, this time by the maximal overlap, i.e. the number of operations minus one.

The other component of our cohesion metric, the *activity information cohesion*, focuses on all data elements that are used either as input or as output by any operation within the respective activity. It determines how many data elements are used more than once relative to all data elements used. It does so by counting all different data elements that appear in the intersection of a pair of operations, considering all pairs. This number is divided by the total number of data elements in the activity, i.e. $|\hat{t}|$.

Definition 7.2.3 (Activity Information Cohesion)**.** Let PDM be a PDM with its set of data elements D and its set of operations O. For an activity t on PDM, i.e. $t \subseteq O$, the information cohesion $\mu(t)$ is defined as follows:

$$\mu(t) = \begin{cases} \frac{|\{d \in D | \exists_{(d_1, ds_1),(d_2, ds_2) \in t}(d \in ((\{d_1\} \cup ds_1) \cap (\{d_2\} \cup ds_2)) \wedge (d_1 \neq d_2))\}|}{|\hat{t}|} & , \text{ for } |\hat{t}| > 0 \\ 0 & , \text{ for } |\hat{t}| = 0 \end{cases}$$

The total cohesion of an activity is now given as the product of both the relation and information cohesion. This is to reflect that in our opinion an activity has to score high on both cohesion aspects to say it is cohesive. In other words, the operations clustered together should (i) be interrelated to each other and (ii) information should be shared to a certain degree.

Definition 7.2.4 (Activity Cohesion)**.** Let PDM be a PDM with its set of data elements D and its set of operations O. For an activity t on PDM, i.e. $t \subseteq O$, the activity cohesion $c(t)$ is defined as follows:

$$c(t) = \lambda(t) \cdot \mu(t)$$

In the process model for the social benefits example, the activity relation cohesion, the activity information cohesion and the total activity cohesion of activity t_G, shown in Figure 7.3, are:

$$\begin{aligned} \lambda(t_G) &= \frac{|\{((i10, \{i13, i14, i34, i37, i42\}), (i34, \{i36, i37, i41\})), ...\}|}{3 \cdot 2} \\ &= \frac{4}{6} = \frac{2}{3} \approx 0.667 \\ \mu(t_G) &= \frac{|\{i34, i37, i42\}|}{9} = \frac{3}{9} = \frac{1}{3} \approx 0.333 \\ c(t_G) &= \frac{2}{3} \cdot \frac{1}{3} = \frac{2}{9} \approx 0.222 \end{aligned}$$

Table 7.2 shows the cohesion values for all activities in the process model of the social benefits example. The overall cohesion of a process model can then be determined by the average activity cohesion.

Definition 7.2.5 (Process Cohesion)**.** Let PDM be a PDM with its set of data elements D and its set of operations O. For a process model PM which consists of a set of activities T on the PDM, the average cohesion, or process cohesion ch, is defined as follows:

$$ch = \frac{\sum_{t \in T} c(t)}{|T|}$$

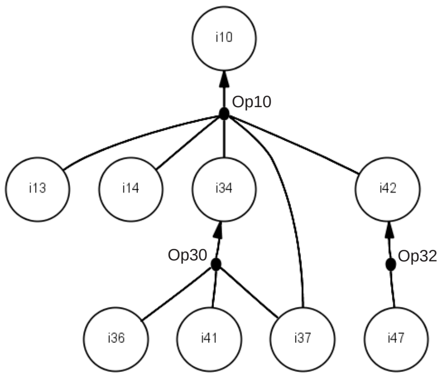


Figure 7.3: *Activity* $t_G = \{(i10, \{i13, i14, i34, i37, i42\}), (i34, \{i36, i37, i41\}), (i42, \{i47\})\}$ *of the process design for the social benefits example (cf. Figure 7.2).*

Activity	Activity cohesion $(c(t))$	Activity relation cohesion $(\lambda(t))$	Activity information cohesion $(\mu(t))$
t_A	0.0	0.0	0.0
t_B	0.082	0.314	0.261
t_C	0.0	0.0	0.0
t_D	0.0	0.0	0.0
t_E	0.0	0.0	0.0
t_F	0.0	0.0	0.0
t_G	0.222	0.667	0.333
t_H	0.0	0.0	0.0
t_I	0.091	0.5	0.182
t_J	0.145	0.4	0.364
t_K	0.125	1.0	0.125
t_L	0.0	0.0	0.0
t_M	0.15	0.5	0.3

Table 7.2: *The values for activity cohesion* $(c(t))$*, activity relation cohesion* $(\lambda(t))$*, and activity information cohesion* $(\mu(t))$ *for each of the activities of the social benefits example.*

As an extension and a natural counterpart of cohesion we also define a metric for the degree *coupling* in a process. Coupling focuses on how strongly the activities in a workflow process are related, or connected, to each other. A certain activity is connected to another if and only if they share one or more data elements. The coupling metric determines the number of related activities for each activity. First, the average coupling is determined by adding the number of connections for all activities and dividing this number by the total number of activities. Now all pairs of activities have been counted twice. To get a relative score for this metric, the average coupling is divided by the maximum number of coupling, i.e. the number of activities minus one.

Definition 7.2.6 (Process Coupling)**.** Let PDM be a PDM with its set of data elements D and its set of operations O. And let PM be a process model which consists of a set of activities (T) on PDM. Then, the process coupling cp for PM is defined as follows:

$$cp = \begin{cases} \frac{|\{(t_1,t_2)\in T\times T \mid t_1\neq t_2 \wedge (\hat{t_1}\cap\hat{t_2})\neq\emptyset\}|}{|T|\cdot(|T|-1)} & , \text{ for } |T| > 1 \\ 0 & , \text{ for } |T| \leq 1 \end{cases}$$

Inspired by the work of Selby and Basili [257], we now define a *coupling/cohesion ratio*. This ratio enables the comparison between various design alternatives.

Definition 7.2.7 (Process Coupling/Cohesion Ratio)**.** Let PDM be a PDM with its set of data elements D and its set of operations O. And let PM be a process model which consists of a set of activities (T) on PDM. Then, the process coupling/cohesion ratio ρ for PM is defined as follows:

$$\rho = \frac{cp}{ch}$$

Again we can calculate the metric values for our social benefits example. The process cohesion of the process design is (see Table 7.2 for the cohesion values per activity):

$$\begin{aligned} ch &= \frac{0+0.082+0+0+0+0+0.222+0+0.091+0.145+0.125+0+0.15}{13} \\ &\approx 0.063 \end{aligned}$$

In this design, activity t_G is for example coupled with 7 other activities. It is coupled with activities t_A, t_J and t_M through data element $i41$, with activity t_B through data elements $i13$, $i14$, $i36$, $i37$, and $i47$, with activity t_H through $i10$, with activity t_I through $i10$, $i36$, and $i37$, and with activity t_K through $i36$. The total process coupling is:

$$\begin{aligned} cp &= \frac{|\{(t_A,t_B),(t_A,t_I),(t_A,t_J),(t_A,t_K),(t_A,t_M),(t_B,t_A),(t_B,t_C),...,(t_L,t_M)\}|}{13*12} \\ &= \frac{116}{156} \approx 0.744 \end{aligned}$$

The coupling/cohesion ratio of the process model then is:

$$\rho = \frac{0.744}{0.063} \approx 11.81$$

The previously defined metrics can be used to find the preferable workflow design among a number of alternative designs. The design with the smallest process coupling/cohesion ratio is the most favorable design. Note that these metrics do not indicate how the alternative designs can be determined. The ratio can only help to choose the best option from already specified alternatives.

We use the social benefit example again to explain the comparison of two alternative process models using the coupling/cohesion ratio. Besides the process model of Figure 7.2, the project team also designed an alternative process model, which is reported on in [233] and which is shown in Figure 7.4. Note that the first part of the process model (activities t_A until t_H) is the same as in the first process design. The other activities are different and smaller than

Design	Process cohesion (ch)	Process coupling (cp)	Coupling/Cohesion ratio (ρ)
1 (see Figure 7.2)	0.063	0.744	11.81
2 (see Figure 7.4)	0.038	0.571	13.605

Table 7.3: *The values for process cohesion, process coupling, and the coupling/cohesion ratio for both alternative process models of the social benefits example.*

in the original design. The values for process cohesion, coupling and the coupling/cohesion ratio for both designs are shown in Table 7.3.

The coupling/cohesion ratio value for the first process model design is lower than the value for the second design. Therefore, the first design is preferable, which seems to be confirmed by our intuition that activities should neither be too small nor too large. The second design has a low value of cohesion because the activities are too small: most of them only consist of one operation. However, the value for coupling in the second design is better than in the first design. This is mainly due to activities t_J and t_M in the first design. They contain the same operations, except for one, which creates a high coupling between the activities in the process model. However, the good value for coupling in the second design is not enough to compensate for the low cohesion value in the coupling/cohesion ratio. Thus, having a lower value for the coupling/cohesion ratio, the first design is preferred. Note that the project team implemented this design without knowing of these coupling and cohesion metrics.

In this section, we have presented coupling and cohesion metrics to evaluate a process model based on the content of the activities that are designed by grouping operations of the PDM. In Section 7.4, our tool support for the calculation of the coupling and cohesion values for a process model in ProM is presented. Moreover, Section 8.2 contains an application of these metrics to a real life process design case. In the next section, we look at a more general metric that is applied on the level of the process model itself.

7.3 Cross-Connectivity Metric

In this section, we present a metric to evaluate a process model as a whole. It only considers the connection between nodes in the model (specified by the arcs) and does not focus on the content of activities given by operations and data elements from the PDM. Therefore, this metric is applicable in a broader area than PBWD only. Besides its basis in software metrics, the metric we propose is also inspired by cognitive theory. It considers the cognitive effort of a user for understanding the process model. Although the explanation of the understandability of a model is one of the objectives of business process metrics, little literature exists on measuring the cognitive effort needed to understand a model. To break away from this tendency, we draw inspiration from the *Cognitive Dimensions Framework*, as first introduced in [111]. The motivation behind this framework is to use research findings from applied psychology for assisting designers of notational systems. Designers can use the framework to evaluate their designs with respect to the impact that they will have on the users of these designs. Since its introduction, it has been widely adopted in the evaluation and design of information artifacts; for an overview of results, see [50].

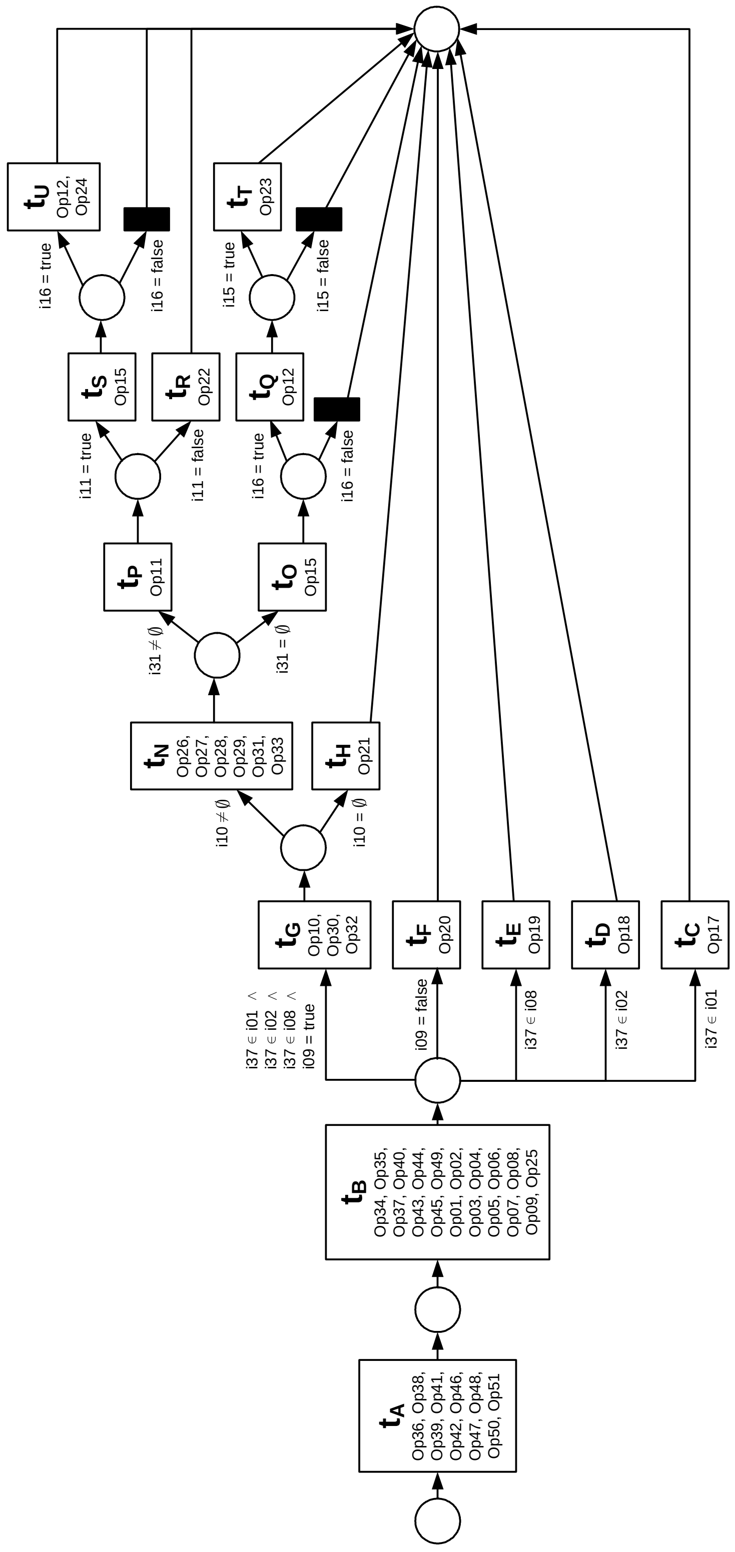


Figure 7.4: *The process model of the alternative design for the social benefits example.*

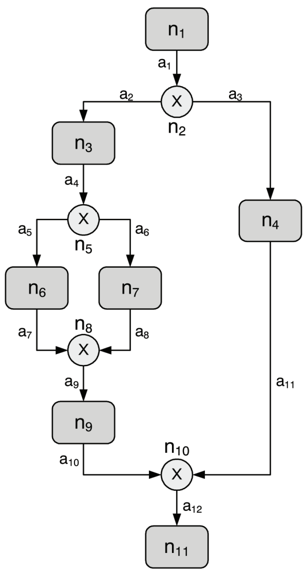


Figure 7.5: *An example process model.*

The most important dimension of this framework for our purpose is that of the *hard mental operations* that may be incurred through a particular notation, i.e. the high demand on a user's cognitive resources. Reading a process model implies some hard mental operations with respect to the behavioral relationships between model elements that have to be constructed in the mind of the reader. In particular, it is quite difficult - even for experts - to understand whether pairs of activities in a model with a high degree of parallelism and choices are exclusive or not. Furthermore, even if activities are on a directed path, it is not immediately clear on which other elements they depend if there are many routing elements in between them. The Cross-Connectivity (CC) metric that we define below aims to quantify the ease of understanding this interplay of any pair of model elements. It builds on the weakest-link metaphor assuming that the understanding of a relationship between an element pair can only be as easy, in the best case, as the most difficult part. Therefore, we identify suitable weights for nodes and arcs along a path between two model elements. Our assertion then is *that a lower (higher) CC-value is assigned to those models that are more (less) likely to include errors, because they are more (less) difficult to understand for both stakeholders and model designers.*

Below a set of definitions is given, which together form the basis of the Cross-Connectivity metric. The term 'Cross-Connectivity' is chosen because the strength of the *connections* between nodes is considered *across* all nodes in the model. To appreciate the formalization below, it is important to note that the CC-metric expresses the sum of the connectivity between all pairs of nodes in a process model, relative to the theoretical maximum number of paths between all nodes. For example, in the process model of Figure 7.5 all pairs $\langle n_1, n_2\rangle$, $\langle n_1, n_3\rangle$, $\langle n_2, n_7\rangle$, etc. are considered. Secondly, we assume that the path with the highest connectivity between two nodes determines the strength of the overall connectivity between those nodes. In the process model of Figure 7.5, for instance, there are several paths from node n_1 to n_{11}, i.e. one path via node n_4, one path via node n_6, and one path via node n_7. Since the path via node n_4 does not contain any choices, it links nodes n_1 and n_{11} stronger than one of the other paths.

Thirdly, the tightness of a path (i.e. the degree of connectivity) is determined by the product of the valuations of the arcs connecting the nodes on the path. In the example, the tightness of the path from n_1 to n_{11} via n_4 depends on the valuations of the arcs a_1, a_3, a_{11}, and a_{12}. Note that a single weak link has a strong influence on the *entire* connection, because we take the product of all valuations (rather than e.g. the sum). Finally, differences in the types of nodes that a path consists of determine the tightness of the arcs connecting nodes. For example, an AND connector on a path gives a stronger relation than an XOR connector. At the end of the formalization, an illustrative example is given of the application of the CC-metric for small process models, showing how the metric can be used to select from alternatives.

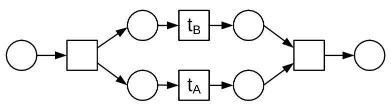
Definition 7.3.1 (Weight of a Node)**.** Let a process model be given as a graph consisting of a set of nodes ($n_1, n_2, ... \in N$) and a set of directed arcs ($a_1, a_2, ... \in A$). A node can be of one of two types: (i) activity, e.g. $t_1, t_2 \in T$, and (ii) connector, e.g. $c_1, c_2 \in C$. Thus, $N = T \cup C$, and $T \cap C = \emptyset$. The weight of a node n, $w(n)$, is defined as follows:

$$w(n) \quad = \quad \begin{cases} 1 & , \text{ if } n \in C \wedge n \text{ is of type AND} \\ \frac{1}{d} & , \text{ if } n \in C \wedge n \text{ is of type XOR} \\ \frac{1}{2^d-1} + (\frac{2^d-2}{2^d-1} \cdot \frac{1}{d}) & , \text{ if } n \in C \wedge n \text{ is of type OR} \\ 1 & , \text{ if } n \in T \end{cases}$$

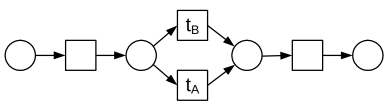
with d the degree of the node (i.e. the total number of ingoing *and* outgoing arcs of the node, cf. Definition 2.3.6).

There are three remarks we would like to make to explain the above definition. First, note that the definition above assumes that the process model consists of activities and connectors. Activities have at most one input and one output arc while connectors can have multiple input and output arcs. A connector of type AND with multiple input arcs is a so-called AND-join, i.e. it synchronizes the various flows leading to the join. The OR-split connector has a behavior in-between an XOR-split (one output arc is selected) and AND-split (all output arcs are selected). A connector can be both a join and a split (i.e. having multiple input and multiple output arcs), provided that both are of the same type. Secondly, note that we treat all model nodes as unique elements, even though their (business) semantics may be the same. In this way, for example, we support the inclusion of duplicate activities. Finally, Definition 7.3.1 does not assume a concrete process modeling language with well-defined semantics. It captures those routing elements that can be expressed with standard process modeling languages such as EPCs, UML Activity Diagrams, Petri nets, BPMN, or YAWL [17] (see also Figure 7.6).

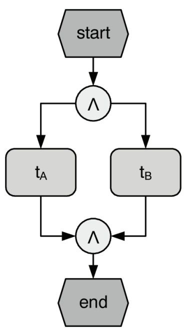
Most of the values for $w(n)$ in Definition 7.3.1 are natural given the intent of this metric, e.g. arcs connected to an AND-connector have a higher weight than arcs connected to an XOR-connector because the latter involves considering optionality. The only value that requires some explanation is the value for the OR-connector. For the OR-connector it is not clear a priori how many of the arcs will be traversed during an execution of the process, e.g. in case of an OR-split with two outgoing arcs either one of the arcs can be traversed, or both of the arcs may be used. This behavior is reflected in the definition of the weight for an OR-connector. The number of all possible combinations of d arcs is: $2^d - 1$. Only one of these combinations (i.e. 1 out of $2^d - 1$) is similar to the situation in which the node would have been an AND, namely the situation in which all arcs are traversed. This particular combination gets a weight of 1 (since that is the weight for an AND-connector from Definition 7.3.1). Therefore, the first part of the formula for the OR-connector is: $\frac{1}{2^d-1} \cdot 1 = \frac{1}{2^d-1}$. Each one of the $(2^d-1)-1$ other combinations of arcs can be seen as a separate XOR-node with weight $\frac{1}{d}$. Thus, in $2^d - 2$ out



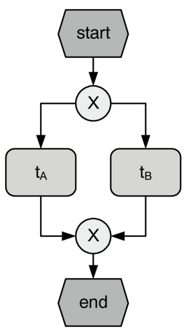
(a) AND-split and AND-join in Petri net notation.



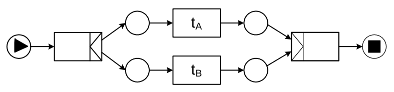
(b) XOR-split and XOR-join in Petri net notation.



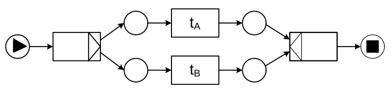
(c) AND-split and AND-join in EPC notation.



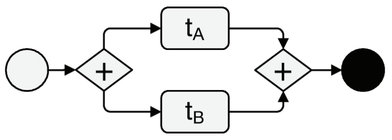
(d) XOR-split and XOR-join in EPC notation.



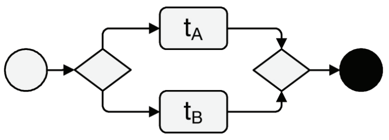
(e) AND-split and AND-join in YAWL notation.



(f) XOR-split and XOR-join in YAWL notation.



(g) AND-split and AND-join in BPMN notation.



(h) XOR-split and XOR-join in BPMN notation.

Figure 7.6: *AND-split, AND-join, XOR-split, and XOR-join routing constructs displayed in different modeling languages.*

of $2^d - 1$ combinations a weight of $\frac{1}{d}$ is added, which leads to the second part of the formula $(\frac{2^d-2}{2^d-1} \cdot \frac{1}{d})$.

The following definition shows that the weight of an arc is based on the weight of the corresponding nodes.

Definition 7.3.2 (Weight of an Arc)**.** Let a process model be given by a set of nodes (N) and a set of directed arcs (A). Each directed arc, $a \in A$, has a source node (denoted by $src(a)$) and a destination node (denoted by $dest(a)$), cf. Definition 2.3.1. The weight of arc a, $W(a)$, is defined as follows:

$$W(a) = w(src(a)) \cdot w(dest(a))$$

The weight of an arc is determined by the weight of its two nodes. To balance the value, the node weights are multiplied.

Definition 7.3.3 (Value of a Path)**.** Let a process model be given by a set of nodes (N) and a set of directed arcs (A). A path σ from node n_1 to node n_x is given by the sequence of nodes and directed arcs that should be followed from n_1 to n_x (cf. Definition 2.3.7): $\sigma = \langle n_1, a_1, n_2, a_2, ..., a_{x-1}, n_x \rangle$. The value for a path σ, $v(\sigma)$, is the product of the weights of all arcs in the path:

$$v(\sigma) = W(a_1) \cdot W(a_2) \cdot ... \cdot W(a_{x-1})$$

Note that the weight of each internal node, i.e. $n_2, ..., n_{x-1}$ is considered twice.

Definition 7.3.4 (Value of a Connection)**.** Let a process model be given by a set of nodes (N) and a set of directed arcs (A) and let P_{n_1,n_2} be the set of paths from node n_1 to n_2. The value of the connection from n_1 to n_2, $V(n_1, n_2)$, is the maximum value of all paths connecting n_1 and n_2:

$$V(n_1, n_2) = \max_{\sigma \in P_{n_1,n_2}} v(\sigma)$$

If no path exists between node n_1 and n_2, then $V(n_1, n_2)$ is defined to be 0. Also note that loops in a path should not be considered more than once, since the value of the connection will not be higher if the loop is followed more than once in the particular path.

Based on the above valuation of connectivity (i.e. the tightness of the connection between two nodes), we define the *Cross-Connectivity metric*.

Definition 7.3.5 (Cross-Connectivity (CC))**.** Let a process model be given by a set of nodes (N) and a set of directed arcs (A). The Cross-Connectivity metric is then defined as follows:

$$CC = \frac{\sum_{n_1,n_2 \in N} V(n_1, n_2)}{|N| \cdot (|N| - 1)}$$

To illustrate the use of the CC-metric an example is elaborated. Figure 7.7 contains a process model with five activities (i.e. $T = \{t_A, t_B, t_C, t_D, t_E\}$), three connectors (i.e. $C = \{XOR, AND, OR\}$) and seven directed arcs (i.e. $A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$). To calculate the value for Cross-Connectivity the weight for each node is calculated first (see Table 7.4). Then, the weight for each arc is calculated:

$$\begin{aligned}
W(a_1) &= w(t_A) \cdot w(XOR) = 1 \cdot \frac{1}{3} = \frac{1}{3} \\
W(a_2) &= w(t_B) \cdot w(XOR) = 1 \cdot \frac{1}{3} = \frac{1}{3} \\
W(a_3) &= w(XOR) \cdot w(AND) = \frac{1}{3} \cdot 1 = \frac{1}{3} \\
W(a_4) &= w(t_C) \cdot w(AND) = 1 \cdot 1 = 1 \\
W(a_5) &= w(AND) \cdot w(OR) = 1 \cdot \frac{3}{7} = \frac{3}{7} \\
W(a_6) &= w(OR) \cdot w(t_D) = \frac{3}{7} \cdot 1 = \frac{3}{7} \\
W(a_7) &= w(OR) \cdot w(t_E) = \frac{3}{7} \cdot 1 = \frac{3}{7}
\end{aligned}$$

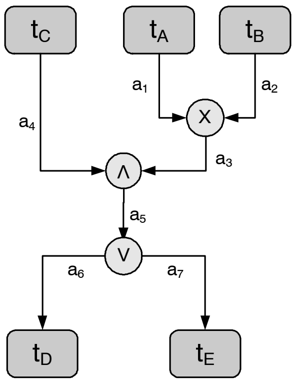


Figure 7.7: *A simple process model with five activities and three connectors.* $T = \{t_A, t_B, t_C, t_D, t_E\}$, $C = \{XOR, AND, OR\}$, $A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$.

Node (n)	Degree (d)	Weight ($w(n)$)
t_A	1	1
t_B	1	1
t_C	1	1
t_D	1	1
t_E	1	1
XOR	3	$\frac{1}{3}$
AND	3	1
OR	3	$\frac{1}{2^3-1} + \frac{2^3-2}{2^3-1} \cdot \frac{1}{3} = \frac{3}{7}$

Table 7.4: *The degrees and weights for the nodes in the process model of Figure 7.7.*

	t_A	t_B	t_C	t_D	t_E	XOR	AND	OR	Total
t_A	0	0	0	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{3}$	$\frac{1}{9}$	$\frac{1}{21}$	$\frac{235}{441}$
t_B	0	0	0	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{3}$	$\frac{1}{9}$	$\frac{1}{21}$	$\frac{235}{441}$
t_C	0	0	0	$\frac{9}{49}$	$\frac{9}{49}$	0	1	$\frac{3}{7}$	$\frac{88}{49}$
t_D	0	0	0	0	0	0	0	0	0
t_E	0	0	0	0	0	0	0	0	0
XOR	0	0	0	$\frac{3}{49}$	$\frac{3}{49}$	0	$\frac{1}{3}$	$\frac{1}{7}$	$\frac{88}{147}$
AND	0	0	0	$\frac{9}{49}$	$\frac{9}{49}$	0	0	$\frac{3}{7}$	$\frac{39}{49}$
OR	0	0	0	$\frac{3}{7}$	$\frac{3}{7}$	0	0	0	$\frac{6}{7}$

Table 7.5: *The values for the connections between all pairs of nodes of the process model of Figure 7.7.*

The paths between each pair of nodes are determined and the value for the connection between the pair of nodes is computed. For example, node t_A and node t_D are connected through the path $\langle t_A, a_1, XOR, a_3, AND, a_5, OR, a_6, t_D\rangle$. In this case, this is the only path from t_A to t_D. Thus, the value of this path is the maximum value over all paths from A to D:

$$\begin{aligned} V(t_A, t_D) &= v(\langle t_A, a_1, XOR, a_3, AND, a_5, OR, a_6, t_D\rangle) \\ &= W(a_1)\cdot W(a_3)\cdot W(a_5)\cdot W(a_6) \\ &= \frac{1}{3}\cdot\frac{1}{3}\cdot\frac{3}{7}\cdot\frac{3}{7} = \frac{1}{49}. \end{aligned}$$

Similarly, the value for the connection from the XOR-node to the OR-node is computed:

$$\begin{aligned} V(XOR, OR) &= v(\langle XOR, a_3, AND, a_5, OR\rangle) \\ &= W(a_3)\cdot W(a_5) = \frac{1}{3}\cdot\frac{3}{7} = \frac{1}{7}. \end{aligned}$$

Table 7.5 shows an overview of all connection values. Finally, the CC-value is determined as the sum of the values for all connections, divided by the number of nodes times the number of nodes minus one:

$$CC = \frac{\frac{235}{441} + \frac{235}{441} + \frac{88}{49} + 0 + 0 + \frac{88}{147} + \frac{39}{49} + \frac{6}{7}}{8\cdot 7} = \frac{\frac{2255}{441}}{56} \approx 0.09131.$$

Now that the mechanics behind the CC-metric have been established, it is worthwhile to explore how it can help to identify process models that are preferable among several designs. Similar to the use of the coupling/cohesion ratio in the previous section, our interest at this point is in a model's CC-value *relative* to that of another, alternative process model.

Consider the two process models that are shown in Figure 7.8. Both models express the same business logic as the initial model in Figure 7.7, but their CC-values are different. The process model on the left hand side in Figure 7.8 is block-structured, i.e. it differs from the initial model in the sense that the AND-join and XOR-join at the top of the model are matched by corresponding splits. Intuitively, one may expect that a block-structure will positively affect model comprehension. Indeed, as the links between the various nodes become tighter, this is expressed by a higher CC-value: 0.12486 versus a value of 0.09131 of the initial model.

The process model on the right hand side in Figure 7.8 is different from the initial process model of Figure 7.7 in the sense that it reorders the top connectors. It expresses (t_C AND t_A) XOR (t_C AND t_B) instead of t_C AND (t_A XOR t_B), the former being the more elaborate expression with a duplicate for task t_C.[2] So, the process model on the right hand side is expected to be slightly more difficult to understand than the initial model, which is supported by a lower CC-value: 0.08503 versus the initial model's CC-value of 0.09131.

Additionally, we have computed the CC-values for the two alternative process model designs of the social benefits case, which are presented in figures 7.2 and 7.4. The CC-values are 0.0273 and 0.0183, respectively for the first and second design. Thus, the first design is preferable with respect to understandability, since it has the highest value for the CC-metric.

In this section we have introduced a business process metric that is inspired by cognitive theory. It measures the degree of connectivity of a process model by examining the strength of the

[2]Please recall that in the computation of the CC-metric all model elements are treated as unique elements.

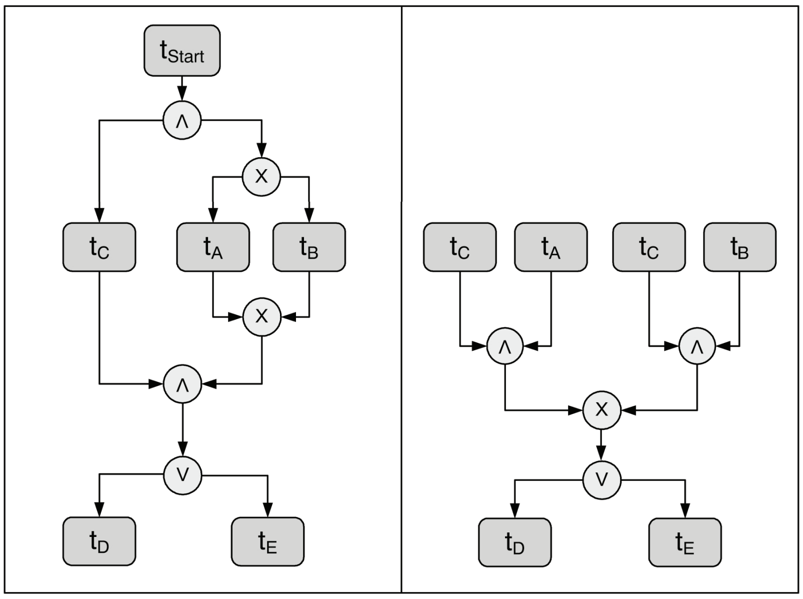
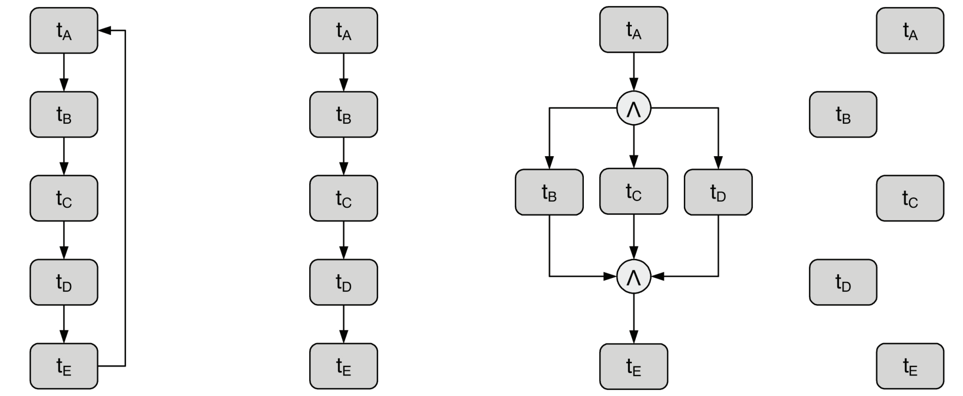


Figure 7.8: *Two alternative process models to the example of Figure 7.7. For the left model CC = 0.12486, for the right model CC = 0.08503, while for the original model in Figure 7.7, CC = 0.09131.*



(a) $CC = 1.000$ (b) $CC = 0.500$ (c) $CC = 0.310$ (d) $CC = 0.000$

Figure 7.9: *A number of example process models and their CC-value. In the best case situation, i.e.* $CC = 1.000$, *all process elements are connected to each other. In the worst case situation, i.e.* $CC = 0.000$, *none of the process elements is connected to another process element.*

connections between all pairs of nodes in the process model. In the next section, the ProM tool support for the calculation of the values for the CC-metric of a process model is discussed. An empirical evaluation of the CC-metric is included in Section 8.4.

7.4 Tool Support in ProM

This section elaborates on the tool support for business process metrics in ProM. For both sets of metrics that are described above, we have developed analysis plugins in the ProM framework. First, the plugin to calculate the cohesion and coupling metrics is explained, followed by a description of the plugin that determines the value for the CC-metric for a given process model.

7.4.1 Cohesion and Coupling Metrics

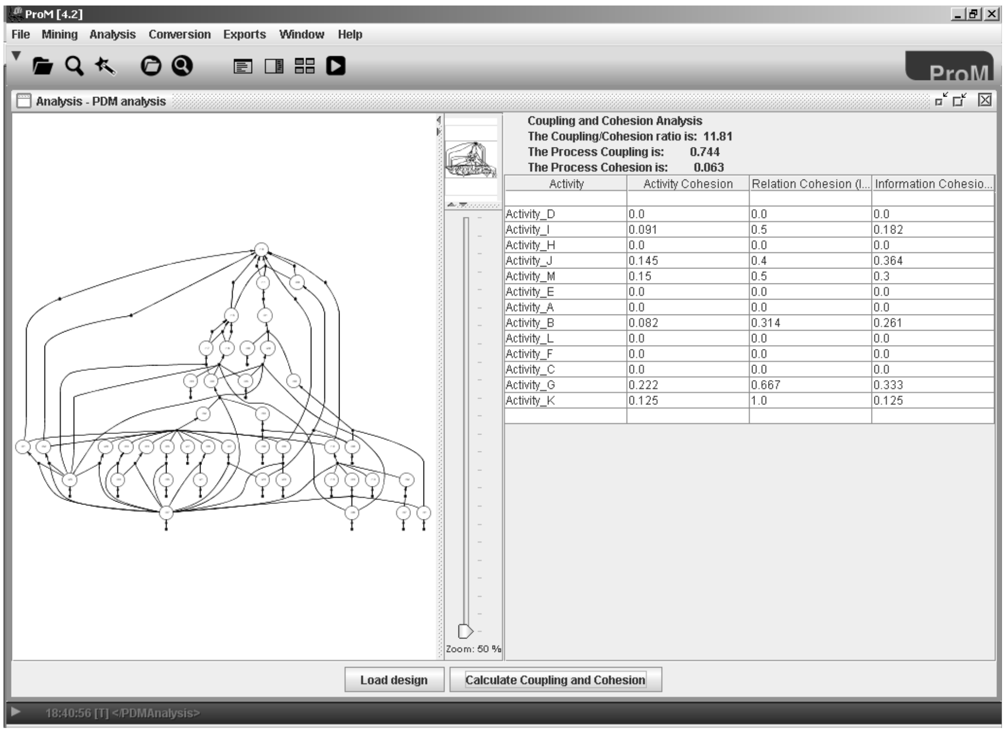
In ProM, we have developed an analysis plugin to support the calculation of cohesion and coupling metrics for a process model which is designed based on a PDM. Figure 7.10 shows a screenshot of this plugin. To use the plugin, first of all, a PDM has to be loaded into the ProM framework. Then, the analysis plugin can be started and a *design of a process model* can be loaded as an XML file. This design represents the grouping of operations and data elements and contains a list of activities. Each activity consists of zero or more operations of the PDM. When the design has been loaded in the ProM framework, the cohesion and coupling values can be computed by using the “Calculate Coupling and Cohesion” button.

7.4.2 Cross-Connectivity Metric

The ProM framework also contains an analysis plugin which computes the values for a number of business process metrics for EPCs. In Figure 7.11 a screenshot of this plugin is shown. Besides a calculation of the CC-metric, the plugin also calculates the Control Flow Complexity measure [62, 63], the Density metric [174], and some simple size metrics, such as the number of events, or the number of functions. The plugin can be started after loading an EPC in ProM. This EPC does not have to satisfy the strict requirements described in Section 2.6, e.g. functions may be directly connected to each other in this plugin. However, note that the other EPC plugins that are available in ProM may not work correctly if an EPC is loaded that does not satisfy the formal requirements. The calculation of the CC-metric is currently only available for EPC-like models. However, similar plugins to calculate the CC-metric for process models modeled in other languages can be easily developed. Moreover, existing conversion plugins may be used to convert models into an EPC.

7.5 Related Work

This section describes the related work on business process metrics. This related work can be organized in three categories: software metrics, data model metrics and business process metrics. Both data model metrics and business process metrics have a basis in software metrics.


ProM [4.2]
File Mining Analysis Conversion Exports Window Help
ProM
Analysis - PDM analysis
Zoom: 50 %
Load design
Calculate Coupling and Cohesion
18:40:56 [T] <PDMAnalysis>

Coupling and Cohesion Analysis
The Coupling/Cohesion ratio is: 11.81
The Process Coupling is: 0.744
The Process Cohesion is: 0.063

Activity	Activity Cohesion	Relation Cohesion (I...	Information Cohesio...
Activity_D	0.0	0.0	0.0
Activity_I	0.091	0.5	0.182
Activity_H	0.0	0.0	0.0
Activity_J	0.145	0.4	0.364
Activity_M	0.15	0.5	0.3
Activity_E	0.0	0.0	0.0
Activity_A	0.0	0.0	0.0
Activity_B	0.082	0.314	0.261
Activity_L	0.0	0.0	0.0
Activity_F	0.0	0.0	0.0
Activity_C	0.0	0.0	0.0
Activity_G	0.222	0.667	0.333
Activity_K	0.125	1.0	0.125

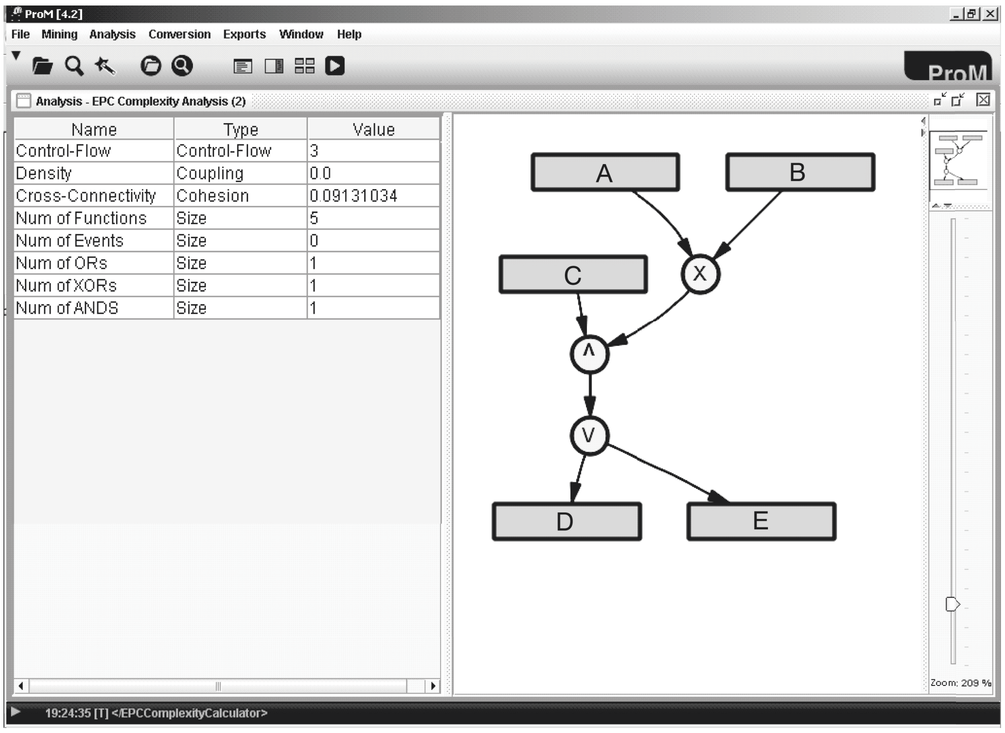
Figure 7.10: *A screenshot of the ProM plugin that calculates the coupling and cohesion measures for a process model designed on a PDM.*

7.5.1 Software Metrics

A vast amount of literature on software engineering metrics exists. As we have seen in Section 7.1, these metrics can be categorized into five categories [71, 259, 271]: (i) cohesion, (ii) coupling, (iii) complexity, (iv) modularity, and (v) size, of which cohesion and coupling are considered to be the most important [192]. We will briefly describe related work on these two types of metrics, since it has been the inspiration for our process coupling and cohesion metrics. Furthermore, we touch upon some of the influential metrics in the other categories of software metrics. These metrics all focus on procedural software programs. Finally, a brief overview of metrics for object-oriented programs is given. Surveys of existing software metrics can be found in e.g. [141, 185, 306].

Cohesion

The most popular cohesion metrics are described below. Stevens, Myers en Constantine [266] define seven types of cohesion within a software module (ranked from lowest to highest): coincidental, logical, temporal, procedural, communicational, sequential, and functional. When there is no meaningful relationship among the elements in a module the cohesiveness between those elements is coincidental and low. On the other end of the spectrum, functional cohesion is the strongest type of binding between elements in a module; all elements are then related to the performance of a single function. Stevens et al. have not operationalized their notions of cohesion. They only give textual definitions of the seven types of cohesion. Also, they have


ProM [4.2]
File Mining Analysis Conversion Exports Window Help
ProM
Analysis - EPC Complexity Analysis (2)
A
B
C
X
^
V
D
E
Zoom: 209 %
19:24:35 [T] </EPCComplexityCalculator>

Name	Type	Value
Control-Flow	Control-Flow	3
Density	Coupling	0.0
Cross-Connectivity	Cohesion	0.09131034
Num of Functions	Size	5
Num of Events	Size	0
Num of ORs	Size	1
Num of XORs	Size	1
Num of ANDS	Size	1

Figure 7.11: *A screenshot of the ProM plugin that calculates a number of process metrics for an EPC. The value for the CC-metric in this example is 0.09131034.*

not provided definitions of what an element of a module exactly is, e.g. a constant, variable, statement, procedure call, etc.

Lakhotia [158] solves the latter problem by defining an element of a module as an output variable. He uses a *variable dependence graph* to summarize the relationships between variables of a module and presents a number of logical rules for computing the cohesion of a module based on the seven types of cohesion by Stevens et al. and the variable dependence graph.

Emerson [94] reclassifies the seven types of cohesion defined by Stevens et al. Since he finds that software designers in practice often use a less refined set of cohesion levels, he proposes three categories: *Type I* ={functional, sequential, communicational}, *Type II* = {procedural, temporal}, and *Type III* = {logical, coincidental}. He represents a program as a flow graph and constructs a *reference set* for each variable. He then defines the cohesion metric based on the flow graph and the reference sets.

Also, Ott and Thuss [204] reclassify the seven types of cohesion into four categories: *low* = {coincidental, temporal}, *control* = {logical, procedural}, *data* = {communicational}, and *high* = {sequential, functional}. They use *program slicing* to determine the cohesion of a module. A *slice* of a program at a statement s with respect to a variable v consists of all statements of the program that may affect the value of v at statement s. Bieman and Ott [48] also use this slice-based approach to develop three cohesion measures that indicate the extent to which a module approaches the ideal of functional cohesion: *weak functional cohesion (WFC)*, *strong functional cohesion (SFC)*, and *adhesiveness*. Additionally, Bieman and Kang [47] introduce the notion of a so-called *input-output dependence graph (IODG)* which shows the relationships between the input and output elements of a module. Based on this graph they define three more slice-based measures: *loose cohesiveness (LC)*, *tight cohesiveness (TC)*, and *module cohesiveness (MC)*.

None of these approaches to calculate module cohesion are directly applicable to calculating the cohesion of activities in process models. Although data is central in our cohesion and coupling metrics, we do not distinguish different types of associations between data elements in one activity. For many software cohesion metrics these different types of cohesion between elements are essential.

Coupling

Although coupling is presented as an important metric in literature, we have found few metrics that determine coupling between modules in procedural programs.

Fenton and Melton [96] introduce *six types of module coupling* ranked from good to bad: (i) no coupling, (ii) data coupling, (iii) stamp coupling, (iv) control coupling, (v) common coupling, and (vi) content coupling. Every pair of modules in a software system has one of these types of coupling. If a pair has more than one connection, the connection having the strongest coupling counts. Based on the values for coupling between any two modules, a global coupling metric is defined on a system of modules.

Selby and Basili [257] define coupling as the number of *data bindings* between routines within a cluster and those outside of the cluster. In other words, the number of connections from the module to other modules of the system is counted. This number is combined with the strength of a module to get a coupling/strength ratio for the module.

The approach by Selby and Basili has been the basis for our coupling metric. Since the data bindings between modules are essential, this approach was almost directly applicable to our approach in which activities are groups of operations and data elements. We do not distinguish different types of coupling between activities.

Various researchers carried out studies to gather empirical evidence that coupling and cohesion metrics do indeed correlate with the quality of a software design. Bieman and Kang [47, 144, 145], in particular, have shown various examples how cohesion metrics can be used to restructure a software design. Also, in [257] evidence is presented that low coupling and high cohesion are desirable. By calculating coupling/strength ratios of a number of routines in a software library tool it was found that routines with low coupling/strength ratios had significantly fewer errors than routines with high coupling/strength ratios. In [61], a number of Fortran modules from a National Aeronautics and Space Administration project were examined. It was found that 50% of high-strength (high cohesion) modules were free of errors, whereas only 18 percent of low-strength modules were free of errors. No relationship was observed between error rate and coupling. The results of [61] indicate that modules with more descendants (modules that call many other modules) have a higher error rate. This work provides two important insights: (i) high cohesion reduces fault rate, and (ii) modules with many descendants (high coupling) are more error prone than those with few descendants. From these three empirical studies, we again conclude that both coupling and cohesion are important measures for the quality of a software design.

Other Metrics for Procedural Programs

The number of *Lines of Code* is probably the oldest and most widely used measure of size.

Halstead [120] defines a number of metrics for e.g. the length, the volume and the difficulty of a software program. These definitions are based on basic measurements of the number of operators and operands in the program. The basic measurements he identifies are: (i) the number of distinct operators in the program, (ii) the number of distinct operands, (iii) the total number of occurrences of operators in the program, (iv) the total number of occurrences of operands in the program, and (v) the number of conceptual input/output operands in the program.

The *cyclomatic complexity number* by McCabe [171, 172] measures the complexity of the control flow graph of a software program. It captures the number of paths in a program from start to end. The basic assumption is that the higher the number of paths in a program, the higher its control flow complexity.

Henry and Kafura [126] propose a complexity metric that is based on the information flow between modules in a software program. This *information flow complexity* metric considers three parameters: (i) the length of the program, (ii) the *fan-in*, which is the number of calls to a module plus the number of reads to global data structures, and (iii) the *fan-out*, which is the number of calls to other modules plus the number of write operations to global data structures.

We have seen already that coupling and cohesion are empirically proven to be important measures for the quality of a design. Also for the other metrics a number of studies demonstrate a significant correlation with the number of errors in the software design, e.g. [258] (Halstead's metrics). Additionally, their theoretical soundness has been investigated in e.g. [298] (Halstead's metrics, McCabe's cyclomatic complexity).

Object-Oriented Metrics

The software quality metrics described above all focus on *procedural programs*. After the introduction of object-oriented programming languages also *object-oriented software metrics* have been developed. In this section, only a few of these metrics are discussed. It is notable that coupling and cohesion are also important concepts in the development of object-oriented software metrics.

The most widely used object-oriented metrics are defined by Chidamber and Kemerer [67]. They introduce six metrics for object-oriented programs: (i) weighted methods per class, (ii) depth of inheritance tree, (iii) number of children, (iv) coupling between object classes, (v) response for a class, and (vi) lack of cohesion in methods. They also identify coupling and cohesion as important concepts for object-oriented designs.

To reuse their procedural cohesion measures for object-oriented programs, Bieman and Kang [46] add a new type of cohesion to the list of Stevens et al. [266]: *data cohesion*. Based on this type of cohesion they define the *tight class cohesion (TCC)* metric and the *loose class cohesion (LCC)* metric.

Simon, Löffler and Lewerentz [262] introduce a *distance based cohesion* metric for computing cohesion in object-oriented programs. This metric is based on the properties of the methods in the program and measures their similarity. Methods with similar characteristics should be put together in one object or module.

Many other object-oriented cohesion and coupling metrics have been developed, e.g. [26, 54, 91, 130]. Comprehensive overviews of these object-oriented software metrics can be found in [52, 53, 273, 306]. Reports on the theoretical and empirical validation of object-oriented metrics are available in e.g. [52, 53, 123].

7.5.2 Data Model Metrics

The adaptation of software metrics to assess the quality of other kind of designs has also been recognized in the area of database design. Several researchers have defined metrics to evaluate a database design specified by an ER-diagram or UML-diagram. An overview of these data model metrics is given in [218].

Gray et al. [110] propose an *ER metric* and an *area metric* to evaluate the quality of an *entity relationship (ER)* diagram. The goal of these two metrics is to provide database designers with quantitative support to compare design alternatives. They suggest that these metrics can be used to determine the effort required to implement a database design. Moreover, these metrics should help to identify problems in the design. No empirical evaluation of the metrics was carried out.

Kesh [148] considers both ontological and behavioral components to determine the quality of a data design. His method consists of three steps. First, the scores for the individual ontological components should be determined for a data design. These components include *suitability*, *soundness*, *consistency*, *conciseness*, *completeness*, *cohesiveness*, and *validity*. Then, the scores for each behavioral component (e.g. *usability*, *maintainability*, and *performance*) are determined by combining the relevant ontological components following the formulae defined. The performance is for instance dependent on the conciseness and completeness of the design. Finally, based on the behavioral components the score for model quality can be calculated.

Moody [183] defines a comprehensive set of quality metrics for a database design, including objective as well as subjective metrics. He defines eight quality factors (i.e. completeness, integrity, flexibility, understandability, correctness, simplicity, integration, and implementability) that all are measured by different metrics. Completeness, for instance, is measured by the number of items in the data model that do not correspond to user requirements. Understandability is measured by the user rating of understandability of the model. The number of entities is used as a measure for simplicity.

Genero et al. [106] focus on measuring the maintainability of a database design. They define three types of metrics for an entity relationship diagram: *entity metrics*, *attribute metrics*, and *relationship metrics*. For instance, the *total number of entities* in the ER-diagram is an entity metric. One of the attribute metrics is the total number of composite attributes and the total number of binary relationships in the ER-diagram is a relationship metric. The metrics are theoretically validated [106, 105] using the frameworks by Briand et al. [54] and Zuse [309]. Also, a first attempt at an empirical validation is made using a case study [104] and a controlled experiment [103].

Piattini et al. [107, 219] define six metrics to evaluate the structure of an ER-diagram. The *RvsE metric* measures the relation between the number of relationships and the number of entities in the ER-diagram. The *DA metric* is the number of derived attributes divided by the maximum number of derived attributes. The *CA metric* measures the number composite attributes relative to the total number of attributes. The *RR metric* is the number of redundant relationships in the ER-diagram relative to the total number of relationships. The *M:NRel metric* measures the number of 'M:N' relationships compared to the total number of relationships and the *ISaRel metric* assesses the complexity of generalization-specialization relationships in the ER-diagram. A controlled experiment was carried out for validation of these metrics [102].

Although these data model metrics have been developed for the evaluation of database designs, some similarities with software metrics and business process metrics exist. For instance, simple size metrics, such as the number of entities in the ER-diagram, are used. Also, cohesion is an important notion in database design, e.g. Kesh [148] defines a measure for cohesion. Finally, the various kinds of relationships between the elements in a diagram play a role in the metrics of e.g. Piattini et al. [107, 219], just like Stevens et al. [266] have defined different kinds of connections between the elements of a software module.

7.5.3 Business Process Metrics

Most of the work on metrics in the area of business processes is of a more recent date and is still under development. The early development of process model metrics has been greatly inspired by and based on software quality metrics. In the tradition of this work, there are some works in the 1990s that are mainly rooted in software quality measurement.

Daneva et al. [77] introduce a set of complexity indicators for EPC models based on the visual attributes of the model: *function cohesion*, *event cohesion* and *cohesion of a logical connector*. From their validation with 11 EPCs they conclude that their metrics help to identify error-prone model fragments.

Morasca [184] proposes a set of simple metrics for software specifications designed with Petri-nets. He identifies size, length, structural complexity, and coupling as interesting attributes of a design. Each of these attributes are operationalized by different metrics. For instance, he proposes to use the *number of transitions* and the *number of places* as measures of size, and the *maximum minimal distance*, i.e. the diameter of the Petri net (cf. Definition 2.3.10), to measure the length of the model. The *number of base paths*, the *concurrent contribution* and *sequential contribution* measure the structural complexity and the *number of outbound and inbound arcs* of a subnet are used to determine coupling. Morasca's contribution is theoretical and does not present any empirical validation.

Latva Koivisto [161] proposes a number of *graph complexity* metrics to evaluate business processes. Among these metrics are the *coefficient of network complexity (CNC)*, the *cyclomatic number*, the *complexity index*, the *restrictiveness estimator*, and the *number of trees in the graph*.

In [230], Reijers introduces a first version of a cohesion metric based on the PDM. As an empirical validation of this simple cohesion metric, the application of the metric to a number of process models is compared with the decisions of 14 experienced process designers. The designers were shown ten design dilemma's, inspired by practical models, presenting them two alternative designs for each dilemma. Each of the designers indicated whether he preferred the one alternative over the other. A strong correlation is established between their answers and the outcomes of the simple cohesion metric. This simple cohesion metric has been the basis for the development of the metrics that are described in Section 7.2 of this thesis. It is extended with the possibility to handle alternative operations with the same output element and we have complemented the cohesion metric with a coupling metric.

Cardoso [63] has developed a *Control Flow Complexity (CFC)* metric which was validated against Weyuker's [298] complexity axioms [62] and tested with respect to its correlation with perceived complexity [64]. In his motivation of the CFC-metric Cardoso focuses on the cognitive effort to understand a process model. He refers to the *mental states* that may be generated by a process model and the different types of routing elements [63]. In contrast to our CC-metric, the CFC-metric does not consider the connections between different model elements, but it focuses on routing elements in isolation.

In [65], Cardoso et al. describe the application of several software complexity metrics to business processes. They use the Lines of Code metric, McCabe's cyclomatic complexity [171, 172], Halstead complexity [120], and the Information Flow metric by Henry and Kafura [126] to show how these metrics can be adapted to business process models. In addition, this survey identifies cognitive motivation as a potential backbone for the further development of business process metrics.

Rolón et al. [243, 244, 245] propose a set of metrics for evaluating business process models represented in BPMN. Their proposal is based on the application and adaptation of the FMESP software measurement framework [101]. In total they apply twelve metrics of the FMESP framework directly, e.g. the *number of activities (NA)* and the *number of precedence dependencies between activities (NDA)*, and they present 15 derived metrics specifically for BPMN processes, e.g. the *number of start events* in the process model (TNSE) and the *connectivity level between activities (CLA)*. Finally, they present a plan for conducting a number of experiments for evaluating the proposed metrics. At the time of writing this thesis no results of these experiments were available yet.

Mendling et al. take an experimental approach towards process model metrics that is driven by the explanatory power of a metric in an empirical setting. In [175, 177] they have tested 28 business process metrics (including size, density[3], structuredness, coefficient of connectivity, average connector degree, control flow complexity, etc.) as error predictors on a set of over 2000 process models from different samples. All metrics, except for density and the maximum degree of a connector, are confirmed to be correlated to error-proneness as expected. Another result of this study is a logistic regression model that is able to classify 90% of the process models correctly. Additionally, a survey on the understandability of process models is reported by Mendling, Reijers and Cardoso [178] in relation to the set of metrics mentioned in the previous study. A similar empirical evaluation of our CC-metric with respect to error prediction and understandability is provided in the next chapter of this thesis.

For an overview of process model metrics in general we refer to [65, 114, 175, 275]. Note that in the area of business process metrics no classification framework exists yet and that the term complexity is often used as a general term, indicating e.g. size or coupling.

7.6 Summary

In this chapter we have shown the potential of adopting metrics to business process models based on the similarities between process models and software program designs. We have introduced two sets of business process metrics. The first set of metrics builds on cohesion and coupling metrics in the software domain. The proposed metrics calculate the cohesiveness of an activity and the coupling between activities based on the content of the activities which is specified by the PDM. The second set of metrics is inspired by the theory of hard mental operations in the Cognitive Dimensions Framework. The CC-metric measures the tightness of connections between all activities in a process model. Because it focuses on the process model as a whole, its applicability is not limited to process models designed by PBWD. Both sets of metrics provide a means to evaluate a process model design and to compare different designs for the same process with each other.

[3]This notion of density is different from the notion presented in [174].

Chapter 8

Evaluation

This chapter contains a number of case studies, practical applications and evaluations of the theory presented in this thesis. The aim of this chapter is to show that 'product-based design and support of workflow processes' is feasible and that it helps to realize improvements to business processes. The first three sections describe case studies. In Section 8.1, a general application of PBWD to redesign an annual reporting process at a bank is presented. Section 8.2 shows how the coupling and cohesion metrics of Chapter 7 can help to decide on the granularity of the activities in a process model for student grant applications. In the third section, the practical application of the algorithms of Chapter 5 is illustrated using a redesign project for the process of granting firework ignition permissions. In the final section, a different type of evaluation is presented for the CC-metric of Chapter 7. This metric is empirically evaluated with respect to its validity as a metric to predict errors and understandability of process models.

8.1 Annual Reports for Mutual Funds

In this section, we report on a case study applying the PBWD method at ING group, a Dutch financial institution. The process redesign project was carried out in 2005 and focused on the redesign of the annual reporting process for mutual funds within the business unit ING Investment Management Europe (ING IM). ING IM is part of ING Investment Management, the principal asset manager of ING group. Besides the business unit ING Investment Management Europe, there are similar business units for the Americas and the Asia-Pacific region. In Figure 8.1, a time line of the project at ING IM is shown.

The main reason for redesigning the annual reporting process was a change of requirements. As a result of a number of newly introduced regulations, the performance requirements on the process changed (see [290]). For instance, the Pension and Insurance Chamber (PVK, in Dutch: Pensioen- en Verzekeringskamer) required that, from 2005, all pension funds present their annual figures before the first of July in the year following the year that is reported on, e.g. annual reports over 2005 should be completed before July 1st, 2006. In the original situation it took one year to complete all reports.

Moreover, the distribution of dividends takes place in May of each year. In the original situation, the approval for the majority of the annual fund reports was given later. This led to post-payments of extra dividend, which damaged the professional image of the mutual fund

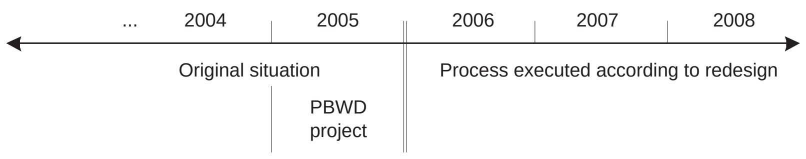


Figure 8.1: *A time line for the redesign project at ING IM. The project was carried out in 2005. The original situation describes the process before it was redesigned. The redesign was implemented in 2006. Annual reports are produced in the year following the reporting period. Thus, the annual reports of the years 2005-2007 have been produced in 2006-2008 according to the new process design.*

as well as that of the bank. An earlier completion of the process and approval of the annual reports was therefore desirable.

Besides the changes in regulations, another reason for redesigning the process was that the number of mutual funds increased over the years and that the structure of these funds became more complex. This led to a higher work load. It was expected that the work load would continue to increase in the coming years (following the redesign project in 2005) because of the introduction of more complex investment products.

Finally, it was recognized that the annual reporting process was not functioning well over the years preceding the redesign project. To examine the problems and issues in the process, an analysis of the original situation was made first. This analysis is described below.

8.1.1 Process Analysis

The ING IM business unit of the ING group manages assets for institutional clients, fund distributors and the ING labels, and can be seen as a portal for all investment actions. It has about 400 mutual funds with a total invested capital of over 146 billion Euros [168]. ING IM reports on its financial situation each year by means of annual reports that are made per mutual fund.

In the original situation, such an annual report is produced based on a template report, which is a framework containing the information that is required by the law and regulations. The template is authorized by accountants before the specific values for all data elements are filled in. After closing of the financial year, the financial data for each mutual fund are incorporated by the department of Accounting and Reporting (A&R) and are checked for their content. Next, the parts that contain standard text, e.g. the introduction or the overview of the mutual fund, are added by the department Mutual Funds Netherlands (MFN). The annual report is then checked and approved by managers of the two departments.

When approved, the annual report can be sent to an internal accountant for an extra check. After processing the remarks of the internal accountant, the annual report is assessed by an external accountant. His remarks and changes are processed by the department. The annual report is updated with the latest information and a final version is prepared. This version is possibly checked again by the internal accountant and necessary adjustments are made. Then, the external accountant checks the annual report, which can still lead to changes in the report. After the approval of the external accountant, the annual reports are compiled to booklets and sent to the print service.

A process model describing this original situation is shown in Figure 8.2. It covers the order of activities in the original situation as well as the resource groups that carry out the activities. In

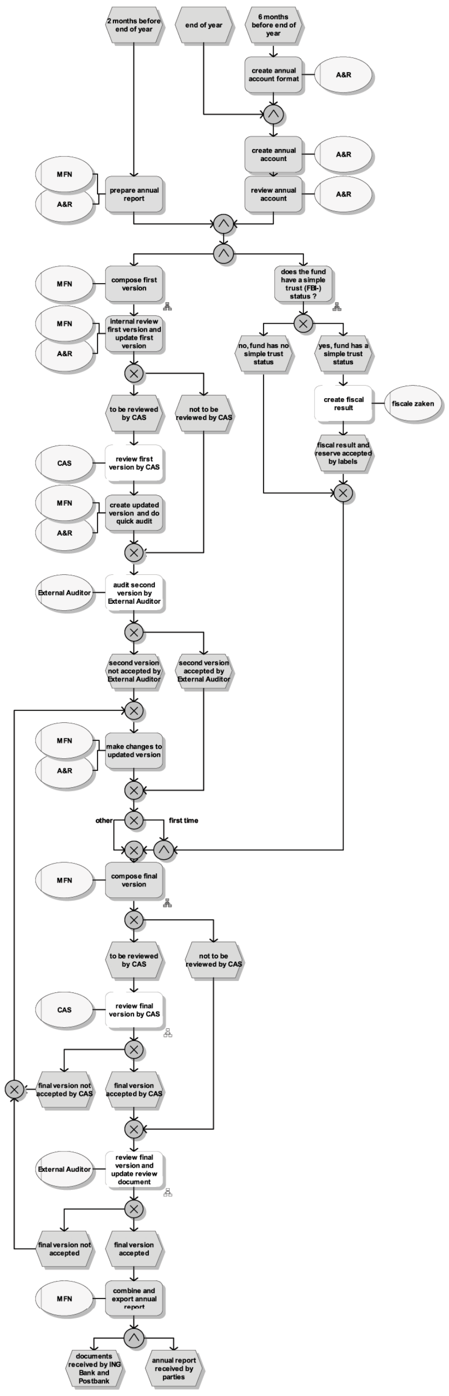

2 months before end of year
end of year
6 months before end of year
create annual account format
A&R
create annual account
A&R
review annual account
A&R
MFN
A&R
prepare annual report
MFN
compose first version
does the fund have a simple trust (FBI-) status ?
MFN
A&R
internal review first version and update first version
no, fund has no simple trust status
yes, fund has a simple trust status
create fiscal result
fiscale zaken
to be reviewed by CAS
not to be reviewed by CAS
fiscal result and reserve accepted by labels
CAS
review first version by CAS
MFN
A&R
create updated version and do quick audit
External Auditor
audit second version by External Auditor
second version not accepted by External Auditor
second version accepted by External Auditor
MFN
A&R
make changes to updated version
other
first time
MFN
compose final version
to be reviewed by CAS
not to be reviewed by CAS
CAS
review final version by CAS
final version not accepted by CAS
final version accepted by CAS
External Auditor
review final version and update review document
final version not accepted
final version accepted
MFN
combine and export annual report
documents received by ING Bank and Postbank
annual report received by parties

Figure 8.2: *The original process model (modeled as an EPC) of the annual reporting process at ING IM [290].*

addition, the systems that are used to execute the activities, and the input and output documents of each activity are listed. The process model contains a number of sub processes on which more detail is provided in [290].

Two important issues came to light in the analysis of the original situation: (i) the many handovers of work and responsibilities between the different departments involved in the process, and (ii) the number of control activities and checks in the process. For the creation and correction of an annual report always two departments (A&R and MFN) are needed. In the original situation there are at least 30 handovers of work between departments and 46 between persons. The handing over of work may lead to extra set-up times and extra waiting times [231]. Moreover, errors are more likely to occur due to these handovers. Thus, frequent work handovers may indicate a low efficiency of the process.

Secondly, there are relatively many control activities in the process (i.e. 6 of the 16 activities in the process model of Figure 8.2 are checks). These checks are necessary because many errors and mistakes are observed. The errors mainly relate to the content of the report, the lay-out, the naming, and the structure. Many of the control activities lead to inefficient double changes: in the annual report itself and in the source system from which the data element values are retrieved.

Because of the many handovers, and the number of corrections and mistakes in the draft versions of the annual report, the completion of all annual reports in the original situation takes one year. The main goal of the redesign project is to "redesign the process of annual reports such that, with the same capacity, the throughput speed is boosted and the process is less vulnerable for errors" [290].

To come to a redesign of the process that meets these specific goals, the PBWD approach was selected. The main reasons to choose for this redesign method were the following: (i) a data oriented approach was suitable since the process is very data intensive, i.e. many pieces of information are processed, (ii) a clean-sheet approach to redesign the process was desirable because of the many superfluous activities and inefficient checks.

During the redesign project, the steps of the PBWD method were followed as described in Section 1.4. First, the product, i.e. the set of annual reports for different mutual funds, was analyzed and represented in a PDM. Based on the product description a new process model was constructed and evaluated in a simulation study. These steps are described in more detail below.

8.1.2 Product Analysis

After an analysis of the original process, the 'product' generated in this process was analyzed. A PDM was constructed by analyzing the set of all annual reports. This product analysis was done in a top-down manner, starting from the annual report and adding more detail in each level of the PDM. Figure 8.3 shows an overview of this PDM.

There are three different types of annual reports: (i) the annual report of a master fund, (ii) the annual report of a feeder fund, and (iii) the annual report of an umbrella fund. Master funds directly deal at the stock market or invest in other master funds. Feeder funds invest in one or several master funds. Similar master and feeder funds may form an umbrella fund. The umbrella fund is a legal structure which makes it possible to distribute different series of shares for each specific sub fund. Despite the different types of funds, the annual reports of these funds are similar. Therefore, one PDM was designed for all types of funds together.

The PDM shown in Figure 8.3 has a simple and flat structure, i.e. there are only a few levels and many input elements lead to one output element. This can be explained by the fact that most of the data element values are present in ING's information systems and in other information sources such as the annual reports of the previous year. This information may be directly transformed to data element values in the annual report without intermediate steps. The PDM therefore does not contain any complex computations or derivations. In this case, the PDM can be seen as a hierarchical description of the data needed, in which the operations describe a 'consists of' relationship between the output element and the input elements. For example, in the PDM shown in Figure 8.3, the annual report comprises the annual accounts ($i08$), a general introduction ($i09$), an executive report ($i10$), the concerns of the board ($i11$), and the so-called *speed pages* ($i12$)[1]. The annual accounts ($i08$) are compiled of data on shares ($i26$) among others. And the data on shares is a set consisting of e.g. the price of the share at the beginning of the reporting period ($i62$) and the price of the share at the end of the reporting period ($i67$).

After the design of the PDM, the data elements were classified based on their content. The following types of data elements are distinguished [290]:

· *Fixed text* - these data elements are generic and are present in all annual reports.

· *Standard text* - these data elements are the same for the annual reports of the same type (i.e. master fund, feeder fund, or umbrella fund).

· *Conditional text* - this is standard text which is not applicable to all annual reports of a certain type and may therefore not be present.

· *Specific text* - these data elements contain specific information for a mutual fund.

· *Calculations* - these data elements contain information that is obtained by calculation.

· *Manual input* - these data elements contain information that is manually put into the annual report.

The annual report for a master fund for example contains a total of 581 data elements, of which 34% is fixed text, 5% standard text, 4% conditional text, 3% specific text, 35% calculations, and 19% manual input. The values for these data elements are obtained from three types of sources: textual sources (43%), system sources (24%), and manual sources (33%). A relatively high percentage of information has to be manually inserted or corrected. This manual information is mainly related to data from the annual report of the mutual fund of the previous year and data from information systems that do not have an automated interface to extract data. The manual input of data is prone to errors and should therefore be reduced if possible.

The PDM resulting from this product analysis has been the basis for the development of a process model, as is explained in the next section.

8.1.3 Process Model Design

At the time this case study was executed, the algorithms of Chapter 5 were not available yet. Therefore, a process model was manually designed by looking at the structure of the PDM. There are relatively few operations in the PDM compared to the number of data elements,

[1]Speed pages are digital documents containing performance information of the mutual funds. They are used for communication with the investor [290].

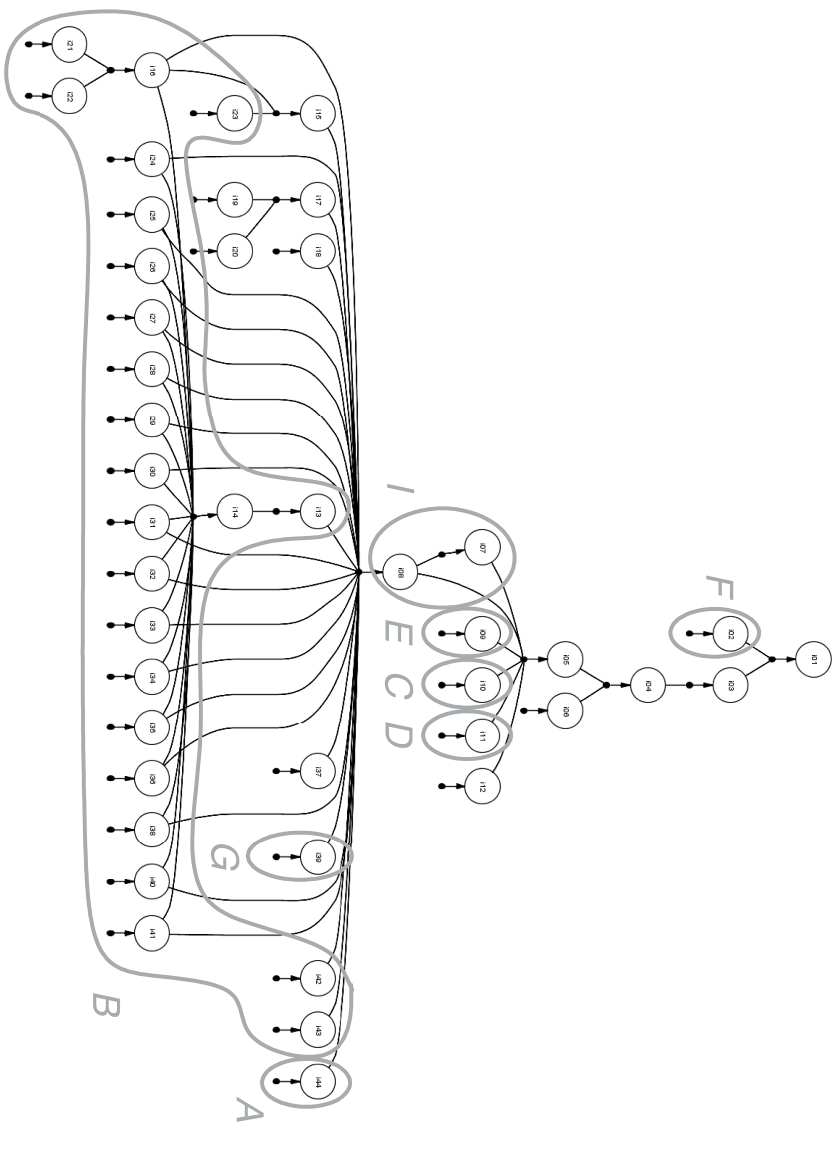


Figure 8.3: *The (high-level) PDM for the annual reporting process at ING IM. Note that the leaf operations in this PDM can be specified further, i.e. they do have input elements. However, for reasons of readability these extra elements are not shown here. In total there are 581 data elements in the PDM for the annual reporting process. For a complete and detailed PDM see [290].*

Data element	Description
i01	Approved annual report
i02	Evidence of ‘nauwe banden’ (transactions with affiliates)
i03	Updated annual report
i04	Remarks on annual report
i05	Annual report
i06	Financial argumentation
i07	Fiscal result
i08	Annual accounts
i09	General introduction
i10	Manager’s fund report
i11	Manager’s shares
i12	Speed pages
i13	Corrections on control sheet
i14	Control sheet
i15	Corrected ultimo account values
i16	Allocations
i17	Account values at the beginning of the reporting period
i18	Annual accounts of the previous year
i19	Corrections and allocations on the ultimo account values of the previous reporting period
i20	Ultimo account values of the previous reporting period
i21	Daily statements
i22	Ultimo account values of the current reporting period
i23	Corrections to the account value
i24	Data of purchase of own mutual funds
i25	Data on Net Asset Value (NAV)
i26	Data on shares
i27	Data on obligations
i28	Data on convertible bonds
i29	Data on participations
i30	Data on loans
i31	Data on deposits
i32	Data on options
i33	Data on the fund capital
i34	Overview of participations
i35	Overview of futures
i36	Overview of portfolio
i37	Average bank balance
i38	Attributed result
i39	Overview of exchange rates
i40	Overview of swaps
i41	Overview corporate actions (stock in loan)
i42	Overview exchange rate position
i43	Expense ratio
i44	Fixed text

Table 8.1: *Description of data elements for the PDM of the annual reporting process shown in Figure 8.3.*

since the operations take a large number of input data elements to produce one output element. Therefore, it was decided to directly translate operations into activities. The resulting process model follows the dependencies between operations in the PDM. Although the design step in the product based design method (cf. Section 1.4) prescribes the derivation of several process models with different activity orders, no alternative process models were generated. The order of execution of the leaf operations in the PDM for the annual reports process is unimportant since all operations have to be executed and there are no knock-out elements[2].

The process obtained by transforming the PDM only contains the steps that assemble the annual report based on the information needed. Preparation and control steps are not included in the PDM, although they may be legally required. Therefore, these activities were added to the process model. A high level overview of the redesign of the annual reporting process is given in Figure 8.4. The relationship between the process model and the PDM can be mainly recognized in the sub processes *Update tool according to new requirements (A)*, *Upload data (B)*, *Create and validate FER overview (G)*, *Generate annual accounts (H)*, *Determine fiscal result (I)*, *Create general introduction (E)*, *Create manager's fund report (C)*, *Determine manager's shares (D)*, *Generate annual report and send to KB-team (J)*, and *'Nauwe banden' (F)* as indicated by the PDM (Figure 8.3) and the process model of the redesign (Figure 8.4).

Although the redesign project at ING IM was already finished, we have applied the algorithms of Chapter 5 to the PDM for comparison afterwards. Figures 8.5-8.7 show some of the process models generated with the algorithms. Note that the process models generated by algorithms Charlie and Echo have a similar structure compared to the final process model of Figure 8.4, since these algorithms also directly translate operations to activities.

8.1.4 Conclusions

In the redesigned process model, that was constructed based on the analysis of the product, the number of work handovers has been significantly reduced (-60%) and so has the number of control activities (-18%). This was expected to lead to a decrease in errors, since many errors occurred as a result of the handover of work. Also, a simulation study of the redesigned process was developed that showed that the new process performs 50% faster [290].

At the moment of writing this thesis, the redesigned process has been in place for three years (i.e. the annual reports over the years 2005-2007 have been created with this process). The expected performance improvement has been realized. It now takes six months to complete all annual reports in stead of twelve. Moreover, the quality of (the draft versions of) the annual reports has substantially improved. The quality is measured by the number of errors found by external accountants who check the annual report. The most important errors are related to the content of the annual report, i.e. wrong numbers and figures. In the original situation 25% of the annual reports had to be corrected based on their content. After the redesign project, no corrections to the content of the annual reports occurred anymore. This quality improvement was not only appreciated at ING IM internally, but also by the external accountants.

This case shows that analyzing the product has played an important role in the insight in and redesign of the annual reporting process for mutual funds at ING IM. In addition to the reduction of the number of work handovers, control activities and throughput time, also important

[2]A *knock-out* element is a data element that may lead to a direct termination of (a part of) the process if its value satisfies a certain condition, e.g. data element H in the mortgage example of Section 3.2.2 is a knock-out element since the process can be terminated immediately if the value of H turns out to be 'negative' [7, 231].

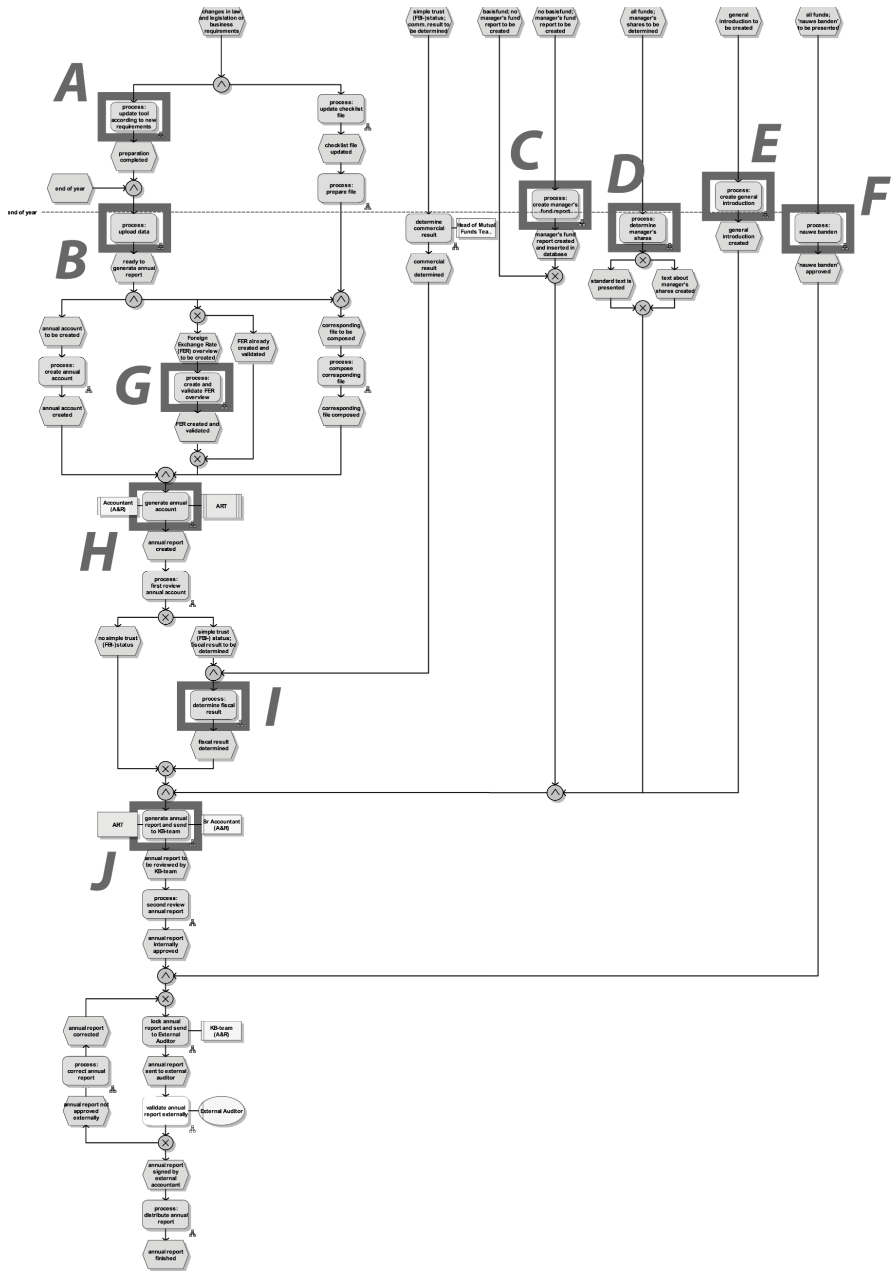

changes in law and legislation or business requirements
A
process: update tool according to new requirements
preparation completed
end of year
process: update checklist file
checklist file updated
process: prepare file
end of year
process: upload data
B
ready to generate annual report
annual account to be created
process: create annual account
annual account created
Foreign Exchange Rate (FER) overview to be created
FER already created and validated
G
process: create and validate FER overview
FER created and validated
corresponding file to be composed
process: compose corresponding file
corresponding file composed
simple trust (FBI-)status; comm. result to be determined
determine commercial result
Head of Mutual Funds Tea..
commercial result determined
basisfund; no manager's fund report to be created
no basisfund; manager's fund report to be created
C
process: create manager's fund report
manager's fund report created and inserted in database
all funds; manager's shares to be determined
D
process: determine manager's shares
standard text is presented
text about manager's shares created
general introduction to be created
E
process: create general introduction
general introduction created
all funds; 'nauwe banden' to be presented
F
process: nauwe banden
'nauwe banden' approved
Accountant (A&R)
generate annual account
ART
H
annual report created
process: first review annual account
no simple trust (FBI-)status
simple trust (FBI-) status; fiscal result to be determined
process: determine fiscal result
I
fiscal result determined
ART
generate annual report and send to KB-team
Sr Accountant (A&R)
J
annual report to be reviewed by KB-team
process: second review annual report
annual report internally approved
annual report corrected
process: correct annual report
annual report not approved externally
lock annual report and send to External Auditor
KB-team (A&R)
annual report sent to external auditor
validate annual report externally
External Auditor
annual report signed by external accountant
process: distribute annual report
annual report finished

Figure 8.4: *The redesigned process model for the annual reporting process at ING IM (reproduced from [290]). The parts of the PDM in Figure 8.3 are indicated by A-J.*

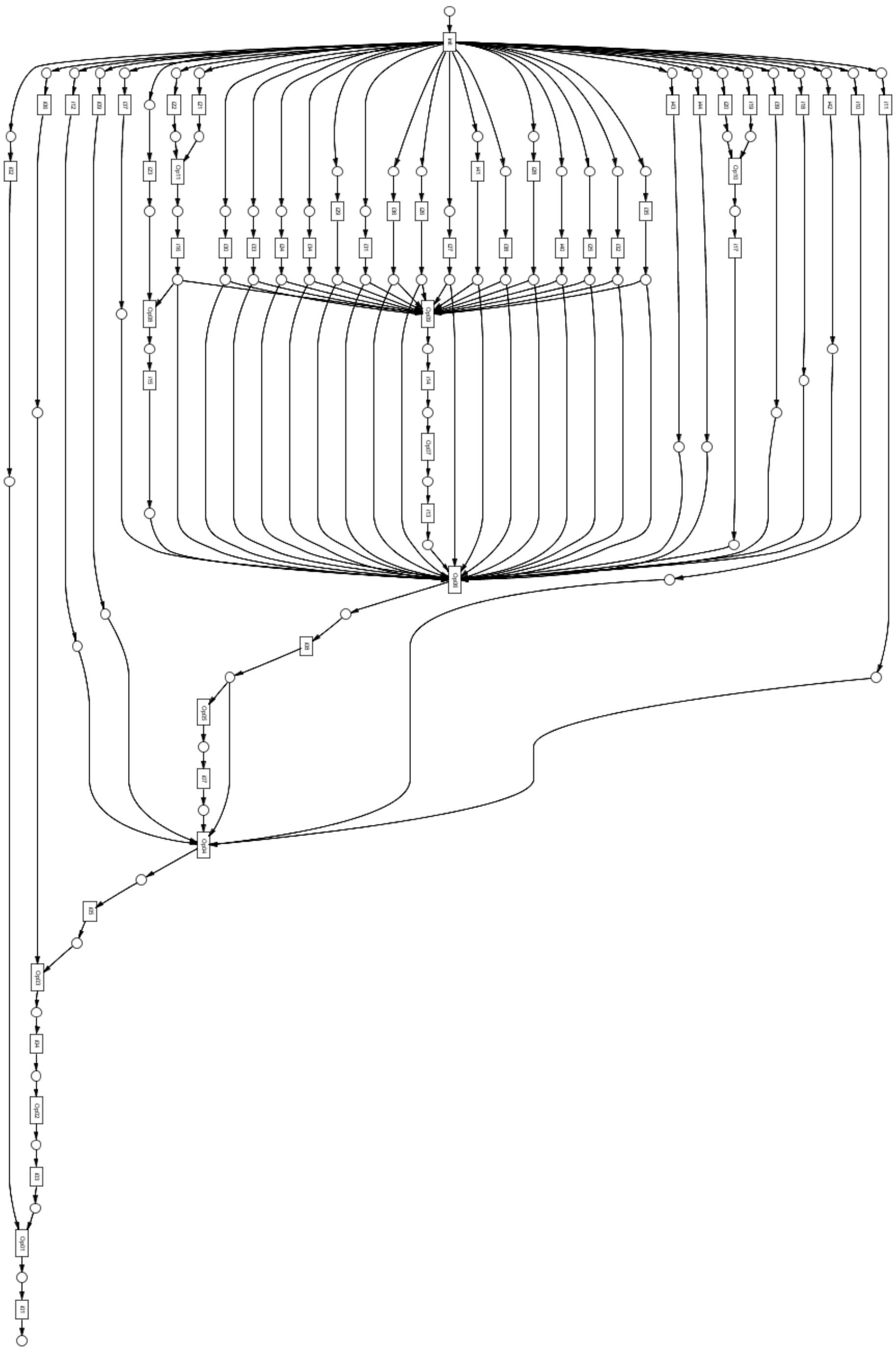


Figure 8.5: *The process model for the ING IM case, generated by algorithm Charlie. Note the similar structure of this process model to the manually designed process model of Figure 8.4.*

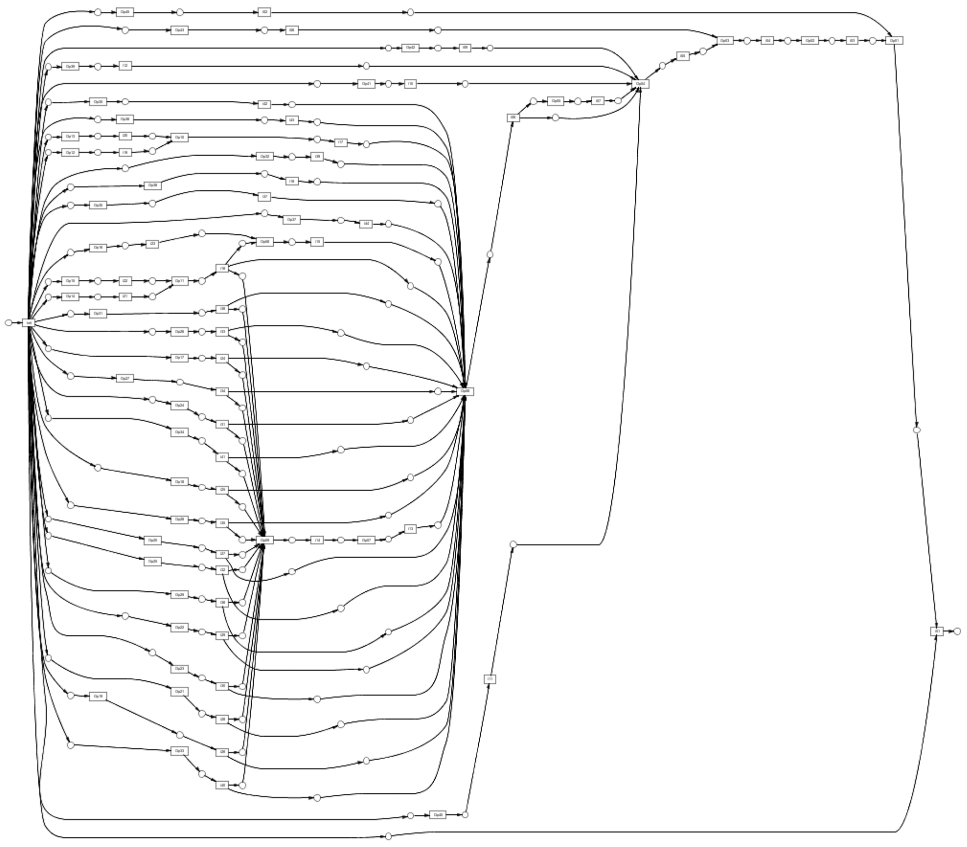


Figure 8.6: *The process model for the ING IM case, generated by algorithm Echo. Note the similar structure of this process model to the final process model of Figure 8.4.*

insights in the original process were obtained. For instance, it was concluded that the number of control activities was unnecessarily high because of many manual mistakes and unclear procedures.

Although the PDM did not directly lead to a complete process model because of the need for additional checks, it has been an important basis for it. Moreover, the product analysis and the creation of a PDM assisted the process owners and process designers in obtaining insight in improvement opportunities of the process. For example, an important observation from the analysis of the product is that the data obtained from information systems was not always reliable. Improving the quality of the data present in these systems already has led to fewer errors, and fewer control activities, and therefore fewer change iterations and a shorter throughput time.

8.2 Student Grants

In this section, a realistic example is described of applying the cohesion and coupling metrics in a design project. The process under consideration is the handling of student grant applications. In the Netherlands every student in professional and higher education (i.e. college or university)

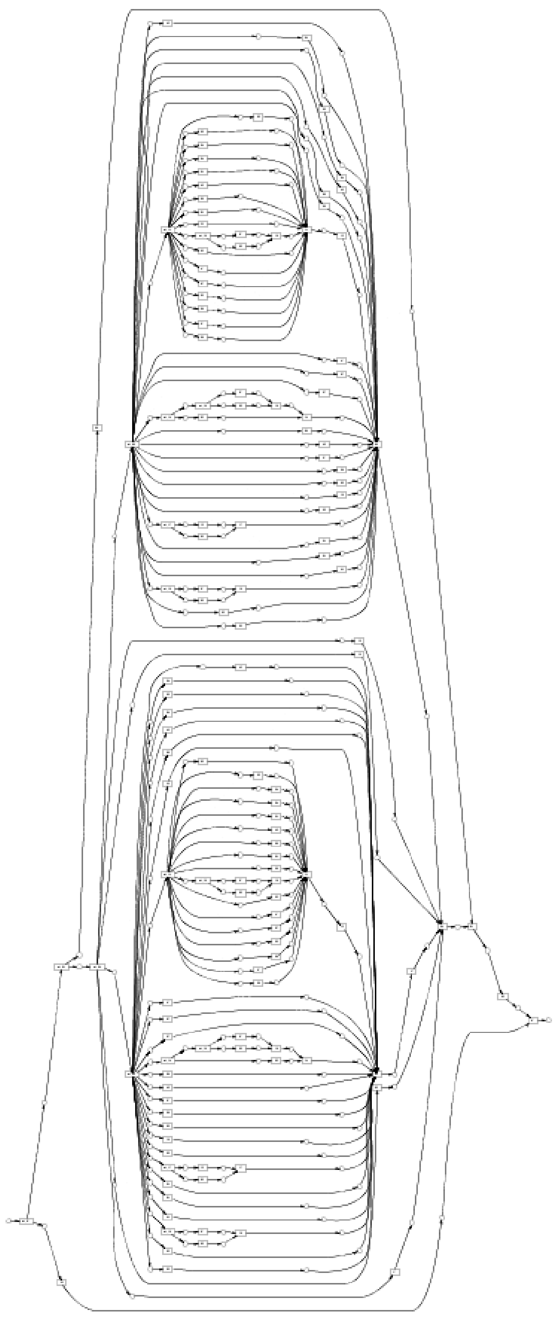


Figure 8.7: *The process model for the ING IM case, generated by algorithm Alpha. Note that the process model contains the same structure twice. Because data elements $i24$ - $i41$ are used in two (very large) operations, the process model contains two transitions per data element.*

is eligible to receive financial support from the government. Requests for student grants are assessed by a special institution in the Netherlands, called the IBG.

The design process for this case started with the analysis of the product, i.e. the decision on which amount of student grant an applicant is entitled to receive, and the design of a product data model. The information on the data elements was mainly retrieved from rules and regulations, the form that an applicant has to fill out, the IBG home page [133] and the brochures students receive about the application for a student grant. Below the PDM is discussed. Next, three alternative designs are generated manually. Using the coupling/cohesion ratio introduced in Section 7.2 the three designs are assessed and the best design is selected.

8.2.1 Product Data Model

A student grant comprises four types of grant: (i) a basic grant, (ii) a supplementary grant, (iii) a loan, and (iv) a credit for tuition fees. Below, these four types of grants are further explained. Also, the PDM for the student grant case is shown in Figure 8.8. Tables 8.2 and 8.3 contain an explanation of the data elements and operations of this PDM.

The basic grant ($i40$) is granted to every applicant who satisfies the legal conditions on e.g. nationality ($i21$) and age ($i23$) to receive a student grant from the government. The amount of basic grant may vary, based on the living situation of the applicant ($i33$). Table 8.4 shows the different amounts of basic grant that are allowed.

The supplementary grant ($i39$) is dependent on the income of the parents of the applicant ($i30$). The information on income is retrieved from the Dutch tax authorities. If the parents' earnings exceed a certain threshold, they are supposed to financially support their child and no supplementary grant is awarded. If the income of the parents is below the treshold, the applicant may receive a supplementary grant. In some cases the parents still are supposed to pay a parental contribution to their child based on their income. The amount of supplementary grant is maximized to an annually set limit (e.g. see Table 8.4).

If the applicant thinks the amount of basic and supplementary grant does not suffice to cover all expenses, he may receive an additional (interest-bearing) loan ($i41$). The maximum amount of loan ($i38$) that can be requested depends on the kind of education ($i48$). The applicant may request for any amount between zero and this maximum amount ($i37$). If he does not request a loan at all the requested amount of loan ($i37$) is set to 0. Finally, an additional credit ($i43$) can be granted to applicants who attend higher education. This additional credit covers the cost of the tuition fee ($i45$) of the school or institution the applicant is registered at.

Note that many of the input elements in the PDM are used in several operations, e.g. $i27$ and $i33$. The PDM does not have a tree structure like in the ING case, but it has an acyclic graph structure.

8.2.2 Process Model Design

When this case study was carried out, the algorithms of Chapter 5 for the automatic generation of process models based on a PDM were not available yet. Therefore, after the product analysis, three alternative process model designs were developed by manually grouping operations and their data elements together (see figures 8.10 - 8.12). The manual design of the first process model started with an investigation of the leaf elements of the PDM. All of these elements are provided by the applicant and can therefore be put together in one activity. Based on this input information the value for a knock-out element ($i27$) can be determined. Data element

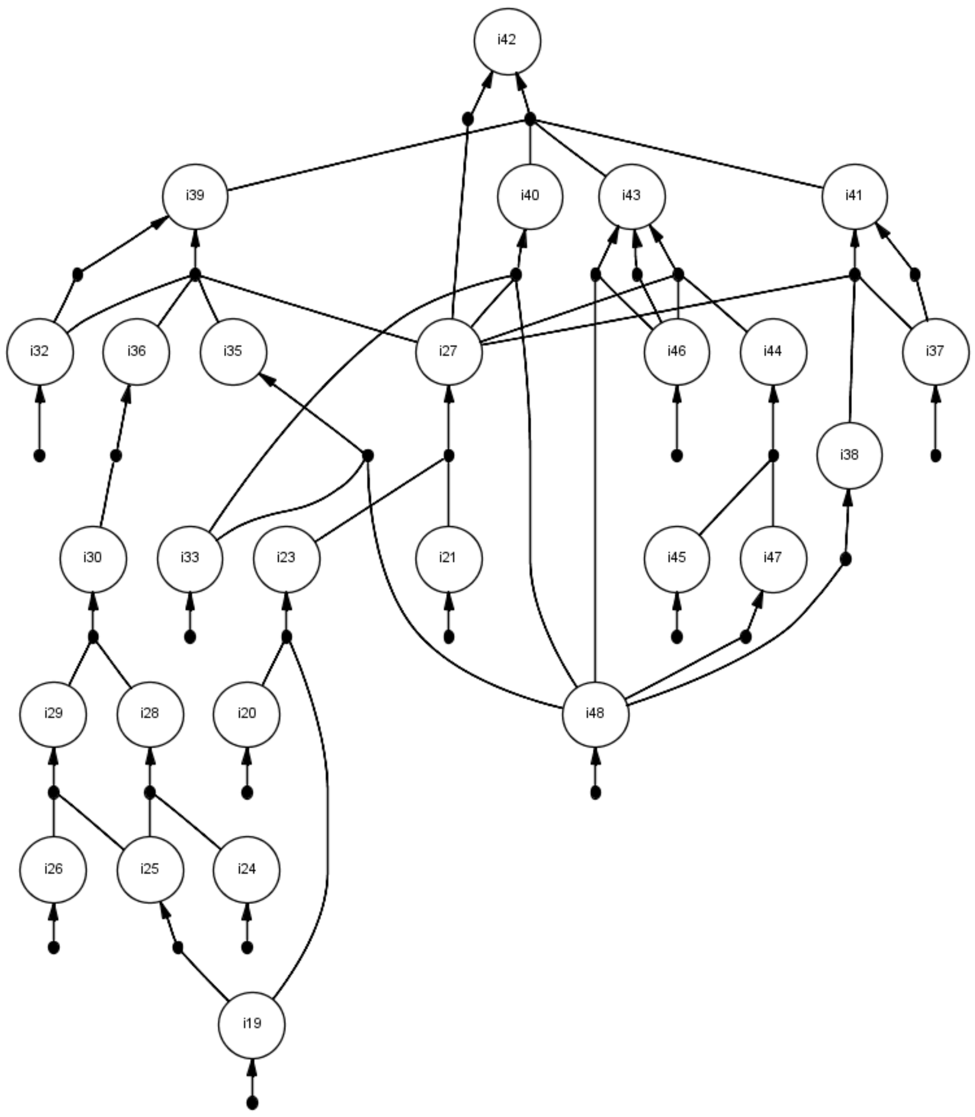


Figure 8.8: *The PDM of the student grant case. A description of the data elements and operations can be found in tables 8.2 and 8.3.*

Data element	Description
i01	Last name of applicant
i02	First name(s) of applicant
i03	Address of applicant
i04	Phone number of applicant
i05	Gender of applicant
i06	Social Security Number of applicant
i07	Country of residence of applicant
i08	Birth place of applicant
i09	Bank account number of applicant
i11	Signature of applicant
i19	Date of request
i20	Birth date of applicant
i21	Nationality of applicant
i23	Age of applicant
i24	Social Security Number of father of applicant
i25	Reference year for tax authority
i26	Social Security Number of mother of applicant
i27	Applicant has the right to receive a student grant
i28	Income of father of applicant
i29	Income of mother of applicant
i30	Income of parents of applicant
i31	Applicant has the right to receive a supplementary grant
i32	Applicant has requested supplementary grant
i33	Living situation of applicant (i.e. living at home, or living away from home)
i35	Maximum amount of supplementary grant
i36	Parental contribution
i37	Requested amount of loan by applicant
i38	Maximum amount of loan
i39	Amount of supplementary grant assigned to applicant
i40	Amount of basic grant assigned to applicant
i41	Amount of loan assigned to applicant
i42	Total amount of student grant assigned to applicant
i43	Amount of credit for tuition fees assigned to applicant
i44	Maximum amount of credit for tuition fees
i45	Tuition fees of educational institution of applicant
i46	Requested amount of credit for tuition fees
i47	Tuition fees declared by law
i48	Kind of education (i.e. higher education, or professional training) of applicant

Table 8.2: *Data elements and their description for the student grant case. Note that data elements $i01$-$i11$ are only used for a correct handling of the case. They are not involved in the determination of the end product.*

ID	Output	Input	Execution condition
Op01	i30	{i28,i29}	
Op02	i28	{i24,i25}	
Op03	i29	{i25,i26}	
Op05	i25	{i19}	
Op06	i23	{i19,i20}	
Op07	i27	{i21,i23}	
Op08	i40	{i27,i33,i48}	i27 = true
Op10	i35	{i33,i48}	
Op11	i36	{i30}	
Op12	i39	{i27,i32,i35,i36}	(i27 = true) $\wedge$ (i32 = true)
Op13	i39	{i32}	i32 = false
Op15	i38	{i33,i48}	
Op17	i41	{i27,i37,i38}	(i27 = true) $\wedge$ (i37 >0)
Op18	i42	{i27}	i27 = false
Op19	i42	{i39,i40,i41,i43}	
Op20	i43	{i46,i48}	(i46>0) $\wedge$ (i48 = 'professional education')
Op21	i43	{i27,i44,i46}	(i27 = false) $\wedge$ (i46>0)
Op22	i44	{i45,i47}	
Op23	i45	{i48}	i48 = 'higher education'
Op24	i41	{i37}	i37 $\leq$ 0
Op25	i43	{i46}	i46 $\leq$ 0
Op27	i19	$\emptyset$	
Op28	i20	$\emptyset$	
Op29	i21	$\emptyset$	
Op30	i24	$\emptyset$	
Op31	i26	$\emptyset$	
Op32	i33	$\emptyset$	
Op33	i37	$\emptyset$	
Op34	i46	$\emptyset$	
Op35	i47	$\emptyset$	
Op36	i48	$\emptyset$	
Op38	i32	$\emptyset$	

Table 8.3: *Operations and their attributes for the student grant case.*

	Professional education		Higher education	
Kind of grant	Living at home	Living away from home	Living at home	Living away from home
Basic grant	72.39	236.22	91.81	255.64
Supplementary grant	296.39	315.52	208.63	227.76
Loan	157.68	157.68	279.69	279.69
Credit for tuition fees	-	-	128.17	128.17

Table 8.4: *The (maximum) amounts of student grant that can be received each month (in Euros, valid in 2008) [133].*

$$
\begin{aligned}
D &= \{i19, i20, i21, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i35, i36, i37, i38,\\
&\quad i39, i40, i41, i42, i43, i44, i45, i46, i47, i48\},\\
O &= \{(i30, \{i28, i29\}), (i28, \{i24, i25\}), (i29, \{i25, i26\}), (i25, \{i19\}),\\
&\quad (i23, \{i19, i20\}), (i27, \{i21, i23\}), (i40, \{i27, i33, i48\}), (i35, \{i33, i48\}),\\
&\quad (i36, \{i30\}), (i39, \{i27, i32, i35, i36\}), (i39, \{i32\}), (i38, \{i33, i48\}),\\
&\quad (i41, \{i27, i37, i38\}), (i42, \{i27\}), (i42, \{i39, i40, i41, i43\}), (i43, \{i46, i48\}),\\
&\quad (i43, \{i27, i44, i46\}), (i44, \{i45, i47\}), (i45, \{i48\}), (i41, \{i37\}), (i43, \{i46\}),\\
&\quad (i19, \emptyset), (i20, \emptyset), (i21, \emptyset), (i24, \emptyset), (i26, \emptyset), (i33, \emptyset), (i37, \emptyset), (i46, \emptyset),\\
&\quad (i47, \emptyset), (i48, \emptyset), (i32, \emptyset)\},\\
C &= \{i27 = true,\ i27 = false,\ i32 = true,\ i32 = false,\ i37 \leq 0,\ i37 > 0,\\
&\quad i46 \leq 0,\ i46 > 0,\ i48 = \text{‘professional education’}, i48 = \text{‘higher education’}\},\\
W &= \emptyset,\\
root &= i42,\\
cost &= \emptyset,\\
time &= \emptyset,\\
cond &= \{((i40, \{i27, i33, i48\}), i27 = true),\\
&\quad ((i39, \{i27, i32, i35, i36\}), (i27 = true) \wedge (i32 = true)),\\
&\quad ((i39, \{i32\}), i32 = false),\\
&\quad ((i41, \{i27, i37, i38\}), (i27 = true) \wedge (i37 > 0)), ((i42, \{i27\}), i27 = false),\\
&\quad ((i43, \{i46, i48\}), (i46 > 0) \wedge (i48 = \text{‘professional education’})),\\
&\quad ((i43, \{i27, i44, i46\}), (i27 = false) \wedge (i46 > 0)),\\
&\quad ((i45, \{i48\}), i48 = \text{‘higher education’}), ((i41, \{i37\}), i37 \leq 0),\\
&\quad ((i43, \{i46\}), i46 \leq 0)\},\\
fprob &= \emptyset,\\
res &= \emptyset.
\end{aligned}
$$

Figure 8.9: *The formal definition of the PDM for the student grant case. Note that the data elements of Table 8.2 that are not used in the PDM are not included in the formal definition.*

$i27$ indicates whether the applicant has a right to receive a student grant based on a number of conditions. If the applicant does not satisfy these conditions, no student grant is awarded and there is no need to compute the amounts of sub grants. The determination of a value for $i27$ should be done as early as possible and is placed directly after the first activity. Next, the determination of the four sub grants are independent of each other and therefore are put in parallel branches. All operations that are involved in the determination of a value for a certain sub grant are put together in one activity, except for the supplementary grant. The determination of the supplementary grant consists of two activities because the first part (i.e. determining the parents' income) involves contact with an external party (i.e. the tax authority).

The process model of this first design is shown in Figure 8.10. It starts with an activity t_A which retrieves a value for all leaf data elements in the PDM, e.g. $i19$, $i24$, $i48$. These values are provided by the applicant by filling out a (digital) form. Next, in activity t_B it is checked whether the applicant satisfies the basic conditions to receive a student grant, i.e. the applicant should have the Dutch nationality and should be no older than 30 years. If these conditions are not satisfied, the process can immediately go to a final activity t_I in which the amount of student grant is set to 0 and the request is rejected (knock-out). If the applicant does satisfy the conditions, four parallel branches are started, each of them concerned with one of the parts of the student grant. In activity t_C the amount of basic grant is determined, in activity t_F the amount of loan, and in activity t_G the amount of credit for tuition fees. The calculation of the amount of supplementary grant is split in two activities: activity t_D determines the income of the parents, followed by the computation of the amount of supplementary grant in activity t_E. When all four calculations have been completed, a final activity t_H adds all grants to obtain the total amount of grant. Then, the process is finished.

The second and third design are based on this first design. An issue in the design of the first process model was the granularity of the activities in the branch of the supplementary grant. Perhaps it would be better to have just one activity in all branches. Therefore, the activities t_D and t_E are merged to one larger activity t_{DE} in the second design of Figure 8.11. The calculation of the supplementary grant is a very large part of the PDM (see Figure 8.8). Thus, most of the work is done in activities t_D and t_E. Therefore, it may be wise to split these into smaller activities in a third design. The process model for the third design is shown in Figure 8.12. The content of activities t_D and t_E in the first design, t_{DE} in the second design, and t_{D1}, t_{D2}, t_{D3}, t_{D4}, and t_E in the third design are explained in more detail in Figure 8.14. Note that in these three process models the four different types of sub grants are clearly recognizable since each type is produced in a separate branch of the AND-split construct.

All three process designs satisfy the basic correctness criteria of Section 4.2. We will only illustrate this for the second design. The process model of the second design in Figure 8.11 has 25 possible execution traces:

$$
\begin{array}{ll}
\Gamma = \{ & \\
\quad \sigma_1 = \langle t_A, t_B, t_I \rangle, & \sigma_{14} = \langle t_A, t_B, t_F, t_C, t_{DE}, t_G, t_H \rangle, \\
\quad \sigma_2 = \langle t_A, t_B, t_C, t_F, t_{DE}, t_G, t_H \rangle, & \sigma_{15} = \langle t_A, t_B, t_F, t_C, t_G, t_{DE}, t_H \rangle, \\
\quad \sigma_3 = \langle t_A, t_B, t_C, t_F, t_G, t_{DE}, t_H \rangle, & \sigma_{16} = \langle t_A, t_B, t_F, t_{DE}, t_C, t_G, t_H \rangle, \\
\quad \sigma_4 = \langle t_A, t_B, t_C, t_{DE}, t_F, t_G, t_H \rangle, & \sigma_{17} = \langle t_A, t_B, t_F, t_{DE}, t_G, t_C, t_H \rangle, \\
\quad \sigma_5 = \langle t_A, t_B, t_C, t_{DE}, t_G, t_F, t_H \rangle, & \sigma_{18} = \langle t_A, t_B, t_F, t_G, t_{DE}, t_C, t_H \rangle, \\
\quad \sigma_6 = \langle t_A, t_B, t_C, t_G, t_{DE}, t_F, t_H \rangle, & \sigma_{19} = \langle t_A, t_B, t_F, t_G, t_C, t_{DE}, t_H \rangle, \\
\quad \sigma_7 = \langle t_A, t_B, t_C, t_G, t_F, t_{DE}, t_H \rangle, & \sigma_{20} = \langle t_A, t_B, t_G, t_C, t_{DE}, t_F, t_H \rangle, \\
\quad \sigma_8 = \langle t_A, t_B, t_{DE}, t_C, t_F, t_G, t_H \rangle, & \sigma_{21} = \langle t_A, t_B, t_G, t_C, t_F, t_{DE}, t_H \rangle, \\
\quad \sigma_9 = \langle t_A, t_B, t_{DE}, t_C, t_G, t_F, t_H \rangle, & \sigma_{22} = \langle t_A, t_B, t_G, t_F, t_C, t_{DE}, t_H \rangle \\
\quad \sigma_{10} = \langle t_A, t_B, t_{DE}, t_F, t_C, t_G, t_H \rangle & \sigma_{23} = \langle t_A, t_B, t_G, t_F, t_{DE}, t_C, t_H \rangle \\
\quad \sigma_{11} = \langle t_A, t_B, t_{DE}, t_F, t_G, t_C, t_H \rangle & \sigma_{24} = \langle t_A, t_B, t_G, t_{DE}, t_C, t_F, t_H \rangle \\
\quad \sigma_{12} = \langle t_A, t_B, t_{DE}, t_G, t_C, t_F, t_H \rangle & \sigma_{25} = \langle t_A, t_B, t_G, t_{DE}, t_F, t_C, t_H \rangle \\
\quad \sigma_{13} = \langle t_A, t_B, t_{DE}, t_G, t_F, t_C, t_H \rangle & \qquad \qquad \}
\end{array}
$$

To verify the basic correctness of the process model with respect to the PDM, the execution traces are checked against a number of activity dependency expressions. The activity dependency expressions are derived from the input and output data sets of the activities in the process

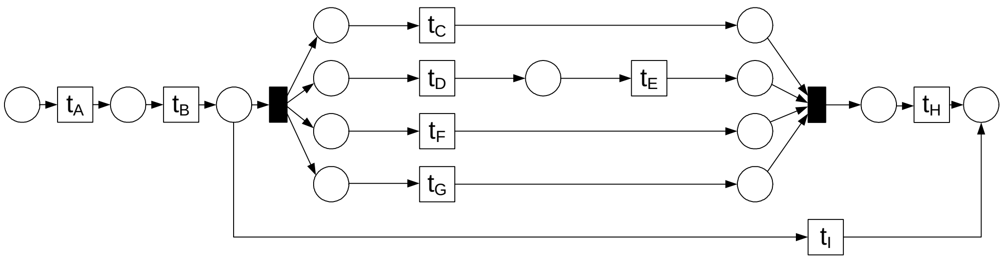


Figure 8.10: *The first design of the student grant process. The content of each activity in terms of operations and their data elements is specified in Figure 8.13.*

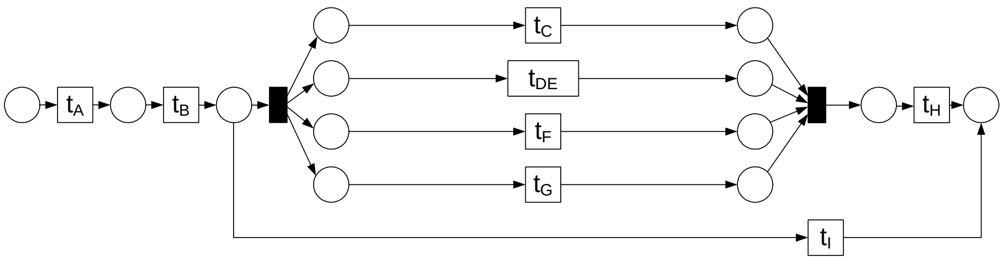


Figure 8.11: *The second design of the student grant process. The content of each activity in terms of operations and their data elements is specified in Figure 8.13.*

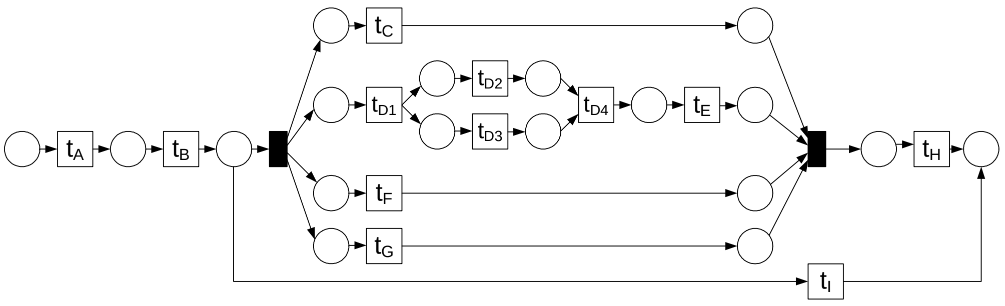


Figure 8.12: *The third design of the student grant process. The content of each activity in terms of operations and their data elements is specified in Figure 8.13.*

$$
\begin{aligned}
t_A &= \{Op27, Op28, Op29, Op30, Op31, Op32, Op33, Op34,\\
&\quad Op35, Op36, Op38\}\\
I_{t_A} &= \emptyset\\
\Omega_{t_A} &= \{i19, i20, i21, i24, i26, i32, i33, i37, i45, i46, i48\}\\
t_B &= \{Op06, Op07\}\\
I_{t_B} &= \{i19, i20, i21\}\\
\Omega_{t_B} &= \{i23, i27\}\\
t_C &= \{Op08\}\\
I_{t_C} &= \{i27, i33, i48\}\\
\Omega_{t_C} &= \{i40\}\\
t_D &= \{Op01, Op02, Op03, Op05\}\\
I_{t_D} &= \{i19, i24, i26\}\\
\Omega_{t_D} &= \{i25, i28, i29, i30\}\\
t_E &= \{Op01, Op02, Op03, Op05, Op10, Op11, Op12, Op13\}\\
I_{t_E} &= \{i30, i32, i33, i48\}\\
\Omega_{t_E} &= \{i35, i36, i39\}\\
t_{D1} &= \{Op05\}\\
I_{t_{D1}} &= \{i19\}\\
\Omega_{t_{D1}} &= \{i25\}\\
t_{D2} &= \{Op02\}\\
I_{t_{D2}} &= \{i24, i25\}\\
\Omega_{t_{D2}} &= \{i28\}
\end{aligned}
$$

$$
\begin{aligned}
t_{D3} &= \{Op03\}\\
I_{t_{D3}} &= \{i25, i26\}\\
\Omega_{t_{D3}} &= \{i29\}\\
t_{D4} &= \{Op01\}\\
I_{t_{D4}} &= \{i28, i29\}\\
\Omega_{t_{D4}} &= \{i30\}\\
t_{DE} &= \{Op01, Op02, Op03, Op05, Op10, Op11, Op12, Op13\}\\
I_{t_{DE}} &= \{i19, i24, i26, i27, i32, i33, i48\}\\
\Omega_{t_{DE}} &= \{i25, i28, i29, i30, i35, i36, i39\}\\
t_F &= \{Op15, Op17, Op24\}\\
I_{t_F} &= \{i27, i37, i48\}\\
\Omega_{t_F} &= \{i38, i41\}\\
t_G &= \{Op20, Op21, Op22, Op23, Op25\}\\
I_{t_G} &= \{i27, i45, i46, i47, i48\}\\
\Omega_{t_G} &= \{i43, i44\}\\
t_H &= \{Op19\}\\
I_{t_H} &= \{i39, i40, i41, i43\}\\
\Omega_{t_H} &= \{i42\}\\
t_I &= \{Op18\}\\
I_{t_I} &= \{i27\}\\
\Omega_{t_I} &= \{i42\}
\end{aligned}
$$

Figure 8.13: *The content of the activities of the three process designs shown in figures 8.10-8.12, including the input and output data sets for each activity.*

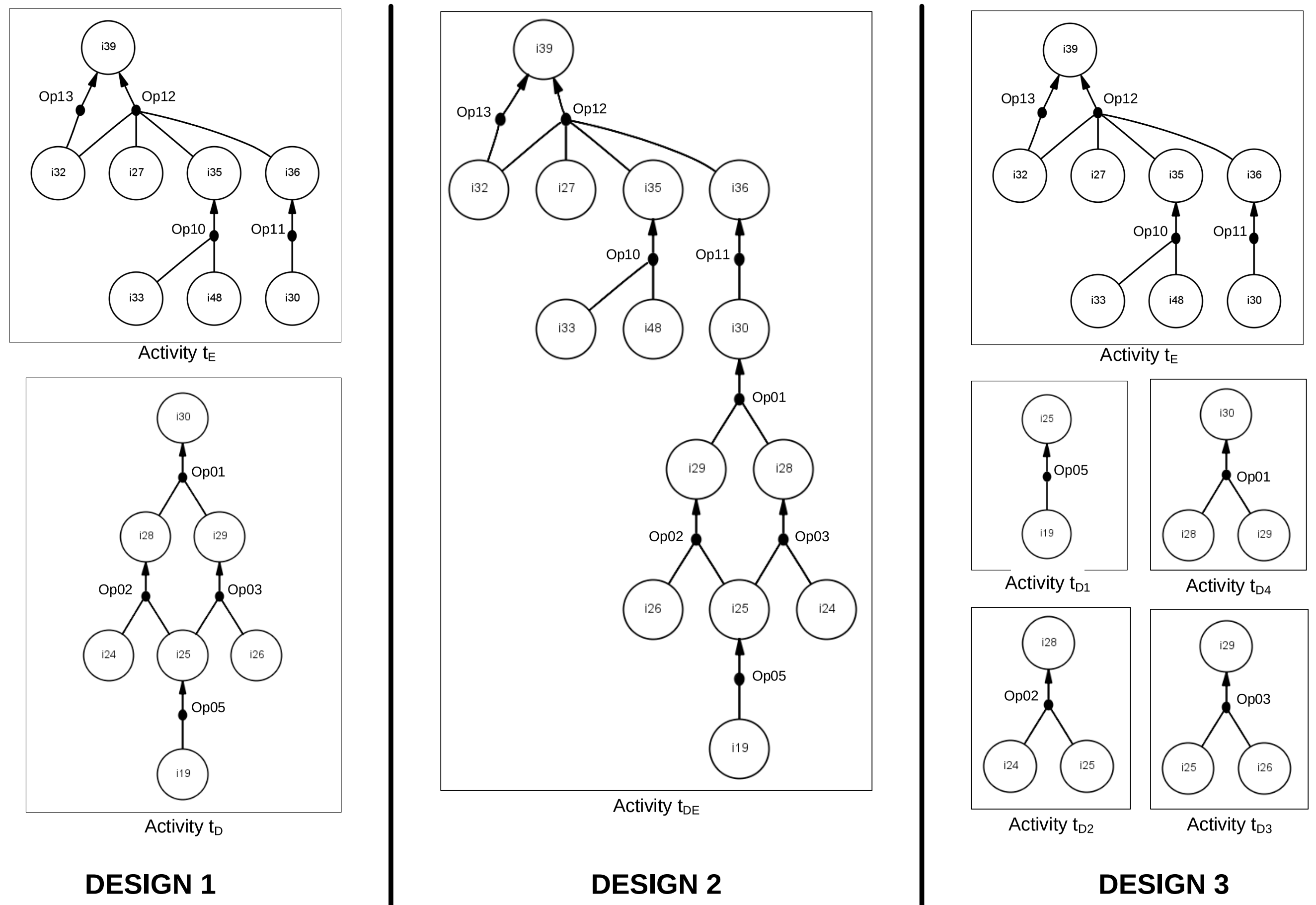


Figure 8.14: *The difference between the three designs lies in the granularity of the activities to determine the amount of supplementary grant. The first design has two activities, t_D and t_E. In the second design, these two activities are merged to one activity t_{DE}. In the third design, activity t_D is split into smaller activities: $t_{D1}, t_{D2}, t_{D3}, t_{D4}$, and t_E.*

Design	Process coupling/cohesion ratio (ρ)	Process cohesion (ch)	Process coupling (cp)
1 (Figure 8.10)	8.104	0.096	0.778
2 (Figure 8.11)	13.738	0.065	0.893
3 (Figure 8.12)	13.250	0.040	0.530

Table 8.5: *The values for process coupling and process cohesion metrics, and the process coupling/cohesion ratio for each of the three designs. The table shows that Design 1 is best in terms of the coupling/cohesion ratio.*

model. Figure 8.13 shows these data sets for all activities in the second design. The activity dependency expressions $\xi_{t_A}, \xi_{t_B}, \xi_{t_C}, \xi_{t_{DE}}, \xi_{t_F}, \xi_{t_G}, \xi_{t_H}, \xi_{t_I}$ are then combined to one expression Ξ:

$$\begin{aligned}\Xi \;=\; & (t_A \mapsto t_B) \wedge (t_A \mapsto t_C \wedge t_B \mapsto t_C) \wedge (t_A \mapsto t_{DE} \wedge t_B \mapsto t_{DE}) \wedge \\ & (t_A \mapsto t_F \wedge t_B \mapsto t_F) \wedge (t_A \mapsto t_G \wedge t_B \mapsto t_G) \wedge \\ & (t_C \mapsto t_H \wedge t_{DE} \mapsto t_H \wedge t_F \mapsto t_H \wedge t_G \mapsto t_H) \wedge (t_B \mapsto t_I)\end{aligned}$$

in which $\xi_{t_B} = (t_A \mapsto t_B)$, $\xi_{t_C} = (t_A \mapsto t_C \wedge t_B \mapsto t_C)$, etc.

Based on the execution traces and the activity dependency expressions it can now be verified whether the design satisfies the basic correctness criteria. The first criterion, all data elements in the PDM are produced by an activity, is satisfied. Secondly, all execution traces, $\sigma_1 ... \sigma_{25}$, respect the activity dependencies and the activity dependency expressions evaluate to true for all execution traces. Finally, none of the operations in the PDM can be executed more than once in any execution trace. Thus, the second design satisfies the basic correctness requirements and is consistent with the PDM.

To decide on which process model is best, the cohesion and coupling values have been calculated for each of the three designs. Table 8.5 shows that the first design has the lowest coupling/cohesion ratio and therefore is the preferred design. This outcome supports the intuition that the activities in the process model should neither be too small as in Design 3, nor too large as in Design 2.

As an illustration afterwards, we have also generated some process models with the algorithms presented in Chapter 5. The process models generated by algorithm Alpha and Echo are shown in figures 8.15 and 8.16, respectively. Note that in both generated process models the structure of the four different parts of student grant (i.e. basic grant, supplementary grant, loan, and credit for tuition fees) is clearly recognizable.

8.2.3 Conclusion

As this example illustrates, the choice of granularity of activities in a process model can be guided by the coupling and cohesion metrics. The coupling/cohesion ratio was used in this example as a relative measure to compare three designs for the same process.

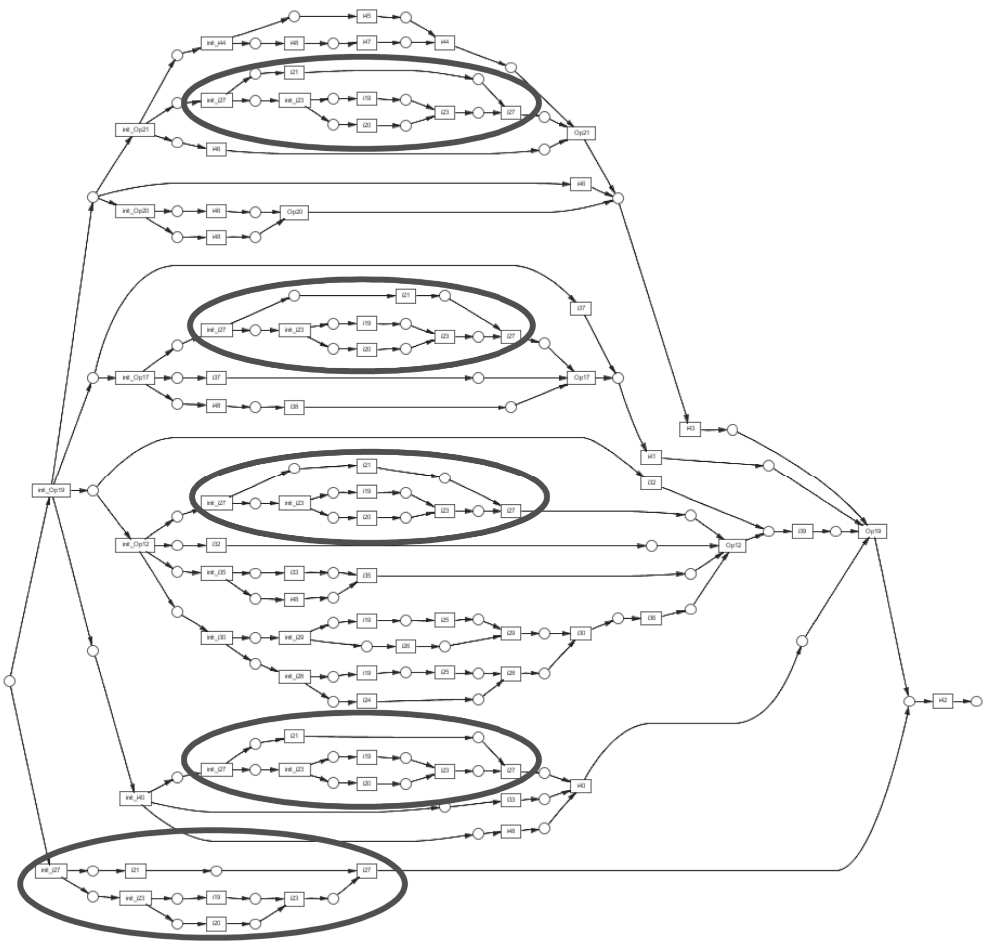


Figure 8.15: *The process model for the IBG case, generated by algorithm Alpha. Note that the four different parts of a student grant are clearly recognizable in the upper four branches of the AND-split* $init\_Op19$*. Also, note that the part of the process to determine the value for* $i27$ *is repeated five times in the model (indicated by ellipses).*

8.3 Firework Ignition Permissions

The case study described in this section is an application of PBWD and the algorithms presented in Chapter 5. This redesign project was carried out in 2007/2008 at the Provincie Noord-Brabant (PNB), which is one of the 12 Provincial Councils in the Netherlands. The responsibilities of PNB include the granting of subsidies, permissions and licenses with respect to environmental policies, and the supervision and enforcement of these policies. PNB deals with those requests that cannot be handled by municipalities since e.g. the request covers an area outside the municipal borders. One can for instance apply for a subsidy to invest in sustainable energy or nature conservation. Also, permissions to e.g. exploit a cattle farm or to store flammable products are granted by PNB. When an applicant applies for a permission, the details of the request are assessed with respect to law, regulations, security rules, concerns of

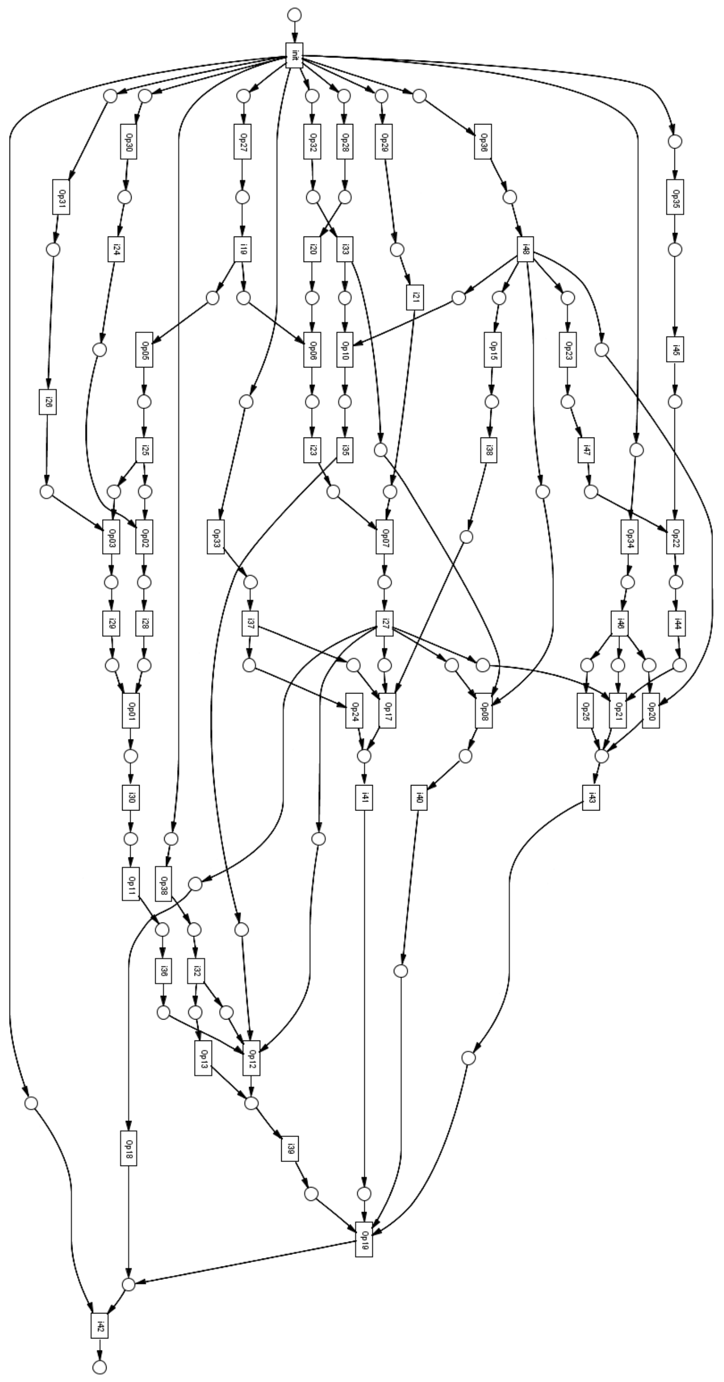


Figure 8.16: *The process model for the IBG case, generated by algorithm Echo. Note that the four different parts of a student grant are recognizable in the four input places of transition Op*19.

neighboring parties, etc.

PNB is redesigning some of its subsidy and permission processes to be of better service to their clients. They want to improve the throughput time of the process and the ease of applying for a permission. The process under consideration in this case study, is that of assessing an application to get permission to ignite fireworks at special events. If the ignition only involves consumer fireworks and it takes place at the turn of the year, then no permission is needed. For the ignition of (professional) fireworks at special events, e.g. at the opening or closing of a fair, or at a theater performance, a special permission is needed.

In addition to the improvement in service, the redesign project was also motivated by the fact that some rules and regulations on granting permissions have changed. PNB has imposed a rule to publish the grant of a permission, such that stakeholders are aware of it and can possibly object to it. It is required by law that a permission should then be published six weeks in advance of the event. A punctual processing of requests therefore is desirable. More details on this project can be found in [288].

The PBWD approach was used in this project because (i) the product of the process was unclear and an analysis of the product would therefore be helpful, (ii) the process is information intensive, and (iii) a clean-sheet approach to design the process seemed suitable since no process model was available yet. First of all, the original situation was analyzed. Next, the product was analyzed and a PDM was designed. Finally, one of the algorithms from Chapter 5 was selected to generate a process model. Because the process model is at the same level of detail as the PDM, this process model was used as a basis for the final design in which activities were grouped to form sub processes.

8.3.1 Process Analysis

In the original situation, i.e. before the redesign project was carried out, the process only contained manual steps. A checklist was available to assess the completeness of a request. This checklist contained questions such as: (i) Has the request been submitted in time? (ii) Does the applicant have a valid certificate of competence to ignite fireworks? (iii) Are there any buildings, waterways or national roads within the fireworks zone? and (iv) Is the firework ignited manually or automatically? These questions served as guidance for the first part of the process for the persons evaluating the request. The procedure to be followed after filling out the checklist was only partly described in a document which is called the *'quality system'*. Most of the knowledge on the procedure to grant firework permissions, however, was held by the persons executing the process.

By means of an analysis of the procedure description in the 'quality system' and a number of interviews with the relevant employees, a model of the original process was drawn up (see Figure 8.17). When a request for a firework ignition permission is received, the application first is assessed for its completeness (based on the checklist). If necessary, additional information is requested from the applicant. If the additional information is not received in time, the firework ignition permission is rejected. Otherwise, a draft of the decision on the permission is drawn and is legally checked. This concept decision is then sent to a number of external parties with expert knowledge, e.g. the fire brigade, the mayor, the Department of Waterways and Public Works, and the labor inspectorate. These parties are requested to assess the concept decision and give their advice or statement of no objection. Then the final decision is established and sent to the applicant.

Approximately 150 permissions for firework ignitions are granted each year. The request

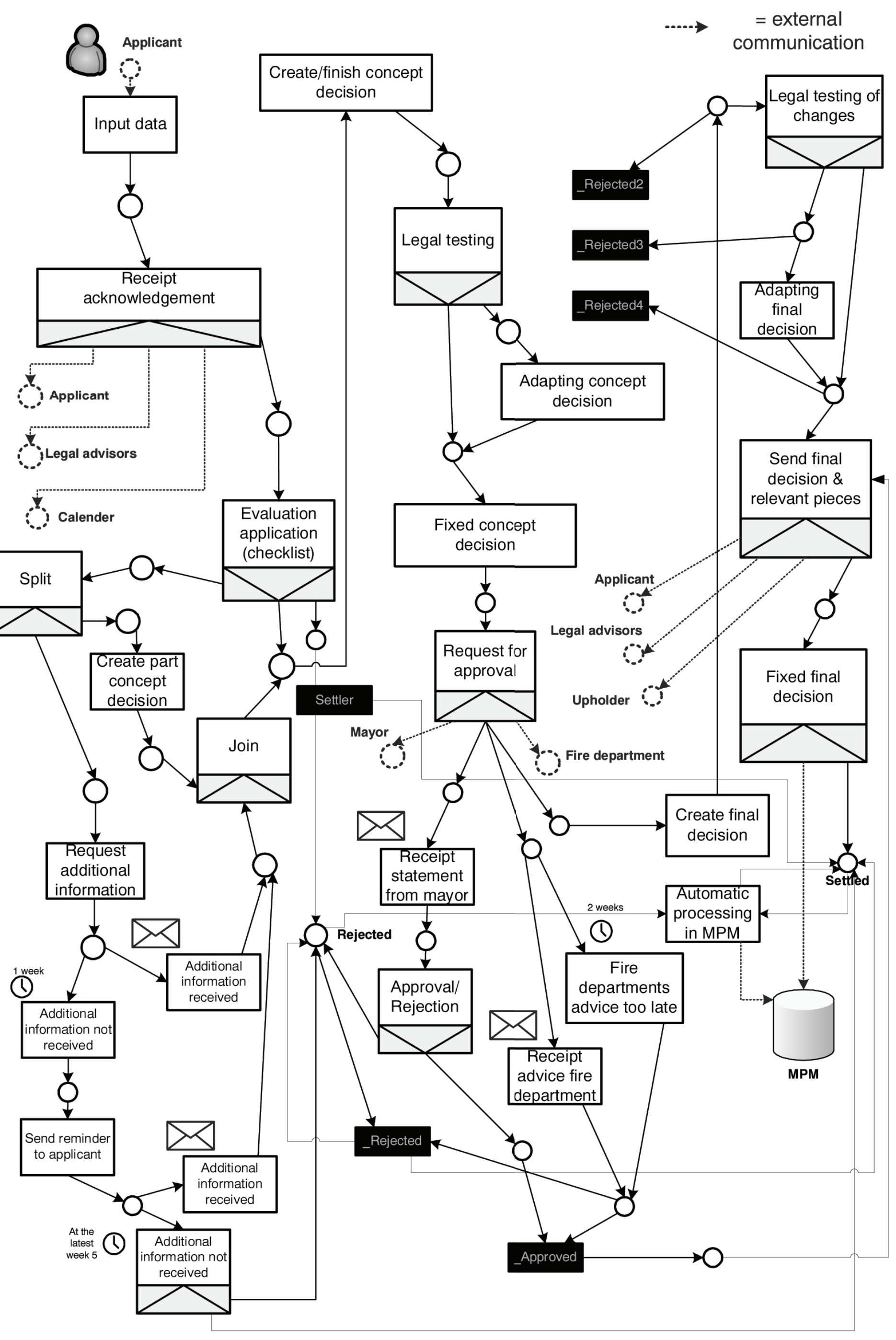

= external communication
Applicant
Input data
Create/finish concept decision
Legal testing of changes
_Rejected2
_Rejected3
_Rejected4
Legal testing
Adapting final decision
Receipt acknowledgement
Adapting concept decision
Applicant
Legal advisors
Calender
Send final decision & relevant pieces
Evaluation application (checklist)
Fixed concept decision
Split
Applicant
Legal advisors
Upholder
Request for approval
Create part concept decision
Settler
Fixed final decision
Join
Mayor
Fire department
Create final decision
Request additional information
Receipt statement from mayor
Settled
Automatic processing in MPM
2 weeks
Rejected
1 week
Additional information received
Additional information not received
Approval/ Rejection
Fire departments advice too late
MPM
Receipt advice fire department
Send reminder to applicant
_Rejected
Additional information received
At the latest week 5
Additional information not received
_Approved

Figure 8.17: *The process model of the original procedure for assessing the request of a firework ignition permission [288]. Note that the modeling language that is used has some similarities with YAWL. However, the model is very informal and is only used for communication with stakeholders.*

for a firework ignition permission has to be submitted at least 14 weeks before the date of the fireworks event. Because the intention to grant a firework ignition permission has to be published six weeks before the date of the fireworks, the process within PNB may take about eight weeks. The majority (around 90%) of the requests are processed within this time. The bottlenecks in the process are the parts in which information from external parties is required (i.e. the request for additional information from the applicant, and the requests for advice of the external expert parties) and the two time consuming activities in which the concept and the final version of the order are drawn up.

8.3.2 Product Data Model

The product of the process is the approval (or rejection) of a permission for the ignition of fireworks. To be granted permission, first of all, the applicant must have a firework licence. This licence is a continuous proof of the applicant's ability to work with fireworks in a safe and professional way. The applicant usually is a company. Besides the firework licence, the applicant also has to indicate which persons are going to ignite the fireworks during the event. These persons should also have a certificate of competence to prove their knowledge about fireworks and their skills to safely ignite fireworks. In addition, information on e.g. the insurance for the event, the exact time and location, and the types of firework that are to be used, has to be provided.

In Figure 8.18 the PDM of the firework ignition permission process at PNB is shown. The PDM has been designed based on the information obtained from the procedure description in the 'quality system', the interviews with employees, and relevant official orders, i.e. the Fireworks Decree [291] and the Working Conditions Decree [35]. Most of the leaf elements in the PDM are derived from the official orders, while the intermediate elements mostly are retrieved from the information about the process obtained by the analysis of the original situation.

Note that the final decision ($i02$) can be produced by many alternative operations. Because it is decided per case which external expert parties should give their approval of the concept decision, there are many alternative ways to produce the end product. The statement of no objection of the mayor ($i03$) is a compulsory element and is needed for the approval of any permission. The need for other advices, e.g. from the Air Force or the Department of Waterways and Public Works, is dependent on the situation. Sometimes, even an advice of an external expert party is requested but never received, which indicates the need for a flexible treatment of these expert advices.

8.3.3 Process Model Design

Based on the PDM described in the previous section a process model was derived using the algorithms of Section 5.2. To select the best algorithm, all dimensions of the framework were assessed and ranked. Based on this ranking, the algorithms that did not satisfy the requirements were excluded in a stepwise fashion, as is explained below.

1. Concurrency
   The possibility to simultaneously execute activities in the process model is considered to be very important since this is an opportunity to reduce the throughput time. For instance, if the advice of external expert parties is not requested in parallel this will significantly increase the throughput time of the process. By executing activities in parallel no unnecessary waiting time is introduced. The algorithms which generate a process model in which concurrent execution is possible are algorithms Alpha, Bravo, Charlie, Delta, and Echo.

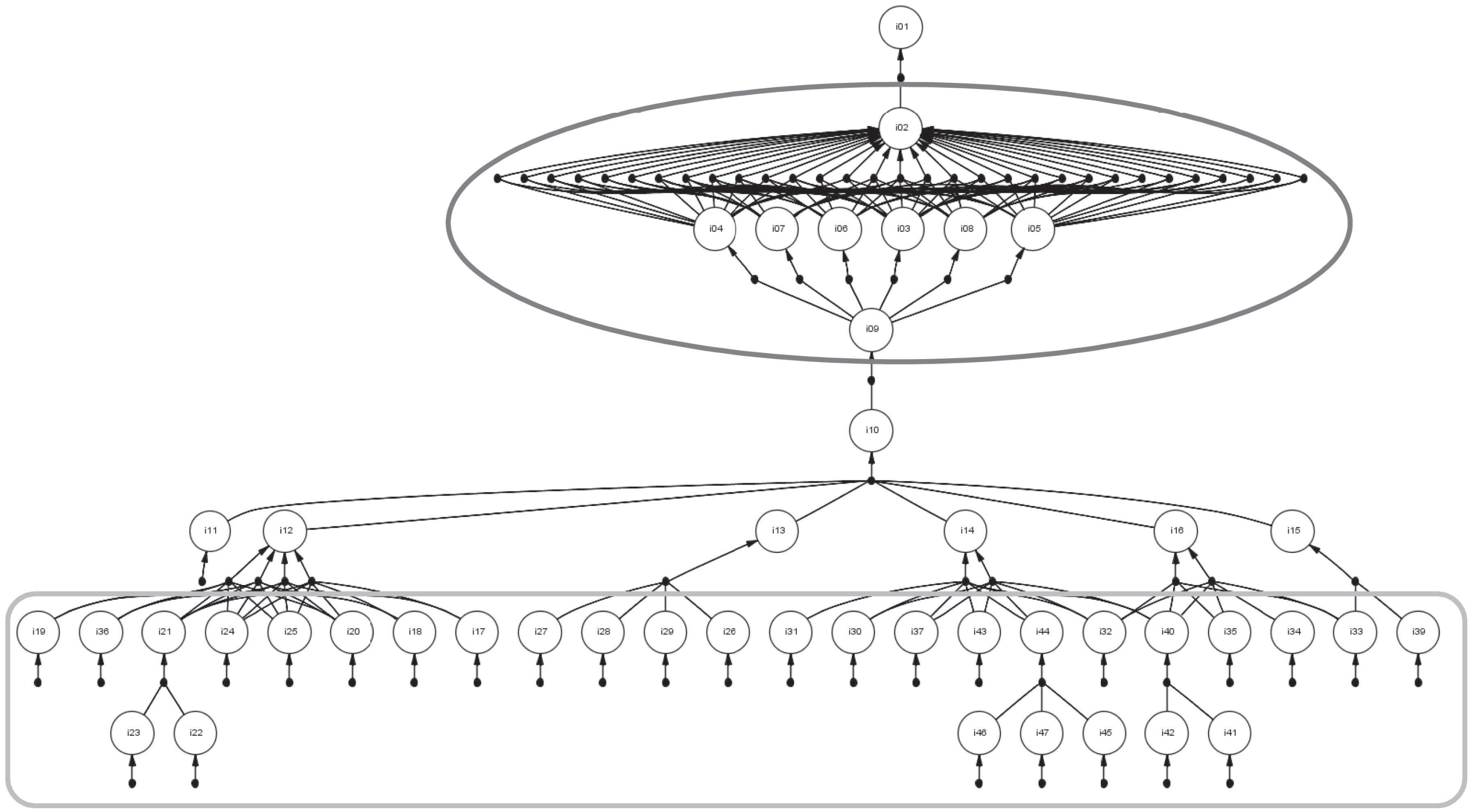


Figure 8.18: *The PDM for the process of granting firework ignition licences at PNB. The information on the input elements (indicated by a box) needed to assess the request, are mainly derived from the information in official decrees. Also note the construct (indicated by an ellipse) at the top of the PDM. Because it is decided per case which expert parties should give their approval on the concept order (only $i03$ is a compulsory approval), there are many alternative ways to produce data element $i02$.*

Data element	Description
i01	Fixed final decision on the firework ignition permission
i02	Final decision on the firework ignition permission
i03	Statement of no objection of mayor (M)
i04	Advice of the fire department (FD)
i05	Advice of the Provincial Executive (PE)
i06	Advice of the labor inspectorate (LI)
i07	Advice of the Department of Waterways and Public Works (WPW)
i08	Advice of the air force (AF)
i09	Fixed concept of decision
i10	Concept of decision
i11	Standard regulations
i12	Acceptance of applicant
i13	Validation of insurance
i14	Acceptance of request
i15	Acceptance of the list of firework pieces
i16	Acceptance of location
i17	List of holders of a firework licence
i18	List of holders of a certificate of competence
i19	Authority that has granted the firework licence
i20	Contact addresses
i21	Company
i22	Address of the company
i23	Post box number of the company
i24	Firework licence of applicant
i25	Certificate of competence of the person(s) who will be igniting the firework.
i26	Copy of valid insurance policy
i27	Duration of insurance
i28	Insurance company
i29	Insurance policy number
i30	Agreements parties concerned
i31	Additional information
i32	Table with work, dangers and measures
i33	List of firework pieces
i34	Additional information
i35	Visit on location
i36	KIWA site
i37	Contact information of the organization of the event
i39	VROM table
i40	Location
i41	Address of event
i42	Map of the environment of the event
i43	Kind of event
i44	Date and time of event
i45	Start time building and construction work
i46	Start time ignition of firework
i47	End time ignition of firework

Table 8.6: *Description of data elements for the PDM of the firework ignition permission process.*

2. Moment of choice
   The choices that are made for each case are unpredictable. For instance, the part of the process in which the external expert parties are consulted for their advice requires the freedom of making late choices. Before the start of a case it is not known which expert parties have to evaluate the concept of the decision because the selection of these parties is dependent on case-specific information that is provided at the start of the case or produced during the execution of the case. It may even be possible that one of the selected expert organizations does not return its advice in time. In that case it must be possible to proceed without the advice. Thus, the *moment of choice* should be as late as possible and algorithms Charlie, Delta, Echo, or Foxtrot are preferred based on this dimension of the framework. Thus, Alpha, Bravo and Golf are excluded, and from the list of Step 1 only Charlie, Delta and Echo are left.

3. Eagerness
   A process model can be constructed in such a way that it may only produce the values for data elements that are strictly needed. But its structure may also allow to execute more operations and produce more data element values than strictly necessary. The eagerness is then classified as overcomplete. Within this class the algorithms can be restricted, i.e. after determination of a value for the root element the execution is stopped, or unrestricted, i.e. after determination of a value for the root element still other steps can be executed. The freedom of producing more data element values than strictly necessary is considered to be important in this situation. This may for instance be important to reduce the throughput time if the retrieval of one of the alternative advices fails and an advice has to be requested from another external expert party. If the request to this new party is done already in parallel with the other requests less time is lost because of the unsuccessful request for the advice of the first party.
   In addition, it may not be desirable to select an overcomplete and unrestricted process model that may continue producing data element values after a value for the end product is determined. However, this problem is avoidable and can for instance be solved by manually correcting the process model or by closing the case immediately after producing a value for the end product. Therefore, both categories of overcomplete eagerness are considered. Thus, based on the eagerness criterion algorithms Charlie, Delta, Echo, and Foxtrot may be selected and the list of candidate algorithms resulting from the previous step is unchanged: Charlie, Delta, and Echo.

4. Representation
   The current algorithms leave a choice for the representation of the process model between a Petri net and a YAWL model. However, the preferred system (FLOWer) which was selected to support the process is not capable of dealing with cancellation regions in a process model and thus does not support the YAWL language. Choosing this language for representing the generated model would therefore lead to a high amount of rework before the model can be used for system enactment. Moreover, the YAWL language is also considered to be too complex for communication purposes with stakeholders. Finally, the specific algorithm that generates a YAWL model, i.e. algorithm Delta, provides rather large models. In these models it is difficult to get an overview of the process and to possibly recognize sub processes. The readability and manageability for both designers and implementers of the process model is poor. Since algorithm Delta provides a YAWL model which is not easily translated to a Petri net, it is excluded, and algorithms Charlie and Echo remain.

5. Order
   The order of construction of the process model can be top-down, bottom-up or middle-out. No motivations exist to clearly favor one of these dimensions. Since the middle-out order of construction actually does not prefer one order over the other it seems to be the most appropriate. Therefore, algorithms Charlie, Delta and Echo have priority over the other algorithms, but the others are not ruled out completely. Thus, from the list in the previous step still algorithms Charlie and Echo are good candidates to be used for the generation of a process model.

6. Focus
   The focus of the construction of a process model can either be on the data elements or on the operations of the PDM. There is no reason to prefer one of these two perspectives over the other. Therefore based on the focus criterium all algorithms are good candidates for the generation of a process. Thus, algorithm Charlie and Echo are still feasible alternatives.

From the selection based on the ranking of the dimensions in the classification framework, it was concluded that algorithms Charlie and Echo are most suitable as a basis for the design of a process model for the firework ignition permission process. Both algorithms were applied to the PDM. The resulting process models are shown in figures 8.19 and 8.21. The structure of both process models is similar. Two parts can be recognized in both models. In the first part, a value for all input elements is determined leading to a concept version of the decision ($i10$). The second part contains all combinations of advices that are requested from the external expert parties.

The process model generated by algorithm Charlie looks simpler than the process model of algorithm Echo. However, it contains a deadlock situation (see Figure 8.19 and Section 5.3.2). The token in the output place of e.g. transitions $i32$, is required as input to all of the transitions $Op09$, $Op10$, $Op12$, and $Op13$. $Op09$ and $Op10$ are alternatives to each other, just as $Op12$ and $Op13$, but always one of the operations in these two pairs has to be executed. This means that if e.g. $Op09$ has been executed, either $Op12$ or $Op13$ has to be executed. But that is impossible since the token from $i32$ was already consumed by $Op09$ and the process deadlocks. This structural problem can be solved in general by adding where necessary (i) arcs that return tokens to the places where they were taken from and (ii) some extra control places that make sure a certain transition can only fire once (cf. algorithm Delta). Or, it can be solved manually for a specific case. Figure 8.20 shows the manually corrected process model of Figure 8.19.

The process model generated by algorithm Echo also has a structural problem (cf. Section 5.3.2). In this process model, tokens may be left in the model when a value for the end product is already determined. Thus, after the end of the process some transitions may still be enabled. For the process model of Figure 8.21, this can be seen in particular in the part where one of the alternative operations is chosen for expert advices. In principle all of these branches are enabled (there is a token that enables each of the transitions $i03$, $i04$, $i05$, $i06$, $i07$, and $i08$), but not necessarily each of these expert advices has to be used. If the advice of a certain external expert party, e.g. the labor inspectorate ($i06$), is not needed a token is left in the model that enables this transition. In general, this problem can be solved by closing a case as soon as the end product is produced such that no activities can be executed anymore. But in this application the problem can be solved by manually adapting the process model and explicitly specifying the different choices between the external expert advices in a sub process of the final design (see Figure 8.24).

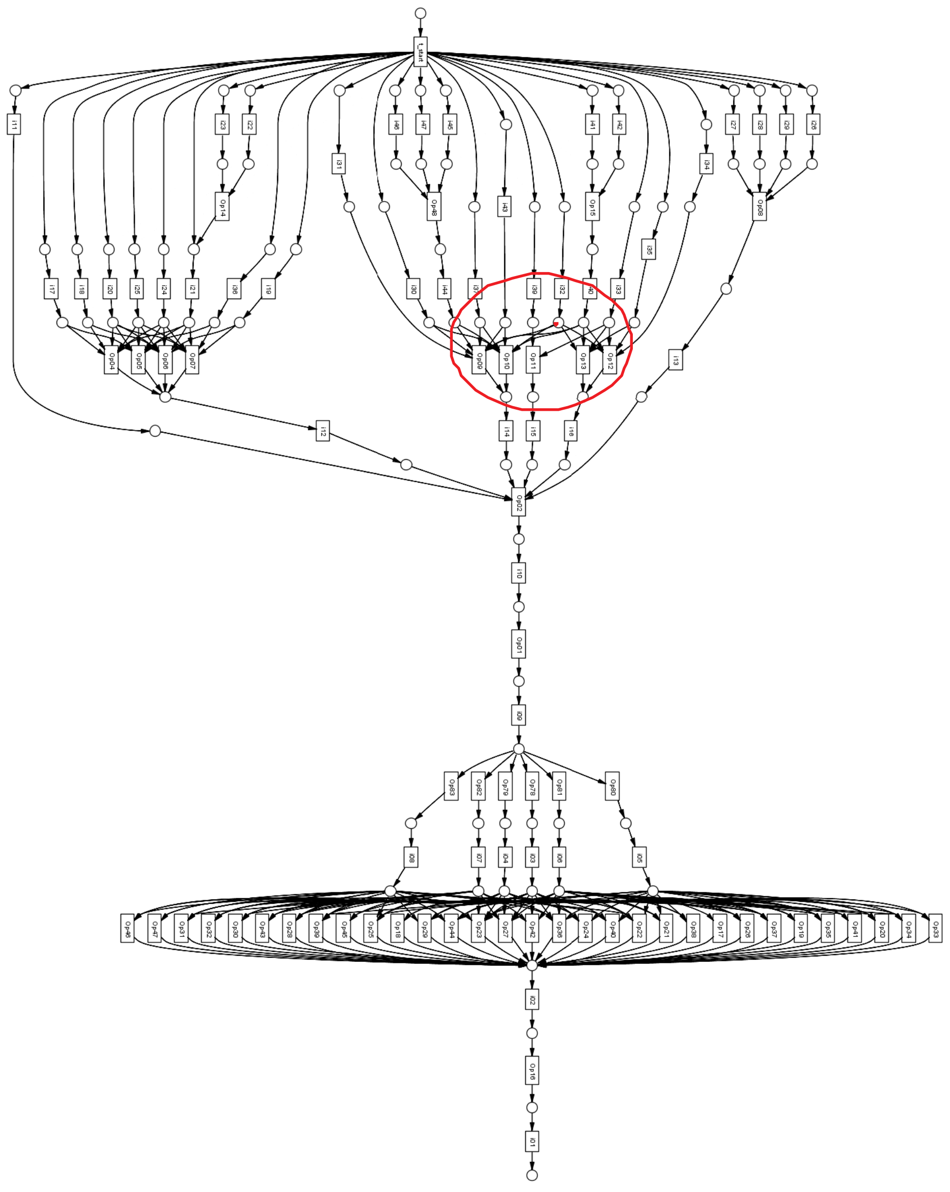


Figure 8.19: *The process model for the PNB case generated by algorithm Charlie. Note that the process model contains a deadlock (indicated by a red circle).*

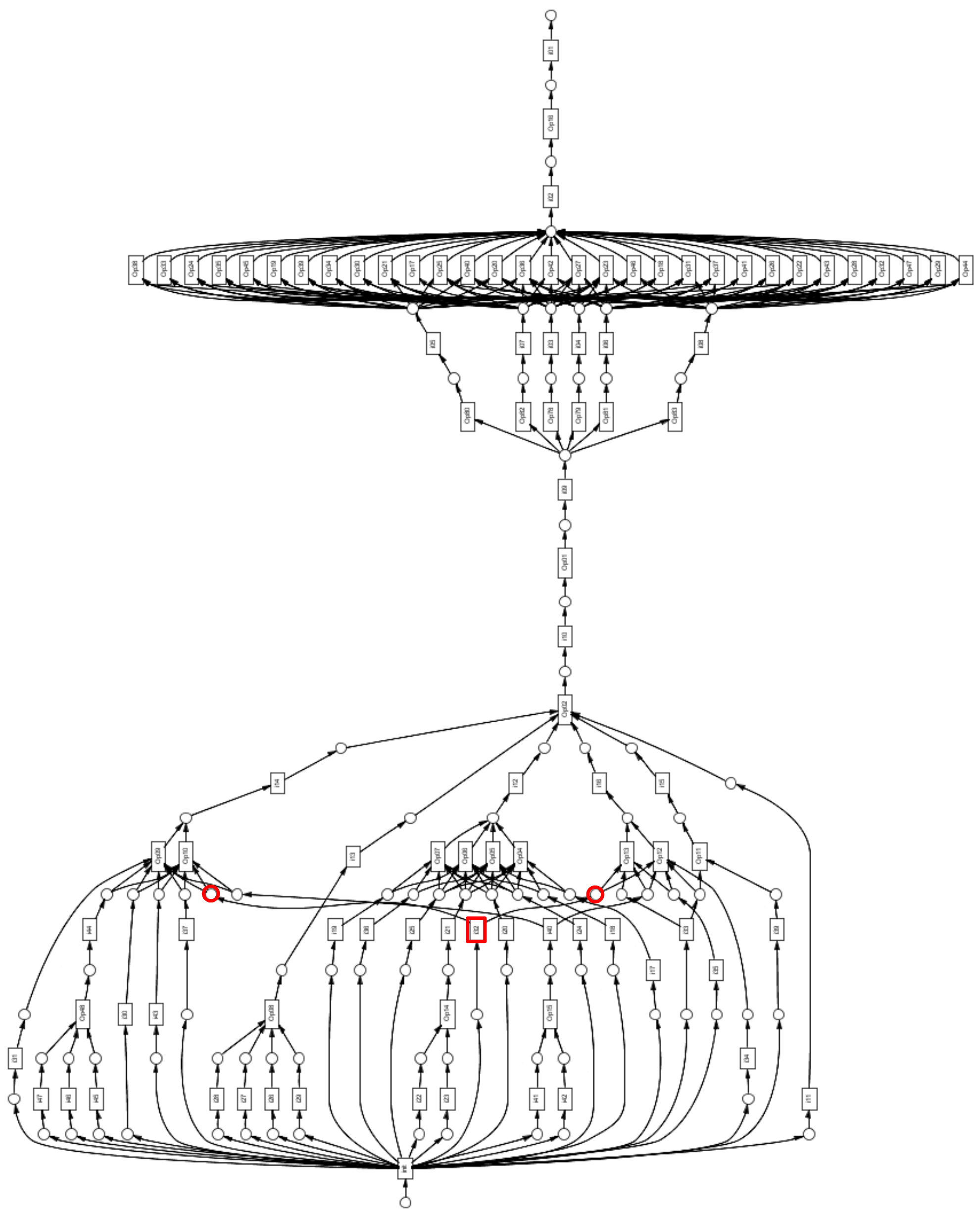


Figure 8.20: *The corrected process model of Figure 8.19. For each data element that is used in multiple operations extra output places are added, e.g. transition i*32 *now has two output places.*

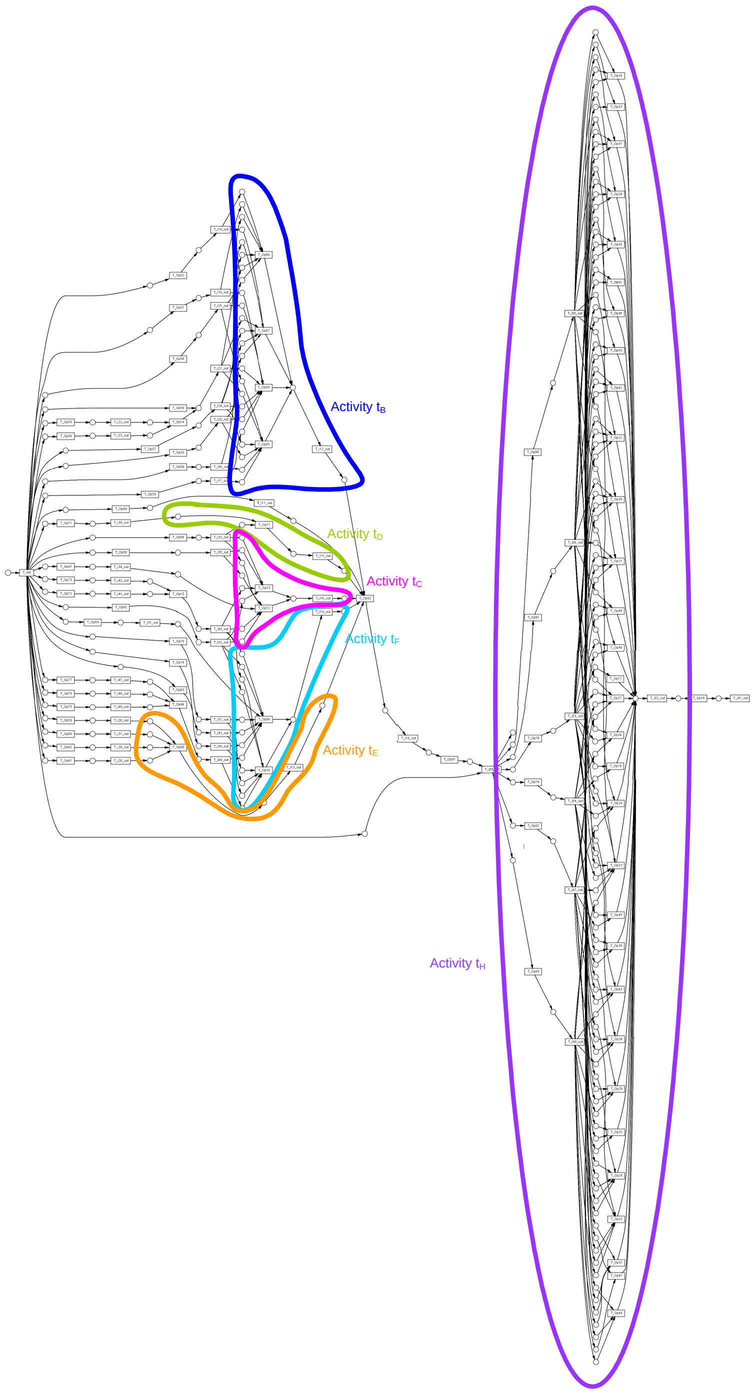


Figure 8.21: *The process model for the PNB case generated by algorithm Echo.*

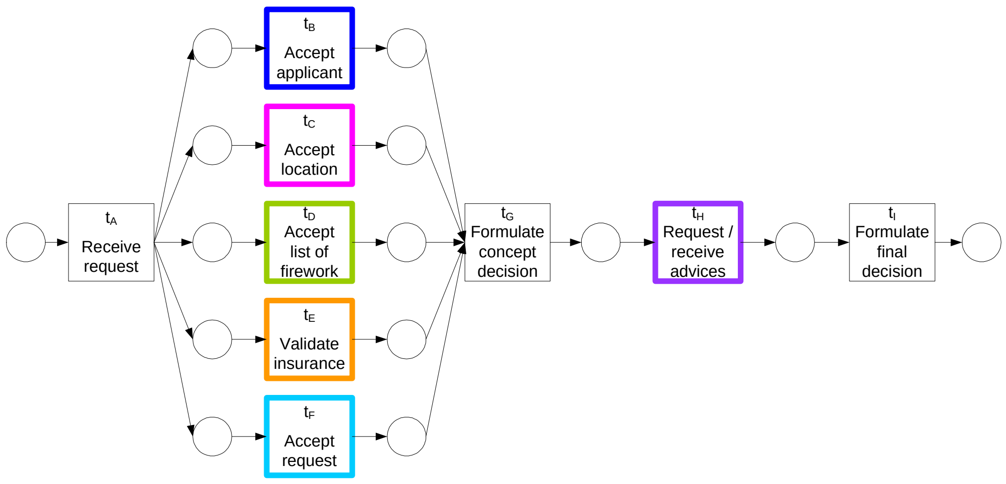

t_B Accept applicant
t_C Accept location
t_A Receive request
t_D Accept list of firework
t_G Formulate concept decision
t_H Request / receive advices
t_I Formulate final decision
t_E Validate insurance
t_F Accept request

Figure 8.22: *The final design for the firework ignition permission process. The colored activities are sub processes. Their content is specified in Figure 8.21 by the same color. Note that the other activities (t_A, t_G, t_I) also cover parts of the process model in Figure 8.21 and the PDM in Figure 8.23, but for reasons of readability these parts are not explicitly indicated.*

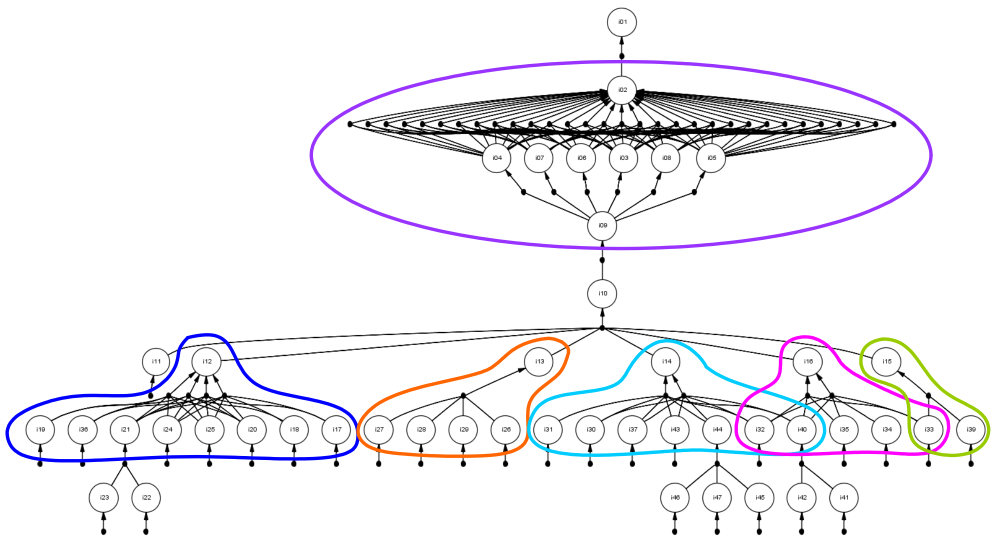


Figure 8.23: *The PDM for the process of granting firework ignition permission at PNB. The grouping of operations and their data elements from the final redesign of Figure 8.22 are indicated by the colors corresponding to the colors of the activities in the final process model.*

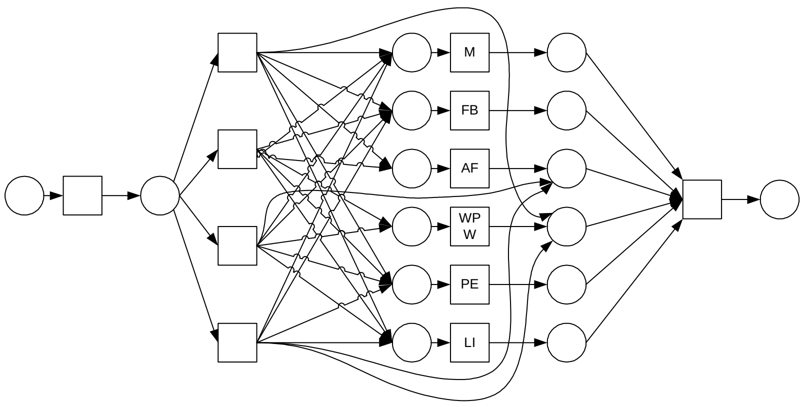


Figure 8.24: *The corrected sub process for activity t_H. The choices between external advisors that occur most frequently are explicitly modeled. For instance, in all cases an advice of the Fire Brigade (FB) is requested, but the advice of the Air Force (AF) and Department of Waterways and Public Works (WPW) is not always needed.*

Design	Process coupling/cohesion ratio (ρ)	Process cohesion (ch)	Process coupling (cp)
1 (Figure 8.22)	11.583	0.036	0.417

Table 8.7: *The values for process coupling and process cohesion metrics, and the process coupling/cohesion ratio for the process model design of the PNB case of Figure 8.22. There are no alternative process model designs that can be compared with this one.*

Solving the issues in the process model of algorithm Charlie requires a good understanding and knowledge of Petri nets and the soundness notion. The solution to the process model of algorithm Echo was considered to be simpler and therefore algorithm Echo was selected. The resulting process model is very detailed. Therefore, some of the activities have been clustered into sub processes. The final process model for the permission process is shown in Figure 8.22. The groups of transitions that form sub processes in the final design are also indicated in the process model generated by the algorithm, cf. Figure 8.21, and in the PDM, cf. Figure 8.23. As an illustration, we have also computed the coupling and cohesion values for this process model design, since each activity is a group of operations. The values for these metrics can be found in Table 8.7.

8.3.4 Conclusions

At the time of writing this thesis, the process redesign was not implemented yet. Therefore, no data is available yet on the performance improvement of the process. However, the feedback on the product driven approach to redesign the process by employees, process experts and other stakeholders is very positive. The analysis of the product has given them important insights about their product, and they have a better overview of all activities related to the allowance of firework ignitions. Moreover, the redesign project team also appreciaed the smooth course of the project. They found that it is easier to communicate with stakeholders about the product they are producing than about the steps the process comprises, because the product is tangible and less ambiguous.

The process model that was developed during this redesign project will be used to support the process with a workflow management system. Based on the positive experiences with the project so far, some of the other 11 Provincial authorities in the Netherlands will join the project and benefit from this redesign. Finally, PNB plans to apply the PBWD method to redesign projects of the other subsidy and permission processes.

In the above three case studies we have applied PBWD to practical redesign cases. In the next section, a different type of evaluation is presented for the CC-metric.

8.4 Empirical Evaluation of the CC-metric

In this section, we report on how the CC-metric has been subjected to a thorough empirical evaluation. First, we describe the evaluation with respect to the metric's capability to predict error probabilities in process models. In the next section, we comment on its suitability to explain which process models are easier to understand than others. For both evaluations the same set of test data is used as in two previous studies on business process metrics and error prediction [175, 177, 179] and business process metrics and understandability [178]. This way, the validation of the CC-metric can be compared to that of other metrics. In both empirical evaluations first a correlation analysis is performed (using Spearman's rank correlation coefficient[3] [264]). In a second step, a logistic regression analysis[4] is done.

8.4.1 Validation for Error Prediction

An indication for a metric's predictive power is that it can accurately distinguish between models with errors and without errors. Because this evaluation uses a large set of EPCs, we use the EPC soundness criterion as defined in [176] for determining whether a model has errors or not and assume that a decrease in CC is likely to result in more errors. Therefore, our hypothesis is:

H1: A decrease in CC implies an increase in error probability.

To evaluate this hypothesis, the set of EPC models of the *SAP Reference Model* is used. The development of the SAP reference model started in 1992 and first models were presented at CEBIT'93 [147, p.VII]. Since then, it was developed further until version 4.6 of SAP R/3, which was released in 2000. The SAP reference model includes 604 non-trivial EPCs. The advantage of considering this set of models is that there is extensive literature available that explains its creation, e.g. [147]. Furthermore, it is frequently cited in research papers as a typical reference model and has been used in previous quantitative analyses, as e.g. reported in [175, 177, 179]. This way, our results can be compared to these related works.

The results of the evaluation of 28 different metrics can be summarized as follows [175]:

[3]In contrast to e.g. Pearson's product-moment correlation coefficient, Spearman's rank correlation coefficient is a non-parametric measure of correlation because it does not assume that the population fits any parametrized distribution such as the normal distribution [70, 260].

[4]Regression analysis is a statistical technique that can be used to analyze the relationship between a single dependent variable and several independent variables. Logistic regression is a special form of regression in which the dependent variable is a binary variable [119].

· The Spearman rank correlation coefficient shows that all metrics have a correlation with the occurrence of errors in the models, e.g. the CFC metric has a positive correlation of 0.39 with the error probability, and the STRUCTUREDNESS metric has a negative correlation of -0.36. The significance of all correlations is good with more than 99% confidence.

· The multivariate logistic regression model includes seven of the 28 metrics (COEFFICIENT OF CONNECTIVITY, CONNECTOR MISMATCH, CYCLICITY, SEPARABILITY, STRUCTUREDNESS, CONNECTOR HETEROGENITY, and DIAMETER) as parameters.

For the validation of the CC-metric we use the same set of EPC models to test the metric's capability of error prediction. As a first step, we again execute a *correlation analysis*. In particular, we investigate to what extent the CC-metric is capable of ranking non-error and error models. This capability can be estimated using the rank correlation coefficient ρ by Spearman [264][5]. For CC it is -0.434. For this metric there is a strong and 99% significant correlation with a negative direction, which matches the expectation of the hypothesis, i.e. H1 holds.

In a second step, we also use *multivariate logistic regression* to develop an estimation model to predict errors in process models based on their values for the business process metrics. This approach estimates the coefficients of a linear combination of input parameters for predicting event versus non-event based on a logistic function. In our case, we predict error versus non-error for the EPCs in the SAP reference model based on the CC-metric and a constant. The accuracy of the estimated model is assessed based on the significance level of the estimated coefficients, the percentage of cases that are classified correctly, and the share of the variation that is explained by the regression. This share is typically measured using the Nagelkerke R^2 ranging from 0 to 1 (1 being the best possible value). The estimated coefficient should have a significance that is below 5% indicating that it is significantly different from zero. For technical details of logistic regression we refer to [131]. For applications in predicting errors in process models see [175, 177, 179].

A univariate logistic regression for CC was calculated first. Table 8.8 shows that CC alone already yields a high Nagelkerke R^2 of 0.586. The negative coefficient matches the expectation of hypothesis H1. Furthermore, we stepwise introduced other metrics to the model. We used those metrics that were found in [177] as the best combination to predict errors in EPCs. In this context, it is interesting to note that adding these metrics yields quite similar coefficients for them as in the predicting function of [177]. This suggests that the CC-metric indeed measures a process model aspect that is orthogonal to metrics that have been defined before.

8.4.2 Validation for Understandability

Besides the evaluation of the CC-metric for error prediction, we also look at its capability to explain understandability of process models. We assume that a model with a high CC-value is better understandable since the parts of the model are tightly coupled to each other. Therefore, our hypothesis is that:

H2: An increase in CC implies an increase in understandability.

To evaluate the capability of the CC-metric to explain which process models are easier to understand than others, we used the empirical data described in [178]. This data was obtained in a project that aims at the analysis of the impact of both model and personal characteristics on

[5] $-1 \leq \rho \leq 1$. A negative ρ indicates a negative correlation; a positive ρ indicates a positive correlation. If $\rho = 0$ there is no correlation between the data sets. If $\rho = -1$ or $\rho = 1$ there is complete correlation between the data sets.

Step	Parameter	Regression Coefficient	Standard Error	Wald	Significance	Nagelkerke R^2	Classification
1	CC	-13.813	1.229	126.386	0.000	0.586	0.791
...	...	...	...	...	...	...	...
5	CC	-10.478	2.931	12.783	0.000	0.847	0.916
	Structuredness	-9.500	1.028	85.328	0.000		
	Diameter	0.139	0.032	18.829	0.000		
	Cyclicity	6.237	1.857	11.281	0.001		
	CNC	5.541	0.935	35.145	0.000		

Table 8.8: *Multivariate Logistic Regression Models which include the CC-metric. The* parameter *column describes the business process metrics that are included in the logistic regression model. The* coefficient *is the regression coefficient used in the logistic regression function for each of the metrics. Note that the sign of the coefficient should correspond to the direction of the correlation of the parameter and the dependent variable. The* standard error *indicates the standard error of the regression coefficient. It denotes the expected range of the coefficient across multiple samples of the data. The* Wald *statistic is a way to test the significance of the estimated regression coefficients. It indicates the impact of the parameter in the logistic regression model. The* significance *indicates whether there is a significant correlation of the separate parameter with the regression result. The value of significance should be less than 0.05 at a 95% confidence to indicate a significant correlation. Only significant parameters are included in the regression model.* Nagelkerke's R^2 *[193] value is used to assess the precision of the logistic regression model. It ranges from 0 to 1 and indicates which fraction of the variability in the dependent variable is explained by the parameters. Thus, the higher* R^2 *the better the regression model 'fits'. Finally, the* classification *gives the fraction of the cases that are correctly classified.*

the understandability of process models. In particular, a set of 20 model characteristics were investigated, which have been proposed and formally defined in [175]. Among these are, for example, the DENSITY and STRUCTUREDNESS of a process model, which respectively relate to (i) the number of arcs in the model relative to the theoretical maximum of a fully connected model and (ii) the extent to which a process model is built by nesting blocks of matching join and split routing elements.

In total, 73 students filled out a questionnaire in the fall of 2006. A set of 12 process models from practice, each having the same number of tasks (25), formed the basis of the questionnaire. In addition, for each of these models two slightly different variants were constructed by changing the type of some routing elements (e.g. a particular XOR-split in a AND-split), which led to a total of 12 models that had to be evaluated by the students. Inspired by a survey on the relative complexity of different modelling techniques [251], it was decided to use an EPC-like notation in our questionnaire to minimize the impact of the notation on understandability.

As part of the models' evaluation, students were asked to answer questions like "If task K is executed for a case, can task L be executed for the same case?" and "Can tasks G, O, and P all be executed for the same case?" Apart from closed questions, an additional open question was included as well: "Is there any problem with this process (e.g. proper completion, deadlock, etc.)?" As a follow up to the latter question, students were asked to elaborate and specify the perceived problem, if any. The correct answers for the questions were determined with the EPC analysis tools introduced in [176]. While the closed answers were evaluated automatically, the open answers had to be interpreted and matched with the errors detected by the tools.

The evaluation of the 12 models by the 73 students led to a total of 847 complete model evaluations. On this basis, a SCORE variable could be calculated per model as the mean sum of correct answers it received. This SCORE variable served as a way to make understandability operational. For the 12 models under consideration, the SCORE ranged between 5.5 to 7.4 on a scale of 0 to 9.

From the earlier analysis of these results [178], the following main conclusions were drawn with respect to model characteristics:

- Of the 20 factors considered, five model factors exhibited the hypothesized relation with SCORE, i.e. (1) #OR-JOINS, (2) DENSITY, (3) AVERAGE CONNECTOR DEGREE, (4) MISMATCH, and (5) CONNECTOR HETEROGENEITY. Of these five model characteristics, only the correlations between DENSITY (the ratio between the actual number of arcs and the theoretical maximal number of arcs) and SCORE (-0.618) and between AVERAGE CONNECTOR DEGREE (the average number of input and output arcs of the routing elements in a model) and SCORE (-0.674) correlated significantly, with respective P-values of 0.032 and 0.016 (i.e. < 0.05).
- Of all linear regression models on the basis of a combination of these five model factors, the regression model that only used AVERAGE CONNECTOR DEGREE displayed the best explanatory power for the variability in SCORE, with an adjusted R^2=45% (Nagelkerke's coefficient of determination).

To evaluate the CC-metric, it was incorporated in the above analysis. Based on the correlation and logistic regression analysis, we arrive at the following conclusions:

- Just like the five model factors that emerged from the original analysis, the CC-metric displays the expected positive correlation with SCORE.

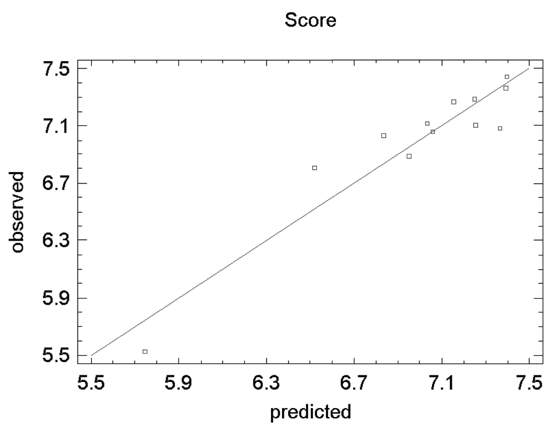


Figure 8.25: *Linear regression model explaining the mean* SCORE *for the 12 process models.*

- · Unlike the density and average connector degree factors, the correlation between SCORE and the CC-metric (0.549) is *not* significant at a 95% confidence interval as the P-value of 0.065 slightly exceeds the 0.05 confidence interval.
- · A regression model with a *much better* explanatory power for the variation in SCORE could be developed by *including* the CC-metric: the adjusted R^2 of the original model increased from 45% to 76% in the new regression model. In particular, by combining the #OR-JOINS, DENSITY, AVERAGE CONNECTOR DEGREE, MISMATCH factors and the CC-metric this result could be achieved. A visualization of this regression model can be seen in Figure 8.25.

What this analysis suggests is that CC on its own is slightly less powerful as an indicator for process model understandability than the two best candidate metrics available, but it may deliver a better explanation of the variation in understandability across models when combined with existing metrics.

8.5 Summary

The three case studies presented in this chapter show how the ideas presented in this thesis can be applied in redesign projects. The case of the annual reporting process at ING IM shows the success of a PBWD approach by a reduction of the throughput time by 50% and a substantial increase in quality. Secondly, the student grants example shows how the cohesion and coupling metrics can be used to decide on which alternative process model is the best. Finally, in the case study of the firework ignition permission process at PNB, an application of the algorithms presented in Chapter 5 has been shown.

In addition to these case studies, also an empirical evaluation of the CC-metric has been presented based on the SAP reference model. It has been tested how powerful the metric is to predict errors and understandability of process models. The evaluation shows the validity of the CC-metric as a tool for the design of (better) process models.

Chapter 9

Conclusion

This chapter concludes this thesis by providing an overview of the main contributions, a discussion of the limitations of the work presented, and an outlook on future work. The focus of the research described in this thesis has been on the development of tools for product-based workflow design and support. The main goal in this research was to develop 'intelligent' tools that provide design guidance. A number of contributions to the field are described in this thesis. First of all, we have provided a *formal definition* for a PDM and a *basic notion of correctness* of a process model with respect to a given PDM. Secondly, we have developed seven algorithms for *generating process models* from a PDM. In Chapter 6, we have focused on the *direct execution* of a PDM. The direct execution of a PDM makes the design of a separate process model superfluous. Next, we have introduced two sets of *business process metrics* that facilitate the evaluation and comparison of alternative process models for the same process. Finally, throughout the thesis we have shown the feasibility of our ideas by developing *tool support* within the ProM Framework. After a summary of these contributions and their limitations, this chapter provides a general discussion of the PBWD method and the challenges for future work.

9.1 Correctness of Process Models

The first contribution made in this thesis is a formal definition of a PDM, and a basic notion of correctness for process models that are designed based on a PDM. The manual design of a process model involves the grouping of operations and data elements of the PDM into activities. Each of the activities has a set of input and a set of output elements. Based on the formal definition of a PDM we have defined a basic notion of correctness for a process model with respect to the given PDM. The process model should satisfy three requirements: (i) it should contain a way to produce each data element in the PDM, (ii) it should respect the data dependencies in the PDM, and (iii) it should never have a possibility to execute an operation more than once in a path from the start to the end of the process.

This basic notion of correctness may help a process designer to evaluate the process model that was (manually) designed based on the PDM. After the process model has been designed, it can be assessed on the three basic correctness requirements. If one of these requirements is not satisfied, there is a problem in the data flow of the process model. Thus, the correctness notion supports the evaluation phase of a PBWD project that is executed in the design phase of a BPR program.

As we have seen in Chapter 5, our basic notion of correctness does not yet distinguish between alternative input data sets for an activity. Therefore, the activity dependencies that are derived from the input/output data dependencies may be too restrictive. Process models with an activity that contains several alternative ways to produce the value of the same data element, may not satisfy the second correctness requirement, while actually no input/output dependencies are violated if only one of the alternative ways is chosen. However, the identification of alternative input data sets based on the formal definition of a PDM is not straightforward. Besides that, it is questionable whether it is a good idea to combine alternative operations in one activity. Alternative operations are often used to model situations in which a certain condition on the value of a data element is not satisfied. It is not very natural to combine all these alternative operations in one activity, since they each represent alternative ways to terminate the process.

9.2 Generating Process Models from a PDM

In Chapter 5, we have introduced seven algorithms that automatically generate process models from a given PDM. This is an approach that fits well with current workflow technology since the generated process model forms the basis for the use of an existing workflow system to support the business process. The presented algorithms generate process models with different properties and can be classified based on a number of characteristics. These characteristics focus on two different perspectives. On the one hand one can adopt the perspective of the *construction* of a process model from the PDM. On the other hand, the *process execution* perspective is distinguished, which focuses on the characteristics of the process model that is generated. The classification framework describes the similarities and differences between the algorithms.

The seven algorithms may help a process designer to create a process model based on a PDM in the design phase of a PBWD project. However, since each of the algorithms has some pitfalls, the generated model should be seen as an initial design that needs to be further refined by the designer. The algorithms describe some straightforward approaches to translate a PDM and are purely based on the dependencies in the PDM. The resulting process models may be adapted using additional information on e.g. failure probabilities or data element values. Or it may be used for communication with stakeholders. The classification framework may provide some first guidance in the selection of one of the seven algorithms for a certain situation.

All of the algorithms generate process models which may have some issues, e.g. related to soundness (algorithms Charlie and Echo), the occurrence of duplicate activities (algorithm Alpha and Bravo), or readability (algorithms Delta, Foxtrot and Golf). The process designer who uses the algorithms, therefore, should have quite some technical and theoretical knowledge on process modeling to make a proper selection among the different algorithms and should be able to resolve soundness problems, for instance.

Another limitation of the algorithms is that they produce process models that have the same level of detail as the PDM. The algorithms do not automatically group operations into activities. Thus, the activities in the process model are as detailed as the operations in the PDM. This may lead to a process model that is overly detailed. The activities then still have to be clustered into sub processes to form logical pieces of work for the persons executing the process. The PNB case in Chapter 8 illustrates the manual determination of these sub processes. There is no concrete guidance available yet to identify groups of activities as sub processes (or to identify groups of operations and their data elements in the PDM). However, it is fairly straightforward to implement a 'brute force' algorithm that computes all possible groupings of operations and selects the best alternative, e.g. based on the cohesion and coupling metrics presented in Chapter 7.

Finally, the existing guidance for selecting the most suitable algorithm in a specific situation is purely based on the technical characteristics in the classification framework. A direction for future work is the identification of higher level characteristics that may help a less experienced designer to choose between algorithms without knowing their technical characteristics in detail. One could for instance think of a question like: Is a short throughput time more important than low execution cost? If so, an algorithm that generates process models which allow for concurrent executions and the deferring of choices may be preferred. Another question could be: Is the process model needed for communication? If so, the process model should be readable and its size should be limited. Also, the animation of an execution of the PDM according to a specific process model may give insight in the characteristics of the different process models. Future work may focus on these aspects.

9.3 Direct Execution of a PDM

A third contribution of this thesis is the support for the *direct execution of a PDM* developed in Chapter 6. For the direct execution of a PDM no process model is derived and the problems identified in Chapter 5 can easily be circumvented. Based on the data element values that are available for a case, it is calculated which operations are executable. Typically, more than one operation can be executed for a single case. To decide which of these executable operations is best as a next step to be executed, we have introduced several selection strategies.

The first set of strategies are simple selection strategies that focus on the optimal decision within the set of executable operations, e.g. the operation with the lowest cost, shortest duration, or lowest failure probability, is selected. These simple selection strategies can easily be computed at runtime and allow for switching between strategies during the execution of a case. However, they do not take into account the effect a decision may have on future steps in the process and therefore may not provide the best overall decision strategy.

The second approach to select a next step is capable of dealing with global optimization of the selection by using Markov decision strategies. The PDM is translated to a Markov Decision Process (MDP). The solution of this MDP is a strategy that provides a best overall decision for any set of available data elements. A limitation of the use of Markov decision strategies is that the computation of the strategy is rather time consuming and may be intractable. Therefore, it is impractical to define and solve a new MDP at each stage of the execution of a case. For large PDMs it may even take too much time to compute the decision strategy at design time. To overcome this problem, we have identified a number of possibilities to restrict the size of the MDP state space. Using these restrictions, the presented decision strategy may be near-optimal since a number of states may not have been considered.

The direct execution of a PDM is more dynamic and flexible than the execution of a workflow process based on a process model. Based on the available data element values for a specific case, it can be dynamically decided which operation in the PDM is executed next. Moreover, a case that has been partially executed can be easily transferred to a modified product description. The available data element values then determine the state of the case under the new description. This allows for more flexibility.

The direct execution of a PDM is a way to directly enact the workflow process without deriving a process model first. Thus, in the process design phase of the BPR life cycle, the process designer only has to design a PDM. This PDM can then be used directly for the enactment of the process in a workflow management system (cf. the process enactment phase of the BPR life cycle). Moreover, such a system is 'intelligent' because it provides execution recommendations that guide the user through the execution of the process.

A limitation of our approach for the direct execution of a PDM is that we do not consider parallel executions of operations. The concurrent execution of operations would lead to difficulties stemming from different values for the same data element, as described in Chapter 4. In general, these data updates are an issue for the use of PDMs (see Section 9.6) and other data-driven approaches. Note that current data-driven workflow management systems, e.g. the case-handling system FLOWer, prevent parallel executions for the same case by blocking the whole case as soon as someone is working on it.

Finally, in our approach to the direct execution of a PDM, we focus on the optimization of a single case. The best decisions are defined in terms of the performance objective on the case level, e.g. the minimization of throughput time or cost for a single case. Optimization on the process level with respect to e.g. the utilization of the process, or the optimal distribution of work among resources, is not considered yet. Kress et al. [152, 153] propose an approach to optimize the whole process with an agent based system. However, they do not consider the performance of a single case. Future work could therefore focus on combining these two optimization perspectives.

The direct execution of a PDM and the generation of process models based on the PDM are rather different ways to enact a workflow process in a product-based fashion. Upon the generation of a process model, current workflow technology can be deployed to support the workflow process. However, we conclude that the automatic generation of (sound and correct) process models is far from straightforward. Chapter 5 has given some insights in the shortcomings of a number of natural and straightforward algorithms that derive process models purely based on the structure of a PDM. These shortcomings have been the motivation for the development of support for the direct execution of the PDM. This direct execution is more flexible and dynamic and ensures a correct execution. However, there is no connection with existing (commercial) tools, systems and approaches to support workflow processes yet, except for our prototype presented in Section 6.6. This makes it difficult to use this approach in practical situations at this moment.

9.4 Business Process Metrics

Another contribution to the development of tools for PBWD guidance, is the definition of two sets of business process metrics. Business process metrics can be used to evaluate and compare process models in the evaluation phase of a PBWD project that is executed in the design phase of the BPR life cycle (see sections 1.1 and 1.4). They provide guidance to select the best process design among several alternatives.

The first set of business process metrics we have introduced, cohesion and coupling metrics, focuses on the content of activities and the links between activities in terms of data. The content of an activity is defined by operations and their data elements in the PDM. Also, the links between activities are formed by input/output data relations. We have adopted the design paradigm of *high cohesion - loose coupling* from the area of software metrics and defined cohesion and coupling metrics to measure the cohesiveness of the data elements and operations within an activity and the coupling based on data elements that are shared between activities. Activities should be as cohesive as possible, representing logical pieces of work. The coupling between activities should not be too strong since this may lead to many work handovers and possibly more errors. This first set of metrics has been inspired by existing software metrics [257, 306]. Moreover, our coupling metric has been adopted by other researchers [248] to provide guidance for the identification of process components in object-centric business process implementations.

Besides the adoption of software metrics, the development of new business process metrics that capture other characteristics of a process model which cannot be measured by software metrics is also emerging. The second set of metrics (Cross-Connectivity)we have presented in this thesis has been inspired by cognitive theory and focuses on the cognitive effort that is needed to understand a process model. These metrics do not consider the data perspective or content of activities in a process model, but focus on the tightness of the links between the elements in the process model. It is assumed that a well-structured and tightly connected process model is easier to understand and contains fewer errors than an unstructured process model. Because these metrics do not focus on the data perspective, they can also be used for process models that are not designed based on a PDM.

The empirical evaluations of business process metrics that have been carried out have focused on the capability of a specific metric as an error predictor and a predictor for understandability of process models. There are only a few practical applications of business process metrics in actual redesign processes (e.g. the student grants example of Section 8.2 where the cohesion and coupling metrics are used to decide on the best alternative design). To make the business process metrics more attractive and easier to use, some benchmark values should be defined for each metric. At this moment we can only state that e.g. a high CC-value is better than a low CC-value, but we are not able to give minimum or preferred levels for the CC-metric yet. Future work should focus on this.

9.5 Tool Support

The final contribution of this thesis is the development of a prototype for each of the contributions described above. As we have seen in Section 3.4, current workflow technology is not yet ready for product-based workflow design and support. Therefore, we have developed tool support in the ProM Framework to show the feasibility of our ideas. However, the implementation of our ideas in the ProM Framework should be seen as a prototype and proof-of-concept, although it has links with many actual systems via the import and export of process models in various modeling languages. A next step towards ‘intelligent’ tool support for the PBWD methodology would be the incorporation of one or more of our approaches in (commercial) workflow management systems.

9.6 Product Based Workflow Design

Adopting a broader perspective, PBWD has been proven to be a successful BPR methodology with a grat potential to improve the business process, e.g. shorter throughput times and higher quality levels. Its practical use is not very widespread yet, partly because of the lack of tool support for this method. We believe that the tools and approaches presented in this thesis provide support and guidance for process designers and we hope they will facilitate a broader use of the PBWD method.

However, there are still some issues that need to be addressed. In this thesis, we assume that data element values are available for the total duration of the case once they have been determined and that they cannot be overwritten by another operation producing a possibly different value for the same data element (cf. Assumption 4.3.4). A relaxation of this assumption that allows for updates of data elements would be realistic, but would also introduce a number of issues related to versioning and rollbacks. An interesting direction for future work would be to investigate the impact of allowing updates to data element values and to develop a way to deal with these updates. Perhaps this would also enable the use of ‘data element lifecycles’, which would link the PDM to the work of Müller et al. [186, 187, 188] and Küster et al. [157, 248, 249, 250] (see also Section 3.5).

Moreover, as we have seen in the ING IM case, which is described in Section 8.1, it may be necessary that the outcome of certain activities in a process model is checked (e.g. because this check is required by law). The PDM does not distinguish between these kind of operations. Therefore, these check activities are not automatically provided by the generated process model and should be manually added. This would be another direction for future research.

Last but not least, the resource perspective has not received a lot of attention yet, although we have extended the definition of a PDM with the possibility to add resource information. Resource information is neither used in the generation and evaluation of process models, nor in the direct execution of a PDM. Since resource availability also is important in the execution of a process, we feel this is an important issue for future work. Kress et al. [152, 153] have made a first attempt to develop a system for the direct execution of a PDM which optimizes the workload and utilization of the whole system, including the resources. Unfortunately, by considering on the whole system, they loose the focus on the optimization of the case. The combination of the two perspectives, i.e. optimization of the case and optimization of the process, may be an interesting direction for future developments.

9.7 Summary

In this thesis, we have presented a number of approaches for the development of ‘intelligent’ tools for the product-based design and support of workflow processes. The basic notion of correctness of a process model with respect to the PDM, the seven algorithms to derive process models from a PDM, and the business process metrics, may help process designers in the design phase of the BPR life cycle to create process models based on a product description. The direct execution of a PDM provides a means to directly deploy the PDM and therefore covers the process implementation and process enactment phase of the BPR life cycle. Each of these approaches has been shown to be feasible by the development of appropriate tool support in the ProM Framework. The contributions in this thesis provide a good foundation for dealing with further research challenges to bring the PBWD methodology in practice.

Appendix A

XML Schemes of Example PDMs

A.1 XML Schema Definition for a PDM

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="PDM">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Name"/>
                <xs:element ref="DataElement" maxOccurs="unbounded"/>
                <xs:element ref="Resource" maxOccurs="unbounded"/>
                <xs:element ref="Operation" maxOccurs="unbounded"/>
                <xs:element ref="RootElementRef"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="DataElement">
        <xs:complexType>
            <xs:sequence minOccurs="0">
                <xs:element ref="Data" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="DataElementID" type="xs:ID" use="required"/>
            <xs:attribute name="Description" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Resource">
        <xs:complexType>
            <xs:attribute name="ResourceID" type="xs:ID" use="required"/>
            <xs:attribute name="Description" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Operation">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Input">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="DataElementRef" type="xs:IDREF"
                                        minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Output">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="DataElementRef" type="xs:IDREF"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Cost" minOccurs="0"/>
                <xs:element name="Time" minOccurs="0"/>
                <xs:element name="Condition" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="Probability" minOccurs="0"/>
                <xs:element name="ResourceRef" type="xs:IDREF" minOccurs="0"/>
                <xs:element ref="Data" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="OperationID" type="xs:ID" use="required"/>
            <xs:attribute name="Descriptioin" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Data"/>
    <xs:element name="RootElementRef" type="xs:IDREF"/>
    <xs:element name="Name"/>
</xs:schema>
```

A.2 Example 1: Mortgage

```
<?xml version="1.0" encoding="UTF-8"?>
<PDM xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=
    "D:\My Documents\Implementatie\XML examples\Pdm.xsd">
    <Name>Mortgage</Name>
    <DataElement DataElementID="A"/>
    <DataElement DataElementID="B"/>
    <DataElement DataElementID="C"/>
    <DataElement DataElementID="D"/>
    <DataElement DataElementID="E"/>
    <DataElement DataElementID="F"/>
    <DataElement DataElementID="G"/>
    <DataElement DataElementID="H"/>
    <Resource ResourceID="w1"/>
    <Operation OperationID="Op01">
        <Input>
            <DataElementRef>B</DataElementRef>
            <DataElementRef>C</DataElementRef>
            <DataElementRef>D</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>A</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>5</Cost>
        <Time>1</Time>
        <Probability>0.05</Probability>
    </Operation>
    <Operation OperationID="Op02">
        <Input>
            <DataElementRef>F</DataElementRef>
            <DataElementRef>G</DataElementRef>
            <DataElementRef>H</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>C</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>5</Cost>
        <Time>4</Time>
        <Probability>0.05</Probability>
    </Operation>
    <Operation OperationID="Op03">
        <Input>
            <DataElementRef>H</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>A</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>H=negative</Condition>
        <Cost>9</Cost>
        <Time>3</Time>
        <Probability>0.05</Probability>
    </Operation>
    <Operation OperationID="Op04">
        <Input>
            <DataElementRef>E</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>A</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>2</Cost>
        <Time>2</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op05">
        <Input/>
        <Output>
            <DataElementRef>B</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op06">
        <Input/>
        <Output>
            <DataElementRef>D</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op07">
```

```
        <Input/>
        <Output>
            <DataElementRef>E</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>1</Cost>
        <Time>1</Time>
        <Probability>0.5</Probability>
    </Operation>
    <Operation OperationID="Op08">
        <Input/>
        <Output>
            <DataElementRef>F</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op09">
        <Input/>
        <Output>
            <DataElementRef>G</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>0</Cost>
        <Time>2</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op10">
        <Input/>
        <Output>
            <DataElementRef>H</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Condition>true</Condition>
        <Cost>3</Cost>
        <Time>10</Time>
        <Probability>0.15</Probability>
    </Operation>
    <RootElementRef>A</RootElementRef>
</PDM>
```

A.3 Example 2: Naturalization

```
<?xml version="1.0" encoding="UTF-8"?>
<PDM xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=
    "D:\My Documents\Implementatie\XML examples\Pdm.xsd">
    <Name>Naturalization to Dutch citizen</Name>
    <DataElement DataElementID="A"/>
    <DataElement DataElementID="B"/>
    <DataElement DataElementID="C"/>
    <DataElement DataElementID="D"/>
    <DataElement DataElementID="E"/>
    <DataElement DataElementID="F"/>
    <DataElement DataElementID="G"/>
    <Resource ResourceID="w"/>
    <Operation OperationID="Op01">
        <Input>
            <DataElementRef>B</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>A</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op02">
        <Input>
            <DataElementRef>D</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>A</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op03">
        <Input>
            <DataElementRef>C</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>A</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op04">
        <Input>
            <DataElementRef>E</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>A</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op05">
        <Input>
            <DataElementRef>C</DataElementRef>
            <DataElementRef>D</DataElementRef>
            <DataElementRef>B</DataElementRef>
            <DataElementRef>E</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>A</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op06">
        <Input/>
        <Output>
            <DataElementRef>D</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op07">
        <Input>
            <DataElementRef>G</DataElementRef>
            <DataElementRef>F</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>E</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op08">
        <Input/>
        <Output>
            <DataElementRef>B</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op09">
        <Input/>
        <Output>
            <DataElementRef>C</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op10">
        <Input/>
        <Output>
            <DataElementRef>F</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <Operation OperationID="Op11">
        <Input/>
        <Output>
            <DataElementRef>G</DataElementRef>
        </Output>
        <ResourceRef>w</ResourceRef>
    </Operation>
    <RootElementRef>A</RootElementRef>
</PDM>
```

A.4 Example 3: Unemployment benefits

```
<?xml version="1.0" encoding="UTF-8"?>
<PDM xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=
    "D:\My Documents\Implementatie\XML examples\Pdm.xsd">
    <Name>GAK</Name>
    <DataElement DataElementID="i01"/>
    <DataElement DataElementID="i02"/>
    <DataElement DataElementID="i03"/>
    <DataElement DataElementID="i04"/>
    <DataElement DataElementID="i05"/>
    <DataElement DataElementID="i06"/>
    <DataElement DataElementID="i07"/>
    <DataElement DataElementID="i08"/>
    <DataElement DataElementID="i09"/>
    <DataElement DataElementID="i10"/>
    <DataElement DataElementID="i11"/>
    <DataElement DataElementID="i13"/>
    <DataElement DataElementID="i14"/>
    <DataElement DataElementID="i15"/>
    <DataElement DataElementID="i16"/>
    <DataElement DataElementID="i17"/>
    <DataElement DataElementID="i18"/>
    <DataElement DataElementID="i21"/>
    <DataElement DataElementID="i23"/>
    <DataElement DataElementID="i24"/>
    <DataElement DataElementID="i25"/>
    <DataElement DataElementID="i27"/>
    <DataElement DataElementID="i28"/>
    <DataElement DataElementID="i29"/>
    <DataElement DataElementID="i30"/>
    <DataElement DataElementID="i31"/>
    <DataElement DataElementID="i32"/>
    <DataElement DataElementID="i33"/>
    <DataElement DataElementID="i34"/>
    <DataElement DataElementID="i35"/>
    <DataElement DataElementID="i36"/>
    <DataElement DataElementID="i37"/>
    <DataElement DataElementID="i39"/>
    <DataElement DataElementID="i40"/>
    <DataElement DataElementID="i41"/>
    <DataElement DataElementID="i42"/>
    <DataElement DataElementID="i43"/>
    <DataElement DataElementID="i44"/>
    <DataElement DataElementID="i45"/>
    <DataElement DataElementID="i47"/>
    <DataElement DataElementID="i48"/>
    <DataElement DataElementID="i49"/>
    <Resource ResourceID="w1"/>
    <Operation OperationID="Op01">
        <Input>
            <DataElementRef>i25</DataElementRef>
            <DataElementRef>i37</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i01</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op02">
        <Input>
            <DataElementRef>i25</DataElementRef>
            <DataElementRef>i37</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i02</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op03">
        <Input>
            <DataElementRef>i33</DataElementRef>
            <DataElementRef>i37</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i03</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op04">
        <Input>
            <DataElementRef>i33</DataElementRef>
            <DataElementRef>i37</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i04</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op05">
        <Input>
            <DataElementRef>i37</DataElementRef>
            <DataElementRef>i45</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i05</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op06">
        <Input>
            <DataElementRef>i21</DataElementRef>
            <DataElementRef>i37</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i06</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op07">
        <Input>
            <DataElementRef>i24</DataElementRef>
            <DataElementRef>i37</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i07</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op08">
```

```
        <Input>
            <DataElementRef>i23</DataElementRef>
            <DataElementRef>i37</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i08</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op09">
        <Input>
            <DataElementRef>i24</DataElementRef>
            <DataElementRef>i39</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i09</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op10">
        <Input>
            <DataElementRef>i13</DataElementRef>
            <DataElementRef>i14</DataElementRef>
            <DataElementRef>i34</DataElementRef>
            <DataElementRef>i37</DataElementRef>
            <DataElementRef>i42</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i10</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op11">
        <Input>
            <DataElementRef>i31</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i11</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>60</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op12">
        <Input>
            <DataElementRef>i16</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i15</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op13">
        <Input>
            <DataElementRef>i17</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i15</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op14">
        <Input>
            <DataElementRef>i16</DataElementRef>
            <DataElementRef>i17</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i15</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op15">
        <Input>
            <DataElementRef>i25</DataElementRef>
            <DataElementRef>i30</DataElementRef>
            <DataElementRef>i35</DataElementRef>
            <DataElementRef>i36</DataElementRef>
            <DataElementRef>i44</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i16</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>561</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op16">
        <Input>
            <DataElementRef>i25</DataElementRef>
            <DataElementRef>i30</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i17</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op17">
        <Input>
            <DataElementRef>i01</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i18</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op18">
        <Input>
            <DataElementRef>i02</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i18</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op19">
        <Input>
            <DataElementRef>i08</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i18</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op20">
        <Input>
            <DataElementRef>i09</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i18</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op21">
        <Input>
            <DataElementRef>i10</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i18</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op22">
        <Input>
            <DataElementRef>i11</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i18</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op23">
        <Input>
            <DataElementRef>i15</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i18</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op24">
        <Input>
            <DataElementRef>i09</DataElementRef>
            <DataElementRef>i11</DataElementRef>
            <DataElementRef>i15</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i18</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op25">
        <Input>
            <DataElementRef>i25</DataElementRef>
            <DataElementRef>i37</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i28</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op26">
        <Input>
            <DataElementRef>i25</DataElementRef>
            <DataElementRef>i30</DataElementRef>
            <DataElementRef>i35</DataElementRef>
            <DataElementRef>i36</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i29</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op27">
        <Input>
            <DataElementRef>i32</DataElementRef>
            <DataElementRef>i37</DataElementRef>
            <DataElementRef>i43</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i30</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op28">
        <Input>
            <DataElementRef>i29</DataElementRef>
            <DataElementRef>i40</DataElementRef>
            <DataElementRef>i48</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i31</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op29">
        <Input>
            <DataElementRef>i01</DataElementRef>
            <DataElementRef>i02</DataElementRef>
            <DataElementRef>i03</DataElementRef>
            <DataElementRef>i04</DataElementRef>
            <DataElementRef>i05</DataElementRef>
            <DataElementRef>i06</DataElementRef>
            <DataElementRef>i07</DataElementRef>
            <DataElementRef>i08</DataElementRef>
            <DataElementRef>i10</DataElementRef>
            <DataElementRef>i27</DataElementRef>
            <DataElementRef>i28</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i32</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op30">
        <Input>
            <DataElementRef>i36</DataElementRef>
            <DataElementRef>i37</DataElementRef>
            <DataElementRef>i41</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i34</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>420</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op31">
        <Input>
            <DataElementRef>i39</DataElementRef>
            <DataElementRef>i41</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i40</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>30</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op32">
        <Input>
            <DataElementRef>i47</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i42</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>30</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O34">
        <Input/>
        <Output>
            <DataElementRef>i13</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>60</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O35">
        <Input/>
        <Output>
            <DataElementRef>i14</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>8</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O36">
        <Input/>
        <Output>
            <DataElementRef>i21</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O37">
        <Input/>
        <Output>
            <DataElementRef>i23</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>67</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O38">
        <Input/>
        <Output>
            <DataElementRef>i24</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O39">
        <Input/>
        <Output>
            <DataElementRef>i25</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O40">
        <Input/>
        <Output>
            <DataElementRef>i27</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>8</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O41">
        <Input/>
        <Output>
            <DataElementRef>i33</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O42">
        <Input/>
        <Output>
            <DataElementRef>i35</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O43">
        <Input/>
        <Output>
```

```
            <DataElementRef>i36</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>100</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O44">
        <Input/>
        <Output>
            <DataElementRef>i37</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>167</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O45">
        <Input/>
        <Output>
            <DataElementRef>i39</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>17</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O46">
        <Input/>
        <Output>
            <DataElementRef>i41</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O47">
        <Input/>
        <Output>
            <DataElementRef>i44</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O48">
        <Input/>
        <Output>
            <DataElementRef>i45</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O49">
        <Input/>
        <Output>
            <DataElementRef>i47</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>33</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O50">
        <Input/>
        <Output>
            <DataElementRef>i48</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="O51">
        <Input/>
        <Output>
            <DataElementRef>i49</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
        <Cost>0</Cost>
        <Time>0</Time>
        <Probability>0.0</Probability>
    </Operation>
    <Operation OperationID="Op33">
        <Input>
            <DataElementRef>i39</DataElementRef>
            <DataElementRef>i49</DataElementRef>
        </Input>
        <Output>
            <DataElementRef>i43</DataElementRef>
        </Output>
        <ResourceRef>w1</ResourceRef>
    </Operation>
    <RootElementRef>i18</RootElementRef>
</PDM>
```

Appendix B

Explanations of the Algorithms

This appendix contains a detailed explanation of each of the seven algorithms of Chapter 5. Each section contains an overview of the classification of the algorithm, a stepwise description, and a list of remarks. The small example PDM in Figure B.1 is used to illustrate the step-by-step elaboration of each algorithm.

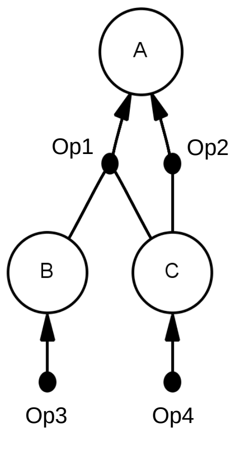


__Figure B.1:__ The simple example PDM.

B.1 Algorithm Alpha

In this section algorithm Alpha is explained in more detail. Algorithm Alpha is classified in the framework of Section 5.1 as follows:

Construction perspective	
1. Representation	Petri net
2. Order	Top-down
3. Focus	Data elements
Process execution perspective	
1. Moment of choice	Early
2. Eagerness	Strict
3. Concurrency	Possible

Stepwise description of algorithm Alpha

· The algorithm starts with the root element X (in the example this is data element A). A transition X is created for this data element. Also, an input place i and an output place o are created and connected to transition X (STEP 1).

· Then, if data element X is not a leaf element, a transition $init\_X$ is added and the ingoing arc to transition X is changed to point to transition $init\_X$ (STEP 2).

  – Next, if data element X is produced by more than one operation, a transition for each operation is created and a single input place and a single output place to these transitions are added and connected. The $init\_X$ transition is connected to the input place, and the output place is connected to transition X (STEP 3). Then:

    $*$ if the operation has several input data elements, an $init\_OpX$ transition is added and the ingoing arc from transition OpX is redirected to transition $init\_OpX$ (STEP 4a) and, an input place, a transition, and an output place are created for each input data element and connected to the $init\_OpX$ and OpX transitions (STEP 4b).

    $*$ else, if the operation OpX only has one input data element, the corresponding transition is renamed to the input element (see e.g. STEP 5 in Figure B.6).

  – Else, if data element X is produced by only one operation OpX, we look at the number of input elements of operation OpX. For each input data element Y a transition is created with one input place i_Y and one output place o_Y. The $init\_X$ transition is connected to place i_Y, place i_Y is connected to transition Y, transition Y is connected to place o_Y, and place o_Y is connected to transition X.

· Else, if data element X is a leaf element, nothing should be done.

· The above steps are repeated for all data elements that are input to operations producing X, as if the input data element was a new root element (STEP 6).

· Finally, a simplification rule is applied to reduce the size of the resulting Petri net. Each part of the model which is of the form of a single input place connected to an 'init_X' transition connected to a single output place, can be replaced by just one place (STEP 7).

Note that STEP 4 and 5 can be executed in an arbitrary order.

Remarks

· The process models generated by algorithm Alpha are sound.

· However, they do not necessarily satisfy the basic correctness criteria with respect to the PDM because it is possible to violate the data dependencies (second correctness condition). See also Section 5.3.

Example

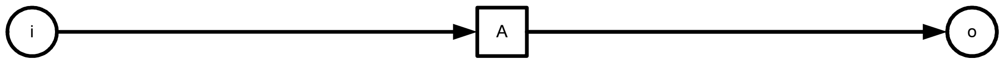


Figure B.2: *STEP 1: A transition A is created for the root element A of the PDM. Also, an input place i is created and connected with transition A and an output place o is created and transition A is connected to this output place.*

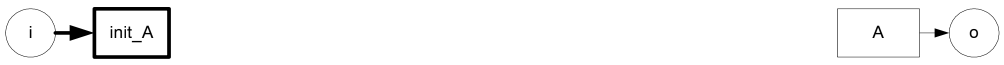


Figure B.3: *STEP 2: Data element A is not a leaf element and can be produced by more than one operation. Therefore, a transition $init\_A$ is added. The ingoing arc to transition A is redirected to transition $init\_A$.*

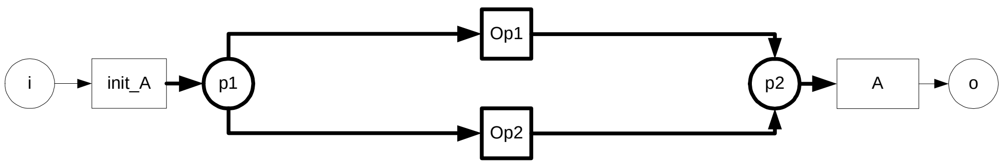


Figure B.4: *STEP 3: Data element A can be produced by two alternative operations: $Op1$ and $Op2$. Therefore, an input place $p1$, an output place $p2$ and two transitions $Op1$ and $Op2$ are created. The $init\_A$ transition is connected to the input place, the input place is connected to each of the transitions $Op1$ and $Op2$, these transitions are connected to the output place, and the output place is connected to transition A.*

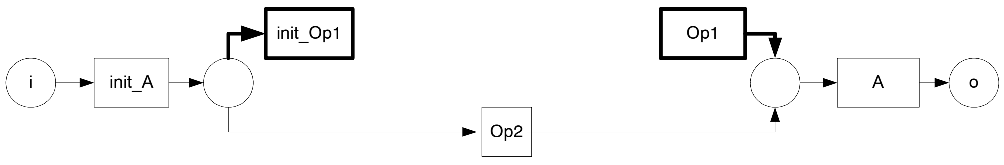


Figure B.5: *STEP 4a: Operation $Op1$ has two input data elements: B and C. An $init\_Op1$ transition is added and the ingoing arc of transition $Op1$ is redirected to transition $init\_Op1$.*

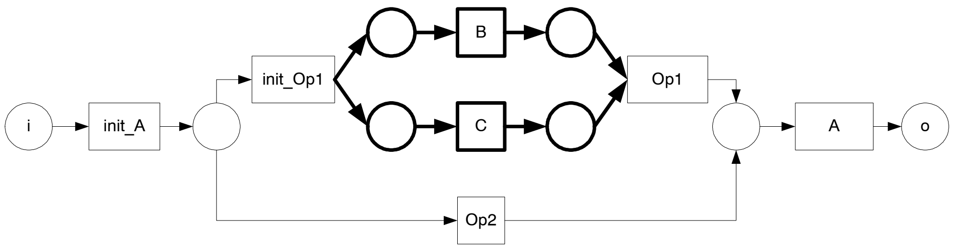


Figure B.6: *STEP 4b: An input place, a transition, and an output place are created for both input data elements B and C of $Op1$ and they are connected to the $init\_Op1$ and $Op1$ transitions.*

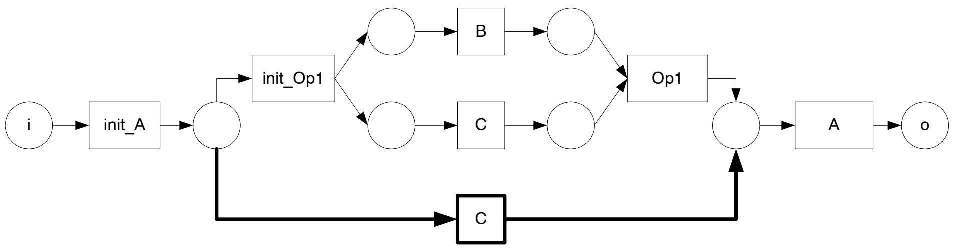


Figure B.7: *STEP 5: Transition $Op2$ is left. Operation $Op2$ has only one input element: C. The transition is renamed to the input data element: transition C.*

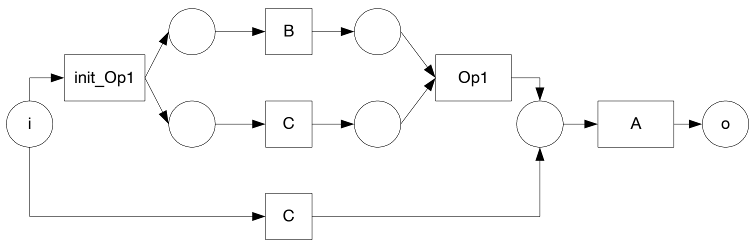


Figure B.8: *STEP 6: The simplification rule is applied and the part of the input place i connected to transition $init\_A$ connected to the output place of $init\_A$ is replaced by only one place i.*

B.2 Algorithm Bravo

In this section algorithm Bravo is explained in more detail. Algorithm Bravo is classified in the framework of Section 5.1 as follows:

Construction perspective	
1. Representation	Petri net
2. Order	Top-down
3. Focus	Operations
Process execution perspective	
1. Moment of choice	Early
2. Eagerness	Strict
3. Concurrency	Possible

Stepwise description of algorithm Bravo

· One start place (i) and one end place (o) is created (STEP 1). The output place represents a data element: the root element.

· Now, the algorithm fills the space between the two places by looking at the data element that is represented by the output place. One of the operations producing this data element is selected. A transition is created for this operation together. Then, based on the number of input elements to the operation, one of three steps is executed:

  – If the number of input data elements is one, a place pi is added and connected to the transition. Then the space between start place i and output place pi is filled as described in this algorithm (STEP 2).
  – If the number of input data elements to this operation is zero, the start place i is connected to the transition (STEP 3).
  – If the number of input data elements to this operation is more than one, an initializing transition is added and the input place i is connected to this transition (STEP 4). Two places pi and po are added for every input data element and the initializing transition is connected to pi, and po is connected to the transition for the operation (STEP 5). Then, the space between input place pi and output place po is filled as described in this algorithm.

· These steps are repeated for all operations producing the root element (STEPS 6-8).

Note that the selection of operations is arbitrary. Thus, STEP 2-3 could be interchanged with STEP 4-8. For similar reasons also STEP 5-6 could be interchanged with STEP 7-8.

Remarks

· The process models generated by algorithm Bravo are sound (see also Section 5.3).

· The process models generated by algorithm Bravo are also consistent with the PDM (see also Section 5.3).

Example

Figure B.9: *STEP 1: A start place i and an end place o are created. The output place represents the root element A.*

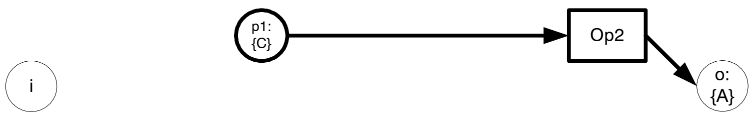


Figure B.10: *STEP 2: Operation $Op2$ is selected because it produces the root element A. A transition $Op2$ is added to the model and it is connected to the output place o. Operation $Op2$ has one input element: C. Therefore one input place $p1$ is added and connected to transition $Op2$.*

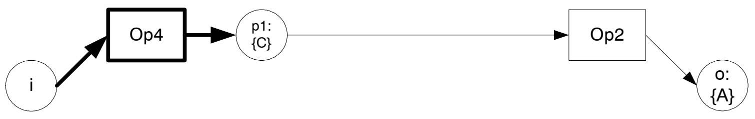


Figure B.11: *STEP 3: Operation $Op4$ is selected because it produces data element C. Transition $Op4$ is added and connected to place $p1$ which represents the availability of a value for C. Since operation $Op4$ does not have input elements, the input place i is connected to transition $Op4$.*

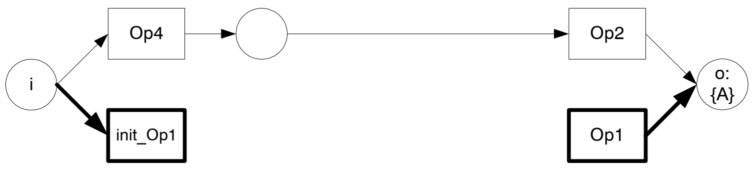


Figure B.12: *STEP 4: Now operation $Op1$ is selected because it produces a value for$A$. Transition $Op1$ is added to the model and connected with place o. Since operation $Op1$ has more than one input element, also a transition $init\_Op1$ is added. Place i is connected to $init\_Op1$.*

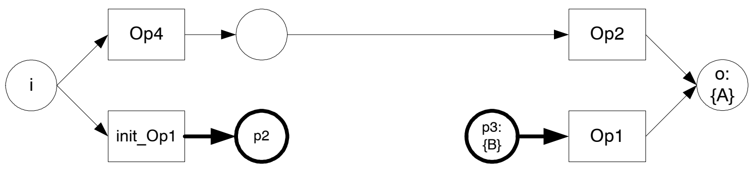


Figure B.13: *STEP 5: For each input element of operation $Op1$ two places are added to the model. First, data element B is selected. Place $p2$ and place $p3$ are added to the model. Transition $init\_Op1$ is connected to place $p2$ and place $p3$ is connected to transition $Op1$. In between the places $p2$ and $p3$ the determination of a value for data element B is worked out (see next step).*

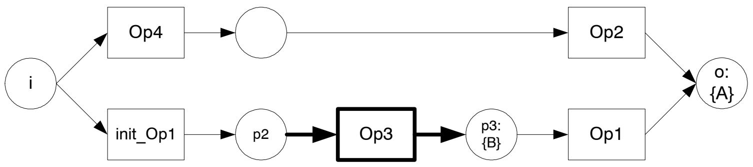


Figure B.14: *STEP 6: Operation $Op3$ produces a value for data element B. A transition $Op3$ is created and connected to place $p3$. Since operation $Op3$ does not have any input elements, place $p2$ is connected to transition $Op3$.*

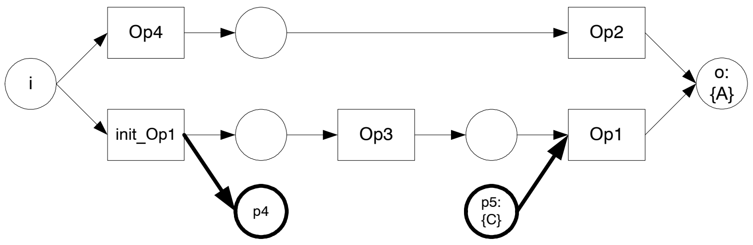


Figure B.15: *STEP 7: The second input element op operation $Op1$ is data element C. Two places $p4$ and $p5$ are added to the model. Transition $init\_Op1$ is connected to place $p4$ and place $p5$ is connected to transition $Op1$. In between these two places $p4$ and $p5$ the determination of a value for data element C is worked out (see the next step).*

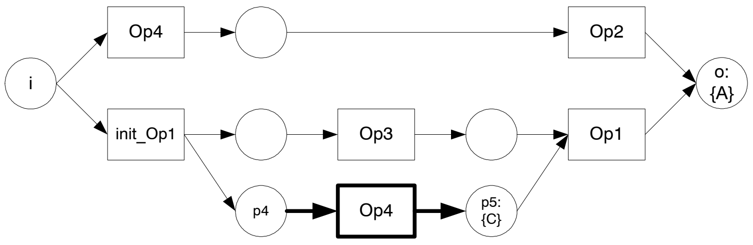


Figure B.16: *STEP 8: Operation $Op4$ produces a value for data element C. A transition $Op4$ is added and connected to place $p5$. Since operation $Op4$ does not have any input elements, place $p4$ is directly connected to transition $Op4$. Since no other operations that produce a value for data element A are left, the process model is ready.*

B.3 Algorithm Charlie

In this section algorithm Charlie is explained in more detail. Algorithm Charlie is classified in the framework of Section 5.1 as follows:

Construction perspective	
1. Representation	Petri net
2. Order	Middle-out
3. Focus	Data elements
Process execution perspective	
1. Moment of choice	Late
2. Eagerness	Overcomplete and unrestricted
3. Concurrency	Possible

Stepwise description of algorithm Charlie

- For each data element in the PDM, a component consisting of a transition with one input and one output place is created (STEP 1).
- Next, for each operation in the PDM a transition is added to the model.
- The input places to a transition which corresponds to an operation, are the output places of the transitions producing the input elements of the operation. Thus, these output places are connected to the transitions. The output place of a transition which corresponds to an operation is the input place of the transition producing the output element of the operation (STEP 2a-2b).
- Finally, a start place and initializing transition are added and connected to the input place of all transitions producing leaf elements (STEP 3). The initializing transition is an AND-split, such that all transitions producing a leaf element are enabled.

Remarks

- The process model generated by algorithm Charlie is not necessarily sound because it may contain deadlocks and does not necessarily have a proper completion (see Section 5.3).
- Neither does it satisfy the basic correctness criteria with respect to the PDM by default (see also Section 5.3).

Example

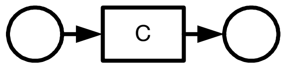


Figure B.17: *STEP 1a: A transition with an input and output place is added for data element C.*

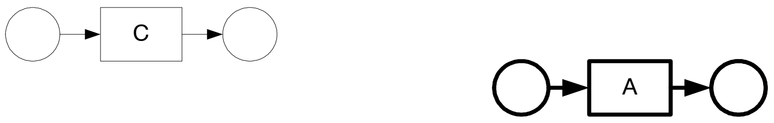


Figure B.18: *STEP 1b: A transition with an input and output place is added for data element A.*

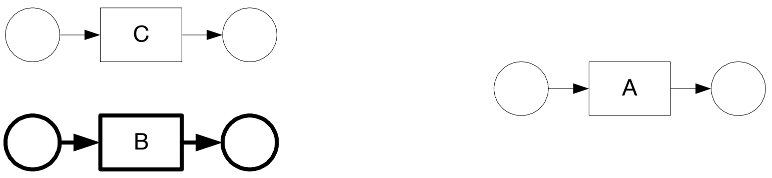


Figure B.19: *STEP 1c: A transition with an input and output place is added for data element B.*

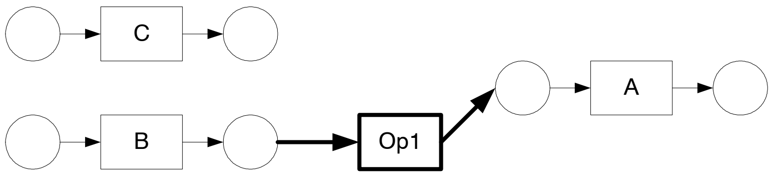


Figure B.20: *STEP 2a: A transition for operation $Op1$ is added. The transition is connected with its output place, i.e. the input place of the transition corresponding to the output element of operation $Op1$: A. The input place of the transition $Op1$ is the output place of the transition corresponding to the input element of $Op1$: B.*

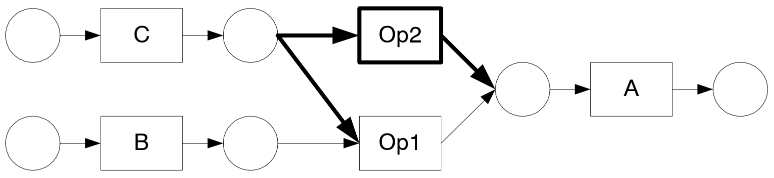


Figure B.21: *STEP 2b: A transition for operation $Op2$ is added. The transition is connected with its output place, i.e. the input place of the transition corresponding to the output element of operation $Op1$: A. The input places of the transition $Op1$ are the output places of the transitions corresponding to the input elements of $Op1$: B and C.*

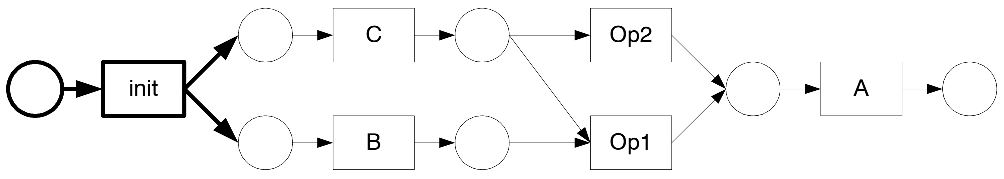


Figure B.22: *STEP 3: The start place and initializing transition are added and connected to the input places of all transitions corresponding to a leaf data element in the PDM.*

B.4 Algorithm Delta

In this section algorithm Delta is explained in more detail. Algorithm Delta is classified in the framework of Section 5.1 as follows:

Construction perspective	
1. Representation	YAWL model
2. Order	Middle-out
3. Focus	Data elements
Process execution perspective	
1. Moment of choice	Late
2. Eagerness	Overcomplete but restricted
3. Concurrency	Possible

Note that the models generated by algorithm Delta are represented in the YAWL language (cf. Section 2.5).

Stepwise description of algorithm Delta

· For each data element in the PDM, a component consisting of a YAWL taks with one input and one output YAWL condition is created (STEP 1).

· Next, for each operation in the PDM a YAWL task is added to the model.

· The input conditions to a task which corresponds to an operation, are the output conditions of the tasks producing the input elements of the operation. Thus, these output conditions are connected to the tasks. The output condition of a task which corresponds to an operation is the input condition of the task producing the output element of the operation. Moreover, the task is also connected to each of its input conditions, i.e. it puts back the tokens it took (STEP 2).

· Then, a start condition and initializing task are added and connected to the input condition of all tasks producing leaf elements (STEP 3). The initializing task is an AND-split, such that all tasks producing a leaf element are enabled.

· Next, control conditions are created for each task to make sure the task can only be executed once (STEP 4).

· Finally, a cancellation region is added to the task producing the root element (STEP 5).

Remarks

· Algorithm Delta has the same basis as algorithm Charlie. The differences with process models generated by algorithm Charlie are: (i) the arcs that put the tokens back in the input conditions of each task that corresponds to an operation from the PDM, (ii) the extra control conditions that ensure that each task is only executed once, (iii) the cancellation region with the final task, and (iv) the representation as a YAWL model (which allows for the use of cancellation regions).

· The process model generated by algorithm Delta is sound because the problems in the process models generated by algorithm Charlie are solved by the extra arcs, control conditions and cancellation region (see also Section 5.3).

· The process model does not satisfy the basic correctness criteria with respect to the PDM by default because the data dependencies may be violated (see also Section 5.3).

· Note that the models generated by algorithm Delta become very large because of the extra control conditions and cancellation arcs.

Example

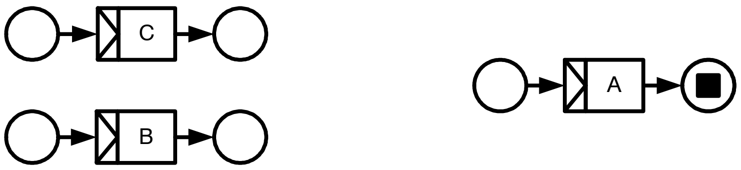


Figure B.23: *STEP 1: A task with one input and one output condition is added for every data element in the PDM (cf. STEP 1 of algorithm Charlie in figures B.17-B.19).*

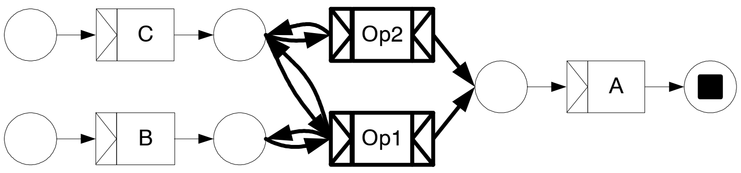


Figure B.24: *STEP 2: A task for each operation is added. These tasks are connected with the right output condition, and the right input conditions. (cf. STEP 2 of algorithm Charlie in figures B.20 and B.21). Note that each input condition of a tasks here also is an output condition.*

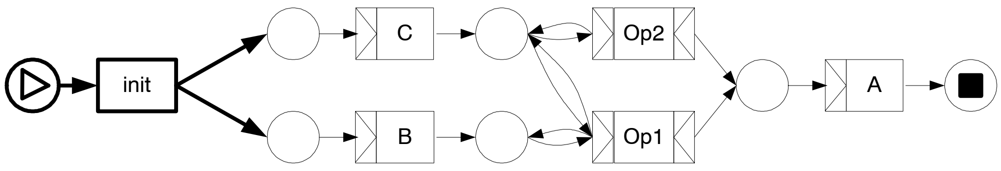


Figure B.25: *STEP 3: The start condition and initializing task are added and connected to the input places of all tasks corresponding to a leaf data element in the PDM (cf. STEP 3 of algorithm Charlie in Figure B.22).*

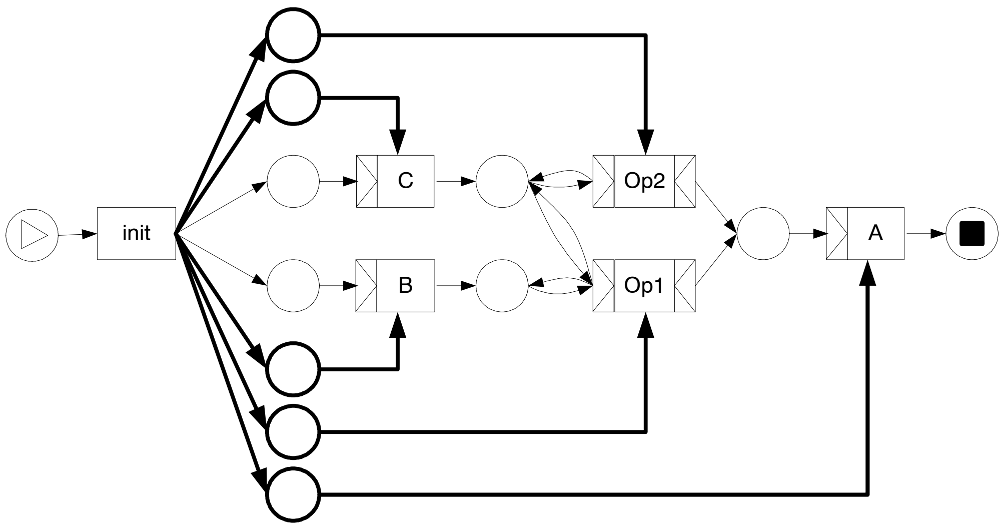


Figure B.26: *STEP 4: Control places are added and connected to all tasks except for the initializing task to enforce that each task can only fire once.*

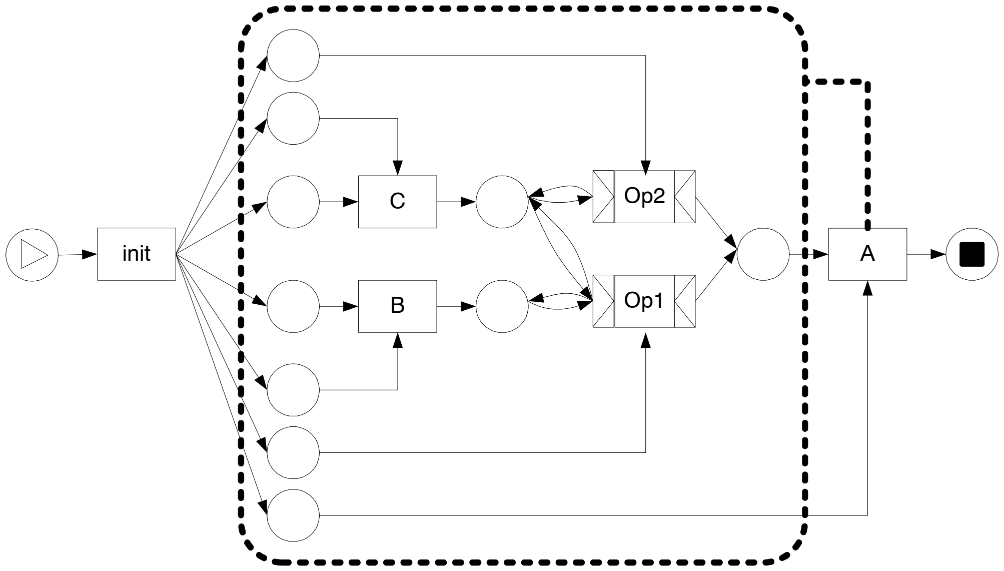


Figure B.27: *STEP 5: All conditions and tasks between the initializing task and the final task producing the end product are added to the cancellation region.*

B.5 Algorithm Echo

In this section algorithm Echo is explained in more detail. Algorithm Echo is classified in the framework of Section 5.1 as follows:

Construction perspective	
1. Representation	Petri net
2. Order	Middle-out
3. Focus	Operations
Process execution perspective	
1. Moment of choice	Late
2. Eagerness	Overcomplete and unrestricted
3. Concurrency	Possible

Stepwise description of algorithm Echo

· An input place, an output place and an initializing transition are created. The input place is connected to the initializing transition (STEP 1).

· Next, for each data element in the PDM a place and a transition are created. The place is connected to the transition (STEP 2).

· Then, one operation is selected and a transition 'O' is created for this operation. Transition 'O' is connected to the input place of the transition corresponding to the output element of the operation. Also, a number of input places to transition 'O' are added: for each input element of the operation one input place to the transition is created and connected. In case the operation has no input elements, one input place is created. Finally, the transitions corresponding to the input elements of the operation are connected to the corresponding input place (STEP 3a).

· The previous step is repeated for all operations in the PDM (STEP 3b, 3c, and 3d).

· Finally, an extra place p is added to ensure that the root element can be produced only once. The initializing transition is connected to place p and p is connected to the transition producing the root element (STEP 4).

Note that STEPS 3a-3d can be executed in any arbitrary order.

Remarks

· The process models generated by algorithm Echo are not necessarily sound. Because all branches are enabled by the initialization transition, there may be tokens left in the model after the final transition has fired. However, the process models are lazy sound because of the extra control place and the production of sufficient tokens (see Section 5.3).

· The process models generated by algorithm Echo are by default correct with respect to the PDM (see Section 5.3).

Example

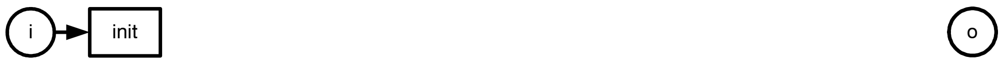


Figure B.28: *STEP 1: An input place i, an output place o and an initializing transition are created. The input place i is connected with the initializing transition.*

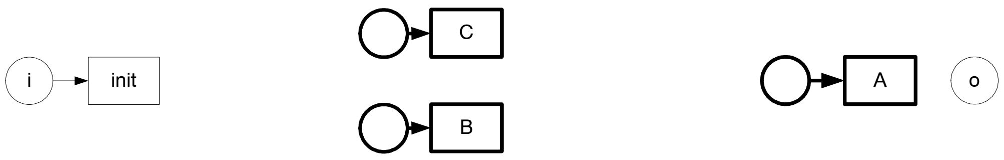


Figure B.29: *STEP 2: For every data element in the PDM, a place connected to a transition are added to the model. This is done in an arbitrary order, similar to STEPS 1a-1c in algorithm Charlie (see Figure B.17-B.19).*

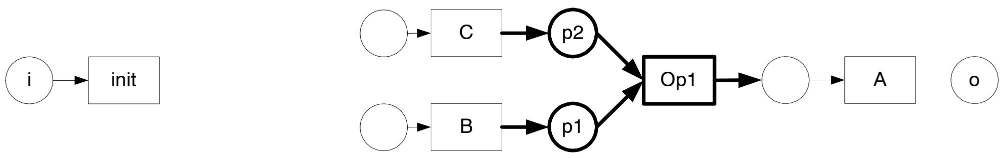


Figure B.30: *STEP 3a: One operation is randomly selected from the PDM: $Op1$. For this operation a transition $Op1$ is added to the model. Then this transition is connected to the input place of transition A because A is the output element of operation $Op1$. Also two input places $p1$ and $p2$ to transition $Op1$ are added to the model, because operation $Op1$ has two input elements: B and C. Finally, transitions B and C are connected to the places $p1$ and $p2$ respectively.*

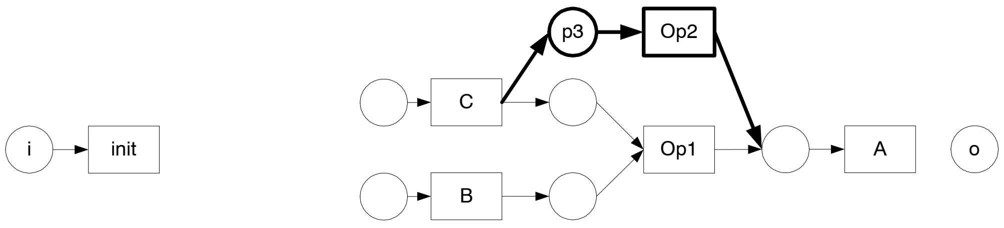


Figure B.31: *STEP 3b: Another operation is selected from the PDM: $Op2$. Transition $Op2$ is added to the model. Then this transition is connected to the input place of transition A because A is the output element of operation $Op2$. Since operation $Op2$ only has one input element, i.e. C, one input place $p3$ is added and connected to transition $Op2$. Finally, transition C is connected to place $p3$.*

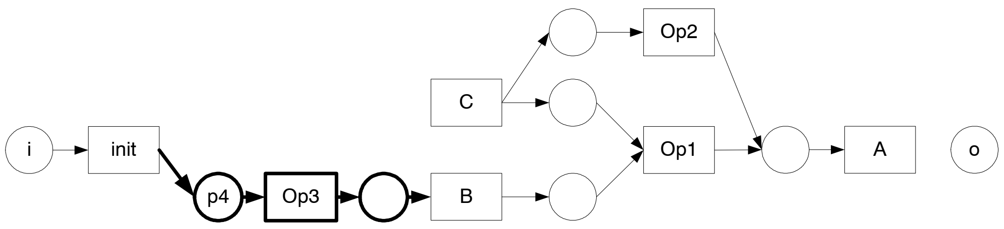


Figure B.32: *STEP 3c: Another operation is selected from the PDM: $Op3$. Transition $Op3$ is added to the model. Then this transition is connected to the input place of transition B because B is the output element of operation $Op3$. Since operation $Op3$ has no input elements, one input place $p4$ is added and connected to transition $Op3$. Also, the initializing transition is connected to place $p4$.*

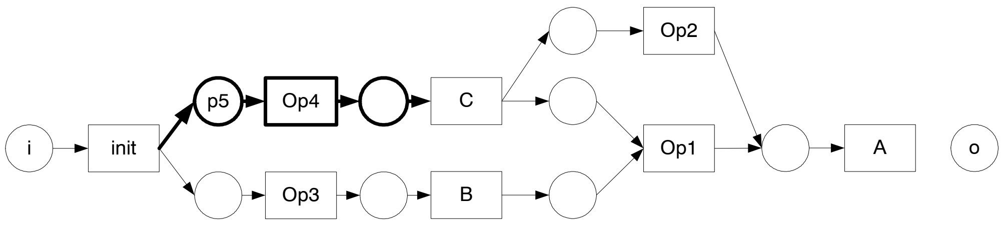


Figure B.33: *STEP 3d: The final operation is selected from the PDM: $Op4$. Transition $Op4$ is added to the model. Then this transition is connected to the input place of transition C because C is the output element of operation $Op4$. Since operation $Op4$ has no input elements, one input place $p5$ is added and connected to transition $Op3$. Also, the initializing transition is connected to place $p4$.*

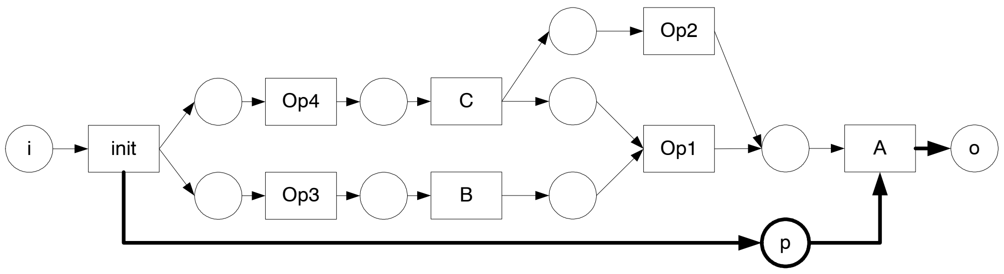


Figure B.34: *STEP 4: An extra place p is added to the model to ensure that transition A, producing the root element, can only fire once. The initializing transition is connected to place p and p is connected to transition A. Finally, transition A is connected to output place o.*

B.6 Algorithm Foxtrot

In this section algorithm Foxtrot is explained in more detail. Algorithm Foxtrot is classified in the framework of Section 5.1 as follows:

Construction perspective	
1. Representation	Petri net
2. Order	Bottom-up
3. Focus	Operations
Process execution perspective	
1. Moment of choice	Late
2. Eagerness	Overcomplete and unrestricted
3. Concurrency	Impossible

Stepwise description of algorithm Foxtrot

· Start with just one start place and one end place (STEP 1). A place represents the available data elements, i.e. the start place represents the empty set and the end place represents the set of all data elements.

· Then, select a leaf element. A transition and an output place for this transition are created and connected. Also, the input place is connected to the transition (STEP 2).

  – We start reasoning further from the output place of the transition. One of the operations of which all input elements are available and which is not executed yet is selected. For this operation a transition is added and connected to a newly created place if this place was not existing yet (STEP 3, STEP 4). Otherwise the transition is connected to the already existing place.

  – If no further steps are possible from the place we have reached we look back to the places we have reached whether another branch can be started from one of this places. This branch is also elaborated step-by-step as described above (STEP 5).

· These steps are executed for all leaf elements (STEP 6, STEP 7, STEP 8, STEP 9).

· Finally, all places containing the root element are connected with the end place via a silent transition (STEP 10).

Remarks

· The process models generated by algorithm Foxtrot describe all possible execution paths of the PDM. Therefore, the models are very large.

· Moreover, the model is completely sequential, no concurrent execution of activities for the same case is possible,

· The process models are sound and correct with respect to the PDM (see Section 5.3).

Example

Figure B.35: *STEP 1: An input place i and an output place o are created. The input place i represents the situation in which no data element values are available, denoted by $\{\}$.*

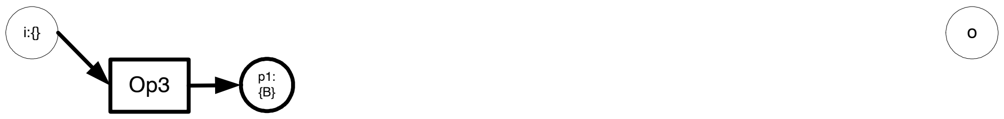


Figure B.36: *STEP 2: One of the leaf operation is selected: $Op3$. A transition $Op3$ is created for this operation, together with an output place $p1$ representing the the availability of a value for data element B. Place i is connected to transition $Op3$ and transition $Op3$ is connected to place $p1$.*

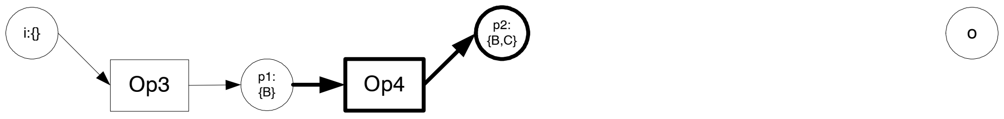


Figure B.37: *STEP 3: Operation $Op4$ is now selected because it is executable from the situation in which a value for element B is available. A transition $Op4$ is created for this operation, together with an output place $p2$ representing the the availability of a value for data element B and C. Place $p1$ is connected to transition $Op4$ and transition $Op4$ is connected to place $p2$.*

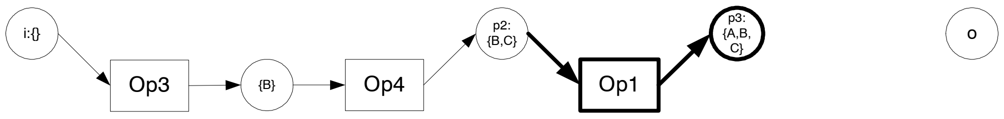


Figure B.38: *STEP 4: One of the operations that can be executed from the situation in which a value for data elements B and C is available, is operation $Op1$. This operation is selected and a transition $Op1$ is added, together with an output place $p3$ representing the availability of a value for data elements A, B, and C. Place $p2$ is connected to transition $Op1$ and transition $Op1$ is connected to place $p3$.*

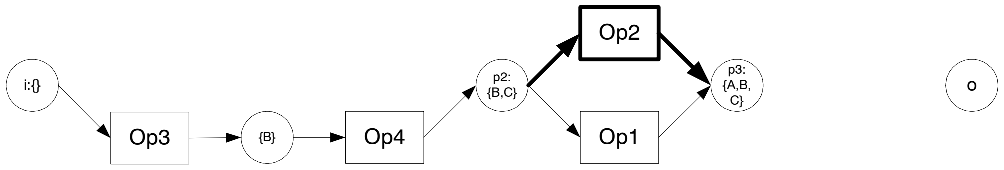


Figure B.39: *STEP 5: From the situation in which a value for data elements A, B, and C is present (cf. $\{A, B, C\}$ in place $p3$) no other operations are executable anymore, since there is a value available for all data elements in the PDM. We take a step back and look at the situation in which B and C are available. Another operation that can be executed based on these values is operation $Op2$. Thus, a transition $Op2$ is created. The situation after executing operation $Op2$ is $\{A, B, C\}$ and is already represented by place $p3$. Thus, transition $Op2$ is connected to place $p3$, and place $p2$ is connected to transition $Op2$.*

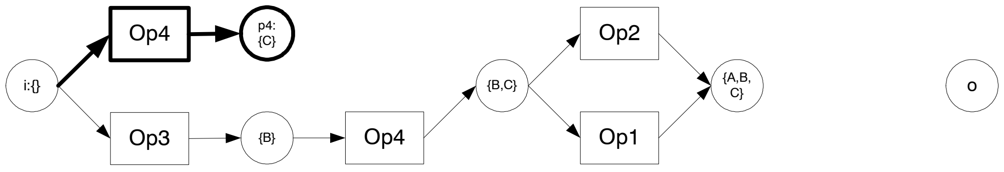


Figure B.40: *STEP 6: Now there are no other executable operations for the situations $\{B, C\}$, and $\{B\}$. So we go back to the situation in which there are no data element values available, i.e. $\{\}$. Another leaf operation, $Op4$, can be executed. A new transition $Op4$ is created for this operation, together with an output place $p4$ representing the the availability of a value for data element C. Place i is connected to transition $Op4$ and transition $Op4$ is connected to place $p4$.*

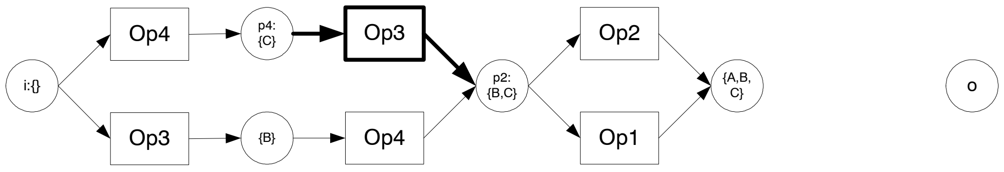


Figure B.41: *STEP 7: From the situation $\{C\}$ in place $p4$, operation $Op3$ can be executed. A new transition $Op3$ is created. The situation after executing operation $Op3$ is already represented by place $p2$: $\{B, C\}$. Therefore, transition $Op3$ is connected to place $p2$, and place i is connected to transition $Op3$.*

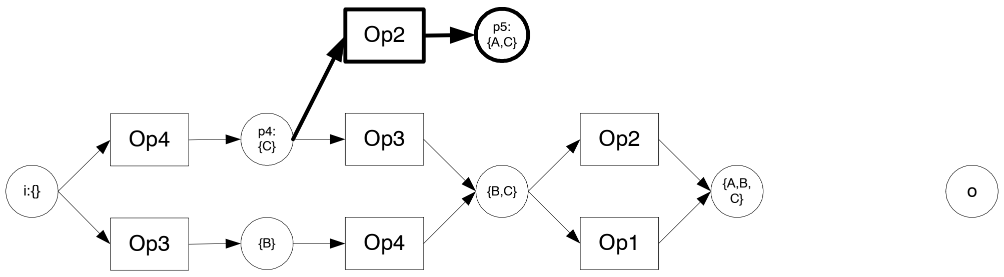


Figure B.42: *STEP 8: From the situation $\{C\}$ in place $p4$, also operation $Op2$ can be executed. A new transition $Op2$ is created, together with an output place $p5$ representing the the availability of a value for data elements A and C. Place $p4$ is connected to transition $Op2$ and transition $Op2$ is connected to place $p5$.*

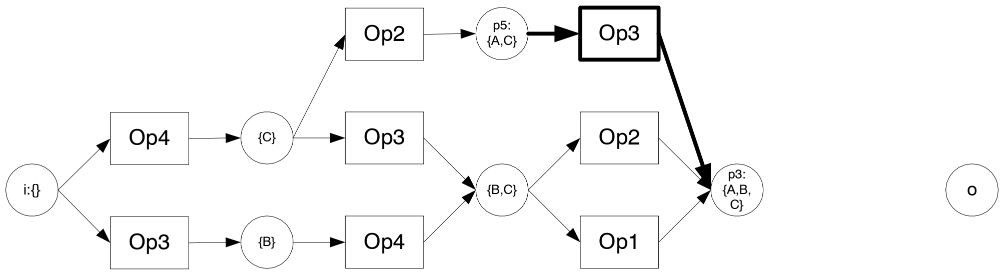


Figure B.43: *STEP 9: From the situation $\{A, C\}$ in place $p5$, operation $Op3$ can be executed. A new transition $Op3$ is created. The situation after executing operation $Op3$ is already represented by place $p3$: $\{A, B, C\}$. Therefore, transition $Op3$ is connected to place $p3$. Finally, place $p5$ is connected to transition $Op3$.*

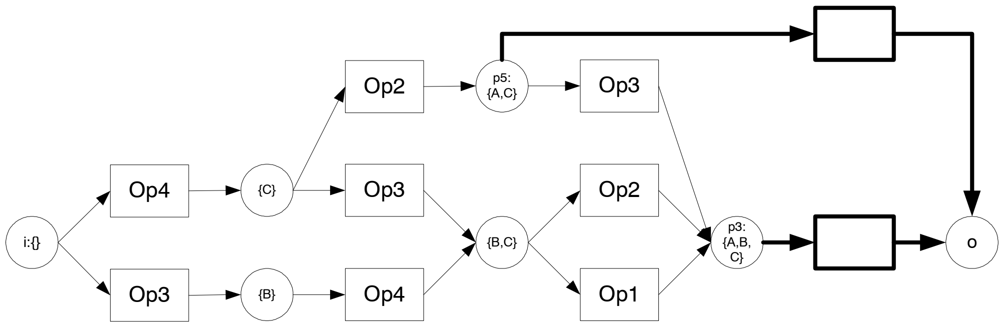


Figure B.44: *STEP 10: Finally, all places containing describing a situation in which a value for the root element A is available (places $p3$ and $p5$) are connected with the end place via a silent transition, i.e. a transition that is only used for a correct routing, but does not represent the execution of an operation from the PDM.*

B.7 Algorithm Golf

In this section algorithm Golf is explained in more detail. Algorithm Golf is classified in the framework of Section 5.1 as follows:

Construction perspective	
1. Representation	Petri net
2. Order	Top-down
3. Focus	Operations
Process execution perspective	
1. Moment of choice	Early
2. Eagerness	Strict
3. Concurrency	Impossible

Stepwise description of algorithm Golf

· First all minimal execution paths (see Section 5.1.1) are determined (STEP 1). A minimal execution path is a path in the PDM in which all operations contribute to the determination of the end product, i.e. if one of the operation would be left out of the execution path, the end product cannot be produced anymore based on the given path. A minimal execution path can be determined by 'walking' through the PDM in a top-down manner.

· Then, a start place and an end place are added to the process model (STEP 2).

· One of the paths is selected and build in a bottom-up manner:

  – The first step in the path is selected and a transition and output place of this transition are created. Also, the input place of the model is connected to the transition (STEP 3a).

  – Then, a next step in the execution sequence is selected. Again, a transition is added for this operation. Also, an output place, representing the available data elements is added and the transition is connected to this place. These steps are repeated until the path is finished (STEPS 3b-3c).

  – When the path is finished a silent transition connecting the final place of the path with the end place of the process model is added (STEP 3d).

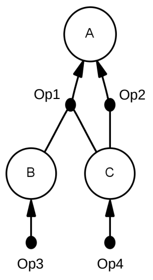
· These steps are repeated for all minimal execution paths. While repeating these steps the existence of transitions and places is checked. If a place representing a number of available data elements or an overlapping part of the path was already created before, this component in the model is used instead of creating a new place or transition (STEPS 4 and 5).

Remarks

· The process models generated by algorithm Golf are sound and correct with respect to the PDM (see Section 5.3).

· Note that the process model generated by algorithm Golf is very similar to an elaboration of the parallel behavior in the process model generated by algorithm Bravo.

Example



$$\begin{aligned}\sigma_1 &= \langle Op4, Op3, Op1\rangle\\ \sigma_2 &= \langle Op3, Op4, Op1\rangle\\ \sigma_3 &= \langle Op4, Op2\rangle\end{aligned}$$

Figure B.45: *STEP 1: The execution paths of the PDM are determined top down. These execution paths are elaborated one by one in a bottom up way that is similar to algorithm Foxtrot.*

Figure B.46: *STEP 2: An input place i and an output place o are created. The input place i represents the situation in which no data element values are available, denoted by $\{\}$ (cf. STEP1 in algorithm Foxtrot in Figure B.35).*

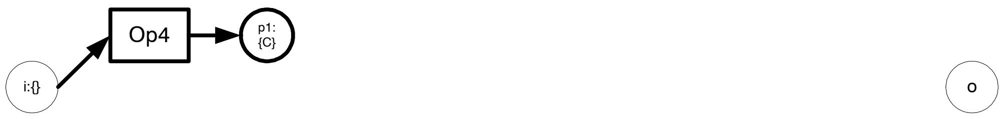


Figure B.47: *STEP 3a: The first execution path σ_1 is selected and it operations are added to the model stepwise. The first operation that is executed in this execution path is $Op4$. A transition $Op4$ and an output place $p1$ representing the situation in which a value for the output element of operation $Op4$, i.e. C is available, are created. The input place i is connected to transition $Op4$ and transition $Op4$ is connected to place $p1$.*

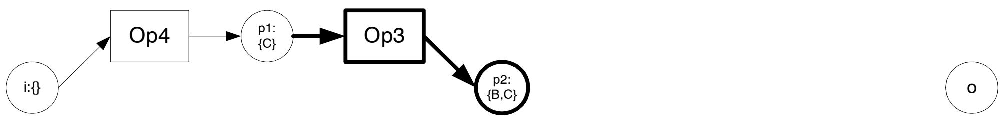


Figure B.48: *STEP 3b: The second operation in execution path σ_1 is operation $Op3$. A transition $Op3$ is created, together with an output place $p2$ representing the situation in with a value for data elements B and C is available. Place $p1$ is connected to transition $Op3$ and transition $Op3$ is connected to place $p2$.*

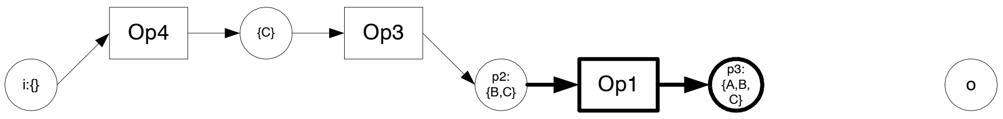


Figure B.49: *STEP 3c: The third operation in execution path σ_1 is operation $Op1$. A transition $Op1$ is created, together with an output place $p3$ representing the situation in with a value for data elements A, B and C is available. Place $p2$ is connected to transition $Op1$ and transition $Op1$ is connected to place $p3$.*

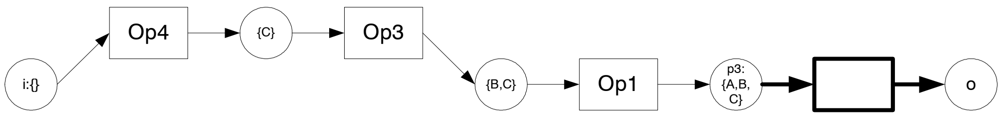


Figure B.50: *STEP 3d: Now, all operations in execution path σ_1 have been added and place $p3$ is connected to the output place o via a silent transition.*

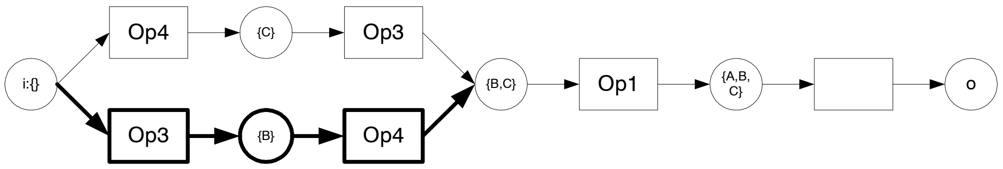


Figure B.51: *STEP 4: The second execution path $\sigma_2 = \langle Op3, Op4, Op1 \rangle$ is added to the model in a similar way as the first execution path was worked out in STEPS 3a-3d in figures B.47-B.50 above.*

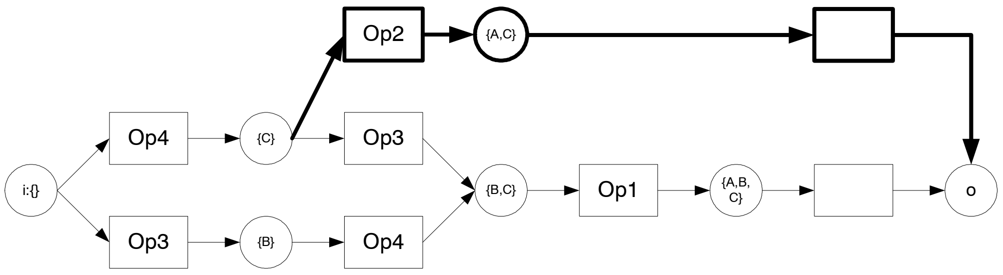


Figure B.52: *STEP 5: The third execution path $\sigma_3 = \langle Op4, Op2 \rangle$ is added to the model.*

Appendix C

CPN Model of Functional Design

In Chapter 6 we have introduced a CPN model, developed in CPN tools [74, 137, 138], which describes the functional design of a tool for the direct execution of a PDM. This appendix contains a more detailed explanation of the CPN model. It consists of two levels: the main level and a lower level, which describes the transitions on the main level.

C.1 Main Level

The main level of the CPN model is depicted in Figure C.1. It contains eight places and three transitions. All three transitions are defined by sub processes in which the actions to be taken are described in detail. The main execution stream in the model is indicated by thick lines: first the executable operations are calculated, then one of the executable operations is selected and executed and this procedure is repeated until the end product of the workflow process is produced or no executable operations remain.

The operations of the PDM are initially stored in the place *not yet executable operations*. An operation is represented by a tuple containing the identifier, the output element and the input elements, and a list of cost and time attributes, e.g. $(Op01, A, \{B, C, D\}, (5.0, 1.0, 0.05))$. The place *available data elements* contains all data elements of which the value has been determined. Also, the places *ready for calculation*, *total cost*, and *total duration* each contain a token. The token in the former indicates that a calculation for the case can be started, while the latter two places are used to monitor the cumulative cost and processing time for the case.

When a case is started the executable operations are determined first, i.e. transition *calculate executable operations* fires (see also Section C.2). This results in a subdivision of the list of operations: operations that are executable are put in the list in place *executable operations* and non-executable operations are put back in the *not yet executable operations* place. Also, a token is put in the place *ready for selection*. Then, transition *select operation* can fire. In the sub process related to this transition (see Section C.3), a selection strategy (e.g. random, lowest cost, shortest processing time, first in list) is specified which prescribes which of the operations in *executable operations* should be selected. The selected operation is put in the place *selected operation* and the other operations are left in *executable operations*. Then, the selected operation is executed by the *execute operation* transition (see Section C.4). A time delay is introduced to account for the processing time of the operation. The total cost, as well as the total duration monitors of the case are increased by the execution cost and the processing time of the operation respectively. Finally, the output data element is produced with a certain

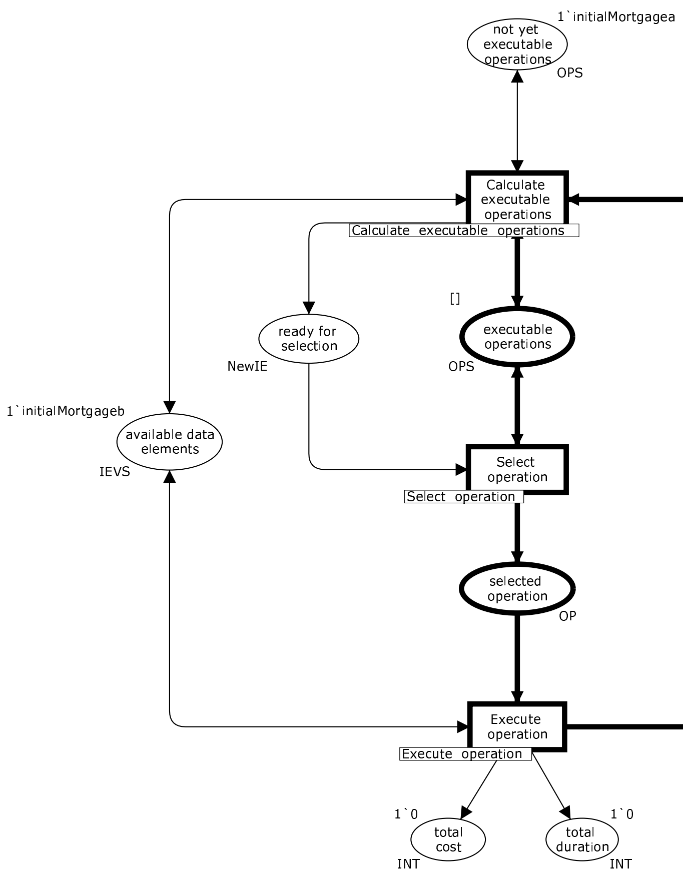


Figure C.1: *The main level of the CPN model (see also Figure 6.3).*

probability. If the execution of the operation was successful, the data element and its value are stored in the place *available data elements*. If not, no data element is added to the place *available data elements*.

As soon as one of the operations has been performed, the set of executable operations can be determined again. This is done based on the list of operations in *not yet executable operations* as well as on the list of operations in *executable operations*. Note that some of the operations in *not yet executable operations* may now have become executable because of new information produced in the previous step and that some of the operations in *executable operations* may no longer be executable since a value for their output data element was produced by another operation. Again, one operation is selected from the list of executable operations and the operation is executed, either successfully or unsuccessfully. This cycle is repeated until the end product of the PDM is produced or no executable operations remain.

In the next sections the subprocesses for the transitions *calculate executable operations*, *select operation*, and *execute operation* are detailed.

C.2 Calculation of Executable Operations

As explained above, the *calculate executable operations* transition determines which of the operations from the list of *not yet executable operations* are executable based on the list of available data elements. As is shown in Figure C.2, the *calculate executable operations* transition takes the list of available data elements and the list of not yet executable operations as inputs. For each operation in the list it determines to which output place it should be sent.

If the condition of an operation is not satisfied, i.e. if the value of one of the input elements is not satisfying a pre-defined condition, the operation should not become executable and is sent to place *condition not satisfied*. Such an operation can never become executable anymore. If the output element of the operation is already in the list of available data elements then the operation is put in the place *data element already known*. There is no need to determine the value again. If one or more of the required input data elements are not available yet, i.e. a required item is not in the list of available data elements, the operation is put back in *not yet executable operations*. Finally, the operations which satisfy the predefined condition on the input data element values, of which the output element is not already known, and of which all input elements are in the list of available data elements are sent to the *executable operations* place.

Note that *calculate executable operations* also recalculates the operations that where put in the place *executable operations* in one of the earlier iterations, because they can become superfluous (e.g. already determined or not satisfying the condition after the calculation of the value of one of the input elements). Besides, if the end product is determined, there is no need to perform more operations. Therefore, a guard is added to *calculate executable operations* which only allows this transition to fire if the end product (‘*root*’) is not yet in the list of available data elements.

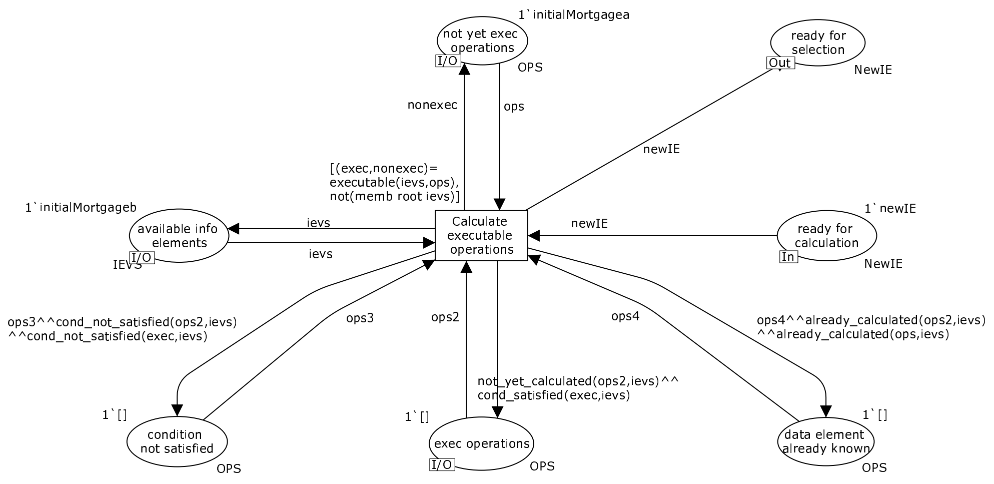

1`initialMortgagea
not yet exec operations
I/O
OPS
ready for selection
Out
NewIE
nonexec
ops
newIE
[(exec,nonexec)= executable(ievs,ops), not(memb root ievs)]
1`initialMortgageb
available info elements
I/O
IEVS
ievs
ievs
Calculate executable operations
newIE
1`newIE
ready for calculation
In
NewIE
ops3^^cond_not_satisfied(ops2,ievs) ^^cond_not_satisfied(exec,ievs)
ops3
ops2
ops4
ops4^^already_calculated(ops2,ievs) ^^already_calculated(ops,ievs)
not_yet_calculated(ops2,ievs)^^ cond_satisfied(exec,ievs)
1`[]
condition not satisfied
OPS
1`[]
exec operations
I/O
OPS
1`[]
data element already known
OPS

Figure C.2: *The the subprocess for the calculate executable operations transition.*

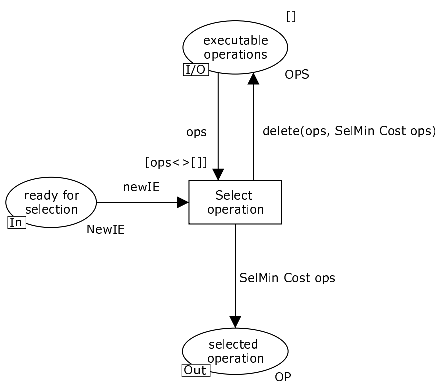


Figure C.3: *The subprocess for the selection of the operation with the lowest costs.*

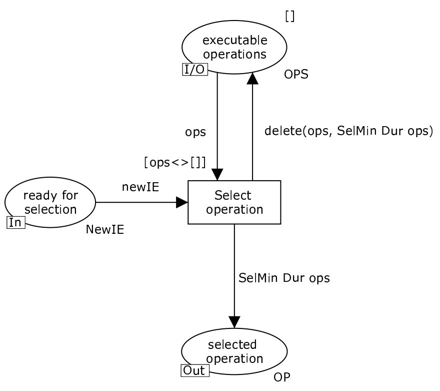


Figure C.4: *The subprocess for the selection of the operation with the shortest processing time.*

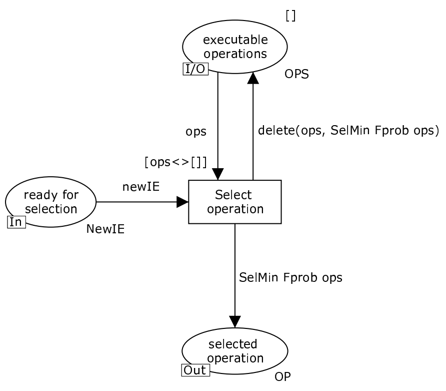


Figure C.5: *The subprocess for selecting the operation with the lowest failure probability.*

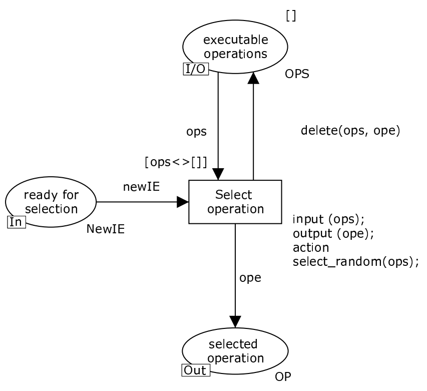


Figure C.6: *The subprocess for the random selection of an operation.*

C.3 Selection of operation

After the executable operations have been determined by the *calculate executable operations* transition (i.e. there is a token available in *ready for selection* and *executable operations* contains a list of operations), one of these operations can be selected for execution. Several strategies can be used to determine which operation from the list of executable operations should be selected. These strategies have been introduced in Section 6.3; here we elaborate on the following four:

· *Lowest cost* (see Figure C.3) - The list of executable operations is taken as an input. The *SelMin Cost* function (see also Section C.5) identifies an operation from the list with the lowest value for the *Cost* attribute. This operation is sent to the *execute operation* transition and it is deleted from the list of executable operations.

· *Shortest processing time* (see Figure C.4) - Similar to the selection of an element with the lowest cost by the *SelMin Cost* function, an element with the shortest processing time is selected by the function *SelMin Dur*.

· *Lowest failure probability* (see Figure C.5) - Similar to the selection of an element with the lowest cost by the *SelMin Cost* function, an element with the smallest failure probability is selected by the function *SelMin Fprob*.

· *Random* (see Figure C.6) - The random selection of an element from the list is done by a function *select_random* (see also Section C.5) that takes an arbitrary element from the list.

The way in which the operation to be executed is selected influences the performance of the process of making the workflow end product. We need performance measures to compare the effect of different selection strategies. To assess the performance of a certain strategy we limit ourselves to total cost and total duration, although other performance indicators (e.g. waiting time and resource utilization) can be imagined.

C.4 Execution of Operation

After selecting an operation for execution, this operation is executed in two steps (see Figure C.7). First the execution is started and the *total cost* and *total duration* monitors are updated. The execution of this operation takes some time (the time delay is determined by the attribute *duration*). Next, the execution is ended, either successfully or unsuccessfully. Whether the execution is successful is determined by a function *prob1()* that randomly generates a value. This value is compared to the failure probability of the operation. If the value is higher than the value of the failure probability, then the execution is decided to be successful and the output data element of the operation is put in the list of available data elements together with its newly determined value (the simple function *calculation* determines this new value, see Section C.5). If the the value is lower than the failure probability, the execution of the operation is failed and no data element value is produced. The *end execution* transition also puts back a token in

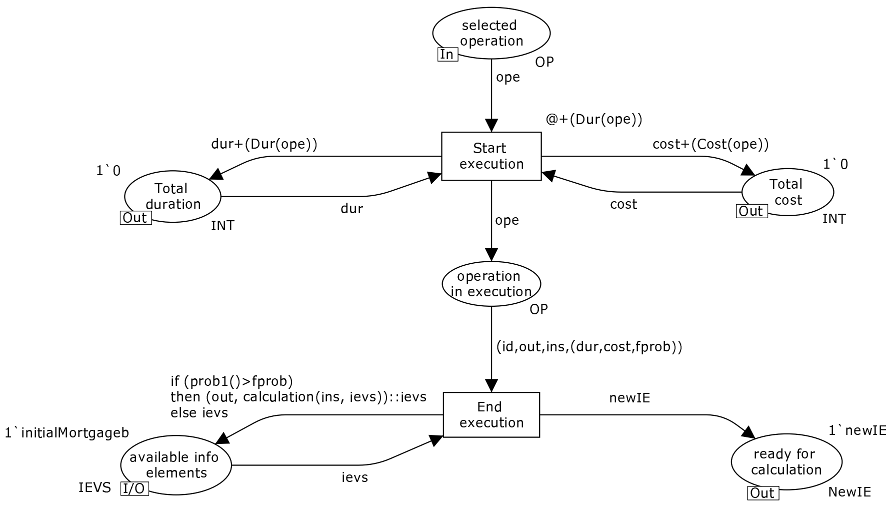


Figure C.7: *The subprocess for the execute operation transition.*

ready for calculation, so that the loop can start again by calculating the executable operations for the next possible step.

C.5 Declarations

This section contains the declarations of the colorsets, variables, values and functions of the CPN model described in the previous sections. Note that the value declarations are specific for the mortgage example which is described in Section 3.2.2.

Colorset Declarations

```
colset IE = INT;                    colset IEVS = list IEV;
colset ID = STRING;                 colset DURATION = INT;
colset NewIE = with newIE;          colset COST = INT;
colset VAL = INT;                   colset ATTR = product DURATION*COST*FPROB;
colset IEV = product IE*VAL;        colset OP = product ID*IE*IES*ATTR timed;
colset IES = list IE;               colset OPS = list OP;
colset PROB = int with 0..100;      colset FPROB = INT;
```

Variable Declarations

```
var out: IE;                        var exec, nonexec:OPS;
var ins : IES;                      var id:ID;
var attr:ATTR;                      var dur: DURATION;
var cost: COST;                     var ope:OP;
var ops,ops2,ops3,ops4: OPS;        var ievs:IEVS;
```

Value Declarations

```
(* ‘root’ is the end product of the workflow process*)
val root=1;

(*value initialMortgagea contains all operations from the product data model
for the mortgage example, including their ID, output element, input elements,
and duration, cost and failure probability attributes*)
val initialMortgagea=[("op01", 1,[2,3,4],(1,5,5)),("op02", 3,[6,7,8],(4,5,5)),
                      ("op03", 1,[8],(3,9,5)),("op04", 1,[5],(2,2,0)),
                      ("op05", 2,[],(0,0,0)),("op06", 4,[],(0,0,0)),
                      ("op07", 5,[],(1,1,50)),("op08", 6,[],(0,0,0)),
                      ("op09", 7,[],(2,0,0)),("op10", 8,[],(10,3,15))]

(*value initialMortgageb contains all initially available data elements with
their corresponding value. In the mortgage example no data element values are
available at the start*)
val initialMortgageb=[]
```

Function Declarations

```
(* memb checks whether x is an element of the list l *)
fun memb x l = List.exists (fn (y,_)=>(y=x)) l;

(* membs checks whether all elements of the first list are elements of the
second list *)
fun membs l1 l2 = List.all (fn x => memb x l2) l1;

(* prob1 generates a random number between 0 and 100 which represents the
failure probability *)
fun prob1() = PROB.ran();

(*divides the list ops into those operations that are ready for execution
and those that are not; it returns a pair of lists*)
fun executable (ievs:IEVS, ops:OPS) =
List.partition (fn (_,_,ies',_) => (membs ies' ievs)) ops;

(*returns a list containing the elements of list ops of which the output
element and its value already are a member of list ievs*)
fun already_calculated(ops,ievs) =
List.filter (fn (_,ie,_,_) =>  memb ie ievs) ops;

(*returns a list containing the elements of list ops of which the output
element and its value is not a member of list ievs*)
fun not_yet_calculated(ops,ievs) =
List.filter(fn (_,ie,_,_) => not(memb ie ievs)) ops;

(*returns the value of data element ie*)
fun get_value (ie:IE, (ie2,v)::ievs) =
if ie=ie2 then v else get_value(ie,ievs)
| get_value(ie:IE, []) = 0;

(*checks for operation with id id, whether the condition for execution is
satisfied or not*)
fun check(id:ID, ievs:IEVS) =
case id of
"op01" => true
|"op02" => true
|"op03" => true
|"op04" => true
|"op05" => true
|"op06" => true
|"op07" => true
|"op08" => true
|"op09" => true
|"op10" => true
|_=>true;

(*returns a list containing those elements of ops that don't satisfy the
condition for execution*)
fun cond_not_satisfied(ops,ievs) =
List.filter (fn (id,_,_,_)=> not(check(id,ievs))) ops;

(*returns a list containing those elements of ops that satisfy the condition
of execution*)
fun cond_satisfied(ops,ievs) =
List.filter (fn (id,_,_,_) => (check(id,ievs))) ops;

(*calculates the value for the newly produced data element by adding the
values of the input elements when the input set contains two or more elements
and by changing the sign in front of the value when the input set contains
only a single element*)
fun calculation(one::two::ins, ievs:IEVS) =
get_value(one, ievs) + get_value(two,ievs) + calculation(ins, ievs)
| calculation([one], ievs) = ~(get_value(one,ievs))
| calculation([], ievs) = 0;

(*deletes element ops1 from list ops*)
fun delete(ops, op1) = List.filter (fn ope => (ope<>op1)) ops
```

```
(* returns the operation ope for which Val(ope) is minimal,fails if it is
called with an empty list *)
fun SelMin Val (op1::op2::tail) = if Val(op1) < Val(op2)
           then (SelMin Val (op1::tail)) else (SelMin Val (op2::tail))
| SelMin Val (op1::[]) = op1
| SelMin Val ([]) = raise Match;

(* returns the cost of an operation *)
fun Cost((id,out,ins,(dur,cost,fprob)):OP) = cost;

(* returns the duration of an operation *)
fun Dur((id,out,ins,(dur,cost,fprob)):OP) = dur;

(* returns the failure probability of an operation *)
fun Fprob((id,out,ins,(dur,cost,fprob)):OP) = fprob;

(* Selects an arbitrary element from the list l *)
fun select_random(l)  = List.nth (l,discrete(0,(List.length l)-1))
```

Bibliography

[1] W.M.P. van der Aalst. Designing Workflows Based on Product Structures. In K. Li, S. Olariu, Y. Pan, and I. Stojmenovic, editors, *Proceedings of the ninth IASTED International Conference on Parallel and Distributed Computing Systems*, pages 337–342. IASTED/Acta Press, Anaheim, 1997.

[2] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.

[3] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[4] W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.

[5] W.M.P. van der Aalst. On the Automatic Generation of Workflow Processes based on Product Structures. *Computers in Industry*, 39:97–111, 1999.

[6] W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.

[7] W.M.P. van der Aalst. Reengineering Knock-out Processes. *Decision Support Systems*, 30(4):451–468, 2001.

[8] W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Workflow Modeling using Proclets. In O. Etzion and P. Scheuermann, editors, *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 198–209. Springer-Verlag, Berlin, 2000.

[9] W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, 10(4):443–482, 2001.

[10] W.M.P. van der Aalst and P.J.S. Berens. Beyond Workflow Management: Product-Driven Case Handling. In S. Ellis, T. Rodden, and I. Zigurs, editors, *International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2001)*, pages 42–51. ACM Press, New York, 2001.

[11] W.M.P. van der Aalst, P. de Crom, R. Goverde, K.M. van Hee, W. Hofman, H. Reijers, and R.A. van der Toorn. ExSpect 6.4: An Executable Specification Tool for Hierarchical Colored Petri Nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 455–464. Springer-Verlag, Berlin, 2000.

[12] W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.

[13] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2004.

[14] W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. BPM Center report BPM-08-02, BPMcenter.org, 2008.

[15] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[16] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns Home Page. http://www.workflowpatterns.com.

[17] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[18] W.M.P. van der Aalst and S. Jablonski. Dealing with Workflow Change: Identification of Issues and Solutions. *International Journal of Computer Systems, Science, and Engineering*, 15(5):267–276, 2000.

[19] W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Genetic Process Mining. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 48–69. Springer-Verlag, Berlin, 2005.

[20] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

[21] W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.

[22] W.M.P. van der Aalst and A.J.M.M. Weijters. Process Mining: A Research Agenda. *Computers in Industry*, 53(3):231–244, 2004.

[23] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. QUT Technical report, FIT-TR-2003-03, Queensland University of Technology, Brisbane, 2003. (Accepted for publication in IEEE Transactions on Knowledge and Data Engineering.).

[24] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[25] W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.

[26] F.B. Abreu and R. Carapuca. Candidate Metrics for Object-Oriented Software within a Taxonomy Framework. *Journal of Systems and Software*, 23(1):87–96, 1994.

[27] M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Facilitating Flexibility and Dynamic Exception Handling in Workflows. In O. Belo, J. Eder, O. Pastor, and J. Falcao e Cunha, editors, *Proceedings of the CAiSE'05 Forum*, pages 45–50. FEUP, Porto, Portugal, 2005.

[28] G. Adomavicius and A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[29] A. Agostini and G. De Michelis. Simple Workflow Models. In W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors, *Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98)*, volume 98/7 of *Computing Science Reports*, pages 146–164. Eindhoven University of Technology, Eindhoven, 1998.

[30] A. Agostini and G. De Michelis. Improving Flexibility of Workflow Management Systems. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 218–234. Springer-Verlag, Berlin, 2000.

[31] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Proceedings of the 6th International Conference on Extending Database Technology*, pages 469–483, 1998.

[32] M. Al-Mashari, Z. Irani, and M. Zairi. Business Process Reengineering: A Survey of International Experience. *Business Process Management Journal*, 7(5):437–455, 2001.

[33] T.A. Aldowaisan and K. Gaafar L. Business Process Reengineering: An approach for Process Mapping. *Omega*, 27(5):515–524, 1999.

[34] A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 2006.

[35] Arbeidsomstandighedenbesluit (Working Conditions Decree). http://wetten.overheid.nl (in Dutch).

[36] M. Attaran. Exploring the Relationship between Information Technology and Business Process Reengineering. *Information & Management*, 41(5):585–596, 2004.

[37] C.Y. Baldwin and K.B. Clark. *Design Rules. Volume 1. The Power of Modularity.* MIT Press, Cambridge, 2000.

[38] L. Baresi, F. Casati, S. Castano, M. Fugini, I. Mirbel, and B. Pernici. WIDE Workflow Development Methodology. In D. Georgakopoulos, W. Prinz, and A.L. Wolf, editors, *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration (WACC'99)*, pages 19–28, New York, NY, USA, 1999. ACM.

[39] L. Baresi, F. Casati, S. Castano, M. Fugini, I. Mirbel, B. Pernici, and G. Pozzi. Workflow Design Methodology. In P. Grefen, B. Pernici, and G. Sanchez, editors, *Database Support for Workflow Management: the WIDE Project*, pages 47–94. Kluwer Academic Publishers, 1999.

[40] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton NJ, 1957.

[41] C. Berge. *Graphs and Hypergraphs*. North-Holland Publishing Company, Amsterdam, 1973.

[42] A. Bernstein, M. Klein, and T.W. Malone. The Process Recombinator: A Tool for Generating New Business Process Ideas. In *Organizing Business Knowledge: The MIT Process Handbook*, pages 403–422. MIT Press, Massachusetts, 2003.

[43] J.W.M. Bertrand, J.C. Wortmann, and J. Wijngaard. *Production Control: A Structural and Design Oriented Approach*, volume 11 of *Manufacturing Research and Technology*. Elsevier Science Publishers, Amsterdam, 1990.

[44] K. Bhattacharya, N. Caswell, S. Kumaran, A. Nigam, and F. Wu. Artifact-Centered Operational Modeling: Lessons learned from Engagements. *IBM Systems Journal*, 46(4):703–721, 2007.

[45] K. Bhattacharya, C.E. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Proceedings of the 5th International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, Berlin, 2007.

[46] J.M. Bieman and B.-K. Kang. Cohesion and Reuse in an Object-Oriented System. In *Proceedings of the ACM Symposium on Software Reusability (SIGSOFT'95)*, pages 259–262, 1995.

[47] J.M. Bieman and B.-K. Kang. Measuring Design-Level Cohesion. *IEEE Transactions on Software Engineering*, 24(2):111–124, 1998.

[48] J.M. Bieman and L.M. Ott. Measuring Functional Cohesion. *IEEE Transactions on Software Engineering*, 20(8):644–657, 1994.

[49] D. Billsus and M.J. Pazzani. Learning Collaborative Information Filters. In J.W. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 46–54. Morgan Kaufmann Publishers Inc., 1998.

[50] A.F. Blackwell. Ten Years of Cognitive Dimensions in Visual Languages and Computing. *Journal of Visual Languages and Computing*, 17(4):285–287, 2007.

[51] C. Boutilier, T. Dean, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[52] L.C. Briand, J.W. Daly, and J. Wust. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering*, 3(1):65–117, 1998.

[53] L.C. Briand, J.W. Daly, and J. Wüst. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999.

[54] L.C. Briand, S. Morasca, and V.R. Basili. Property-Based Software Engineering Measurement. *IEEE Transactions on Software Engineering*, 22(6):68–86, 1996.

[55] E.D. Browne, M. Schrefl, and J.R. Warren. A Two Tier, Goal-Driven Workflow Model for the Healthcare Domain. In *Proceedings of the 5th International Conference on Enterprise Information Systems*, volume III - Information Systems Analysis and Specification, pages 32–39, 2003.

[56] E.D. Browne, M. Schrefl, and J.R. Warren. Activity Crediting in Distributed Workflow Environments. In *Proceedings of the 6th International Conference on Enterprise Information Systems*, volume III - Information Systems Analysis and Specification, pages 245–253, 2004.

[57] E.D. Browne, M. Schrefl, and J.R. Warren. Goal-Focused Self-Modifying Workflow in the Healthcare Domain. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, pages 1–10, 2004.

[58] R. Burke. The Wasabi Personal Shopper: A Case-Based Recommender System. In *Proceedings of the 11th Conference on Innovative Applications of Artificial Intelligence*, pages 844 – 849. American Association for Artificial Intelligence, 1999.

[59] R. Burke. A Case-Based Reasoning Approach to Collaborative Filtering. In E. Blanzieri and L. Portinale, editors, *Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning (EWCBR 2000)*, volume 1898 of *Lecture Notes in Computer Science*, pages 370–379. Springer-Verlag, Berlin, 2000.

[60] P. Calvert. An Advanced BPR Methodology with Integrated, Computer-based Tools. In B.C. Glasson et al., editor, *Business Process Re-engineering: Information Systems Opportunities and Challenges*, pages 161–170. Elsevier Science, Amsterdam, 1994.

[61] D.N. Card, V.E. Church, and W.W. Agresti. An Empirical Study of Software Design Practices. *IEEE Transactions on Software Engineering*, 12(2):264–271, 1986.

[62] J. Cardoso. Control-flow Complexity Measurement of Processes and Weyuker's Properties. In *Proceedings of the 6th International Enformatika Conference (IEC 2005)*, pages 213–218, Budapest, Hungary, 2005. International Academy of Sciences.

[63] J. Cardoso. How to Measure the Control-flow Complexity of Web Processes and Workflows. In L. Fischer, editor, *Workflow Handbook 2005*, pages 199–212. Future Strategies, Lighthouse Point, 2005.

[64] J. Cardoso. Process control-flow complexity metric: An empirical validation. In *IEEE International Conference on Services Computing (IEEE SCC 06)*, pages 167–173, Chicago, USA, 2006. IEEE Computer Society.

[65] J. Cardoso, J. Mendling, G. Neumann, and H.A. Reijers. A Discourse on Complexity of Process Models. In J. Eder and S. Dustdar, editors, *Proceedings of the BPM 2006 Workshops, Workshop on Business Process Design BPI 2006, Lecture Notes in Computer Science Volume 4103*, pages 115–126, September 2006.

[66] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211–238, 1998.

[67] S.R. Chidamber and C.F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[68] W.-K. Ching and M.K. Ng. *Markov Chains: Models, Algorithms and Applications*. Springer Science, New York, USA, 2006.

[69] Workflow Management Coalition. WFMC Home Page. http://www.wfmc.org.

[70] J. Cohen, P. Cohen, S.G. West, and L.S. Aiken. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Lawrence Erlbaum, London, 2003.

[71] S.D. Conte, H.E. Dunsmore, and V.Y. Shen. *Software Engineering Metrics and Models*. Benjamin/Cummings Publishing Company, Inc., 1986.

[72] J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[73] COSA. COSA website. http://www.cosa-bpm.com.

[74] CPN Tools Home Page. http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[75] B. Curtis, M.I. Kellner, and J. Over. Process modeling. *Communications of the ACM*, 35(9):75–90, 1992.

[76] P. Dadam, M. Reichert, S. Rinderle, H. Acker, M. Jurisch, U. Kreher, K. Gser, and M. Lauer. ADEPT2 - Next Generation Process Management Technology. In *Proceedings of the 4th Heidelberg Innovation Forum (Invited Paper)*, pages 1–10. D.punkt Verlag, 2007.

[77] M. Daneva, R. Heib, and A.-W. Scheer. Benchmarking Business Process Models. IWi Research Report 136, Institute for Information Systems, University of the Saarland, Germany, 1996.

[78] A. Datta. Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. *Information Systems Research*, 9(3):275–301, 1998.

[79] T.H. Davenport. *Process Innovation : Reengineering Work through Information Technology*. Harvard Business School Press, Boston, 1993.

[80] J. Dehnert. *A Methodology for Workflow Modeling: From Business Process Modeling Towards Sound Workflow Specification*. PhD thesis, TU Berlin, Berlin, Germany, 2003.

[81] J. Dehnert and P. Rittgen. Relaxed Soundness of Business Processes. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer-Verlag, Berlin, 2001.

[82] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.

[83] B.F. van Dongen. *Process Mining and Verification*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2007.

[84] B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.

[85] P. Dourish, W.K. Edwards, J. Howell, A. LaMarca, J. Lamping, K. Petersen, M. Salisbury, D. Terry, and J. Thornton. A Programming Model for Active Documents. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 41–50. ACM Press, 2000.

[86] P. Dourish, W.K. Edwards, A. LaMarca, J. Lamping, K. Petersen, M. Salisbury, D.B. Terry, and J. Thornton. Extending Document Managment Systems with User-Specific Active Properties. *ACM Transaction on Information Systems*, 18(2):140–170, 2000.

[87] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset Nets Between Decidability and Undecidability. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115, Aalborg, Denmark, July 1998. Springer-Verlag.

[88] C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T Nets. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Lectures on Concurrency and Petri Nets*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310, Prague, Czech Republic, July 1999. Springer-Verlag.

[89] M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons, 2005.

[90] P.P.J. Durlinger and R.P.H.G. Bemelmans. *Logistical Techniques*. P.P.J. Durlinger, 1999. (in Dutch).

[91] J. Eder, G. Kappel, and M. Schrefl. Coupling and Cohesion in Object-Oriented Systems. Technical Report, University of Klagenfurt, 1992.

[92] A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.

[93] C.A. Ellis and K. Keddara. A Workflow Change Is a Workflow. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 201–217. Springer-Verlag, Berlin, 2000.

[94] T.J. Emerson. A Discriminant Metric for Module Cohesion. In *Proceedings of the 7th International Conference on Software Engineering (ICSE-7)*, pages 294–303, Piscataway, NJ, USA, 1984. IEEE Press.

[95] R. Eshuis and J. Dehnert. Reactive Petri nets for Workflow Modeling. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 295–314. Springer-Verlag, Berlin, 2003.

[96] N. Fenton and A. Melton. Deriving Structurally Based Software Measures. *Journal of Systems and Software*, 123:177–187, 1990.

[97] FileNET. FileNET website. http://www.filenet.com.

[98] M. Fitting. *First-order Logic and Automated Theorem Proving*. Springer-Verlag, Berlin, 1990.

[99] D.W. Fogarty, J.H. Blackstone, and T.R. Hoffmann. *Production and Inventory Management*. South-Western Publishing Co., Cincinnati, 2nd edition, 1991.

[100] G. Gallo, G. Longo, S. Pallotino, and S. Nguyen. Directed Hypergraphs and Applications. *Discrete Applied Mathematics*, 42:177–201, 1993.

[101] F. García, F. Ruiz, M. Piattini, G. Confora, and C.A. Visaggio. Framework for the Modeling and Evaluation of Software Processes. *Journal of Systems Architecture*, 52:627–639, 2006.

[102] M. Genero, L. Jimènez, and M. Piattini. Measuring the Quality of Entity Relationship Diagrams. In A.H.F. Laender, S.W. Liddle, and V. Storey, editors, *Proceedings of the 19th International Conference on Conceptual Modeling (ER 2000)*, volume 1920 of *Lecture Notes in Computer Science*, pages 513–522. Springer-Verlag, Berlin, 2000.

[103] M. Genero, J. Olivas, M. Piattini, and F. Romero. Knowledge Discovery For Predicting Entity Relationship Diagram Maintainability. In *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE 2000)*, pages 203–211. Knowledge Systems Institute, 2000.

[104] M. Genero, M. Piattini, and C. Calero. An Approach to Evaluate the Complexity of Conceptual Database Models. In *Proceedings of the 2nd European Software Measurement Conference (FESMA 2000)*, pages 1–10, 2000.

[105] M. Genero, M. Piattini, and C. Calero. Formalization of Metrics for Conceptual Data Models. In *Proceedings of the 5th Annual UK Academy for Information Systems Conference (UKAIS 2000)*, pages 99–100. McGraw-Hill, 2000.

[106] M. Genero, M. Piattini, C. Calero, and M. Serrano. Measures to Get Better Quality Databases. In *Proceedings of the 2nd International Conference on Enterprise Information Systems (ICEIS2000)*, pages 49–55, 2000.

[107] M. Genero, G. Poels, and M. Piattini. Defining and Validating Measures for Conceptual Data Model Quality. In A. Banks Pidduck, J. Mylopoulos, C.C. Woo, and M. Tamer Özsu, editors, *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE 2002)*, volume 2348 of *Lecture Notes in Computer Science*, pages 724–727. Springer-Verlag, Berlin, 2002.

[108] C.E. Gerede, K. Bhattacharya, and J. Su. Static Analysis of Business Artifact-centric Operational Models. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007)*, pages 133–140. IEEE Computer Society, Washington, DC, USA, 2007.

[109] C.E. Gerede and J. Su. Specification and Verification of Artifact Behaviors in Business Process Models. In B.J. Krämer, K.-J. Lin, and P. Narasimhan, editors, *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC 2007)*, volume 4749 of *Lecture Notes in Computer Science*, pages 181–192. Springer-Verlag, Berlin, 2007.

[110] R. Gray, B. Carey, N. McGlynn, and A. Pengelly. Design Metrics for Database Systems. *BT Technology Journal*, 9(4):69–79, 1991.

[111] T.R.G. Green and M. Petre. Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework. *Journal of Visual Languages and Computing*, 7(2):131–174, 1996.

[112] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business Process Intelligence. *Computers in Industry*, 53(3):321–343, 2004.

[113] J.L. Gross and J. Yellen. *Handbook of Graph Theory*. CRC Press, Boca Raton, Florida, 2004.

[114] V. Gruhn and R. Laue. Complexity Metrics for Business Process Models. In W. Abramowicz and H.C. Mayer, editors, *Proceedings of the 9th international conference on business information systems (BIS 2006)*, volume 85 of *Lecture Notes in Informatics*, pages 1–12, 2006.

[115] A.S. Gücegliogu and O. Demirörs. Using Software Quality Characteristics to Measure Business Process Quality. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the 3rd International Conference on Business Process Management (BPM 2005)*, volume 3649 of *Lecture Notes in Computers Science*, pages 374–379, 2005.

[116] A.L.P. Guedes and L. Markenzon. Directed Hypergraph Planarity. *Pesquisa Operacional*, 25(3):383–390, 2005.

[117] A. Gunasekaran and B. Kobu. Modelling and Analysis of Business Process Reengineering. *International Journal of Production Research*, 40(11):2521–2546, 2002.

[118] C.W. Günther and Wil M.P. van der Aalst. Mining Activity Clusters From Low-level Event Logs. BPM Center Report BPM-06-11, BPMcenter.org, 2006.

[119] J.F. Hair, W.C. Black, B.J. Babin, R.E. Anderson, and R.L. Tatham. *Multivariate Data Analysis*. Pearson Prentice Hall, New Jersey, 2006.

[120] M.H. Halstead. *Elements of Software Science*. Elsevier, Amsterdam, 1987.

[121] M. Hammer and J. Champy. *Reengineering the Corporation*. Nicolas Brealey Publishing, London, 1993.

[122] G.A. Hansen. *Automated Business Process Reengineering: Using the Power of Visual Simulation Strategies to Improve Performance and Profit*. Prentice-Hall,Englewood Cliffs, 1997.

[123] R. Harrison, S.J. Counsell, and R.V. Nithi. An Evaluation of the MOOD Set of Object-Oriented Software Metrics. *IEEE Transactions on Software Engineering*, 24(6):491–496, 1998.

[124] K.M. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.

[125] K.M. van Hee, N. Sidorova, and M. Voorhoeve. Generalised Soundness of Workflow Nets Is Decidable. In J. Cortadella and W. Reisig, editors, *Application and Theory of Petri Nets 2004*, volume 3099 of *Lecture Notes in Computer Science*, pages 197–215. Springer-Verlag, Berlin, 2004.

[126] S. Henry and D. Kafura. Software Structure Metrics based on Information-flow. *IEEE Transactions on Software Engineering*, 7(5):510–518, 1981.

[127] J. Hernandez. *The SAP R/3 Handbook*. McGraw Hill, 1997.

[128] T. Herrmann, M. Hoffmann, K.U. Loser, and K. Moysich. Semistructured Models are Surprisingly Useful for User-Centered Design. In G. De Michelis, A. Giboin, L. Karsenty, and R. Dieng, editors, *Designing Cooperative Systems. Proceedings of the 4th International Conference on the Design of Cooperative Systems (Coop 2000)*, pages 159–174. IOS Press, Amsterdam, 2000.

[129] A.R. Hevner, S. T. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[130] M. Hitz and B. Montazeri. Measuring Product Attributes of Object-Oriented Systems. In W. Schafer and P. Botella, editors, *Proceedings of the 5th European Software Engineering Conference (ESEC 95)*, volume 989, pages 124–136. Springer-Verlag, Berlin, 1995.

[131] D. Hosmer and S. Lemeshow. *Applied Logistic Regression, 2nd edition*. Wiley & Sons, 2000.

[132] M. Huth and M. Ryan. *Logic in Computer Science: Modeling and Reasoning about Systems*. Cambridge University Press, Cambridge, 2004.

[133] IBG. Home page. http://www.ib-groep.nl.

[134] IBM. IBM WebSphere. www.ibm-306.ibm.com/software/websphere.

[135] IDS Scheer. ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper). IDS Scheer, Saarbrücken, Gemany, http://www.ids-scheer.com, 2002.

[136] E.J. Ignall. A Review of Assembly Line Balancing. *Journal of Industrial Engineering*, 16(4):244–254, 1965.

[137] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1997.

[138] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods*. Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1997.

[139] J. Jiao, M.M. Tseng, Q. Ma, and Y. Zou. Generic Bill-of-Materials-and-Operations for High-Variety Production Management. *Concurrent Engineering: Research and Application*, 8(4):297–322, 2000.

[140] K. Kaan, H.A. Reijers, and P. van der Molen. Introducing Case Management: Opening Workflow Management's Black Box. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 358–367. Springer-Verlag, Berlin, 2006.

[141] D. Kafura. A Survey of Software Metrics. In *ACM '85: Proceedings of the 1985 ACM annual conference on The range of computing : mid-80's perspective*, pages 502–506, New York, NY, USA, 1985. ACM Press.

[142] J. Kamphuis. From Product Data Model to Process Model: Algorithms and their Classification. Master's thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 2008.

[143] J. Kamphuis, I. Vanderfeesten, H.A. Reijers, and M. van Hattem. From Product Data Model to Process Model. BETA Working Paper Series, WP 238, Eindhoven University of Technology, Eindhoven, 2008.

[144] B.-K. Kang and J.M. Bieman. Using Design Cohesion to Visualize, Quantify and Restructure Software. In *Proceedings of the 8th International Conference Software Engineering and Knowledge Engineering*, pages 222–229. Knowledge Systems Institute, Skokie IL, 1996.

[145] B.-K. Kang and J.M. Bieman. A Quantitative Framework for Software Restructuring. *Journal of Software Maintenance*, 11:245–284, 1999.

[146] G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.

[147] G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[148] S. Kesh. Evaluating the Quality of Entity Relationship Models. *Information and Software Technology*, 37(12):681–689, 1995.

[149] W.J. Kettinger, J.T.C. Teng, and S. Guha. Business Process Change: A Study of Methodologies, Techniques and Tools. *MIS Quarterly*, 21(1):55–80, 1997.

[150] M. Klein, C. Dellarocas, and A. Bernstein, editors. *Proceedings of the CSCW-98 Workshop Towards Adaptive Workflow Systems*, New York, NY, USA, November 1998. ACM.

[151] M. Klein, C. Dellarocas, and A. Bernstein, editors. *Adaptive Workflow Systems*, volume 9(3-4) of *Special Issue of the Journal of Computer Supported Cooperative Work*, 2000.

[152] M. Kress, J. Melcher, and D. Seese. Introducing Executable Product Models for the Service Industry. In *Proceedings of the 40th Hawaii International Conference on System Sciences (HICSS '07)*, pages 1–10. IEEE Computer Society, 2007.

[153] M. Kress and D. Seese. Executable Product Models - The Intelligent Way. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (ISIC '07)*, pages 1987 – 1992, 2007.

[154] S. Ku and Y.H. Suh. An Investigation of the K-tree Search Algorithm for Efficient Case Representation and Retrieval. *Expert Systems with Applications*, 11(4):571–581, 1996.

[155] V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman and Hall, London, 1995.

[156] A. Kumar and J. Wang. A Framework for Resource-Based Workflow Management. Working Paper, Penn State University, Smeal College of Business, 2008.

[157] J.M. Küster, K. Ryndina, and H. Gall. Generation of Business Process Models for Object Life Cycle Compliance. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Proceedings of the 5th International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 165–181. Springer-Verlag, Berlin, 2007.

[158] A. Lakhotia. Rule-based Approach to Computing Module Cohesion. In *Proceedings of the 15th International Conference on Software Engineering (ICSE '93)*, pages 35–44. IEEE Computer Society Press, Los Alamitos, CA, USA, 1993.

[159] A. LaMarca, W.K. Edwards, P. Dourish, J. Lamping, I. Smith, and J. Thornton. Taking the Work Out of Workflow: Mechanisms for Document-Centered Collaboration. In *Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work (ECSCW'99)*, 1999.

[160] P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 286–305. Springer-Verlag, Berlin, 1998.

[161] A.M. Latva-Koivisto. Finding a Complexity Measure for Business Process Models. Research report, Helsinki University of Technology, Systems Analysis Laboratory, 2001.

[162] J. Lee and B.T. Pentland. Grammatical Approach to Organizational Design. MIT Center for Coordination Science Working Paper 215, 4144, 2000.

[163] S. Limam Mansar and H.A. Reijers. Best Practices in Business Process Redesign: Validation of a Redesign Framework. *Computers in Industry*, 56(5):457–471, 2005.

[164] S. Limam Mansar and H.A. Reijers. Best Practices in Business Process Redesign: Use and Impact. *Business Process Management Journal*, 13(2):193–213, 2007.

[165] S. Limam Mansar, H.A. Reijers, and F. Marir. Applying Business Process Redesign: A Case-based Reasoning Approach. In F. McGrath and D. Remenyi, editors, *Proceedings of the 4th European Conference On Knowledge Management*, pages 635–644. Academic Conferences Limited, Reading, 2003.

[166] R. Liu, K. Bhattacharya, and F.Y. Wu. Modeling Business Contexture and Behavior Using Business Artifacts. In J. Krogstie, A.L. Opdahl, and G. Sindre, editors, *Proceedings of the the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, volume 4495 of *Lecture Notes in Computer Science*, pages 324–339. Springer-Verlag, Berlin, 2007.

[167] T.W. Malone, K. Crowston, J. Lee, and B. Pentland. Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. *Management Science*, 45(3):425–443, 1999.

[168] ING Investment Management. ING IM website. http://www.ingim.com.

[169] A. Martens. Analyzing Web Service Based Business Processes. In M. Cerioli, editor, *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.

[170] H. Mather. *Bills of Materials*. Dow Jones-Irwin, Hornewood, 1987.

[171] T.J. McCabe. A Complexity Measure. *IEEE Transactins on Software Engineering*, 2(4):308–320, 1976.

[172] T.J. McCabe and C.W. Butler. Design Complexity Measurement and Testing. *Communications of the ACM*, 32:1415–1425, 1989.

[173] J.A. McHugh. *Algorithmic Graph Theory*. Prentice-Hall International, Englewood Cliffs, New Jersey, 1990.

[174] J. Mendling. Testing Density as a Complexity Metric for EPCs. Technical Report JM-2006-11-15, Vienna University of Economics and Business Administration, 2006.

[175] J. Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Vienna University of Economics and Business Administration, Vienna, Austria, May 2007.

[176] J. Mendling and W.M.P. van der Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In J. Krogstie, A.L. Opdahl, and G. Sindre, editors, *Proceedings of the the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, volume 4495 of *LNCS*, pages 439–453. Springer-Verlag, Berlin, 2005.

[177] J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the Occurrence of Errors in Process Models based on Metrics. In R. Meersman and Z. Tari, editors, *OTM Conference 2007, Proceedings, Part I*, volume 4803 of *Lecture Notes in Computer Science*, pages 113–130. Springer-Verlag, Berlin, 2007.

[178] J. Mendling, H.A. Reijers, and J. Cardoso. What Makes Process Models Understandable? In G. Alonso, P. Dadam, and M. Rosemann, editors, *Proceedings of the 5th International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, Berlin, 2007.

[179] J. Mendling, H.M.W. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, and G. Neumann. Detection and Prediction of Errors in EPCs of the SAP Reference Model. *Data and Knowledge Engineering*, 64(1):312–329, January 2008.

[180] R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, UK, 1999.

[181] D.M. Min, J.R. Kim, W.C. Kim, D. Min, and S. Ku. IBRC: Intelligent Bank Reengineering System. *Decision Support Systems*, 18(1):97–105, 1996.

[182] D.C. Montgomery and G.C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley and Sons, New York, 1999.

[183] L. Moody. Metrics for Evaluating the Quality of Entity Relationship Models. In T.W. Ling, S. Ram, and M.-L. Lee, editors, *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)*, volume 1507 of *Lecture Notes in Computer Science*, pages 213–225. Springer-Verlag, Berlin, 1998.

[184] S. Morasca. Measuring Attributes of Concurrent Software Specifications in Petri Nets. In *Proceedings of the 6th International Symposium on Software Metrics*, pages 100–110. IEEE Computer Society, 1999.

[185] S. Morasca. Software Measurement. In *Handbook of Software Engineering and Knowledge Engineering - Volume 1: Fundamentals*, pages 239 – 276, Skokie, IL, USA, 2001. Knowledge Systems Institute.

[186] D. Müller, M. Reichert, and J. Herbst. Flexibility of Data-Driven Process Structures. In J. Eder and S. Dustdar, editors, *Proceedings of the 1st International Workshop on Dynamic Process Management (DPM'06)*, volume 4103 of *Lecture Notes in Computer Science*, pages 181–192. Springer-Verlag, Berlin, 2006.

[187] D. Müller, M. Reichert, and J. Herbst. Data-Driven Modeling and Coordination of Large Process Structures. In R. Meersman and Z. Tari, editors, *Proceedings of the 15th International Conference on Cooperative Information Systems (CoopIS'07)*, volume 4803 of *Lecture Notes in Computer Science*, pages 131–149. Springer-Verlag, Berlin, 2007.

[188] D. Müller, M. Reichert, and J. Herbst. A New Paradigm for the Enactment and Dynamic Adaptation of Data-Driven Process Structures. In Z. Bellahsene and M. Léonard, editors, *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE 2008)*, volume 5074 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, Berlin, 2008.

[189] D. Müller, M. Reichert, J. Herbst, D. Köntges, and A. Neubert. COREPRO-Sim: A Tool for Modeling, Simulating and Adapting Data-Driven Process Structures. In M. Dumas, M. Reichert, and M.-C. Shan, editors, *Proceedings of the 6th International Conference on Business Process Management (BPM 2008 Demonstrations)*, volume 5250 of *Lecture Notes in Computer Science*, pages 394–397. Springer-Verlag, Berlin, 2008.

[190] D. Müller, M. Reichert, J. Herbst, and F. Poppa. Data-Driven Design of Engineering Processes with COREPRO-Modeler. In *16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2007)*, pages 376–378. IEEE Computer Society Press, 2007.

[191] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[192] G.J. Myers. *Composite/Structured Design*. Van Nostrand Reinhold, New York, NY, 1978.

[193] N.J.D. Nagelkerke. A Note on a General Definition of the Coefficient of Determination. *Biometrika*, 78(3):691–692, 1991.

[194] A. Nerode and R.A. Shore. *Logic for Applications*. Springer-Verlag, Berlin, 1993.

[195] M. Netjes, I. Vanderfeesten, and H.A. Reijers. 'Intelligent' Tools for Workflow Process Redesign: a Research Agenda. In C. Bussler et al., editor, *(Post-)Proceedings of the 1st International Workshop on Business Process Design (BPD2005)*, volume 3812 of *Lecture Notes in Computer Science*, pages 444–453. Springer-Verlag, Berlin, 2005.

[196] N. Nissanke. *Introductory Logic and Sets for Computer Scientists*. Addison-Wesley, Harlow, England, 1999.

[197] M.E. Nissen. Redesigning Reengineering trough Measurement-driven Inference. *MIS Quarterly*, 22(4):509–534, 1998.

[198] M.E. Nissen. An Experiment to Asses the Performance of a Redesign Knowledge System. *Journal of Management Information Systems*, 17(3):25–43, 2000.

[199] M.E. Nissen. An Intelligent Tool for Process Redesign: Manufacturing Supply-Chain Applications. *The International Journal of Flexible Manufacturing Systems*, 12(4):321–339, 2000.

[200] Oracle. Oracle Home page. http://www.oracle.com.

[201] J.A. Orlicky. Structuring the Bill of Materials for MRP. *Production and Inventory Management*, pages 19–42, Dec 1972.

[202] J.A. Orlicky. *Material Requirements Planning*. McGraw-Hill, New York, 1975.

[203] L.V. Orman. A Model Management Approach to Business Process Reengineering. *Journal of Management Information Systems*, 15(1):187–212, 1998.

[204] L.M. Ott and J.J. Thuss. The Relationship between Slices and Module Cohesion. In *Proceedings of the 12th International Conference on Software Engineering (ICSE '89)*, pages 198–204. IEEE Computer Society Press, Los Alamitos, CA, USA, 1989.

[205] 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, 2007.

[206] Pallas Athena Home Page. http://www.pallasathena.com.

[207] Pallas Athena. *Case Handling with FLOWer: Beyond workflow*. Pallas Athena BV, Apeldoorn, The Netherlands, 2002.

[208] Pallas Athena. *Flower User Manual*. Pallas Athena BV, Apeldoorn, The Netherlands, 2002.

[209] Pallas Athena. *Protos User Manual*. Pallas Athena BV, Plasmolen, The Netherlands, 2004.

[210] S.S. Panwalkar and W. Iskander. A Survey of Scheduling Rules. *Operations Research*, 25:45–61, 1977.

[211] C. Pedrinaci, J. Domingue, and A.K. Alves de Medeiros. A Core Ontology for Business Process Analysis. In S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, editors, *The Semantic Web: Research and Applications, Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, pages 49–64. Springer-Verlag, Berlin, 2008.

[212] B.T. Pentland. Grammatical Models of Organizational Processes. In *Organizing Business Knowledge: The MIT Process Handbook*, pages 191–214. MIT Press, Massachusetts, 2003.

[213] J. Peppard and P. Rowland. *The Essence of Busines Process Re-engineering*. Prentice Hall, New York, 1995.

[214] M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Control to Users*. PhD thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 2008.

[215] M. Pesic and W.M.P. van der Aalst. A Declarative Approach for Flexible Business Processes. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops, Workshop on Dynamic Process Management (DPM 2006)*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180, Berlin, 2006. Springer-Verlag.

[216] M. Pesic, M.H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 287–300. IEEE Computer Society, 2007.

[217] M. Pesic, M.H. Schonenberg, N. Sidorova, and W.M.P. van der Aalst. Constraint-Based Workflow Models: Change Made Easy. In R. Meersman and Z. Tari, editor, *On the Move to Meaningful Internet Systems 2007: OTM 2007 Confederated International Conferences, Part I.*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94, 2007.

[218] M. Piattini, M. Genero, and C. Calero. Data Model Metrics. In *Handbook of Software Engineering and Knowledge Engineering - Volume 2*, pages 239 – 276. Knowledge Systems Institute, Skokie, IL, USA, 2002.

[219] M. Piattini, M. Genero, C. Calero, M. Polo, and F. Ruiz. Database Quality. In M. Piattini and O. Díaz, editors, *Chapter 14 in Advanced Database Technology and Design*, pages 485–509. Artech House, 2000.

[220] E.A.H. Platier. *A Logistical View on Business Processes: BPR and WFM concepts*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1996. (in Dutch).

[221] G. Poyssick and S. Hannaford. *Workflow Reengineering*. Adobe Press, Mountain View, CA, 1996.

[222] Process Mining Home Page. http://www.processmining.org.

[223] ProM Framework. http://prom.sourceforge.net.

[224] F. Puhlmann and M. Weske. Interaction Soundness for Service Orchestrations. In A. Dan and W. Lamersdorf, editors, *Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC 2006)*, volume 4294 of *Lecture Notes in Computer Science*, pages 302–313. Springer, 2006.

[225] F. Puhlmann and M. Weske. Investigations on Soundness Regarding Lazy Activities. In S. Dustdar, J. Fiadeiro, and A. Sheth, editors, *Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes In Computer Science*, pages 81–96. Springer, 2006.

[226] M.L. Puterman. *Markov Decision Processes*. Wiley, New York, 1994.

[227] M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[228] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive Process Management with ADEPT2. In *Proceedings of the 21st INternational Conference on Data Engineering (ICDE 2005)*, pages 1113–1114. IEEE Computer Society, 2005.

[229] H.A. Reijers. Product-Based Design of Business Processes Applied within the Financial Services. *Journal of Research and Practice in Information Technology*, 34(2):34–46, 2002.

[230] H.A. Reijers. A Cohesion Metric for the Definition of Activities in a Workflow Process. In *Proceedings of the 8th CAiSE/IFIP8.1 International workflop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD 2003)*, pages 116–125, 2003.

[231] H.A. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*, volume 2617 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2003.

[232] H.A. Reijers and S. Limam Mansar. Best Practices in Business Process Redesign: An Overview and Qualitative Evaluation of Successful Redesign Heuristics. *OMEGA - The International Journal of Management Science*, 33(4):283–306, 2005.

[233] H.A. Reijers, S. Limam Mansar, and W.M.P. van der Aalst. Product-Based Workflow Design. *Journal of Management Information systems*, 20(1):229–262, 2003.

[234] H.A. Reijers, J. Rigter, and W.M.P. van der Aalst. The Case Handling Case. *International Journal of Cooperative Information Systems*, 12(3):365–391, 2003.

[235] H.A. Reijers and I. Vanderfeesten. Cohesion and Coupling Metrics for Workflow Process Design. In J. Desel, B. Pernici, and M. Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2004.

[236] H.A. Reijers and I.T.P. Vanderfeesten. Cohesion and Coupling Metrics for Workflow Process Design. In J. Desel and B. Pernici and M. Weske, editor, *Proceedings of the 2nd International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 290–305. Springer-Verlag, Berlin, 2004.

[237] V.E. van Reijswoud, H.B.F. Mulder, and J.L.G. Dietz. Communicative Action-Based Business Process and Information Systems Modelling with DEMO. *Information Systems Journal*, 9(2):117–138, 1999.

[238] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer-Verlag, Berlin, 1985.

[239] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

[240] P. Resnick and H.R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56–58, 1997.

[241] Rijkswet op het Nederlanderschap (Statute Law on Dutch Citizenship). http://wetten.overheid.nl (in Dutch).

[242] S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.

[243] E. Rolón Aguilar, F. García, F. Ruiz, and M. Piattini. An Exploratory Experiment to Validate Measures for Business Process Models. In C. Rolland, O. Pastor, and J.-L. Cavarero, editors, *Proceedings of the First International Conference on Research Challenges in Information Science (RCIS 2007)*, pages 271–280. ACM Press, 2007.

[244] E. Rolón Aguilar, F. Ruiz, F. García, and M. Piattini. Applying Software Metrics to evaluate Business Process Models. *CLEI Electronic Journal*, 9, 2006.

[245] E. Rolón Aguilar, F. Ruiz, F. García, and M. Piattini. Towards a Suite of Metrics for Business Process Models in BPMN. In Y. Manolopoulos, J. Filipe, P. Constantopoulos, and J. Cordeiro, editors, *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006): Databases and Information Systems Integration*, pages 440–443, 2006.

[246] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, Reading, MA, USA, 1998.

[247] R.O. Rupp and J.R. Russell. The Golden Rules of Process Redesign. *Quality Progress*, 27(12):85–92, 1994.

[248] K. Ryndina and J.M. Küster. Predicting Coupling of Object-Centric Business Process Implementations. In M. Dumas, M. Reichert, and M.-C. Shan, editors, *Proceedings of the 6th International Conference on Business Process Management (BPM 2008)*, volume 5240 of *Lecture Notes in Computer Science*, pages 148–163. Springer-Verlag, Berlin, 2008.

[249] K. Ryndina, J.M. Küster, and H. Gall. A Tool for Integrating Object Life Cycle and Business Process Modeling. In M. Adams and S.W. Sadiq, editors, *Proceedings of the BPM Demonstration Program at the 5th International Conference on Business Process Management (BPM'07)*, volume 272 of *CEUR Workshop Proceedings*, pages 1–4. CEUR-WS.org, Aachen, 2007.

[250] K. Ryndina, J.M. Küster, and H. Gall. Consistency of Business Process Models and Object Life Cycles. In Thomas Kühne, editor, *Models in Software Engineering, MoDELS 2006 Workshops*, number 4364 in Lecture Notes in Computer Science, pages 80–90. Springer-Verlag, Berlin, 2007.

[251] K. Sarshar and P. Loos. Comparing the Control-flow of EPC and Petri Net from the Enduser Perspective. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the 3rd International Conference on Business Process Management (BPM 2005), Lecture Notes in Computer Science, Volume 3649*, pages 434–439. Springer-Verlag, Berlin, 2005.

[252] M. Sayal, F. Casati, U. Dayal, and M.C. Shan. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.

[253] A.W. Scheer. *Business Process Engineering, ARIS-Navigator for Reference Models for Industrial Enterprises*. Springer-Verlag, Berlin, 1994.

[254] A.W. Scheer. *ARIS: Business Process Modelling*. Springer-Verlag, Berlin, 2000.

[255] H. Schonenberg, B. Weber, B.F. van Dongen, and W.M.P. van der Aalst. Supporting Flexible Processes Through Log-Based Recommendations. In M. Dumas, M. Reichert, and M.-C. Shan, editors, *Proceedings of the 6th International Conference on Business Process Management (BPM 2008)*, volume 5240 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag, Berlin, 2008.

[256] A. Seidmann and A. Sundararajan. The Effects of Task and Information Asymmetry on Business Process Redesign. *International Journal of Production Economics*, 50(2-3):117–128, 1997.

[257] R.W. Selby and V.R. Basili. Analyzing Error-Prone System Structure. *IEEE Transactions on Software Engineering*, 17:141–152, 1991.

[258] V.Y. Shen, T.-J. Yu, S.M. Thebaut, and L.R. Paulsen. Identifying Error-Prone Software. *IEEE Transactions on Software Engineering*, 11(4):317–324, 1985.

[259] M. Shepperd. *Software Engineering Metrics Volume I: Metrics and Validations*. McGraw-Hill, 1993.

[260] S. Siegel and N.J. Castellan. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, New York, 1988.

[261] E.A. Silver, D.F. Pyke, and R. Peterson. *Inventory Management and Production Planning and Scheduling*. John Wiley and Sons, Hoboken, NJ, 1998.

[262] F. Simon, S. Löffler, and C. Lewerentz. Distance Based Cohesion Measuring. In *Proceedings of the 2nd European Software Measurement Conference (FESMA 1999)*, pages 69–83. Technologisch Instituut Amsterdam, 1999.

[263] C. Sørensen. The Augmented Bill of Materials. In K. Schmidt, editor, *Social Mechanisms of Interaction*, pages 221–236. 1994.

[264] C. Spearman. The Proof and Measurement of Association between Two Things. *American Journal of Psychology*, 15:72–101, 1904.

[265] Staffware. *Staffware Process Suite Version 2 – White Paper*. Staffware PLC, Maidenhead, UK, 2003.

[266] W.P. Stevens, G.J. Myers, and L.L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974.

[267] S.X. Sun, J.L. Zhao, J.F. Nunamaker, and O.R. Liu Sheng. Formulating the Data-Flow Perspective for Business Process Management. *Information Systems Research*, 17(4):374–391, 2006.

[268] TIBCO. TIBCO Staffware. http://www.tibco.com.

[269] H.C. Tijms. *A First Course in Stochastic Models*. Wiley, Chichester, England, 2003.

[270] R. van der Toorn. *Component-Based Software Design with Petri nets: An Approach Based on Inheritance of Behavior*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.

[271] D.A. Troy and S.H. Zweben. Measuring the Quality of Structred Designs. *Journal of Systems and Software*, 2:113–120, 1981.

[272] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The Enterprise Ontology. *Knowledge Engineering Review*, 13(1):31–89, 1998.

[273] V.K. Vaishnavi, S. Purao, and J. Liegle. Object-oriented Product Metrics: A Generic Framework. *Information Sciences*, 177:587–606, 2007.

[274] I. Vanderfeesten, W.M.P. van der Aalst, and H.A. Reijers. Modeling a Product-Based Workflow System in CPN tools. In K. Jensen, editor, *Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005)*, volume 576 of *DAIMI*, pages 99–118, Aarhus, Denmark, October 2005. University of Aarhus.

[275] I. Vanderfeesten, J. Cardoso, J. Mendling, H.A. Reijers, and W.M.P. van der Aalst. Quality Metrics for Business Process Models. In L. Fischer, editor, *BPM and Workflow Handbook 2007*, pages 179–190. Future Strategies, Lighthouse Point, Florida, USA, May 2007.

[276] I. Vanderfeesten, J. Cardoso, and H.A. Reijers. A Weighted Coupling Metric for Business Process Models. In J. Eder, S.L. Tomassen, A. Opdahl, G. Sindre, editor, *Proceedings of the CAiSE 2007 Forum*, volume 247 of *CEUR Workshop Proceedings*, pages 41–44, Trondheim, Norway, June 2007.

[277] I. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. Product-Based Workflow Design with Case Handling Systems. BETA Working Paper Series, WP 189, Eindhoven University of Technology, Eindhoven, 2006.

[278] I. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. An Evaluation of Case Handling Systems for Product-Based Workflow Design. In V. Pedrosa, editor, *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS 2007)*, pages 39–46. INSTICC, 2007.

[279] I. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. Evaluating Workflow Process Designs using Cohesion and Coupling Metrics. *Computers in Industry*, 59(8):420–437, 2008.

[280] I. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. Product-Based Workflow Support: A Recommendation Service for Dynamic Workflow Execution. BPM Center Report BPM-08-03, BPMcenter.org, 2008.

[281] I. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. Product-Based Workflow Support: Dynamic Workflow Execution. In Z. Bellahsène and M. Léonard, editors, *Proceedings of the 20th Conference on Advanced Information Systems Engineering (CAiSE'08)*, volume 5074 of *Lecture Notes in Computer Science*, pages 571–574. Springer-Verlag, Berlin, 2008.

[282] I. Vanderfeesten, H.A. Reijers, J. Mendling, W.M.P. van der Aalst, and J. Cardoso. On a Quest for Good Process Models: The Cross-Connectivity Metric. In Z. Bellahsène and M. Léonard, editors, *Proceedings of the 20th Conference on Advanced Information Systems Engineering (CAiSE'08)*, volume 5074 of *Lecture Notes in Computer Science*, pages 480–494. Springer-Verlag, Berlin, 2008.

[283] E.A. van Veen. *Modeling Product Structures by Generic Bills-of-Materials*. PhD thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 1991.

[284] E.A. van Veen and J.C. Wortmann. Generative Bill of Material Processing Systems. *Production Planning and Control*, 3(3):314–326, 1992.

[285] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.

[286] H.M.W. Verbeek, B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Interoperability in the ProM Framework. In T. Latour and M. Petit, editors, *Proceedings of the EMOI-INTEROP Workshop at the 18th International Conference on Advanced Information Systems Engineering (CAiSE'06)*, pages 619–630. Namur University Press, 2006.

[287] P. Verschuren and R. Hartog. Evaluation in Design-oriented Science. *Quality and Quantity*, 39(6):733–762, 2005.

[288] J. Vogelaar. Design of the Firework Ignitions Permission Process at the Provincie Noord-Brabant. Master's thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 2008. (in Dutch).

[289] T.E. Vollmann, W.L. Berry, and D.C. Whybark. *Manufacturing Planning and Control Systems*. Dow Jones-Irwin, 1984.

[290] R.L. Vos. Improving the Fund Annual Reports Process: a Case Study on the Possibilities of Business Process Redesign. Master's thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 2006. (in Dutch).

[291] Vuurwerkbesluit (Fireworks Decree). http://wetten.overheid.nl (in Dutch).

[292] J. Wang and A. Kumar. A Framework for Document-Driven Workflow Systems. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the 3rd International Conference on Business Process Management (BPM 2005), Lecture Notes in Computer Science, Volume 3649*, pages 285–301. Springer-Verlag, Berlin, 2005.

[293] B. Weber, B.F. van Dongen, M. Pesic, C.W. Guenther, and W.M.P. van der Aalst. Supporting Flexible Processes Through Recommendations Based on History. BETA Working Paper Series, WP 212, Eindhoven University of Technology, Eindhoven, 2007.

[294] A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

[295] A.J.M.M. Weijters, W.M.P. van der Aalst, B.F. van Dongen, C. Gnther, R. Mans, A.K. Alves de Medeiros, A. Rozinat, M. Song, and H.M.W. Verbeek. Process Mining with ProM. In M. Dastani and E. de Jong, editor, *Proceedings of the 19th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, 2007.

[296] Werkloosheidswet (Unemployment Law). http://wetten.overheid.nl (in Dutch).

[297] M. Weske. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In R. Sprague, editor, *Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-34)*. IEEE Computer Society Press, Los Alamitos, California, 2001.

[298] E.J. Weyuker. Evaluating Software Complexity Measures. *IEEE Transactions on Software Engineering*, 14(9):1357–1365, 1988.

[299] S.A. White et al. Business Process Modeling Notation (BPML), Version 1.0, 2004.

[300] R. Wild. *Mass-Production Management*. John Wiley and Sons, London, 1972.

[301] Woflan Home Page. http://www.win.tue.nl/ hverbeek/doku.php?id=projects:woflan:start.

[302] Workflow Patterns Home Page. http://www.workflowpatterns.com.

[303] J.C. Wortmann, D.R. Muntslag, and P.J.M. Timmermans. *Customer-Driven Manufacturing*. Chapman and Hall, London, 1997.

[304] M.T. Wynn. *Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins*. PhD thesis, Queensland University of Technology, Faculty of Information Technology, Brisbane, Australia, 2006.

[305] M.T. Wynn, H.M.W. Verbeek, W.M.P. van der Aalst, A.H. ter Hofstede, and D. Edmond. Business Process Verification - Finally a Reality! *Business Process Management*, 2008 (to appear).

[306] M. Xenos, D. Stavrinoudis, K. Zikouli, and D. Christodoulakis. Object-Oriented Metrics - A Survey. In *Proceedings of the FESMA 2000, Federation of European Software Measurement Associations*, pages 1–10, 2000.

[307] YAWL Home Page. http://www.yawl-system.com/.

[308] E.S. Yu and J. Mylopoulos. From E-R to "A-R" - Modelling Strategic Actor Relationships for Business Process Reengineering. In *Proceedings of the 13th International Conference on the Entity-Relationship Approach*, volume 889 of *Lecture Notes in Computer Science*, pages 548–565. Springer-Verlag, Berlin, 1994.

[309] H. Zuse. *A Framework of Software Measurement*. Walter de Gruyter, Berlin, 1998.

Summary

Organizations improve the efficiency and effectiveness of their business processes by Business Process Redesign (BPR) programs. A BPR program aims at the (re)design of the business process, possibly across departmental borders. Many methods for BPR exist. However, most of them merely support the process designer in drawing the process model. Only a few can be said to be 'intelligent' because they provide actual guidance for the (re)design of a process model by suggesting alternatives or generating a process model automatically. The research described in this thesis focuses on a specific BPR methodology: Product Based Workflow Design (PBWD). PBWD is applied to workflow processes. A workflow process is an administrative process, e.g. the processing of an insurance claim, the request of a loan, or the application for a subsidy.

In this thesis, we present a number of 'intelligent' tools for the product-based design and support of workflow processes. PBWD is a revolutionary (re)design approach (as opposed to evolutionary) and starts the design of a workflow process model from scratch. The workflow product (e.g. a decision on an insurance claim, the approval of a loan, or the grant of a subsidy) is taken as the central concept for the development of a process model. This approach is similar to the use of a Bill-of-Material (BoM) in manufacturing processes. The BoM describes the product to be assembled in terms of its individual parts and the relations between them. A workflow product can also be described by such a structure, which is called the Product Data Model (PDM). In contrast to the BoM, the PDM does not list physical parts but information elements. A PDM contains the data elements that are used to make a decision step-by-step and links these data elements through operations.

Before this research was started, the derivation of process models from a PDM was a manual and therefore time-consuming and error-prone process. In the manual derivation of a process model, data elements and operations are grouped into activities, such that each group is a logical piece of work. The activities are then combined into a process model by defining a flow relation between them. The development of tools that support the process designer in making a process model for supporting a process in a product-based way is the subject of this thesis. We present four different approaches to support the PBWD methodology.

The first approach is a way to evaluate a process model that is (manually) derived from a PDM. Based on the grouping of data elements and operations into activities, a data flow analysis is executed to check for data anomalies. A process model is a correct representation of the corresponding PDM if: (i) it contains a way to produce each data element in the PDM, (ii) it respects the data dependencies in the PDM in its control flow, and (iii) it does not allow to execute an operation more than once in a path from the start to the end of the process. If one of these requirements is violated, the process model is inconsistent with the PDM and may for instance contain an activity in which a data element is used that has not been initialized. This correctness notion is a way to check a process model after it has been designed.

A second contribution of this thesis is the development of seven algorithms to automati-

cally generate process models from a PDM. The algorithms vary on a number of dimensions (described in a classification framework) and result in different process models for the same PDM. From the application of the algorithms to a number of example PDMs, we conclude that the direct translation of a PDM to a (correct and sound) process model is not straightforward. A number of problems related to the structure and behavior of the process model may occur, depending on the specific algorithm used. To correct these problems, the process designer needs to have detailed knowledge on the algorithms and on the process modeling language. However, a process model generated by one of these algorithms may serve as a starting point for further refinement.

To overcome the problems with the algorithms described above, we have examined the direct execution of a PDM. In this approach, no process model is derived and the PDM is executed step-by-step based on the available data elements for a specific case. Several operations may be executable at the same time and only one of them can be selected. To guide the decision, we present two approaches to select the best next step in the execution of the PDM. The first approach introduces simple selection strategies (e.g. lowest cost, shortest processing time) which optimize the selection within the set of executable operations. However, it does not take the effect on future steps into consideration. Hence, the selected operation may lead to a sub-optimal overall execution of the PDM. To solve this issue, we have introduced a mapping from the PDM to a Markov Decision Process (MDP). The solution of such an MDP gives an overall optimal decision strategy. However, since the whole state space of the decision problem has to be calculated, this approach may take a great amount of computation time. The direct execution of a PDM using one of the above strategies provides a more dynamic and flexible support for the workflow process.

Finally, two sets of business process metrics are introduced to evaluate a process model after it has been designed. The first set of metrics consists of cohesion and coupling metrics. The cohesion metric assesses the size (granularity) of the activities in the process model by looking at the cohesiveness of the operations within the activity. The coupling metric measures the coupling of activities. These metrics take the PDM as a basis and focus on the data elements and operations within each activity. Based on the well-known paradigm of loose coupling and strong cohesion from software engineering, the activities in the process model should neither be too small nor too large. Secondly, the Cross-Connectivity (CC-) metric is used to measure the degree of connectivity between the nodes in a process model. The metric considers the cognitive effort a user has to make to understand a process model. It focuses on the connections between activities in the process model and assigns a weight to each connection. The CC-metric does not consider data elements and operations but calculates the tightness of the links between the nodes in the process model. Therefore, its application is not restricted to product-based process models only.

The four tools that are described above all support the product based workflow design methodology in an 'intelligent' way. The algorithms for the automatic generation of process models from a PDM autonomically suggest alternative process models, while the correctness notion and the business process metrics help a process designer to evaluate and compare alternative process models. Finally, the direct execution of a PDM supports a product-based workflow process by guiding a case through all the steps in the PDM. To show the feasibility of the ideas presented in this thesis, a number of prototypes have been developed and implemented in the ProM Framework and an evaluation with a number of case studies is reported on.

Samenvatting

Organisaties verbeteren de efficiëntie en effectiviteit van hun bedrijfsprocessen door middel van Business Process Redesign (BPR) programma's. Een BPR programma heeft als doel het (her)ontwerp van een bedrijfsproces, mogelijk over de grenzen van afdelingen heen. Er zijn veel methoden voor BPR, maar de meeste ondersteunen de procesontwerper enkel in het tekenen van het procesmodel. Slechts een paar van deze methoden kunnen 'intelligent' genoemd worden, omdat ze de procesontwerper helpen bij het maken van een procesmodel door bijvoorbeeld het suggereren van alternatieven of het automatisch genereren van een procesmodel. Het onderzoek dat in dit proefschrift is beschreven richt zich op een specifieke BPR methode: Product Based Workflow Design (PBWD). PBWD richt zich op het (her)ontwerp van zogenaamde workflow processen. Een workflow proces is een administratief proces, bijvoorbeeld het verwerken van een verzekeringsclaim, de aanvraag van een lening of een subsidieaanvraag.

In dit proefschrift presenteren we een aantal 'intelligente' gereedschappen voor productgedreven ontwerp en ondersteuning van workflow processen. PBWD is een revolutionaire (her)ontwerp aanpak (in tegenstelling tot een evolutionaire aanpak) en begint het ontwerp vanaf nul. Het workflow product (bijvoorbeeld de beslissing ten aanzien van een verzekeringsclaim, de goedkeuring van een lening, of de toekenning van een subsidie) is het uitgangspunt bij het ontwerpen van een procesmodel. De aanpak lijkt op het gebruik van een Bill-of-Material (BoM) bij productieprocessen. Zo'n BoM beschrijft hoe het fysieke product gemaakt wordt uit zijn onderdelen. Een workflow product kan met een soortgelijke structuur beschreven worden, die het Product Data Model (PDM) wordt genoemd. In tegenstelling tot de BoM beschrijft een PDM geen fysieke onderdelen maar gegevens. Een PDM bevat de 'data elementen' die gebruikt worden om een beslissing te maken en beschrijft de relaties tussen deze data elementen door middel van 'operaties'.

Voordat dit onderzoek plaatsvond werd het afleiden van procesmodellen op basis van een PDM handmatig gedaan en was het daarom een tijdrovend en foutgevoelig proces. In de handmatige afleiding van een procesmodel worden data elementen en operaties gegroepeerd tot activiteiten, zodanig dat iedere groep een logische eenheid van werk vormt. Deze activiteiten worden dan gecombineerd tot een proces model door het definiëren van een 'flow' relatie tussen de activiteiten. Het ontwikkelen van gereedschappen die de procesontwerper ondersteunen en helpen in dit ontwerpproces is het onderwerp van dit proefschrift. We presenteren vier verschillende gereedschappen om de PBWD methode te ondersteunen.

De eerste aanpak is een manier om een process model dat (handmatig) van een PDM is afgeleid te evalueren. Een 'dataflow' analyse, gebaseerd op de groepering van data elementen en operaties, wordt uitgevoerd om het procesmodel te controleren op fouten in de data-stroom. Een procesmodel is een correcte weergave van het bijbehorende PDM als het (i) een manier bevat om ieder data element uit het PDM te maken, (ii) de afhankelijkheden tussen data elementen in het PDM respecteert en (iii) geen mogelijkheid biedt om een operatie meerdere malen uit te voeren in eenzelfde pad van het begin van het proces tot het eind. Als aan één van deze voorwaarden niet voldaan is, dan is het procesmodel niet consistent met het PDM en kan het bijvoorbeeld een activiteit bevatten waarin een data element gebruikt wordt dat nog niet geïnitialiseerd is. Deze notie van correctheid is een manier om een procesmodel na ontwerp te controleren.

Een tweede bijdrage van dit proefschrift is de ontwikkeling van zeven algoritmes die automatisch een procesmodel genereren op basis van een PDM. De algoritmes verschillen in een aantal dimensies die beschreven zijn in een classificatie raamwerk en resulteren in verschillende procesmodellen voor hetzelfde PDM. We concluderen dat de directe vertaling van een PDM naar een (correct en 'sound') procesmodel niet eenvoudig is omdat, afhankelijk van het specifieke algoritme, een aantal problemen kunnen voorkomen die gerelateerd zijn aan de structuur en het gedrag van het procesmodel. Om deze problemen te verhelpen dient de procesontwerper gedetailleerde kennis te hebben van de algoritmes en de modelleertaal. Het door één van de algoritmes gegenereerde procesmodel kan ook als startpunt gebruikt worden voor verdere verfijning en (her)ontwerp.

Om de hierboven beschreven problemen met de algoritmes te omzeilen beschouwen we de directe uitvoering van een PDM. In deze aanpak wordt geen procesmodel meer afgeleid en wordt het PDM stap-voor-stap uitgevoerd op basis van de beschikbare data elementen voor een specifiek geval. Op hetzelfde moment kunnen meerdere operaties uitvoerbaar zijn. Daarom presenteren we twee benaderingen om de beste volgende stap te kiezen in de uitvoering van het PDM. De eerste aanpak introduceert een aantal eenvoudige selectie strategieën (bijvoorbeeld laagste kosten of kortste doorlooptijd) om de beste keuze binnen de verzameling van uitvoerbare operaties te maken. Deze aanpak houdt echter geen rekening met het effect van de keuze op toekomstige stappen. Daarom kan de selectie van de beste operatie binnen de verzameling van uitvoerbare operaties tot een sub-optimale keuze met betrekking tot de hele uitvoering van het PDM leiden. Om dit probleem op te lossen hebben we een vertaling van een PDM naar een Markov beslissingsprobleem (MDP) geïntroduceerd. De oplossing van zo'n MDP geeft een optimale beslissingsstrategie voor de hele uitvoering van het PDM. Echter, omdat de volledige toestandsruimte van het beslissingsprobleem uitgerekend moet worden is deze benadering tijdrovend. Het direct uitvoeren van een PDM op basis van deze selectie strategieën biedt een meer dynamische en flexibele ondersteuning voor het workflow proces.

Tot slot introduceren we twee verzamelingen van bedrijfsprocesmaten die gebruikt kunnen worden om een procesmodel te evalueren nadat het ontworpen is. De eerste bestaat uit 'cohesie'- en 'koppelings'-maten. De cohesie-maat bepaalt de grootte (granulariteit) van activiteiten in het procesmodel door naar de samenhang van operaties in een activiteit te kijken. De koppelings-maat drukt de mate van koppeling tussen activiteiten uit. Deze maten gaan uit van het PDM en beschouwen de data elementen en operaties in iedere activiteit. Gebaseerd op het bekende paradigma van zwakke koppeling en sterke cohesie uit software engineering wordt ernaar gestreefd dat activiteiten niet te klein noch te groot zijn. Ten tweede gebruiken we de Cross-Connectivity (CC-) maat om de verbondenheid tussen de onderdelen in een procesmodel te meten. Deze maat karakteriseert de cognitieve moeite die een gebruiker moet doen om een procesmodel te begrijpen en weegt de verbindingen tussen de verschillende activiteiten in het procesmodel. De CC-maat kijkt niet naar data elementen en operaties, maar berekent de sterkte van de relaties tussen de elementen in het procesmodel. Daarom is de toepassing van de CC-maat niet alleen geschikt voor product-gedreven procesmodellen.

De vier benaderingen die hierboven beschreven zijn ondersteunen de product-gedreven workflow ontwerpmethode op een 'intelligente' manier. De zeven algoritmes die automatisch een procesmodel genereren vanuit een PDM suggereren alternatieve procesmodellen uit zichzelf, terwijl de notie van correctheid en de bedrijfsprocesmaten een procesontwerper helpen bij het achteraf evalueren en vergelijken van alternatieve procesmodellen. Tot slot kan de directe uitvoering van een PDM een product-gedreven workflow proces ondersteunen door een instantie van het proces stap voor stap door het PDM te leiden. Om de haalbaarheid en uitvoerbaarheid van de aanpakken beschreven in dit proefschrift aan te tonen zijn enkele prototypes ontwikkeld en geïmplementeerd in het ProM Framework en wordt een aantal case studies beschreven waarin de voorgestelde gereedschappen zijn toegepast.

Curriculum Vitae

Irene Vanderfeesten was born in Weert, the Netherlands, on September 9th, 1980. After she completed her secondary education (Gymnasium) at the Philips van Horne Scholengemeenschap in Weert in 1998, she started to study Computer Science at Eindhoven University of Technology. She soon discovered that her interests were in combining the technical aspect of computer science with human and business related perspectives. Therefore, she specialized in business information systems. She conducted her final project in the Information Systems group of the department of Industrial Engineering and Innovation Sciences. After receiving her Master of Science degree in Computer Science in 2004, she continued to work in the same group on a research project with prof.dr.ir. Wil van der Aalst and dr.ir. Hajo Reijers. The aim of the project was to develop 'intelligent' tools for workflow process design with a focus on the Product Based Workflow Design methodology. This thesis summarizes the results of this research project.

Index