Managing Variability in Process-Aware Information Systems

by

Marcello La Rosa
BEng, MEng

A dissertation submitted for the degree of
IF49 Doctor of Philosophy

P. Supervisor: Prof. Marlon Dumas
A. Supervisors: Prof. Arthur ter Hofstede, Prof. Jan Mendling

Faculty of Science and Technology
Queensland University of Technology
Brisbane, Australia

25 March 2009
To Chiara
Keywords

Business process management, process-aware information systems, configurable process model, process configuration, reference model, variability, staged configuration, questionnaire, software family, Petri net, Workflow net, configurable Workflow net, EPC, C-EPC, YAWL, C-YAWL.
Abstract

Configurable process models are integrated representations of multiple variants of a process model in a given domain, e.g. multiple variants of a shipment-to-delivery process in the logistics domain. Configurable process models provide a basis for managing variability and for enabling reuse of process models in Process-Aware Information Systems. Rather than designing process models from scratch, analysts can derive process models by configuring existing ones, thereby reusing proven practices.

This thesis starts with the observation that existing approaches for capturing and managing configurable process models suffer from three shortcomings that affect their usability in practice. Firstly, configuration in existing approaches is performed manually and as such it is error-prone. In particular, analysts are left with the burden of ensuring the correctness of the individualized models. Secondly, existing approaches suffer from a lack of decision support for the selection of configuration alternatives. Consequently, stakeholders involved in the configuration of process models need to possess expertise both in the application domain and in the modeling language employed. This assumption represents an adoption obstacle in domains where users are unfamiliar with modeling notations. Finally, existing approaches for configurable process modeling are limited in scope to control-flow aspects, ignoring other equally important aspects of process models such as object flow and resource management.

Following a design science research method, this thesis addresses the above shortcomings by proposing an integrated framework to manage the configuration of process models. The framework is grounded on three original and interrelated contributions: (i) a conceptual foundation for correctness-preserving configuration of process models; (ii) a questionnaire-driven approach for process model configuration, providing decision support and abstraction from modeling notations; (iii) a meta-model for configurable process models covering control-flow, data objects and resources. While the framework is language-independent, an embodiment of the framework in the context of a process modeling language used in practice is also developed in this thesis. The framework was formally defined and validated using four scenarios taken from different domains. Moreover, a comprehensive toolset was implemented to support the validation of the framework.
# Contents

Keywords i

Abstract iii

Contents v

List of Figures ix

Statement of Original Authorship xiii

Acknowledgments xv

## 1 Introduction

1.1 Problem Area ........................................... 1
  1.1.1 Process-Aware Information Systems .................. 1
  1.1.2 Reference Process Models ............................ 3
  1.1.3 Configurable Process Models ......................... 4

1.2 Problem Statement ...................................... 6

1.3 Objectives and Approach ................................ 8

1.4 Publications ............................................ 11

1.5 Outline .................................................. 13

## 2 Literature Review

2.1 Business Process Modeling Languages .................. 15
  2.1.1 Event-Driven Process Chains ........................ 16
  2.1.2 UML Activity Diagrams ............................... 16
  2.1.3 Business Process Modeling Notation ................. 18
  2.1.4 Web Services Business Process Execution Language . 19
  2.1.5 Yet Another Workflow Language .................... 20

2.2 Software Variability Management ........................ 20
  2.2.1 Software Configuration Management ................. 21
  2.2.2 Feature Diagrams ..................................... 22
2.3 Business Process Variability Modeling .............................................. 23
  2.3.1 Configurable Nodes ................................................................... 24
  2.3.2 Model Projection ........................................................................ 28
  2.3.3 Annotation-based Process Variability ........................................... 29
  2.3.4 Hiding and Blocking .................................................................... 34
2.4 Business Process Flexibility .............................................................. 36
2.5 Summary and Discussion ................................................................. 37

3 Conceptual Foundation for Process Model Configuration ................... 39
  3.1 Workflow nets ................................................................................... 40
    3.1.1 Workflow nets: Syntax ................................................................. 40
    3.1.2 Workflow nets: Semantics ............................................................ 42
  3.2 Process Model Configuration ........................................................... 43
  3.3 Correctness-Preserving Configuration .............................................. 45
    3.3.1 Preserving Syntactic Correctness ................................................... 46
    3.3.2 Preserving Semantic Correctness .................................................. 49
  3.4 Configurable EPCs .......................................................................... 52
    3.4.1 EPCs: Syntax ............................................................................. 54
    3.4.2 EPCs: Semantics ...................................................................... 55
    3.4.3 EPCs: Configuration ................................................................. 56
  3.5 Formal relation between C-EPCs and configurable Workflow nets .... 60
  3.6 Correctness-Preserving Configuration of C-EPCs ............................. 67
    3.6.1 Soundness Issues in C-EPC Configuration .................................... 68
    3.6.2 Deriving C-EPC Process Constraints ............................................ 71
    3.6.3 Tool Support ............................................................................. 73
  3.7 Summary and Discussion ................................................................. 74

4 Questionnaire-Based Approach for Variability Modeling ....................... 77
  4.1 Overview of the Approach ............................................................... 78
    4.1.1 Working Example ..................................................................... 79
    4.1.2 Order Dependencies and Domain Constraints ................................ 80
  4.2 Questionnaire Models ..................................................................... 83
  4.3 Generation of Interactive Questionnaires ......................................... 87
  4.4 Tool Support .................................................................................... 92
    4.4.1 Sample Configuration Process ...................................................... 96
    4.4.2 Performance Experiments ............................................................ 98
  4.5 Summary and Discussion ................................................................. 100

5 Questionnaire-Driven Staged Process Configuration .............................. 105
# List of Figures

1.1 The PAIS lifecycle. ................................................................. 2  
1.2 A configurable process model is an integrated representation of multiple process variants. ........................................ 5  
1.3 Configuration and individualization of process models in the PAIS lifecycle. 6  
1.4 Integrated framework to manage the configuration of business process models. .................................................... 9  

2.1 The picture post-production process in EPCs. ....................... 17  
2.2 The four basic concepts of Software Product Lines. ............... 21  
2.3 The feature diagram for picture post-production. ................... 23  
2.4 A possible configuration for the feature diagram of Figure 2.3. ..... 23  
2.5 The configurable version of the EPC model in Figure 2.1. ........ 26  
2.6 The individualized EPC for a project shot on tape and film, edited via negmatching and finished on film. ......................... 28  
2.7 (a) The picture post-production process model in EPC, with logical terms for the budget levels; (b) the model projection for a low budget project. 30  
2.8 (a) The picture post-production process model in annotated BPMN. (b) A configured process model. ............................. 31  
2.9 The relation between the variation point Prepare medium for editing and the Shooting sub-features from Figure 2.3. ................. 32  
2.10 (a) An annotated UML AD modeling an extract of the picture post-production process, where Presence Conditions refer to the features of Figure 2.3. (b) A possible individualized model. 34  
2.11 Three LTSs: (a) the process model for picture post-production, (b) the configured model for a project shot on Tape, edited Online and delivered on Film, and (c) the individualized model. 35  

3.1 The process model for travel form approval. ............................ 41  
3.2 a) A correct configured Petri net. b) An incorrect configured Petri net. 45  
3.3 The acyclic paths identified for \( p_3 \). .............................................. 48  
3.4 Blocking \( t_3 \) in (a) leads to an unsound WF-net (b). ............... 50  
3.5 The process model for travel form approval in C-EPC. ............... 53  
3.6 Merging of start (a) and end (b) events. ................................. 55
3.7 The configured EPC for international travels from the C-EPC of Figure 3.5.

3.8 Mapping C-EPC connectors to Petri net places, transitions and arcs.

3.9 a) The expanded net for the C-EPC of Figure 3.5 and b) its induced Petri net.

3.10 a) The configured EPC obtained from the expanded C-EPC of Figure 3.9 and b) its induced Petri net.

3.11 a) The WF-net configuration induced by the C-EPC configuration is applied to the induced WF-net of Figure 3.9.b. b) The configured net after enforcing $PC_{N(T)}$.

3.12 The relation between configurable EPCs and configurable WF-nets.

3.13 Configuring a function to $OPT$ in C-EPC means optionally hiding the corresponding transition in the induced Petri net.

3.14 Configuring a C-EPC with $OR$ connectors may lead to unsound individualizations.

3.15 Possible split–join mismatches generated by configuring the $OR$ connector.

3.16 The set of markings produced by the $OR$-split transition includes those produced by the $XOR$-split transition and that produced by the $AND$-split transition.

3.17 A non-configurable $OR$-join handles all configurations of the preceding $OR$-split without generating semantic issues.

3.18 An example of sound C-EPC with no configurable joins. Any configured net obtained from this C-EPC will be sound so long as it will be syntactically correct.

3.19 The staged configuration of the C-EPC in Figure 3.18.

4.1 A questionnaire model is composed of questions and domain facts.

4.2 The questionnaire model for the order fulfilment process.

4.3 The partial order over the questions of Fig 4.2.

4.4 Examples of circular dependencies over facts and questions.

4.5 Questionnaire Designer – showing the model in Figure 4.2.

4.6 Questionnaire Designer - detection of a cyclic dependency among 4 facts.

4.7 Quaestio - the interactive questionnaire tool.

4.8 State $s_1$: the only valid questions are $q_1$ and $q_3$.

4.9 State $s_2$: $f_3$ has been forced to true in $q_1$ in order not to violate $DC_2$.

4.10 State $s_6$: $q_6$ and $q_7$ have been skipped as their facts can only be negated.

4.11 Comparison among SCM, FDs, OVM and questionnaire models (QMs).

5.1 The questionnaire-driven approach for staged correctness-preserving configuration of process models.

5.2 The configurable process model for picture post-production in C-EPC, without configurable joins.

5.3 The questionnaire model for picture post-production.
5.4 Choosing low budget has a number of implications on other decisions.  

5.5 The mapping between questionnaire models and process models is achieved by equating each process fact with a boolean expression over domain facts.  

5.6 The effects of choosing TV and Internet as distribution channels on the process model.  

5.7 The EPC for a low budget post-production project, obtained from the model in Figure 5.2.  

5.8 *Synergy* configuration toolset - overview of the software architecture.  

6.1 Two examples of task-role and task-object associations in iEPC.  

6.2 An exemplification of the reference process model for audio editing at the AFTRS.  

6.3 The associations through the range connector.  

6.4 The role-hierarchy model and the object-hierarchy model for the post-production process model.  

6.5 The configurable version of the model in Figure 6.2.  

6.6 The audio editing process model individualized for a project without music.  

6.7 The range of role and object connectors must be aligned with possible changes in the connector’s degree.  

6.8 *C-iEPC Designer* – showing the model in Figure 6.5.
Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed: __________________________________________

Date: __________________________________________
Acknowledgments

Thanks God this journey is over! It has been quite an intense one. So here I come to the end and all I need to write now are the acknowledgments. Interestingly enough, this is the very first section almost everyone is going to read. At least the layman for sure. Let’s get started then.

First and foremost, I would like to express my sincere gratitude to my supervisory team at the Queensland University of Technology. I really appreciated working with Marlon Dumas, Arthur ter Hofstede and Jan Mendling and strove to learn the most out of their different skills. From Marlon I learned what it means not to get lost in words. If you can describe what you have in mind in simple words, you are already half way through. His practical determination, his problem solving capabilities, his ‘reactivity’ to unexpected issues and his continuous stimulation and suggestions really proved beneficial to my work. Arthur gave me the necessary tools to build rigor into my research. His dedication, technical skills, work ethics and deep knowledge of every single topic on the face of the earth was essential for my research to succeed (and I really mean every topic: you can talk with Arthur about Petri nets as well as gossip or movies – you would be surprised how deep he can get into the topic). I will not (easily) forget those sleepless nights spent proving that theorem or proposition... Jan joined the team in the last year of my research, yet he provided all his research experience and field knowledge to help me achieve my goals. I really liked working with him and hope this collaboration can continue beyond the short timeframe during which he supervised me. And we all know we better do research with Jan as he can publish everything he wants!

Special thanks go to Wil van der Aalst from the Eindhoven University of Technology. Wil was perhaps the first person to realize the passion I was putting into my research studies. He managed to control this passion of mine with his technical knowledge (e.g. he told me that some strange ideas of mine simply couldn’t be achieved) and guided me towards the right approach to kick-start my research. Wil gave me the chance to undertake collaborative research with members of his team at TU/e and invited me to spend some time there. Thanks to him I met Florian Gottschalk, with whom I undertook a fruitful collaboration and started a nice friendship.

This thesis wouldn’t have been possible without the support I received from my colleagues and friends – the members of the BPM research group – who provided a welcoming and challenging environment to work in. A special thank you goes to Michael Rosemann for his expert advice which was inspirational to my work. Also, I would like to thank Chunj Ouyang, Guy Redding and Moe Wynn for their valuable feedback over these years, and Michael Adams and Stephan Clemens for our endless discussions on the YAWL system.
If I were an artist I could say Giorgio Bruno was the one who discovered me (but unfortunately I’m not!). I owe the merits of choosing to pursue research to Giorgio, who supervised me during my Laurea thesis at the Politecnico di Torino. He initiated me to the thrilling topic of business process management and made it possible for me to meet Marlon and thus to start my PhD.

Two more people need to be duly acknowledged for their indirect contribution to this thesis. The first is Pascal Raymond. I never met him but his SBDD solver was vital to prove the practical feasibility of my research findings. Thanks God for good software developers! The second person is Katie Shortland and the team from the Australian Film Television & Radio School for providing me with one of the case studies I used in my thesis. It was instructing and very interesting to work with people from such a different domain.

I cannot forget to mention all my friends who surrounded me during these years here in Brisbane. Guido, Alfredo, Jan, Marta, Jessie, Asgeir, Leni, Sandra, Axel, Greta and all the others (I can’t list you all). Thank you guys for the lovely moments we spent together.

My sincere thanks also go to my parents for their constant support, and for inculcating me with a desire to learn which I hope I will be able to hand down to my children. If I found the courage to leave my dear land and go around the world, I really owe this to them. Even from afar they were always close to me, believed in me and shared with me every single moment of my life here in Australia. My parents-in-law deserve special thanks as well. They really loved me as a son and endorsed my choices even if sometimes these were difficult to cope with (like the one to leave Italy with their daughter). For this, I truly admire them.

Last but most importantly, I’d like to thank my wife Chiara – the partner of all my adventures – to whom I dedicate this achievement. She deserves a special thank you for her love, encouragement, trust, understanding, endurance and admirable support over these years, without which I wouldn’t have made it (wow, she seems to be the perfect wife – then I’m glad she is my wife).

Marcello La Rosa
Brisbane, 24 March 2009
Chapter 1

Introduction

Every block of stone has a statue inside it and it is the task of the sculptor to discover it. Carving is easy, you just go down to the skin and stop. I saw the angel in the marble and carved until I set him free—Michelangelo

1.1 Problem Area

Enterprise Systems (ESs) are extremely function-rich in order to cope with a large number of business requirements. These can be functional requirements describing the core functionalities of an ES, and non-functional requirements such as regulations, scalability considerations and performance issues. This profusion of requirements has driven organizations to exploit process technology to describe and manage ESs in a more structured and systematic way [SM01], leading to a trend towards Process-Aware Information Systems (PAISs) [DAH05].

1.1.1 Process-Aware Information Systems

A PAIS has been broadly defined as a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models [DAH05]. Therefore, at the basis of a PAIS is the concept of process model.

A process model essentially describes the logical and temporal order in which organizational tasks have to be performed to realize a given goal [BKR03]. Many are the benefits brought by the use of explicit process models in PAISs. Firstly, they provide a means of communication between managers and business analysts, who determine the structure of the business process, and IT architects, software developers and system administrators who design, implement and operate the technical infrastructure supporting these processes. Secondly, if an ES is driven by process models and not by code, only the former need to be changed to support evolving or emerging business processes [AJ00]. Furthermore, the explicit representation of the processes supported by an organization allows their automated enactment leading to increased efficiencies by streamlining the use of available resources [AH02]. It also enables management support at different levels by means of simulation and monitoring facilities.

The PAIS lifecycle follows a typical top-down approach based on four main phases,
as illustrated in Figure 1.1 [DAH05]. The first stage is process design. Here, intuitive high-level business processes are modeled for scoping the project and capturing and discussing business requirements and process improvement initiatives with domain experts, such as business representatives. These models can be designed at different levels of abstraction, typically by means of Business Process Modeling Tools, which essentially provide a design-oriented graphical editor. The second stage concerns the implementation (encompassing testing and deployment) of process models. Here, those process models that need to be automated are selected and refined into operational (i.e. machine-readable) process specifications, linking the various process tasks to concrete applications and organizational entities. This can be achieved by using a Workflow Management System (WfMS).

![Figure 1.1: The PAIS lifecycle.](image)

The third stage (execution) regards the enactment of executable process specifications. Once a process specification has been deployed to a WfMS, instances can be triggered by the occurrence of a given event (e.g. the arrival of a purchase order). As a result, one or several tasks belonging to the process instance are enabled and dispatched to relevant people/applications that may perform them, until the whole instance is completed. Finally, in the last stage (diagnosis), operational processes are analyzed to identify problems and to find aspects that can be improved. Project Management Tools, used in the last two phases, are able to log process-related data and to interpret them in real-time and offline.

Recently, a holistic understanding of the PAIS lifecycle has been established through the Business Process Management (BPM) discipline [Wes07]. Although BPM is still a maturing discipline, its principles and underlying technology have already been widely accepted as foundational for many organizational and IT-driven re-design projects [Gar09]: through BPM, organizations can derive significant time and cost savings [BKR03]. However, despite this promising scenario, the modeling of PAISs still presents major challenges. At present in fact, the configuration of ESs to meet all the specific requirements of an organization, a new business area or project, consumes significant resources, as the functionality offered by an ES may continuously need to be aligned with the requirements of new settings.

For example, an organization interested in providing a payroll process to different kinds of customers has to re-design its payroll process model from scratch every time to suit the needs of a class of customers, thus consuming significant efforts from business analysts, IT architects and software developers. This is also true for those organizations that want to diversify their domain-specific businesses over different mar-
kets. A PAIS top-down approach based on little or no reuse of process models as that mentioned above, is inefficient and ineffective. The after-effects of failing or delayed ES implementation projects are severe and may even threaten the organization’s existence [RMAR06].

To the experienced BPM practitioner, it is clear that commonalities are frequently found between business processes across business units within the same organization, across companies in a given industry, or even across industries. For example, the term ‘order-to-cash’ is often used to refer to a business process that goes from the moment a purchase order is received by a supplier, to the moment this purchase order has been fulfilled (and the supplier has received the corresponding payment). Virtually all order-to-cash processes include activities related to invoicing, delivery and payment.

However, while all order-to-cash processes look the same, they end up being different. For example, an order-to-cash process for the delivery of goods (e.g., delivery of office supplies) is different from an order-to-cash process for the delivery of services (e.g., delivery of consultancy services). In the first case, there is often a physical delivery that happens at a discrete point in time, and the condition of the goods can be checked upon receipt. On the other hand, the delivery of a service may occur over a long period of time (say, six months). Over this period, several invoices may be issued for the same original purchase order. Also, evaluating the quality of a consultancy service is often trickier than checking the quality of a box of reams of paper. Not surprisingly, the corresponding order-to-cash process models will have many differences.

Despite such differences, companies have a lot to learn from each other when it comes to analyzing and re-designing their order-to-cash processes. It would be inefficient if every time a company engages in modeling and re-designing its order-to-cash process, it did so from scratch, without considering the common practices for this domain.

1.1.2 Reference Process Models

Reference process models driven by industry consortia, such as SCOR [Ste01, HSW04], VICS [VICSA] or ITIL [OGC], go a long way into identifying common process practices, activities and performance metrics in their respective industries. For instance, VICS describes different variants of the order-to-cash process to cope with a variety of logistics options. In a similar vein, but outside the framework of industry consortia, SAP’s R/3 reference model [CK97, Her05] captures recurrent processes supported by the SAP’s Enterprise Resource Planning platform.

Any BPM practitioner should consider having reference models like the above ones in their toolkit, and use them to streamline the modeling of new business processes. In fact, tool vendors have already seized this opportunity by integrating popular reference process models in their product portfolio, as e.g. in IDS Scheer’s Architecture of Integrated Information Systems (ARIS) [Sch00]. However, their offerings do not go beyond pre-populated repositories of process models and provide little guidance and tool support to help analysts to configure these process models to specific needs and contexts. Other commercial products are documented in natural language (e.g. ITIL), thus they fail to unambiguously identify where and how process models can vary [Add07], or only focus on certain aspects of a process (e.g. SCOR [Pol07]). The fundamental problem is that, beyond independent initiatives aimed at creating reference process models in various industries, there is a lack of standard notations and methods for modeling ref-
ference processes and for enabling systematic reuse of reference process models in BPM projects.

One key factor preventing the widespread adoption and systematic reuse of reference process models is the inherent trade-off between standardization on the one hand, and variation and differentiation on the other. On the one hand, process standardization is desirable because it enables the emergence of best-in-kind designs, it leads to uniform interfaces for customers and business partners, and helps in creating synergies and economies of scale through sharing of business improvements, resources, and IT assets. It also simplifies training of process workers, and facilitates their re-deployment across business units. On the other hand, different business units (and, of course, different companies) have different needs and priorities evolving in different directions over time. Also, standardization across business units operating in different regions is often hindered by specific requirements, such as a credit check being required in some regions while not in others, or the said credit check being performed in different ways depending on the region. Finally, standardization across companies operating in the same industry often clashes with the imperative for competitive differentiation. Thus, what is needed is a configuration framework that allows the standardization of processes while at the same time enabling variations.

1.1.3 Configurable Process Models

The concept of configurable process model – recently put forward by Rosemann and van der Aalst [RA07] – is a step forward towards the systematic reuse of (reference) process models. A configurable process model deals with the question: “How to model business processes that are similar to one another in many ways, yet differ in some other ways from one organization, project or industry to another?” This is achieved by merging multiple variants of a process into a single configurable model. In line with methods from the field of Software Product Line Engineering [PBL05], these alternatives are explicitly captured as variation points. A configurable process model provides a starting point to derive process models for a specific setting (e.g. a company, business unit or project) and thus it is an alternative to designing process models from scratch.

To understand how a configurable process model is constructed, let us consider a concrete example. This example is extracted from a repository of process models for the screen business that we have documented in collaboration with several stakeholders in the post-production industry during the course of this research [LHRS08]. In this industry it is difficult to come up with standardized process models because virtually every screen project has its own characteristics and, thus, its own business processes. Yet, these processes share many commonalities.

Figure 1.2 shows two variants of the post-production process, represented in the Business Process Modeling Notation (BPMN) [OMG08]. These variants reflect two common practices in post-production: the first, ‘shooting on tape’, is typically followed when the project is shot on tape, the other, ‘shooting on film’, is followed when the project is shot on film.

Post-production starts with the preparation of the footage for editing and continues with the offline editing. We can observe that this sequence of activities is common to both the practices, regardless of the shooting medium. However, after this, an online editing is carried out if the footage is shot on tape, while a negmatching is carried
out if the footage is shot on film. Online editing is a cheap editing procedure that is well suited for low-budget movies, which typically shoot on tape. Negmatching offers better quality results although it requires higher costs; thus it is more suitable for high-budget productions which typically shoot on film. The choice between online editing and negmatching represents a variability in the post-production process: depending on drivers such as budget, creativity and type of project, one option or the other needs to be taken.

Figure 1.2: A configurable process model is an integrated representation of multiple process variants.

The right-hand side of Figure 1.2 shows a configurable process model for post-production. This model is a merger between the two process variants with a variation point represented by a configurable gateway. This inclusive split gateway has been marked with a thicker border: unlike a “normal” BPMN gateway, it does not represent a choice or a parallel split that will have an effect when the process is executed or simulated. Instead, a configurable gateway represents a design choice that will need to be made by an analyst to adapt the configurable process model to a particular setting, such as a project or an organization. In the post-production example, the configurable gateway captures the fact that one needs to choose for a given screen project whether to select one path (tape shooting) or the other (film shooting), or possibly both.

Therefore a core feature of configurable process models is the explicit representation of variation points and their variants. A variation point can be indicated in different ways, e.g. it could also be a special activity. A configurable process model will typically feature many variation points, each capturing a decision that needs to be taken during process design. An analyst can configure this model by picking the most suitable variant for each variation point. Once all these decisions have been taken, the configured process is individualized by removing those variants that are no longer relevant, leading to an individualized process model. The latter can be used for further analysis, for simulation, or to produce an executable specification for a given set of requirements. Thus, a configurable process model can foster the adoption of common or ‘best’ practices in a given domain, and reduce the modeling effort.

Let us come back to the difference between a variation point, like the configurable gateway in the above example, and a traditional gateway. The decision associated with
1. Introduction

A variation point is a design-time decision. It is not based on data values available when the process is actually performed (e.g., the amount of an expense in a procurement process), but, rather, on the requirements of the project or organization for which the model is to be used. As mentioned before, many can be the drivers behind the configuration of a process: organizational culture, national or regional regulations, compliance requirements, cost, etc. Therefore, to exploit configurable process models in the PAIS lifecycle, the traditional design phase is split into two phases: one where the configurable process model is designed from the consolidation of selected process variants, and another where the model is actually configured and individualized to fit a particular setting, as shown in Figure 1.3.

Figure 1.3: Configuration and individualization of process models in the PAIS lifecycle.

1.2 Problem Statement

Despite their benefits, configurable process models have not yet succeeded in being adopted to capture reference process models. This is primarily because the concept of process model configuration is not yet mature enough. For example, it is still not clear how interdependencies among variation points should be captured, how individualizations can be achieved and how the correctness of individualized process models can be guaranteed.

Moreover, while a configurable process model represents both standard process and its variations, it does not easily allow analysts to understand what these variations share, what their differences are in relation to the business domain, and thus why and how these differences occur. Last but not least, the scope of process configuration has so far been restricted to the control-flow, tending to oversimplify other important aspects such as human resources and business objects participating in the process.
Specifically, during the initial phase of this research, the following issues in existing approaches for managing configurable process models were identified:

1. **Error-prone configuration** - first and foremost, there is a lack of understanding of what process model configuration means from a language-independent point of view. This uncertainty has led to the employment of manual methods to configure process models, which are error-prone. In particular, these methods cannot guarantee the individualized models to be correct, whether syntactically or semantically. For example, if a model element or an entire path in a configurable process model is removed during configuration, the remaining model elements need to be re-connected to maintain syntactic correctness. Even worse, the configuration of variation points attached to parallel splits, decision points and synchronization points may lead to the introduction of semantic problems (e.g., deadlocks), which are much more difficult to spot. Although this issue is technical in nature, its consequences are very practical. In fact analysts are left with the burden of ensuring the correctness of the individualized process models and of manually fixing errors. This leads to limitations. Firstly, it is not possible to guarantee that a given set of configuration choices always leads to the same individualized model, thus leaving room for interpretation. Secondly, incorrect individualized models cannot be directly used by developers to derive executable process specifications, thus requiring further effort.

2. **Lack of decision support** - configurable process models do not provide support for the actual selection of alternatives. They do not guide the user as to what might be a recommended configuration given the user’s environment. Consequently, it is not easy to estimate the impact of configuration decisions on the process model. The lack of an explicit link between variation points in the process model and business decisions, requires that the user possesses expertise both in the application domain and in the modeling language the process model has been built in. Therefore, the user who carries out the configuration must not only be a domain expert but also be skilled in reading and configuring process models. This assumption represents an adoption obstacle in domains where experts are unfamiliar with modeling notations (e.g. in the screen business). In addition, this lack of abstraction from the modeling notation makes the configuration of industry reference models (e.g. VICS) extremely hard, as these are typically characterized by numerous variation points with complex and intricate interdependencies.

3. **Lack of expressiveness** - configurable process models, and existing languages for representing reference process models, are only concerned with the control-flow aspects of a process, neglecting other equally important aspects. Indeed, business processes provide an integrated view over an organization by bringing together different organizational aspects in a consistent way. Therefore, besides the control-flow, describing which tasks have to be performed and in which order, other aspects (or process perspectives) need to be considered. These primarily include the organizational resources (humans and machines/applications) participating in the process, and the business objects (software and physical artifacts) that are produced and consumed by the process [JB96]. The lack of techniques for representing variations in the various process perspectives leads to limited expressive power. Firstly, it is not possible for process modelers to capture the
unavailability of given entities in the process, such as human resources or physical artifacts. Secondly, it is not possible to understand the impact this can have on the tasks of the process. For example, in post-production, the unavailability of the role Negcutter leads to projects being edited exclusively on tape. Despite the fact that these situations occur quite often in reality, there is currently no technique to capture them in process models.

1.3 Objectives and Approach

In light of the shortcomings identified, the core objectives of this research are to:

- release analysts from the burden of manually checking models for correctness during process configuration;
- enable non-modeling experts to leverage off the benefits of configurable (reference) process models;
- offer modelers holistic support for process model configuration.

The first objective stems from the usage of error-prone methods to configure process models. The second one is motivated by the lack of decision support in existing process model configuration approaches. The last objective is the result of a need to cover the lack of expressiveness in capturing process model configuration. This research aims to achieve the above objectives by:

1. clarifying the concept of process model configuration – by establishing a language-independent conceptual foundation for process model configuration it is possible to reason on the correctness of the individualized models. Techniques can then be designed that guarantee the preservation of model correctness during configuration, thus releasing analysts from using manual methods to check model correctness. These techniques in turn can smooth the transition from process model design to implementation (given that the individualized models are correct), thus aiding developers in preparing the individualized models for execution.

2. shedding light on the relationship between process model configuration and domain decisions – by making explicit how business decisions can impact variation points in configurable process models, approaches can be formulated that enable abstraction from process modeling notations and provide decision support for the actual selection of alternatives. This in turn provides a means to manage the complexity induced by configurable process models, thus enabling non-modeling experts to fully exploit the potential of configurable (reference) process models.

3. exploring the interplays across different process perspectives – by understanding how different aspects of a business process can interact with each other, mechanisms can be defined to enable modelers to capture variability across multiple process perspectives in a coherent and consistent manner. These mechanisms can then be incorporated in a rich meta-model supporting the holistic configuration of business process models.
Following this approach, a satisfactory solution to the shortcomings was achieved by designing, formalizing, implementing and validating an integrated framework to manage the configuration of business process models. The framework is grounded on three original and interrelated contributions, as illustrated in Figure 1.4.

![Figure 1.4: Integrated framework to manage the configuration of business process models.](image)

The first such contribution is a conceptual foundation for process model configuration, which abstracts from the specificities of modeling notations used in practice (e.g. BPMN). Accordingly, a Petri net-based representation of business processes was adopted, namely Workflow nets [Aal97]. This foundation paved the way to the development of a technique for staged process configuration, which checks model correctness at each intermediate step of the configuration procedure, thus guaranteeing the individualized models to be syntactically and semantically correct.

The Petri net-based foundation was validated mathematically through a formalization culminating in propositions and theorems of correctness-preserving criteria. This foundation was then applied to enable staged correctness-preserving configuration of Configurable Event-driven Process Chains (C-EPCs) [RA07]. C-EPCs are a configurable version of EPCs [KNS92, Men08], the latter being a notation widely used in practice to define conceptual process models in the early stages of the PAIS lifecycle. C-EPCs extend EPCs by providing a means to explicitly represent variation points in EPC models. This thesis leverages off the benefits brought by the concept of configurable process model and its incarnation in C-EPCs.

The conceptual foundation was also applied to the realm of workflow languages. This is not discussed in this thesis and is reported in [GAJL08, GL08].

The second contribution is a questionnaire-based approach for variability modeling. This approach proposes to abstract from the notation used to model configurable processes, by capturing the variability of the domain in which a process operates as a collection of interdependent choices. These choices, organized in a questionnaire model, can then be linked to variation points in a process model to drive the configuration of the latter. The approach includes a technique to generate interactive questionnaires from questionnaire models. These questionnaires guide the configuration process by posing relevant questions to the user, consistent with the order dependencies established in the questionnaire model, and prevent the user from entering incorrect answers that would violate the constraints of the application domain. This ensures the alignment of the process model’s configuration with the business rules of the domain. At the same time, by leveraging off the findings on model correctness, the derived models are also guaranteed to be syntactically and semantically correct.
1. Introduction

If on the one hand, the abstraction from the specificities of a modeling notation provides a means to manage the complexity of realistic configuration scenarios, on the other hand the use of questionnaires is particularly suitable to those users who are not proficient in modeling notations. The questionnaire-based approach was applied to configure process models defined in the C-EPC notation, however it was defined in a general manner to be used for the configuration of other types of models, e.g. data models.

The last contribution of this thesis is a meta-model for holistic process configuration, which extends the expressive power of configurable process models to participating resources and business objects. Specifically, the meta-model allows modelers to define complex role-tasks and object-tasks associations, and to capture a range of variations on top of these associations. Furthermore, it enables modelers to define the interplay of variation points across different process perspectives. This study highlighted the intricacies that configurable process models across multiple perspectives create. To overcome this issue, and thus to fully exploit the expressive power of multi-perspective, configurable process models, users can leverage off the benefits of the questionnaire-based approach.

The meta-model was defined as an extension of the C-EPC notation, namely Configurable integrated-EPC (C-iEPC). This can foster the uptake of the meta-model by analysts and modelers in practice, given the widespread use of EPCs in industry. However this meta-model was formalized in an abstract manner so that it can be transposed to other notations.

In line with the principles of the design science research method [HMPR04], this thesis is concerned with the development of a new, purposeful artifact – the above-mentioned framework for business process model configuration. All the concepts, approaches and techniques of this framework were rigorously defined by means of formal methods. Moreover, to establish the practical feasibility of this framework, a comprehensive, open-source toolset was implemented, namely Synergia, supporting all the aspects of this framework. Finally, four reference process models and their respective questionnaire models were drawn from different domains to validate the framework and the scalability of the toolset. Excerpts of these models were used as working examples throughout this thesis. In particular, one such model was constructed in collaboration with subject-matter experts from the Australian Film Television & Radio School [AFTRS].

The various aspects of this framework were exposed to scientific and professional communities, including prestigious forums, an online magazine and the wide network of the screen industry. Furthermore, the web-site Process Configuration [LaR] was set up to provide up-to-date information on the topic of this thesis. The Synergia toolset is available for download from this web-site.

The significance of this thesis is derived from the uptake of reference process models in many current reengineering, software development and software configuration projects. Many organizations are endorsing and adopting reference process models, while others have a strong interest in developing their own internal ones. A new configuration framework which allows reference process models to be easily and effectively configured can then significantly improve the state of the art.
1.4 Publications

This research led to the following publications (categorized by topic):

### On the framework for managing process model configuration:


### On techniques for capturing process variability:


### On the conceptual foundation for process model configuration:


### On the configuration of executable process specifications:


1. Introduction

On the questionnaire-based approach for variability modeling:


On the meta-model for holistic process configuration:


Case studies:


1.5 Outline

This thesis is organized as follows. Chapter 2 positions this research by reviewing current literature on process model configuration and discussing available techniques for representing systems variability in the Software Engineering field. Chapter 3 introduces the conceptual foundation for process model configuration using Workflow nets and then applies the correctness results to the realm of C-EPCs. Chapter 4 exposes the questionnaire-based approach for modeling domain variability and presents a technique for generating interactive questionnaires. Chapter 5 elaborates on the concept of staged process configuration by linking the correctness results of Chapter 3 with the questionnaire-based approach of Chapter 4. Chapter 6 describes the meta-model for holistic process configuration using the C-iEPC notation. Each of these chapters also discusses the implementation of the respective concepts in the Synergia toolset. Finally, Chapter 7 concludes this thesis by summarizing the work presented and discussing possible extensions.
Chapter 2

Literature Review

This chapter positions the research in this thesis by reviewing current literature relevant to the topic of configurable process models. Section 2.1 provides an overview of business process modeling languages. Specifically, the focus is on those languages that have been extended in order to capture variations in business processes. Next, Section 2.2 outlines two main research streams in Software Engineering (SE) that deal with variability management in software families. Section 2.3 compares current approaches for capturing configurable process models based on the objectives identified in Chapter 1. This section also shows how approaches from the SE field can be used to drive the configuration of process models. Finally, Section 2.3 provides an overview of process flexibility – an area of BPM that deals with process models that can change at run-time, as opposed to design-time configuration. Finally, Section 2.3 draws conclusions.

This chapter presents and expands upon work published in [LDH09].

2.1 Business Process Modeling Languages

Business process modeling deals with the explicit representation of business processes at different abstraction levels, by means of suitable graphical notations [Wes07]. Two types of process models are typically identified: business-oriented process models and workflow models. The former are high-level models used for requirements analysis in the early stages of the PAIS lifecycle. These models provide a basis for communication among relevant stakeholders, and as such they must be unambiguous as well as intuitive. On the other hand, workflow models are designed for process automation. They are typically obtained by refining business-oriented process models with information that is relevant for implementation. Their execution is supported by a WfMS.

In the following we provide an overview of languages for business process modeling. Three examples of business-oriented languages are outlined: Event-driven Process Chains, UML Activity Diagrams and the Business Process Modeling Notation. These languages have been extended in different ways to capture configurable process models. To complement this overview, two workflow languages are also presented: the Web Services Business Process Execution Language and Yet Another Workflow Language.
2.1.1 Event-Driven Process Chains

Event-Driven Process Chains (EPCs) [KNS92, Men08] are an easy-to-understand language for modeling business processes [SL05] which were initially developed for the design of the SAP R/3 reference process model [CK97, Her05]. EPCs also became the core modeling language in the ARIS platform [Sch00] and were later used by other vendors for the design of SAP-independent reference models (e.g. the ARIS-based reference models for ITIL [IDSa] or SCOR [IDSb]).

An EPC is a directed graph consisting of events, functions, connectors and arcs linking these elements. Each EPC starts and ends with at least one event. Events are triggers for functions and signal their completion, while functions represent activities to be performed. Each function is preceded and followed by an event. Connectors are used to model alternative and parallel branching and merging. They are splits and joins of the logical types of OR and XOR (for inclusive and exclusive decision and merging, respectively) and AND (for parallelism and synchronization).

Figure 2.1 shows a process model for picture post-production in the EPC notation. This is a more elaborated version of the model shown in Chapter 1 and is the result of a case study we conducted with domain experts from the AFTRS (more details on this study are provided in Chapter 6). In the following we briefly describe this process by means of the EPC notation, as it will be used as a working example throughout this chapter.

Picture post-production is that phase of post-production that deals with the editing of the motion picture. It starts with the receipt of the footage from production, once the shooting has completed. In the EPC model, this is captured by the event Shooting finished followed by the function Receive footage. Next, the preparation of the shooting medium for editing follows. The medium can be tape, film or both. This choice is done depending on which branches are taken at the first OR-split in the model. The control is then synchronized by the subsequent OR-join. Once the footage is ready, the project is first edited on a low-resolution format (function Offline editing), and then on a high-resolution format in the cut stage. A choice can be made for the type of cut between Online editing, Negmatching or both via the second OR-split. The cut is followed by the finishing phase, where the project can be released on tape, film or on both media depending on the choice made at the last OR-split. The project concludes with a finish on new medium (e.g. a DVD or a stream video).

Events, functions and connectors form the core set of modeling elements of EPCs. In extended EPCs (eEPCs) [Sch99, STA05], extra symbols have been added to the original meta-model. These symbols denote organizational resources (e.g. roles or units) and business objects (e.g. documents) which can be attached to functions in any combination. However no specific semantics is given.

2.1.2 UML Activity Diagrams

Another example of process modeling language is UML Activity Diagrams (UML ADs), which is part of the Unified Modeling Language (UML) [OMG07]. UML is a visual, object-oriented and multipurpose modeling language which offers a variety of notations to capture different aspects of software structure and behavior. UML has been standardized by the Object Management Group (OMG) consortium [OMG]. Its eleven
diagram types, although heterogeneous, are put together in a uniform framework by a common meta-model which formally defines the abstract syntax of all the diagram types, and a notation guide which defines a concrete syntax for the meta-model elements. UML was primarily designed to model software systems, however some diagram types, such as UML ADs, can also be used for business process modeling.

In UML ADs a business process can be described by an activity consisting of a coordinated sequencing of nodes, based on control-flow and object-flow [EFHT05]. The control-flow comprises two types of nodes: action nodes and control nodes. An ac-

Figure 2.1: The picture post-production process in EPCs.
tion node can model an activity to be performed or a signal to be received/sent by the process. Control nodes are used to model sequencing and parallel or alternative branching, like splits and joins in EPCs (with the exception of the OR-join, which is not supported by UML ADs). Processes can be organized in a hierarchy by means of compound activities, in order to avoid cluttering the model.

The object-flow is represented via associations of object nodes with activities to denote inputs and outputs. The execution of an activity consumes one object from each of the activity’s input object nodes and produces one object in each of its output object nodes. UML ADs also provide features for modeling streams – object nodes that can store multiple objects at once. Moreover, an activity can be associated with a swimlane which may represent a role or an organizational unit. UML ADs allow multiple swimlanes (or partitions) to be associated with an activity but in this case, the activity is performed by one resource that belongs to all the partitions associated with it.

A worthwhile feature of UML is the possibility to extend its vocabulary by defining profiles, which are sets of stereotypes. Stereotypes define new elements from existing ones (e.g. an activity), by adding new properties that are suitable for a specific context. Later on we will see how this mechanism can be used to model variability.

2.1.3 Business Process Modeling Notation

An emerging notation alternative to UML ADs is the Business Process Modeling Notation (BPMN) [OMG08], which has recently been standardized by the OMG consortium. BPMN was designed with the precise intent to enable both business users and technical developers to model readily understandable graphical representations of business processes. For this purpose, BPMN reuses typical elements of flowcharting techniques, e.g. rounded rectangles and diamonds, which are familiar to most modelers.

The control-flow of a BPMN process is modeled by means of tasks, representing activities, and gateways, representing splits and joins. Like EPCs, BPMN supports splits and joins of type OR, XOR and AND. Events are also available in BPMN. They are partitioned into three types according to their position in the process model. A start event indicates the beginning of a process while one or more end events indicate the completion of the process. Intermediate events are used to capture anything that happens during the execution of a process, but unlike EPCs, they do not need to be strictly alternated with tasks.

Tasks, gateways and events are the core notation categories of BPMN. For each category, extended constructs exist to define advanced process behavior. For example, it is possible to use composite tasks to model subprocesses, or specific types of events to model timeouts, message receipts and exceptions to the normal process flow.

Similar to UML ADs, in BPMN the object-flow is represented via associations of data objects with tasks to denote input and output artefacts. However the semantics of such constructs is underspecified in BPMN. Also, streams of objects cannot be modeled in BPMN. Organizational aspects are represented by means of pools and lanes, like swimlanes in UML ADs. Pools act as containers for process models and represent business process participants. They are typically used to capture the organization in which a process operates. Lanes subpartition pools and can be nested or defined
in a matrix. They are typically used to capture organizational entities, such as the
departments or roles within an organization. Similar to UML ADs, a task is performed
by one resource that belongs to all the lanes associated with it. Apart from this, the
semantics of these constructs is voluntarily unspecified in BPMN, to leave modelers
with the freedom to choose a specific meaning according to the situation at hand.

Formal mechanisms \cite{ODA09} are available to generate mappings between BPMN
elements and constructs in the Web Service Business Process Execution Language.
These mechanisms can be used to facilitate the later execution of BPMN process models.

2.1.4 Web Services Business Process Execution Language

Web Services \cite{ACK03} have emerged in the BPM landscape due to the need to shift
from isolated, tightly coupled, controlled IT applications to a highly distributed, het-
erogeneous environment, characterized by loosely coupled systems \cite{KMC05}. The
Web Service framework lies on an extensible and modular stack of open XML-based
technologies (e.g. WSDL \cite{W3C07b} and SOAP \cite{W3C07a}) which allow providers to
easily describe, compose, access and offer software applications as services deployed to
the Internet.

The Web Service Business Process Execution Language (WS-BPEL or “BPEL”
for short) \cite{OAS07} sits on top of the Web Service stack and is used to describe the
behavior of Web services using business process modeling constructs. For this reason,
BPEL represents a convergence between Web services and business process technology.
Its specification derives from the joint effort of several industry partners and it has been
standardized by the OASIS \cite{OASIS} consortium.

BPEL essentially extends imperative programming languages (e.g. C) with con-
structs for the implementation of Web Services. The language is layered on top of the
WSDL specification since a BPEL process is exposed as a Web Service through WSDL
interfaces, and typed partnerLinks are used to define connections with other partners.
From a process perspective, there is no difference between a client (who triggers the
process) and its partners, since all of them are seen as Web Services.

A BPEL process is a hierarchical structure of basic activities corresponding to atomic
actions for sending, receiving and creating/processing messages. Compound activities
determine the process structure by allowing sequential, parallel and conditional routing,
as well as looping. It is also possible to specify events as external agents, such as a
timeout or a message receipt. Specific activities are also available for exception handling
and recovery.

The language has been designed to specify both abstract and executable processes.
The former, also called behavioral interfaces, provide a specification of the message
exchange flow a service may have with other partners. In an abstract process, private
information that is not relevant to the other participants in the interaction is hidden.
The executable variant, also called orchestration model, provides a process definition
with enough information to be interpreted and executed in a compliant engine.
2. Literature Review

2.1.5 Yet Another Workflow Language

Another example of executable language is Yet Another Workflow Language (YAWL) [AH05]. YAWL is an expressive language to describe, analyze and automate complex business process specifications which builds on top of the research outcomes of the Workflow Patterns Initiative [AHKB03, WPI]. The Workflow Patterns describe various aspects of a process, from the order in which work items have to be executed (control-flow patterns), to the data being manipulated by work items (data patterns) and the strategies to allocate work items to human participants and applications for their execution (resource patterns). This set of patterns was used to conduct a rigorous evaluation of a number of process modeling languages (e.g. EPCs, BPMN, Petri Nets, BPEL). As a result of this analysis, considerable differences and lacunae were detected in the expressiveness of such languages.

This state of affairs motivated the development of a new workflow language that could provide comprehensive support for the workflow patterns while keeping the notation as simple and intuitive as possible. YAWL was realized by extending Petri nets with vital constructs to directly support the workflow patterns. Nevertheless, YAWL is a completely new language with a formal semantics specifically designed to model workflow specifications. This formal semantics has allowed the development of a number of sophisticated verification techniques to catch potentially costly mistakes before deploying YAWL models for execution.

A YAWL model is a hierarchical structure of tasks corresponding to atomic or composite work items (similar to transitions in Petri nets), and conditions, to explicitly represent the notion of state (similar to places in Petri nets). Splits and joins are of type OR, XOR and AND, and are defined as output, respectively, input decorations of a task. Multiple instance tasks and cancelation regions complete the control-flow semantics of YAWL, and are used to model advanced control-flow features.

YAWL relies on global variables to capture the data-flow. These variables can be mapped to task input parameters to determine the content of a work item. At runtime, a work item can be consumed by an external application (automated task) or by a human resource via a Web form (manual task). The data produced is collected and stored in the task output variables, which are then mapped back to global variables. Allocation strategies can be defined to assign work items to human resources, based on the resource patterns. Resources are defined in terms of roles, capabilities and groups, which are drawn from an organizational model.

In the next section we provide an overview of mechanisms that deal with variability management in software families.

2.2 Software Variability Management

Variability management has been widely studied in the field of Software Product Line Engineering (SPLE) [PBL05, Cle06]. Software product lines (or software families) refer to engineering techniques for creating a collection of similar software systems from a shared set of software assets. This is achieved by using common means of production with the purpose of reducing time, effort, cost and complexity of software creation and maintenance [Kru]. Software product lines can be described in terms of four concepts,
illustrated in Figure 2.2 [Kru]:

- Software asset inputs: a library of optional and configurable software assets, to be used in order to create all the software products of a product line;
- Product decisions: a set of optional and variable requirements for the products in a product line;
- Production mechanism: the means for assembling and configuring products from the software asset inputs, using the product decisions;
- Software product outputs: the collection of all the products of a product line that can be produced. The outputs also determine the scope of the product line.

![Figure 2.2: The four basic concepts of Software Product Lines.](image)

Among others, two research streams have emerged in SPLE, namely Software Configuration Management [Pre05] and Feature Diagrams [SHT06].

### 2.2.1 Software Configuration Management

Software Configuration Management (SCM) is a methodology to control and manage a software development project. Work on SCM has led to models and languages to capture how a set of available options impacts upon the way a software system is built from a set of components.

The Adele Configuration Manager [EC94] is an example of SCM language. Adele supports the definition of dependencies between artefacts composing a software family, such as all-or-none or exclusion dependencies between interfaces and realizations thereof (e.g. “only one realization of an interface should be included in any instance of the family”). Such dependencies are expressed in a first-order logic language using attributes defined on objects, where an object represents a software artefact. Building a configuration in Adele involves selecting a collection of objects that satisfy all constraints.

Similarly, in the Proteus Configuration Language (PCL) [TGC95], software entities are annotated with information attributes and variability control attributes. The former provide stable information about an entity, i.e. commonalities, while the latter capture variability in the structure and in the process of building the entities. Variability attributes determine which actions are performed to build a variant of an entity. For
example, in PCL one can capture that a sub-system, such as a graphical interface, is optional, or that a sub-system maps to different sets of program files depending on the value of a variability attribute.

Another example is the CoSMIC (Component Synthesis with Model Integrated Computing) configurable middleware [TGN04]. A key component of CoSMIC is the Options Configuration Modeling Language (OCML), which allows developers to capture hierarchical options that affect the way in which middleware services are configured. Options are similar to variability attributes in PCL, but OCML goes beyond PCL by allowing constraints to be defined over individual options or groups thereof. Like in Adele, constraints in OCML are expressed in a first-order logic language. OCML expressions are fed to an interpreter that prompts users to enter values for each option and raises error messages when the entered values violate a constraint.

2.2.2 Feature Diagrams

Feature Diagrams (FDs) are a family of techniques for describing software product lines in terms of their features. A number of feature modeling languages have been proposed, e.g. [BG97, CE00, Man02, CHE05], since FDs were first introduced as part of the FODA (Feature Oriented Domain Analysis) method [KCH+90]. For a survey of FDs techniques, the reader is referred to [SHT06].

A feature model consists of one or more feature diagrams which are generally represented as tree-structures with high-level features being decomposed into sub-features. A feature represents a system property that is relevant to a stakeholder and is used to capture commonalities or to discriminate among systems in a family [CE00]. A feature includes a set of attributes such as its description, can be mandatory or optional, and can be bound to other features via constraints.

Constraints can be expressed as plain text [BG97] or as arbitrary propositional logic expressions over the values of features [Man02, Bat05], specified by means of a proper grammar (e.g. a limit in the number of sub-features a feature can have). Constraints among the sub-features of a same feature can be graphically represented to model restrictions in the number of sub-features a feature can have. These relations can be: AND (all the sub-features must be selected), XOR (only one sub-feature can be selected) and OR (one or more can be selected). OR relationships can be further specified with an \( n : m \) cardinality [CHE05], where \( n \) indicates the minimum and \( m \) indicates the maximum number of allowed sub-features. For example, the sub-features of “Cut” are bound by an OR relation (it is possible to have more than one type of cut).

Figure 2.3 shows a possible feature diagram for the picture post-production domain, using the notation proposed in [Bat05]. There are features related to the options available for shooting, type of editing, transfer and finish. Some features have been identified as mandatory (with a full circle on top of them) if they are required, while some others have been identified as optional (with an empty circle), if they can be excluded. The feature “Transfer” and its sub-features “Telecine” and “DFM” (digital film master) are optional. Their inclusion depends on the selection of the sub-features of “Editing” and “Finish”, by means of proper constraints.

A configuration specifies a valid scenario in terms of features selected/deselected, i.e.
a scenario that complies with the constraints. Figure 2.4 depicts the feature diagram for picture post-production configured for a project shot on tape, edited online and delivered on film.

Figure 2.4: A possible configuration for the feature diagram of Figure 2.3.

2.3 Business Process Variability Modeling

This section provides an overview of current approaches to capturing variability in business process models for the purpose of modeling configurable processes. Specifically, we refer to those approaches where configuration is achieved by restriction of behavior, i.e. the initial model needs to incorporate all the possible variants that can be selected during configuration. This is in line with the notion of reference process model which is interpreted as a repository of recommended practices for a certain domain [FL03, RA07]. Therefore we exclude from our analysis those approaches which allow the addition or modification of nodes in the process model through abstraction, adaptation or specialization mechanisms, e.g. [BWB03, RBSS05, SRS07, HBR09]. These mechanisms are outside the scope of this thesis.

The approaches presented in this section are evaluated against a set of desirable criteria. These criteria operationalize the objectives of this thesis as discussed in Chapter 1, i.e.: (i) releasing analysts from the burden of manually checking models for correctness during configuration; (ii) enabling non-modeling experts to leverage off the benefits of
configurable process models; (iii) offering modelers holistic support for process model configuration. Accordingly, we have derived the following criteria:

1. Correctness preservation – the ability to preserve the correctness of a configurable process model during individualization. This includes:
   1.1. Syntactic correctness – the ability to guarantee the correct structure of the individualized model, e.g. by avoiding disconnected nodes;
   1.2. Semantic correctness – the ability to guarantee the correct behavior of the individualized model, e.g. by avoiding deadlocks.

2. Decision support – the provision of support for the selection of configuration alternatives. This includes:
   2.1. Language-independence – the possibility of abstracting from the specific notation used to capture the configurable process and to reason in terms of business decisions;
   2.2. Guidance – the availability of business-related information to guide the configuration, e.g. to estimate the necessity or criticality of the possible configurations.

3. Expressiveness – the ability to configure various process perspectives. This includes:
   3.1. Control-flow – the ability to configure the control-flow of a process, e.g. activities, events, connectors;
   3.2. Resources – the ability to configure the resources of a process, e.g. roles, information systems;
   3.3. Objects – the ability to configure the objects of a process, e.g. physical or information artifacts.

4. Individualization algorithm – the provision of an algorithm to individualize a configured process model by making the required model transformations.

5. Tool support – the provision of tool support to design, configure and individualize (according to the approach) configurable process models.

The overview of the various approaches for business process variability modeling follows.

2.3.1 Configurable Nodes

Let us consider again the EPC model in Figure 2.1. This model can be perceived as a ‘big picture’ since it includes all possible ways of carrying out picture post-production. However only a subset of them will actually be used in a post-production project, depending on the project requirements. For example, as mentioned in Chapter 1, a negmatching cannot be performed if the project is not shot on film. So in the EPC model function Negmatching is only relevant if function Prepare film for editing is chosen at the first OR-split, otherwise it must not be performed. Also, function Finish
on new medium is not always required, so in some cases it may need to be removed from the model.

A possible way of specifying these conditions is by means of Configurable EPCs (C-EPCs) [RA07]. C-EPCs extend EPCs by providing a means to explicitly capture variability in EPC process models. This is achieved by identifying a set of variation points (called configurable nodes) in the model, to which variants (called alternatives) can be assigned, as well as constraints (called configuration requirements) to restrict the combination of allowed variants. The active nodes of an EPC’s control-flow, i.e. functions and connectors, can be marked as configurable with a thicker border.

Figure 2.5 shows the C-EPC version for the post-production example, where we have identified 4 configurable functions and 6 configurable connectors to cover all the configuration possibilities in post-production.

Configurable connectors can be configured to an equally or less restrictive connector. In other words, the model resulting from a configuration should have the same or less execution traces than the original model. A configurable OR can be left as a regular OR (no restriction is applied), or restricted to an XOR, to an AND or to one of its outgoing/incoming sequences of nodes. This latter alternative is indicated with $SEQ_n$, where $n$ is the node starting the SEQuence. If the connector is of type split, $n$ must be one of its outgoing nodes; if the connector is of type join, $n$ must be one of its incoming nodes. For example, we can capture the choice of the shooting medium by configuring the first OR-split in Figure 2.5. We can restrict this connector to its left-hand side branch – $SEQ_{e_1}$ – if the choice is tape (this results in branch $SEQ_{e_2}$ being removed); to its right-hand side branch – $SEQ_{e_2}$ – for film (this results in branch $SEQ_{e_1}$ being removed); or to an AND-split if both the media need to be prepared for editing. In these cases we anticipate the decision of the medium at configuration-time. Alternatively, if we configure this connector to an (X)OR-split, we postpone the decision till run-time, when the post-production process is actually enacted. A configurable XOR (not depicted in Figure 2.5) can be set to a regular XOR or to an outgoing/incoming sequence of nodes. For symmetry, an AND connector can also be configured, but it can only be mapped to a regular AND, thus no actual configuration is achieved in this case. These options are summarized in Table 2.1 [RA07].

<table>
<thead>
<tr>
<th>Config. connector</th>
<th>OR</th>
<th>XOR</th>
<th>AND</th>
<th>SEQ_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XOR</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>AND</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Configurable connectors can be configured to equally or less expressive variants.

Configurable functions have three variants: included (ON), excluded (OFF) or conditionally skipped (OPT). The first two alternatives allow one to decide a priori whether to keep the function in or permanently discard it from the process; the last option permits the deferral of this choice until run-time, where the execution of the function can be skipped on an instance-by-instance basis. For example, function Finish on new medium is configurable in Figure 2.5, so we can switch it OFF for those projects where this is not required.
Configuration requirements formalize domain constraints over the alternatives of configurable nodes and are enforced to rule out unfeasible configurations. It is also possible to define configuration guidelines to aid the configuration process. Both requirements and guidelines are expressed as logical predicates and are depicted as tags attached to the involved nodes. Only requirements are mandatory and must hold in order for a configuration to be valid. We have defined seven Requirements and one Guideline in the process of Figure 2.5. For example, Requirement 1 binds the first and the second OR-splits and implies to perform an online editing if the project is only shot on tape. In this way we exclude the option of performing a negmatching if the project is not...
shot on film. Requirements 2 and 3 ensure that the respective OR-joins are configured accordingly. The other requirements restrict the available options for editing and finishing. For example, function Record DFM is carried out to transfer the editing results from tape to film. Therefore, it is only needed when the editing is online and the finish is on film, otherwise it needs to be switched OFF (Requirement 6). Similarly, Transfer in Telecine is only needed to transfer the results of negmatching to tape or new medium (Requirement 5). We have also defined a guideline (Guideline 1) to suggest performing only negmatching if the project is shot on film. In fact, although an online editing is still possible, in this case negmatching offers the best results as the cut is directly done on the film roll.

Once each configurable node has been assigned a variant that complies with the requirements, an individualization algorithm [RA07, MRRA06] can be used to derive an EPC from the configured C-EPC. We can observe that in our example the only nodes that never vary (either directly or by configuring other nodes) are functions Receive footage and Offline editing. These denote commonalities in the post-production process.

Figure 2.4 shows the individualized EPC resulting from configuring the post-production example for a project shot on tape and film, edited via negmatching and delivered on film. Here some connectors have been removed because they were configured to sequences of nodes, as well as all those sequences that are no longer relevant (e.g. the one starting with event Tape editing). Similarly, functions switched OFF have been removed from the model (e.g. Finish on new medium). Provided the initial C-EPC is syntactically correct, this algorithm preserves the syntactic correctness of the individualized models. This is achieved by removing all nodes that are no longer connected to the initial and final events via a path, and by properly reconnecting the remaining nodes. However semantic issues that may arise during configuration are not considered.

C-EPCs provide guidelines to aid users in configuring a process model, but this is counteracted by the lack of abstraction from the modeling notation (and from the intricacies created by the configuration layer). In other words, configuration is achieved by deciding on each individual variation point. In fact, guidelines can only be defined over the variants of these variation points, and not in terms of business choices. Laboratory tests [RRA05] to evaluate the usefulness and easy of use of the C-EPC notation – as it is perceived by experienced users – indicate a need for more decision support. These tests report on the difficulties users experienced in evaluating the potential impact of configuration decisions on the model. Moreover, the behavior of configurable connectors is not easily understood. A first attempt to address these issues is presented in [DRA+05], where the authors extend the original C-EPC specification with further notation elements (e.g. to represent critical configuration nodes). However this extension does not address the fundamental lack of abstraction from model-based configuration which hampers the use of C-EPCs in practice.

Tool support for C-EPCs comprises an interchange format, developed as an extension to the EPC Markup language (EPML) [MN06], and a tool (C-EPC Validator [Men]) to check the compliance of a C-EPC configuration against configuration requirements and guidelines. However, the individualization algorithm has not been implemented and a designer for C-EPC models is not available.
2.3.2 Model Projection

Another approach for capturing variability in process models is exposed in [BDD+04, BDK07]. This approach is based upon the principle of model projection. Since a reference process model typically contains information on multiple application scenarios, it is possible to create a projection for a specific scenario (e.g., a class of users) by fading out those process branches that are not relevant to the scenario in question.

Business characteristics can be used to determine the available application scenarios. For example, in the case of picture post-production, we can identify the business characteristic ‘Shooting type’ (ST), which can be ‘Tape shooting’ (T) or ‘Film shooting’ (F). Another example is ‘Budget Level’ (BL), which yields the scenarios ‘Low budget’ (L), ‘Medium budget’ (M) or ‘High budget’ (H). This is a high-level characteristic since a choice on the budget typically affects a number of decisions in post-production. These characteristics are linked to the elements of a process model by means of configuration parameters defined in the form of simple attributes or logical terms over characteristics. The elements of a process model which are linked to parameters are thus the parts of the model that can vary, although these are not explicitly represented in the process model like in C-EPCs. The language chosen to capture process models is plain eEPCs. Specifically, parameters can be linked to functions, events, organizational resources and objects of an eEPC.
Figure 2.7.a shows the picture post-production model in EPC, where each function and each event has been associated with a logical term referring to the project’s budget. For instance, event Film editing and function Negmatching have been linked to the term \texttt{NOT BL(L | M)}, meaning that these elements are not suitable for a low budget project, due to the high costs involved in editing a project on film (where \texttt{L} stands for the logical \texttt{OR}). On the other hand, function Online editing has the term \texttt{BL(L | M | H)}, meaning that it is suitable for any type of budget.

The projection of a process model to a specific scenario is done by marking as hidden those elements whose parameters evaluate to \texttt{false}. Then an individualization algorithm [Del06] (pp. 141–142) is performed to remove the hidden elements and re-connect the remaining nodes. Figure 2.7.b shows the individualized post-production model for a low budget project. The algorithm can fix simple syntactic issues (e.g. the removal of connectors which have one input and one output, as in the example), but cannot ensure the syntactic and semantic correctness of the resulting models. For example, since both events and functions can be removed, the algorithm does not work in the context of cycles, or when a function between two events is removed. Syntactic issues that cannot be fixed like the above ones, are prompted to the modeler who is demanded to fine-tune the configuration.

Configuration parameters can also be applied to the meta-model layer (i.e. to the eEPC meta-model), to remove process modeling perspectives that are not relevant to a specific scenario. For example, it is possible to hide the resource perspective in the eEPC meta-model. In this way all the roles and organizational units that are associated with functions are removed from the model.

An advantage of this approach is that, unlike C-EPCs, configuration is not carried out at the process model level, but through the evaluation of a set of business characteristics. However the approach does not offer guidance to users when assigning values to configuration parameters. Also, this approach suffers from limited configuration expressiveness. For example, the routing behavior of connectors cannot be restricted, which is instead possible in C-EPCs, and fine-grained configuration of task-resources and task-object associations is not possible, beyond the simple mechanisms offered by eEPCs.

In [DJK+06] the authors show how this approach has been implemented as a toolset that interacts with the ARIS platform. Configuration parameters can be defined and linked to elements of an eEPC, which can be designed in ARIS. Then an interface allows users to select the desired parameters and a projection is performed on the initial EPC to remove irrelevant elements.

### 2.3.3 Annotation-based Process Variability

In this section we outline three approaches which rely on the use of annotations to represent variability in process models.

#### Variant-Rich Process Models

The idea of capturing variability in process models has also been explored in the PE-SOA (Process Family Engineering in Service-Oriented Applications) project [PSWW05,
Figure 2.7: (a) The picture post-production process model in EPC, with logical terms for the budget levels; (b) the model projection for a low budget project.

SP06]. The aim of this project is not to provide a language for representing and configuring process models, but rather to improve the customization of process-oriented software systems, i.e. of systems that are developed from the specification of process models. If the variability of a software system can be directly represented in a process model that describes the system’s behavior, it is then possible to generate code stubs for the system from the individualization of the process model itself. Since the purpose of this proposal is outside the topic of this thesis, we only focus on the way the authors represent process variability.

According to this approach, a variant-rich process model is a process model extended
with stereotype annotations to accommodate variability. Although stereotypes are an extensibility mechanism of UML, in this approach they are applied to both UML ADs and BPMN models. The places of a process model where variability can occur are marked as variation points with the stereotype "<<VarPoint>>". These can be activities in UML ADs and tasks in BPMN. A variation point represents an abstract activity (task), such as Prepare medium for editing, which needs to be realized with a concrete variant ("<<Variant>>") among a set of possible ones. For example, Prepare medium for editing can be realized with the variant Prepare tape for editing, or with Prepare film for editing, or with both of them. It is possible to annotate the default variant for a variation point with the stereotype "<<Default>>". Figure 2.8.a shows the process model for picture post-production in annotated BPMN, where a number of variation points have been identified. For example, Prepare tape for editing has been annotated as the default variant of Prepare medium for editing, because this is the most common choice in post-production.

If the variants are exclusive, i.e. if only one variant can be assigned to a given variation point, the stereotype "<<Abstract>>" is used instead of "<<VarPoint>>". In Figure 2.8.a we assume that the variants Online editing and Negmatching are exclusive, so their variation point Picture cut has been annotated with the tag "<<Abstract>>". As a shortcut, when the variants are exclusive, the default resolution can be depicted directly on the variation point with the stereotype "<<Alternative>>".

A variation point annotated with the stereotype "<<Null>>" indicates optional behavior. It can only be associated to one variant and its resolution is not mandatory. This is the case of the variation point Transfer tape to film which may be resolved with the variant Record DFM, or be completely dropped from the process model. A shortcut for a "<<Null>>" variation point and its variant is to depict the variant straight on the variation point, using the stereotype "<<Optional>>". This is the case of Transfer in Telecine, which subsumes the variation point Transfer film to tape.

Through a configuration each variation point is realized with one or more variants according to its type. Figure 2.8.b shows a fragment of the BPMN process model for post-production configured for a project shot on Tape and edited Online. In this model the variants that are not required have been removed.

Figure 2.8: (a) The picture post-production process model in annotated BPMN. (b) A configured process model.

A variation point annotated with the stereotype "<<Null>>" indicates optional behavior. It can only be associated to one variant and its resolution is not mandatory. This is the case of the variation point Transfer tape to film which may be resolved with the variant Record DFM, or be completely dropped from the process model. A shortcut for a "<<Null>>" variation point and its variant is to depict the variant straight on the variation point, using the stereotype "<<Optional>>". This is the case of Transfer in Telecine, which subsumes the variation point Transfer film to tape.

Through a configuration each variation point is realized with one or more variants according to its type. Figure 2.8.b shows a fragment of the BPMN process model for post-production configured for a project shot on Tape and edited Online. In this model the variants that are not required have been removed.
Language-independence is achieved by linking process variants with features from a feature diagram. In fact, although the initial aim of feature-based approaches was to facilitate the configuration of software families, a feature diagram can also be used for the configuration of process models. This is done by tagging each process variant with the name of a feature, such that when a feature is disabled in a feature configuration, the corresponding variant is removed from the process model. Figure 2.9 shows an example of “tagged” variants for the BPMN model in Figure 2.8, in relation to the features of the diagram in Figure 2.3. Constraints over feature values can be used to capture domain requirements, thus restricting the possible combinations of variants in the process model. However, feature diagrams do not provide guidance for the selection of a suitable set of features.

![Figure 2.9: The relation between the variation point Prepare medium for editing and the Shooting sub-features from Figure 2.3.](image)

This approach does not discuss an individualization algorithm for transforming variation points to those variants that have been selected. This leaves room for interpretation. For example, it is not clear how model elements have to be reconnected after removing a <<Null>> or <<Optional>> variation point, nor is it clear how a variation point should be transformed when more than one of its variants are selected. These transformations may lead to correctness issues which have not been taken into account.

The approach has been implemented as a plugin of the Eclipse IDE [ECa]. This tool provides support for configuring a feature diagram and for applying the results to the underlying process model. However, since an individualization algorithm has not been defined, the configuration of a process model is limited to the removal of the undesired variants.

**Hierarchical Variability Modeling**

A subset of the stereotypes proposed by the PESOA project appears in [RK08]. Specifically, in this approach the focus is on two types of variation points: *optional* and *alternative* ones. An optional variation point (annotated with <<opt_vp>>) allows the selection of at most one variant among the available ones. An alternative variation point (annotated with <<alt_vp>>) allows the selection of exactly one variant.

These annotations can be applied to the control-flow and object-flow of UML ADs. For example, an input object Tape to activity Prepare tape for editing can be realized...
with either Digital tape or Analog tape. Regarding the control-flow, both activities and
decision points can be annotated. However, interdependencies among model variants
cannot be defined beyond those offered by optional and alternative variation points. As
a result, only simple configuration scenarios can be captured.

Moreover, the authors claim to achieve abstraction from the complexity induced by
process configuration by exploiting the hierarchical definition of an AD. For example, it
is possible to mark a composite activity with the stereotype <<variable>> to indicate
that the underlying subprocess includes some variation points and meanwhile to hide
its variability details. This feature is also available in PESOA, but different from
the latter, in this approach configuration is carried out directly on the process model.
Therefore, modeling expertise is still required to configure a hierarchical process model
and hence, language-independence is not achieved. Finally, correctness requirements
are not discussed, an individualization algorithm is not provided and no tool support
is offered to implement the concepts proposed.

Superimposed Variants

The idea of annotating model elements to represent variability has also been investigated
in [CA05]. In this approach, any control-flow element of an UML AD can be annotated
using presence conditions (PCs) and meta-expressions (MEs). PCs indicate if the model
element they refer to should be present or be removed. MEs are used to compute
attributes of model elements relevant to the UML notation (e.g. the name of an activity).
These attribute variations span a finite range of options.

Both PCs and MEs are captured by boolean formulae over the features and feature
attributes of a feature diagram, and are evaluated against a feature configuration.
These formulae can be represented in disjunctive normal form or as XPath [W3C07c]
expressions. UML stereotypes are used to create annotations for these formulas to be
assigned to model elements. For example, the stereotype <<Tape ∨ Film>> indicates
the disjunction of the two features Tape and Film, while the stereotype <<PC>> indicates
an XPath expression, where the expression is specified as a String property of the
stereotype itself. The assignment of stereotypes to modeling elements is done through
rendering mechanisms, such as labels, color schemes or icons.

Figure 2.10.a shows the finishing phase of the picture post-production process in
annotated UML AD; for simplicity, in this example we have only specified PCs. These
PCs have been defined over the features of Figure 2.3 and their annotation has been
rendered with a color and a number. For instance, activity Transfer in Telecine has
been associated with the sub-feature Telecine of feature Transfer (annotated in blue with
label “1”), while the two outgoing flows of the decision point have been associated with
the two Finish sub-features Tape and Film. All the non-labeled elements (in black) have
been associated with the always-true formula. These represent the commonalities of
the model and as such cannot be removed, e.g. the decision point and the ActivityFinal
element.

Process configuration is achieved by evaluating PCs and MEs against a feature
configuration. Those model fragments whose PCs evaluate to false are removed from the
model, while those model attributes that are affected by MEs are changed accordingly
(e.g. an activity name is changed).

Figure 2.10.b shows a possible individualized model of the post-production example
2. Literature Review

Figure 2.10: (a) An annotated UML AD modeling an extract of the picture post-production process, where Presence Conditions refer to the features of Figure 2.3. (b) A possible individualized model.

where only activities Record DFM and Finish on film have been kept. This model can be obtained through an individualization algorithm which applies patches to reconnect modeling elements that have been disconnected during configuration, and simplifications to remove splits and joins that have been left with one inflow and one outflow. However patches are only applied to those nodes that have exactly one inflow/outflow, and an annotation error is raised otherwise. Therefore syntactic correctness is only partly addressed; semantic correctness is not dealt with.

This algorithm has been implemented in an Eclipse plugin which allows users to configure UML ADs via cardinality-based feature diagrams [CHE05].

2.3.4 Hiding and Blocking

We conclude this overview of variability mechanisms for process models with a theoretical approach. This approach, presented in [ADG+06, GAJ07], is motivated by the need for capturing configurable process models independently of vendor-specific modeling notations. In light of this, the authors apply the hiding and blocking operators from the concept of inheritance of workflow behavior [AB02] to Labeled Transition Systems (LTSs). LTSs are a formal abstraction of computing processes, therefore any process model with a formal semantics (e.g. Petri Nets or YAWL) can be mapped to an LTS.

An LTS is a graph composed by nodes – representing states of a process – and directed edges between nodes – representing labeled transitions. A label can denote some event, activity or action that changes the process from one state to another. Two special nodes, indicated with $i$ and $o$ denote the initial and the final state. A traditional choice (i.e. the $(X)OR$ in EPC or BPMN) is modeled as a node with two or more outgoing edges. Figure 2.11.a shows a simplified version of the picture post-production process model as an LTS. For example, a choice between the edges labeled Prepare tape for editing and Prepare film for editing must be made after the first edge.

Hiding and blocking can be applied to configure the edges, which are the active nodes of an LTS. According to the inheritance of workflow behavior, blocking corresponds
Figure 2.11: Three LTSs: (a) the process model for picture post-production, (b) the configured model for a project shot on Tape, edited Online and delivered on Film, and (c) the individualized model.

to encapsulation, i.e. the execution of an atomic action is disabled. In the LTS this means that a blocked edge cannot be taken anymore and the process flow will never reach a subsequent state. Hiding corresponds to abstraction, i.e. the execution of an atomic action becomes unobservable. In the LTS a hidden edge is skipped, but the corresponding path is still possible (the edge’s label becomes silent and is no longer relevant).

The process model in Figure 2.11.a can be configured by selecting the desired parts through the application of hiding and blocking. Figure 2.11.b shows this model configured for a project shot on tape and delivered on film. Accordingly, edges Prepare film for editing, Finish on tape and Telecine transfer have been blocked, while edge Finish on new medium has been hidden. Figure 2.11.c shows the individualized model. This model can be obtained through the application of an individualization algorithm [GAJ07] which takes care of removing all blocked edges and merging the nodes surrounding an hidden edge.

As a consequence of blocking an edge, all the subsequent nodes (and their connecting edges) that depend on the blocked edge become unreachable. For example, in Figure 2.11.c edges Offline editing, Online editing and Negmatching and the nodes in-between (indicated in grey), have become unreachable after blocking Prepare film for editing. Similarly, Finish on tape has become unreachable after blocking Telecine transfer. The syntactic correctness of this model can be obtained by “cleaning-up” the model from all the elements that have become unreachable after configuration. However this does not always work. For example, if edge Finish on new medium were blocked instead of hidden, the final state would no longer be reachable. Yet, if Record DFM were also blocked in Figure 2.11.b, the individualized model would be cut in two parts. Furthermore, since LTSs do not allow the modeling of parallelism and synchronization, they are unsuitable to study semantic issues that may arise during configuration, e.g. deadlocks.
In [GAJ07], the authors also define the notions of optional hiding and optional blocking. These operators extend hiding and blocking by allowing the deferral of the choice whether or not to hide (block) an edge until run-time.

2.4 Business Process Flexibility

Process configuration can be regarded as design-time flexibility, i.e. the ability of changing the business process structure before deploying the model for execution. On the other hand, WfMSs often have to handle deviations and changes during the execution of a process that were not anticipated at design-time. These deviations are commonly known as process exceptions and lead to real-time modifications to the process instances already running. For example, exceptions may occur when people stop performing certain roles or decide to take roles that normally they do not take. Similarly, they can occur as a result of deadline violations or emergencies (e.g. a task may be skipped or fast-tracked because of a special situation). This kind of run-time flexibility, generally denoted with the term process flexibility, is supported by WfMSs such as the YAWL system [AADH04] and ADEPT [RD98, RRKD05].

In the YAWL system, support for flexibility is offered by a dedicated YAWL component called Worklet Dynamic Process Selection Service [AHEA05]. This component is able to substitute a work-item in a YAWL process with a dynamically selected worklet at run-time, in order to handle expected exceptions. A worklet is a discrete YAWL process acting as a sub-net for the selected work-item. Additionally, exlets may be added at run-time to handle unexpected exceptions, and such handling methods automatically become an implicit part of the process specification for all current and future instances of that process. The advantage of such an approach is to provide for continuous evolution of the process while avoiding any need to modify the original process definition.

ADEPT is a WfMS which supports modifications of a process during execution both at the model and instance levels. These modifications are achieved through the application of a set of change patterns [WRR08] which allow one to add, delete and change a sequence of tasks. However, unlike YAWL, these changes are only achieved through manual intervention. Authorized users can also trigger forward and backward ‘jumps’ through a process instance [RDB03].

Some form of flexibility can also be achieved in BPEL, by dynamically assigning the endpoint of a service (i.e. a reference to its physical location) to a partner the process is interacting with. The value of the endpoint can be taken out from a message or variable. Through this mechanism a BPEL instance is able to change a service partner on the fly, provided the corresponding WSDL document specifying the messages and types exchanged by the service is known before the process is deployed.

A first attempt towards the improvement of flexibility support in BPEL is represented by the concept of parameterized process [KLB05]. This extension to BPEL consists of the ability of changing the portTypes and operations of a service partner through the use of parameters in the process model, whose values are resolved at runtime with proper substitution policies. This approach can free BPEL definitions from information on concrete portTypes and operations, decreasing the model complexity and minimizing maintenance efforts.
2.5 Summary and Discussion

This chapter began with an overview of business process modeling languages and then explored variability management in the context of software product families. Afterwards, it compared current approaches for capturing configurable process models against a set of desirable criteria. The chapter concluded with an overview of the process flexibility area.

The results of the comparative analysis are summarized in Table 2.2. The first column lists the six approaches that were considered. Each approach is identified by a reference to its major publication. The second and the third column indicate the variability mechanism adopted by each approach and the language chosen to capture configurable process models. The remaining columns report the results of the analysis for each criterion. Here a “+” indicates a criterion that is fulfilled, a “−” indicates a criterion that is not fulfilled and a “±” indicates partial fulfillment and an “n/a” indicates a criterion that is not applicable.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[RA07]</td>
<td>Conf. nodes</td>
<td>EPC</td>
<td>+</td>
<td>−</td>
<td>±</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>±</td>
<td>+</td>
<td>n/a</td>
</tr>
<tr>
<td>[BDK07]</td>
<td>Projection</td>
<td>eEPC</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>+</td>
<td>±</td>
<td>±</td>
<td>−</td>
<td>+</td>
<td>n/a</td>
</tr>
<tr>
<td>[PSWW05]</td>
<td>Annotation</td>
<td>BPMN, AD</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>n/a</td>
</tr>
<tr>
<td>[RK08]</td>
<td>Annotation</td>
<td>AD</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>±</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>[CA05]</td>
<td>Annotation</td>
<td>AD</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>n/a</td>
</tr>
<tr>
<td>[GAJ07]</td>
<td>Hid. &amp; Blo.</td>
<td>LTS</td>
<td>−</td>
<td>−</td>
<td>n/a</td>
<td>+</td>
<td>−</td>
<td>n/a</td>
<td>±</td>
<td>−</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 2.2: Comparative analysis of approaches for business process variability modeling.

From this table it is evident that while the approaches considered in this analysis exhibit a heterogenous set of methods for capturing variability in process models, none of them addresses all the identified criteria. Specifically, the matter of model correctness is completely neglected. All the approaches rely on manual methods to configure process models. Indeed there are no mechanisms that prevent users from entering configurations that will lead to incorrect individualizations. For example, in [BDK07] and [CA05] users are prompted with a list of syntactic issues detected during individualization, which have to be fixed manually. The only approach that gets closer to ensuring model correctness is C-EPCs [RA07], where syntactic issues get automatically fixed during individualization. However, as we will point out in Chapter 3 (cf. Section 3.6.2) there are situations in which the individualization of a C-EPC generates undesirable results (e.g. an empty net).

Decision support for configuring process models is not prevalent among these approaches. For example, [PSWW05] and [CA05] rely on FDs to achieve independence from process modeling notation. Although features typically capture properties of software systems, they could also be used to capture business decisions that have to be
taken to configure a process model. Nevertheless, guidance is not offered to facilitate the choice of a suitable set of features given a specific setting (e.g. a low-budget project in post-production). Also, there is no control over the order in which features are presented to users. Users are exposed to the whole set of features which they need to select from, and this set can be huge depending on the complexity of the scenario in question (an in-depth analysis of feature diagrams and their supporting tools is provided in Section 4.5 of Chapter 4). Similarly, in [BDK07] business characteristics can be used to create projections of process models, but no guidance is offered to choose from these characteristics. The only approach that provides guidelines is [RA07]. However these guidelines cannot refer to business decisions, but rather to the variation points of a process model.

As for configuration expressiveness, all the approaches define variability mechanisms for the configuration of control-flow aspects of a process. For example, in [RK08] and [RA07] both activities (functions) and decision points (connectors) can be configured while in [PSWW05] only activities can be identified as variation points. In other approaches also passive nodes can be configured, such as events [BDK07] and arcs [CA05]. These latter approaches however lead to further correctness issues which need to be taken into account. Moreover, one could argue the usefulness of configuring passive nodes, given that they should “react” to the configuration of the active nodes they depend upon.

The only approaches that define variability mechanisms for the configuration of resources and object-flow are [BDK07] (both resources and objects) and [RK08] (objects only). However these mechanisms inherit the limits of the meta-model adopted to represent resources and objects in processes (i.e. eEPCs and UML ADs). As a result, only basic features are provided to configure these process perspectives (e.g. dropping a role from an eEPC function).

In conclusion, despite there exist different proposals for capturing variability in business process models, all of them suffer from the three major shortcomings outlined in Chapter 1: (i) error-prone configuration, (ii) lack of decision support and (iii) lack of expressiveness. The remaining chapters of this thesis propose a framework for process model configuration which addresses these shortcomings. In particular, the next chapter builds on top of the results of the theoretical study on process model configuration conducted in [GAJ07]. This allows us to treat the problem independently of vendor-specific notations to develop a conceptual foundation for correctness-preserving configuration of process models.
Chapter 3

Conceptual Foundation for Process Model Configuration

In the Introduction we highlighted the importance of ensuring model correctness during process configuration and in Chapter 2 we pointed out that so far this issue has been neglected by proposals in the area of configurable process modeling. As a result, analysts are left with the burden of ensuring the correctness of the configured models and of manually fixing errors.

This chapter exposes a conceptual foundation for process model configuration. The objective is to formally define the concept of process model configuration and to use this formalization to analyze properties of configurable process models, particularly with respect to correctness. For this purpose we adopt Workflow nets – a class of Petri nets specifically designed to formally represent business processes. By applying the hiding and blocking operators from the concept of inheritance of workflow behavior [AB02], we enhance Workflow nets with variation points, leading to configurable Workflow nets.

We then present the concept of staged process configuration, according to which an individualized process model can be obtained incrementally by the combination of a set of configuration steps, until all the required variation points have been configured. We use this concept to develop a technique to automatically infer process constraints from a configurable process model that, if satisfied by a configuration step, guarantee the syntactic correctness of the resulting model. We prove that for a large class of process models, namely “free-choice” nets [DE95], these constraints also ensure that semantic correctness is preserved. While being formally grounded, this result has practical implications as the majority of constructs of process modeling languages such as EPCs, BPMN or BPEL can be mapped to Petri nets in this class [Aal99, AL08, ODA+09].

Having established a foundation for process model configuration, we then apply it to the realm of C-EPCs. We extend previous results on the mapping between EPCs and Workflow nets [Aal99] to show the formal relation between C-EPCs and configurable Workflow nets, thus confirming the foundational nature of the conceptualization. Finally we show how the notion of staged configuration defined on Workflow nets can be adapted to fit the specificities of C-EPCs, by extending the technique for ensuring the preservation of process correctness to C-EPC models and implementing it in the Synergia toolset.

Against this background, this chapter is structured as follows. Section 3.1 introduces
Workflow nets and their notion of correctness while Section 3.2 presents the notion of configurable Workflow net and configured net. Section 3.3 exposes the concept of staged configuration and presents a technique for interactively ensuring the correctness of configured nets based on the inference of process constraints. Along the same lines, Section 3.4 introduces EPCs, their notion of correctness and configurable EPCs while Section 3.5 discusses the relation between the concepts of hiding and blocking and the notion of configuration of C-EPC models. Section 3.6 concludes the discussion on C-EPCs by applying the concept of staged correctness-preserving configuration to this notation through the extension of the technique for inferring process constraints. The chapter concludes with a summary and a discussion on related work.

This chapter presents and expands upon work published in [ADG+08a, ADG+08b].

3.1 Workflow nets

Petri nets are a formal model of concurrent systems [Pet62, Mur89] which benefit from a rich body of theoretical results, analysis techniques and tools. They have been extensively applied to the formal verification of business process models [VBA01]. These features make Petri nets suitable for establishing a formal foundation for business process model configuration. In addition, mappings exist between process modeling languages used in practice (e.g. EPCs, BPMN, BPEL) and Petri nets. These mappings provide a basis for extending the results outlined in this chapter to concrete process modeling notations.

We use a class of Petri nets, namely Workflow nets, specifically designed for business process modeling. Workflow nets have a single starting point and ending point, which capture the intuition that business processes are instantiated, and each process instance progresses independently through a series of activities until completion. A desirable property is that an instance of a Workflow net always completes properly, i.e. the process model is semantically correct. This is captured by the notion of soundness. In the following, we provide an introduction to workflow nets and soundness.

3.1.1 Workflow nets: Syntax

Petri nets are composed of two types of elements, namely transitions and places, connected by directed arcs. Transitions represent tasks while places represent the status of the system before or after the execution of a transition. Formally [Mur89]:

Definition 3.1 (Petri net, Preset, Postset) A Petri net is a triple $PN = (P, T, F)$, such that:

- $P$ 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).

For each node $n \in P \cup T$, we use $\bullet n$ and $n \bullet$ to denote the set of inputs to $n$ (preset) and the set of outputs of $n$ (postset). Sometimes we also use the preset (postset) of
Figure 3.1: The process model for travel form approval.

A set to indicate the union of all presets (postsets) of the elements in that set, e.g. $X \bullet = \bigcup_{x \in X} x \bullet$.

Figure 3.1 shows a process model for travel requisition approval as a Petri net. Let us briefly describe it. The model consists of two variants: the left one for international travels and the right one for domestic travels. After requesting a quote for international travel, either the employee or an assistant prepares the travel requisition form. In case of the latter, the employee needs to check the form before submitting it for approval. The administrator can then approve or reject the requisition, or make a request for change. At this point, the employee can update the form according to the administrator’s suggestions and re-submit it, or drop the case. In contrast, the application for domestic travels only requires the employee to ask for a quote and to hand over the travel requisition to the administration.

A business process model may be executed a number of times to deal with different cases (e.g. different travel requests in the example). Each of these cases (called process instances) has a distinct start (input) and a distinct end (output). Accordingly, we are only interested in Petri nets with a unique source place (representing the input) and a unique sink place (output), and such that all other nodes are on a directed path between the input and the output places. A Petri net satisfying these conditions represents a structurally correct process model and is known as a Workflow net [Aal97]. Formally:

**Definition 3.2 (Workflow net)** Let $PN = (P,T,F)$ be a Petri net and $F^*$ be the reflexive transitive closure of $F$. $PN$ is a Workflow net (WF-net) iff:

- there exists exactly one input place, i.e. $\exists! p_I \in P$, $p_I = \emptyset$, and
• there exists exactly one output place, i.e. \( \exists!_{p_O \in P} p_O = \emptyset \), and
• each node is on a directed path from the input place to the output place, i.e. \( \forall_{n \in P \cup T} [(p_I, n) \in F^* \land (n, p_O) \in F^*] \).

The Petri net in Figure 3.1 is a WF-net.

### 3.1.2 Workflow nets: Semantics

Behavioral (i.e. semantic) correctness of a WF-net is defined with respect to the states that a process instance can be in during its execution. A state of a WF-net is represented by the marking of its places with tokens. In other words, in a given state, each place is either empty, or it contains one or more tokens (i.e. it is marked). A transition is enabled in a given marking if all the places in the transition’s preset are marked. Once enabled, the transition can fire (i.e. can be executed) by removing a token from each place in the preset and putting a token into each subsequent place of the transition’s postset. This leads to a new state. Formally:

**Definition 3.3 (Marking, Enabling Rule, Firing Rule)** Let \( N = (P, T, F) \) be a WF-net with source place \( p_I \) and sink place \( p_O \):

- \( M : P \rightarrow \mathbb{N} \) is a marking of \( N \) and \( \mathbb{M}(N) \) is the set of markings of \( N \),
- \( M_I \) is the initial marking of \( N \) with one token in place \( p_I \), i.e. \( M_I = [p_I] \),
- \( M_O \) is the final marking of \( N \) with one token in place \( p_O \), i.e. \( M_O = [p_O] \),
- \( M(p) \) returns the number of tokens in place \( p \) if \( p \in \text{dom}(M) \),
- For any two markings \( M, M' \in \mathbb{M}(N) \), \( M \geq M' \) iff \( \forall_{p \in P} M(p) \geq M'(p) \),
- For any transition \( t \in T \) and any marking \( M \in \mathbb{M}(N) \), \( t \) is enabled at \( M \), denoted as \( M[t] \), iff \( \forall_{p \in \bullet t} M(p) \geq 1 \). Marking \( M' \) is reached from \( M \) by firing \( t \) and \( M' = M - \bullet t + t\bullet \),
- For any two markings \( M, M' \in \mathbb{M}(N) \), \( M' \) is reachable from \( M \) in \( N \), denoted as \( M' \in N[M] \), iff there exists a firing sequence \( \sigma = \langle t_1, t_2, ..., t_n \rangle \) leading from \( M \) to \( M' \), and we write \( M \xrightarrow{\sigma}_N M' \). If \( \sigma = \langle t \rangle \), we use the notation \( M \xrightarrow{t}_N M' \). \( N \) can be omitted if clear from the context.

The execution of a process instance starts with the state in which the input place has one token and no other place is marked. The execution of this process instance should then progress through transition firings until a proper completion state. This intuition is captured by three requirements [Aal97]. Firstly, every process instance should always have the option to complete. If a WF-net satisfies this requirement, it will never run into a deadlock or livelock. Secondly, every process instance should eventually reach the state in which there is one token in the output place \( p_O \), and no tokens are left behind in any other place, since this would signal that there is still work to be done. Thirdly, for every transition, there should be at least one execution sequence from the initial marking (where only the input place \( p_I \) is marked) to the final marking (where
only \( p_o \) is marked) that includes at least one firing of this transition. In other words, no transition in the WF-net should be spurious. A WF-net fulfilling these requirements is sound [Aal97]. Formally:

**Definition 3.4 (Sound WF-net)** Let \( N = (P, T, F) \) be a WF-net and \( M_I, M_O \) be the initial and final markings. \( N \) is sound iff:

- option to complete: for every marking \( M \) reachable from \( M_I \), there exists a firing sequence leading from \( M \) to \( M_O \), i.e. \( \forall M \in N[M_I] \; M_O \in N[M] \), and
- proper completion: \( M_O \) is the only marking reachable from \( M_I \) with at least one token in place \( p_o \), i.e. \( \forall M \in N[M_I][M \geq M_O \Rightarrow M = M_O] \), and
- no dead transitions: every transition can be reached by \( M_I \), i.e. \( \forall t \in T \; \exists M \in N[M_I] \; M[t] \).

### 3.2 Process Model Configuration

As shown in Chapter 2, the work in [ADG+06, GAJ07] provides initial insights on how to capture variation points in a language-independent manner for the purpose of representing configurable process models. Specifically, the reasons for building on top of this work are twofold. Firstly, the hiding and blocking operators abstract from vendor-specific process modeling notations and can easily be applied to Petri nets. Secondly, it has been shown that these two operators are atomic and complete, i.e. they are all we need to restrict, hence to configure, the behavior of a process model [AB02]. Accordingly, we define the notion of configurable WF-net, where each transition captures a variation point whose possible values (or variants) are: allowed, hidden and blocked. In this way we generalize from approaches such as C-EPCs, where only selected elements can be marked as variation points.

Hiding a transition refers to skipping its execution while it is fired, without affecting the rest of the process flow. Consider for example the WF-net in Figure 3.1. Some organizations may not require a quote for domestic travels. Thus, the task to request a quote can be skipped from the process model by hiding transition \( t_2 \). The process continues without forcing the employee to request a quote.

Blocking a transition implies to inhibit it in the process model. Blocked transitions cannot forward tokens and all the subsequent transitions will never be executed if they cannot be enabled via other paths. For example, some organizations may only allow domestic travels. This can be achieved by blocking \( t_1 \), thus preventing the process for international travels to be triggered.

If a transition is neither blocked nor hidden, we say it is allowed, meaning nothing changes in the model. To configure a WF-net each transition has to be assigned one value among hidden, blocked or allowed. Formally:

**Definition 3.5 (WF-net Configuration)** Let \( N = (P, T, F) \) be a WF-net, then \( C_N \in T \rightarrow \{allow, hide, block\} \) is a configuration for \( N \). We define:

- \( A_N^C = \{t \in T \mid C_N(t) = allow\} \) as the set of all allowed transitions,
3. Conceptual Foundation for Process Model Configuration

- \( H_N^C = \{ t \in T \mid C_N(t) = \text{hide} \} \) as the set of all hidden transitions,
- \( B_N^C = \{ t \in T \mid C_N(t) = \text{block} \} \) as the set of all blocked transitions.\(^1\)

Based on these configuration values, a configured (i.e. individualized) net is obtained representing the new behavior of the process model. This new Petri net is a restriction of the behavior of the starting model, where all the hidden transitions are replaced by silent \( \text{skip} \) transitions and all the blocked transitions are removed. Also, all the places connected only to blocked transitions and all the flow relations from/to blocked transitions are removed too. Formally:

**Definition 3.6 (Configured Petri net)** Let \( N = (P,T,F) \) be a WF-net and let \( C_N \) be a configuration of \( N \). The resulting configured net \( \beta_N(N,C_N) = (P^C,T^C,F^C) \) is defined as follows:

- \( T^C = (T \setminus (B_N^C \cup H_N^C)) \cup \{ \text{skip}_t \mid t \in H_N^C \} \),
- \( F^C = (F \cap ((P \cup T^C) \times (P \cup T^C))) \cup \{(p, \text{skip}_t) \mid (p,t) \in F \land t \in H_N^C\} \cup \{(\text{skip}_t,p) \mid (t,p) \in F \land t \in H_N^C\} \),
- \( P^C = (P \cup \bigcup_{(x,y) \in F^C} \{x,y\}) \cup \{p_I,p_O\} \).

As an example, Figure 3.2.a shows a configured net derived from the WF-net in Figure 3.1, where the transitions \( t_2 \) and \( t_9 \) have been blocked to allow the complex approval process only. In this new net employees have to prepare the approval form on their own, as \( t_3 \) has been blocked, and cannot drop a form application if a change is requested after approval (\( t_{10} \) has also been blocked). According to Definition 3.6, place \( p_5 \) has been removed since it became disconnected after removing \( t_3 \) and \( t_9 \).

Any choice we make when configuring a process model has to comply with the requirements of the application domain. This may prevent users from configuring the values of transitions freely. For example, in the travel management domain, if an employee decides to submit a travel form for approval there must be at least an option to accept the request and an option to reject it. This is clearly a requirement of the domain, which forbids users to block both \( t_{11} \) and \( t_{12} \) in the process model. In Chapter 4 we show how domain constraints like the above can be encoded as propositional logic expressions. By evaluating each transition’s value against these constraints with a satisfiability (SAT) solver, it is possible to prevent all the configurations which would violate the constraints.

Nonetheless, the set of constraints derived from the domain are in most cases not sufficient to guarantee the syntactic and semantic correctness of the configured model. Indeed, as per Definition 3.6, a configured net can be any Petri net, which means that it can contain elements that are not on a path from \( p_I \) to \( p_O \), or which are completely disconnected. Therefore this net may not satisfy the properties of Workflow nets. For example, forbidding the request for a change by blocking \( t_8 \) in the WF-net of Figure 3.2.a would make \( p_6, t_6, p_3 \) and \( t_5 \) unreachable, yielding the net of Figure 3.2.b. This configured net is not syntactically correct and hence not sound either, according to Definition 3.4. So, as soon as \( t_3 \) and \( t_8 \) are blocked, it would be desirable to suggest the user to block \( t_6 \) and \( t_5 \) too, so as to get rid of the unreachable branch.

\(^1\) \( A_N^C \cap H_N^C \cap B_N^C = \emptyset \) follows from the definition of \( N \).
In the following section we present a technique to automatically derive a set of constraints from a WF-net that preserve the model correctness during configuration.

### 3.3 Correctness-Preserving Configuration

Existing tools such as Woflan [VBA01] support the verification of Petri net-based process models. These tools could be used to check every single configured net that can be derived from a configurable process model. If the net is incorrect, the configuration that has generated this net should be excluded from the set of possible configurations. However this approach is costly, considering that configurable process models can potentially yield a large number of individualized process models. In fact, the number of individualized process models is determined by the number of transitions, each of which has three possible variants, the structure of the model (e.g. if we block the first transition of a chain, the configuration of the transitions further down that chain is of no relevance), and the domain requirements. While the latter two factors constrain the exponential configuration space yielded by the total number of transitions, the remaining number of individualizations may still be very large.

Similar considerations apply to configurable process models defined in languages other than Petri nets. For example, the configurable process model of the case study illustrated in Chapter 6 (cf. Section 6.5), which is captured in C-iEPC, features 183 variation points which lead to 310,000 valid individualizations.

Our aim is therefore to define a technique which allows incorrect configuration steps to be discarded incrementally and without computing all possible configurations of the configurable model. In addition, the technique needs to seamlessly integrate the domain constraints, so that a user can derive a correct process model which also satisfies any
domain constraints.

To this end, we complement the domain constraints with a set of process constraints to guarantee the preservation of syntactic and semantic correctness in the configured net. Both sets of constraints are captured in propositional logic over the nodes of a WF-net and are reduced by a SAT solver. Whenever a value is assigned to a variation point, the current set of constraints is evaluated. If the constraints are satisfied, the configuration step is applied. If on the other hand the constraints are violated, we compute a reduced propositional logic formula, from which we can identify additional variation points that need to be configured simultaneously in order to preserve correctness (e.g., if an edge in the process model is removed, all nodes in a path starting with that edge need to be removed). The set of constraints is incrementally updated after each step of the configuration procedure. In this way we can provide interactive support to the user, by pinpointing the impact of each configuration step on the resulting net and by eliminating unfeasible options.

Domain constraints are discussed in Chapter 4 while the relation between domain constraints and process constraints is illustrated in detail in Chapter 5.

3.3.1 Preserving Syntactic Correctness

In a staged configuration, users make decisions one after another in steps, and the set of configuration options is recalculated after each step. To remain syntactically correct, a WF-net must thus be checked on which configuration options are still viable among the transitions that have not been configured yet. For this, we have to consider the configuration decisions already taken.

To distinguish nodes which remain in the net from nodes which do not, we use a boolean variable for each node. If the variable is set to true, the node remains part of the net; if it is set to false, the node is dropped in the configured net. Accordingly, we assign a blocked transition the value false, while a transition that is allowed or hidden is assigned the value true. Since silent transitions have the same routing behavior as the original transitions, we do not need to distinguish hidden from allowed transitions. All transitions that are not explicitly configured remain as variables (i.e., unset).

According to Definition 3.6, any internal place remains in the net if there is a non-blocked transition in its preset or postset. Translating this definition in boolean logic, if one such transition is true, the place has also to be set to true; if all the connected transitions are false, the place has to be set to false; if some transitions have no value assigned yet, the place remains unset. Since a configuration is defined over the transitions of a net, we have to derive the values of the places. We do that by imposing that each transition set to true implies that for all the places in its preset and in its postset are set to true as well. Formally: $\bigwedge_{t \in \mathcal{T}} [t \Rightarrow \bigwedge_{p \in \bullet t} p \land \bigwedge_{p \in \bullet t} p]$, where with $t$, $p$ we indicate a transition, respectively a place, which is set to true.

Assuming the original net is a WF-net, to guarantee the configured net is still a WF-net, we have to ensure each node that remains in the configured net be on a directed path from $p_I$ to $p_O$. This is the only WF-net requirement to be verified, as $p_I$ and $p_O$ are part of the configured net by definition. This means all the nodes composing the directed path should not be false. For each node, we can decompose this path into two sub-paths: one from $p_I$ to the node in question and the other from the node to $p_O$, and
verify the property over the nodes of each sub-path. However, as per Definition 3.6, we can restrict the verification to the places of each sub-path, by deriving the places’ values from the ones of the transitions. Indeed, if a non-blocked transition has at least one place in its preset on a directed path from \( p_I \) and at least one place in its postset on a directed path to \( p_O \), then the transition is on a directed path from \( p_I \) to \( p_O \). Moreover, when searching for such paths we can restrict our analysis to acyclic paths. In fact a cycle always leads back to a same node, but does not provide any valuable progress from \( p_I \) to \( p_O \). Formally, we define an acyclic path as follows:

**Notation 3.7 (Acyclic Path)** Let \( PN = (P,T,F) \) be a Petri net:

- \( \phi = \{n_1,n_2,...,n_k\} \) is an acyclic path of \( PN \) such that \( (n_i,n_{i+1}) \in F \) for \( 1 \leq i \leq k-1 \) and \( i \neq j \Rightarrow n_i \neq n_j \),
- \( \alpha(\phi) = \{n_1,n_2,...,n_k\} \) is the alphabet of \( \phi \),
- \( \Phi_{PN} \) is the set of all acyclic paths of \( PN \),
- for all \( n \in P \cup T \), \( AC_I(n) = \{ \phi \in \Phi_{PN} \mid \phi = \{p_I,...,n\} \} \) is the set of all acyclic paths from \( p_I \) to \( n \),
- for all \( n \in P \cup T \), \( AC_O(n) = \{ \phi \in \Phi_{PN} \mid \phi = \{n,...,p_O\} \} \) is the set of all acyclic paths from \( n \) to \( p_O \).

The conjunction of the process constraints is called \( PC_N \) and is defined as follows:

**Definition 3.8 (Process Constraint)** Let \( N = (P,T,F) \) be a WF-net. Treating each place and each transition of \( N \) as a propositional variable, the process constraint \( PC_N \) is a propositional logic formula over these variables, given by the conjunction of the following expressions:

- \( p_I \) and \( p_O \) are always true, i.e. \( p_I \land p_O \);
- each place \( p \) implies the disjunction of all acyclic paths from \( p_I \) to \( p \) and the disjunction of all acyclic paths from \( p \) to \( p_O \), i.e. \( \bigwedge_{p \in P} [p \Rightarrow \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{p_I,p\}} n) \land \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{p,p_O\}} n)] \). \(^2\)

For example, four acyclic paths can be identified for \( p_3 \) from the model in Figure 3.2.a: two from \( p_I \) (\( \phi_1 \) and \( \phi_2 \)) and two to \( p_O \) (\( \phi_3 \) and \( \phi_4 \)), as shown in Figure 3.3. From these paths we infer the following boolean expressions:

\[
\phi_1: \ t_1 \land p_1 \land t_4 \land p_4 \land t_7 \land p_7 \land t_8 \land p_6 \land t_6 \\
\phi_2: \ t_3 \land p_2 \land t_4 \land p_4 \land t_7 \land p_7 \land t_8 \land p_6 \land t_6 \\
\phi_3: \ t_5 \land p_4 \land t_7 \land p_7 \land t_{11} \\
\phi_4: \ t_5 \land p_4 \land t_7 \land p_7 \land t_{12}.
\]

As per Definition 3.8, we then impose the formula \( p_3 \Rightarrow (\phi_1 \lor \phi_2) \land (\phi_3 \lor \phi_4) \). We infer this constraint for each place and conjunct all the constraints to determine \( PC_N \). The following theorem shows that any configured net derived from a configuration that satisfies \( PC_N \) is a WF-net.

\(^2\) \( p_I \) and \( p_O \) need not be included in this expression as they are imposed to \textit{true} by the first expression, neither does \( p \) as it is logically redundant.
3. Conceptual Foundation for Process Model Configuration

**Theorem 3.9** Let \( N = (P, T, F) \) be a WF-net and \( PC_N \) be its process constraint. Let \( C_N \) be a configuration of \( N \) and let \( \beta_N(N, C_N) = (PC^c, TC^c, PC^c) \) be the resulting configured net. Let \( v \in T \cup P \rightarrow \{ \text{true}, \text{false} \} \) be a valuation of transitions and places such that \( v(q) = \text{true} \) iff \( q \in T^c \cup P^c \). Then \( \beta_N(N, C_N) \) is a WF-net if and only if \( v \models PC_N \).

**Proof** (\( \Rightarrow \)) Let \( \beta_N(N, C_N) \) be a WF-net and let \( v \in T \cup P \rightarrow \{ \text{true}, \text{false} \} \) such that \( v(n) = \text{true} \) iff \( n \in T^c \cup P^c \). As \( p_I \in PC^c \) and \( p_O \in PC^c \) (Definition 3.6), \( v(p_I) = \text{true} \) and \( v(p_O) = \text{true} \); hence \( v \models p_I \land p_O \). Since \( \beta_N(N, C_N) \) is a WF-net, for all \( p \in PC^c \) there exists at least one directed path from \( p_I \) to \( p \). Let \( \phi \in AC_1(p) \) be such a path, thus for all \( n \in \alpha(\phi) \setminus \{ p_I, p \} \) we have \( n \in P^c \cup T^c \), hence \( v(n) = \text{true} \). Therefore, \( v = \bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n \). Hence, \( v = \bigwedge_{\phi \in AC_1(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n) \). Similarly, as there is at least one path from \( p \) to \( p_O \), \( \bigwedge_{\phi \in AC_1(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n) \), hence \( v = \bigwedge_{\phi \in AC_1(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n) \). Thus, for all \( p \in PC^c \), \( v \models \bigwedge_{\phi \in AC_1(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n) \). As \( p \in P^c \), \( \bigwedge_{\phi \in AC_1(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n) \) and therefore for all \( p \in PC^c \), \( v \models p \Rightarrow \bigwedge_{\phi \in AC_1(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n) \). If \( p \in P^c \) then \( v(p) = \text{false} \) and thus \( v \models p \Rightarrow \bigwedge_{\phi \in AC_1(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n) \). Hence \( v \models \bigwedge_{p \in P} (p \Rightarrow \bigwedge_{\phi \in AC_1(p)} (\bigwedge_{n \in \alpha(\phi) \setminus \{ p_I, p \}} n)) \).

(\( \Leftarrow \)) Let \( v \models PC_N \). Assume \( \beta_N(N, C_N) \) is not a WF-net. Since \( p_I \) and \( p_O \) belong to \( \beta_N(N, C_N) \) by definition, choose \( p \in PC^c \) such that there is either (1) no path from \( p_I \) to \( p \) or (2) no path from \( p \) to \( p_O \). If (1) then for all \( \phi \in AC_1(p) \) there is a node \( n \in \alpha(\phi) \setminus \{ p_I, p \} \) such that \( n \notin P^c \cup T^c \) and hence \( v(n) = \text{false} \). If (2) then for all \( \phi \in AC_0(p) \) there is a node \( n \in \alpha(\phi) \setminus \{ p, p_O \} \) such that \( n \notin P^c \cup T^c \) and hence \( v(n) = \text{false} \).

\( v \models PC_N \) if and only if \( v \not\models p_{I,O} \). Since \( v \models PC_N \), there exists a node \( n \in \alpha(\phi) \setminus \{ p_{I,O} \} \) such that \( v(n) = \text{true} \). From both cases we can conclude \( v \not\models PC_N \).

---

Figure 3.3: The acyclic paths identified for \( p_3 \).
For each configuration step, we determine a valuation $v$ of all the nodes of the net, where the values of the transitions are configured by the user (one for each step) and the values of the places are derived automatically. In order to enforce the WF-net requirements, $v$ has to satisfy $PC_N$ at each configuration step.

The satisfiability of $PC_N$ is clearly an NP-complete problem. To overcome this issue, we propose to use a SAT solver based on Shared Binary Decision Diagrams (SBDDs) [Bry86, MIY90]. SBDDs are canonical forms of boolean formulas for which there are efficient analysis algorithms. They are based on the classical BDDs with the advantage of being always cheaper in size and time computation. Existing SBDD solvers can efficiently deal with systems made up of around one million possibilities [MIY90]. Hence they are reasonably adequate to capture all the configurations produced by a realistic scenario.

We propose to use the solver to obtain a reduced representation of $PC_N$ in conjunctive normal form, where each variable is initially unset. Then we conjunct this formula with each new transition valuation as provided by the user during the configuration process, and further reduce the formula. In this way we do not recalculate $PC_N$ for each configuration step. The solver can only reduce the formula if this is satisfiable, i.e. if the configuration can yield a syntactically correct process model. This may imply to automatically force to true or false the conjunction or disjunction of other transitions which are still unset, in order to keep the formula satisfiable. For example, after blocking $t_8$ in the model of Figure 3.2.a, the solver would force $p_3$ to false, as there is no longer a directed path from $p_I$ to $p_3$ (both the expressions inferred from the paths $\phi_1$ and $\phi_2$ of $p_3$ would evaluate to false). Similarly, $p_6$ would be forced to false. Given that $t_5 \Rightarrow (p_3 \land p_4)$ and $t_6 \Rightarrow (p_6 \land p_3)$, it would follow that both $t_5$ and $t_6$ must also be false for $PC_N$ to be satisfied, i.e. these transitions need to be blocked. Therefore, as a result of blocking $t_8$, the whole branch starting with this transition will be removed to ensure the syntactic correctness of the model. By following this approach invalid configurations are identified at each configuration step and suggestions are proposed to keep the model correct.

### 3.3.2 Preserving Semantic Correctness

In addition to structural correctness, a configured net should also be semantically correct. The example in Figure 3.4 shows that a configured net conforming to the WF-net properties is not automatically sound, even if it is derived from a sound WF-net. The WF-net in (a) is sound: if $t_3$ fires before $t_4$, the token in $p_2$ can reach $p_5$ via $t_3$. However, if $t_3$ is blocked (b), $t_4$ needs to fire before $t_8$ as $t_4$ depends on the token in $p_6$ which is removed when $t_8$ fires. Since this behavior is not enforced in the net, the process might deadlock, and is therefore not sound, although (b) is still a valid WF-net.

Soundness is only defined for WF-nets (Definition 3.2), but it can be generalized to any Petri net with a designated source and sink place. However, it is easy to show that any non WF-net would still violate this generalized soundness notion. Therefore, the process constraint defined in Definition 3.8 is a necessary requirement for soundness.
3. Conceptual Foundation for Process Model Configuration

but as Figure 3.4 shows, it is not sufficient.

Below, we prove that \( PC_N \) is a sufficient requirement to guarantee soundness of a configured net, if the original model is a sound, free-choice WF-net. The restriction to this class of Petri nets provides a good compromise between expressiveness and verification complexity. Not only do free-choice WF-nets have several desirable properties [DE95], but the large majority of constructs of process modeling languages such as EPCs, BPMN or BPEL can be mapped to Petri nets in this class [Aal99, AL08, ODA+09]. A free-choice is defined as follows [Mur89]:

**Definition 3.10 (Free-choice WF-net)** Let \( N = (P,T,F) \) be a Petri net. \( N \) is free-choice (FC) if for every couple of places sharing transitions in their postset, these postsets coincide, i.e. \( \forall p_1,p_2 \in P \ [p_1 \cdot \cap p_2 \cdot \neq \emptyset \Rightarrow p_1 \cdot = p_2 \cdot] \).

Assuming the configurable process model is a sound, FC WF-net, we are able to identify several configuration properties relevant for the preservation of soundness during the configuration process:

**Proposition 3.11 (Properties of Configured WF-net)** Let \( N = (P,T,F) \) be a sound, FC WF-net with source place \( p_I \) and sink place \( p_O \), let \( C_N \) be a configuration of \( N \), and let \( \beta_N(N,C_N) = (P^C, T^C, F^C) \) be the configured net resulting from \( C_N \). If \( \beta_N(N,C_N) \) is a WF-net (i.e. \( PC_N \) evaluates to true), then:

1. \( \forall t \in T^C \ [(\cdot_N t = \cdot_{\beta_N(N,C_N)} t) \wedge (t \cdot_N = t \cdot_{\beta_N(N,C_N)})] \).
2. \( p_I \in P^C \) and \( p_O \in P^C \).
3. \( \forall t \in B^C \ [(\cdot_N t \cap P^C = \emptyset) \vee \exists t' \in T^C \ (\cdot_N t = \cdot_N t')] \) (a blocked transition is either not consuming any tokens from \( P^C \) or there is a transition in \( T^C \) with the same input set).
4. \( \forall \sigma \in T^C^* \ [(M_I \xrightarrow{\sigma} N) \leftrightarrow (M_I \xrightarrow{\sigma} \beta_N(N,C_N))] \) (the input and output sets of transitions in \( T^C \) are the same in both nets, therefore, the respective behaviors are identical when considering only firing sequences \( \sigma \in T^C^* \)).
e) \( \forall \sigma \in T_C. \forall M \ [ (M_I \xrightarrow{\sigma} N, M) \iff (M_I \xrightarrow{\beta_{N(C_N)}(N,C_N)} M) ] \).

f) \( \beta_N(N,C_N)[M_I] \subseteq N[M_I] \) (all firing sequences of \( \beta_N(N,C_N) \) are also possible in \( N \)).

g) \( \beta_N(N,C_N) \) is FC.

h) \( \forall M \in \beta_{N(C_N)}[M_I] \setminus \{ M_0 \} \exists t \in T_C \ [ M[t'] ] (\beta_N(N,C_N) \) has no deadlock markings).

**Proof**

a) Follows directly from the construction of \( \beta_N(N,C_N) \).

b) Idem.

c) Suppose that some \( t \in B^c \) consumes a token from a place \( p \in P^c \) in \( N \). Because \( \beta_N(N,C_N) \) is a WF-net with source place \( p_1 \) and sink place \( p_o \), there has to be a path from \( p \) to \( p_o \). Hence there is a transition \( t' \in T^c \) consuming a token from \( p \).

Hence \( \bullet_{N} t \cap \bullet_{N} t' \neq \emptyset \), thus \( \bullet_{N} t = \bullet_{N} t' \) (\( N \) is FC).

d) Follows directly from (a).

e) Follows directly from (d).

f) Follows directly from (e).

g) Let \( t, t' \in T^c \) such that \( \bullet_{\beta_{N(C_N)}} t \cap \bullet_{\beta_{N(C_N)}} t' \neq \emptyset \). Given that \( \bullet_{N} t' = \bullet_{\beta_{N(C_N)}} t' \) and \( \bullet_{N} t = \bullet_{\beta_{N(C_N)}} t \), we have \( \bullet_{\beta_{N(C_N)}} t \cap \bullet_{\beta_{N(C_N)}} t' = \bullet_{N} t \cap \bullet_{N} t' \neq \emptyset \). Hence \( \bullet_{N} t = \bullet_{N} t' \) and thus \( \bullet_{\beta_{N(C_N)}} t = \bullet_{\beta_{N(C_N)}} t' \). Therefore \( \beta_N(N,C_N) \) is FC.

h) Let \( M \in \beta_{N(C_N)}[M_I] \setminus \{ M_0 \} \). Then using (e) we can deduce \( M_I \xrightarrow{\sigma} N, M \), thus there exists a \( t \in T^c \) such that \( M[t] \) (as \( N \) is sound). If \( t \in T^c \) then we are done. If \( t \in B^c \) then there exists a \( t' \in T^c \) such that \( \bullet_{N} t = \bullet_{\beta_{N(C_N)}} t' \) (e). Thus \( M[t'] \).

While propositions a, b, d, e and f follow directly from the construction of a configured net and hold for non FC WF-nets, propositions c, g, and h are particularly interesting for soundness. The problem in the example of Figure 3.4 is that the configuration may yield an unsound model when a transition is blocked which shares part of its preset with another transition. By definition, in an FC WF-net such a situation cannot exist and therefore a deadlock marking cannot occur (propositions c and h). Further on, the deadlock in the example prevents all tokens from reaching the final place. As the configured net derived from an FC WF-net remains FC (proposition g), the FC property prevents also this problem since it permits any token to move towards the final place.

These properties allow us to prove that if a configured net derived from a sound, FC WF-net is a WF-net, the configured net fulfills the soundness criteria. Formally:

**Theorem 3.12** Let \( N = (P,T,F) \) be a sound, FC WF-net with source place \( p_1 \) and sink place \( p_o \), let \( C_N \) be a configuration of \( N \) and let \( \beta_N(N,C_N) = (P^c,T^c,F^c) \) be the resulting configured net. If \( \beta_N(N,C_N) \) is a WF-net, then \( \beta_N(N,C_N) \) is sound.
Proof Changing a transition into a silent transition (hiding) has no implications for soundness analysis.

- proper completion: since $\beta_N(N,C_N)(M_I) \subseteq N[M_I]$ (Proposition 3.11f), $M_O$ is the only state marking $p_O$.
- option to complete: because $\beta_N(N,C_N)$ is a FC WF-net (Proposition 3.11g), any token can decide to move towards $p_O$. If $p_O$ is marked, all other places are empty ($\beta_N(N,C_N)$ has proper completion). Hence, marking $M_O$ can be reached (and the property holds) or the net is in a deadlock $M$. However, this is not possible as $\beta_N(N,C_N)$ has no deadlock markings (Proposition 3.11h).
- no dead transitions: we define a length function as follows: $L : T^C \rightarrow \mathbb{N}$. If $p_I \in \bullet t$ then $L(t) = 0$. Otherwise $L(t) = 1 + \min_{p \in \bullet t, t' \in \bullet p} L(t')$. Given that every transition in $\beta_N(N,C_N)$ is on a path from $p_I$, the function is well-defined. Using induction we prove $\forall n \in \mathbb{N} \forall t \in T^C \ [L(t) = n \Rightarrow t \text{ is not dead in } \beta_N(N,C_N)]$.

(Base case) If $n = 0$ then $\bullet t = \{p_I\}$ and as $p_I \in P^C$ (Proposition 3.11b), $M_I[t]$, hence $t$ is not dead.

(Induction Hypothesis (IH)) If $t \in T^C$ is such that $L(t) = n + 1$, there exists a transition $t'$ such that $L(t') = n$ and $t' \cap \bullet t \neq \emptyset$. $t'$ is not dead (IH), hence there exists an $M \in \beta_N(N,C_N)(M_I)$ such that $M[t']$. Let $M'$ be such that $M \rightarrow M'$, then $M'$ marks at least one input place (i.e., $p$) of $t$. As $\beta_N(N,C_N)$ has the option to complete, $M' \rightarrow M_O$. This implies that some transition $t''$ exists which removes the token from $p$ in some marking $M'$, hence $p \in \bullet t''$. Therefore $\bullet t \cap \bullet t'' \neq \emptyset$, and thus, given that $\beta_N(N,C_N)$ is FC (Proposition 3.11g) $\bullet t = \bullet t''$. Therefore $M'[t]$ and $t$ is not dead. \qed

By combining Theorem 3.9 and Theorem 3.12 we can state that a configured net is sound if and only if the process constraint $PC_N$ is satisfied for the corresponding configuration. If the configured net is not an FC WF-net, the implication only holds in one direction and in the other direction soundness cannot be guaranteed. In these cases $PC_N$ can still be used to rule out all the syntactically incorrect process models, but conventional analysis tools such as Woflan [VBA01] have to be used in addition.

In the next sections we demonstrate how the conceptual foundation can be used to ground the configuration of business process models defined in the C-EPC notation. Then we show how the correctness results can be exploited for the staged configuration of C-EPC models.

### 3.4 Configurable EPCs

As described in Chapter 2, configuration in C-EPC is achieved by restricting the behavior of configurable nodes, which can be functions and connectors. Figure 3.5 depicts the travel requisition approval process in C-EPC, where we defined four configurable functions and five configurable XOR connectors.

Intuitively, leaving a configurable function $ON$ corresponds to allowing a transition in a configurable Workflow net. Similarly, switching a function $OFF$ corresponds to hiding a transition, since this implies some work to be skipped. Configuring a function
to \textit{OPT} is just a combination of the previous two options, as it permits one to defer the decision of whether to execute or skip the function until run-time, on an instance-by-instance basis. Along the same lines we can draw a parallelism between the restriction of a connector’s routing behavior in C-EPC and the blocking operator in a configurable Workflow net. For example, configuring an XOR-split to one of its outgoing sequences corresponds to blocking all the transitions in the postset of a place except the one that starts the selected sequence.

To better understand this relation between C-EPCs and Workflow nets, we first need to formally define the EPC syntax, its semantics and the concept of configurable EPC.

![Process model for travel form approval in C-EPC.]

\textbf{Figure 3.5:} The process model for travel form approval in C-EPC.
3. Conceptual Foundation for Process Model Configuration

3.4.1 EPCs: Syntax

An EPC is defined as a combination of events, functions and connectors as the nodes of a graph [Aal99]. These nodes are connected through a set of arcs, while each connector can be of type OR, AND and XOR.

**Definition 3.13 (Event-driven Process Chain)** An Event-driven Process Chain (EPC) is a five-tuple \( \Upsilon = (E,F,C,l,A) \) where:

- \( E \) is a finite non-empty set of events,
- \( F \) is a finite non-empty set of functions,
- \( C \) is a finite set of logical connectors \((E \cap F = \emptyset, E \cap C = \emptyset, F \cap C = \emptyset)\),
- \( l \in C \rightarrow \{OR, AND, XOR\} \) is a mapping defining the type of each connector (AND or XOR), and
- \( A \subseteq (E \times F) \cup (F \times E) \cup (E \times C) \cup (C \times E) \cup (F \times C) \cup (C \times F) \cup (C \times C) \) is a set of arcs.

Further on, we define auxiliary sets, such as preset and postset of a node, and predicates, such as a path of nodes, which allow us to describe EPCs in a more compact way.

**Notation 3.14 (EPC auxiliary sets and predicates)** Let \( \Upsilon = (E,F,C,l,A) \) be an EPC. Then:

- \( \forall n \in E \cup F \cup C \quad n = \{x \in E \cup F \cup C \mid (x, n) \in A\} \) is the preset of \( n \),
- \( \forall n \in E \cup F \cup C \quad n^\bullet = \{x \in E \cup F \cup C \mid (n, x) \in A\} \) is the postset of \( n \),
- \( C_{OR} = \{c \in C \mid l(c) = OR\} \) is the set of OR connectors,
- \( C_{AND} = \{c \in C \mid l(c) = AND\} \) is the set of AND connectors,
- \( C_{XOR} = \{c \in C \mid l(c) = XOR\} \) is the set of XOR connectors,
- \( C_S = \{c \in C \mid \|c\| = 1 \land |c^\bullet| > 1\} \) is the set of split connectors,
- \( C_J = \{c \in C \mid \|c\| > 1 \land |c^\bullet| = 1\} \) is the set of join connectors,
- \( C_{EF} \subseteq C \) such that \( c \in C_{EF} \) iff there is a path \( p = \langle n_1, n_2, \ldots, n_k \rangle \) such that \( n_1 \in E, n_2, \ldots, n_k-1 \in C, n_k \in F \) and \( c \in \{n_2, \ldots, n_k-1\} \) is the set of connectors between events and functions, and
- \( C_{FE} \subseteq C \) such that \( c \in C_{FE} \) iff there is a path \( p = \langle n_1, n_2, \ldots, n_k-1, n_k \rangle \) such that \( n_1 \in F, n_2, \ldots, n_k-1 \in C, n_k \in E \) and \( c \in \{n_2, \ldots, n_k-1\} \) is the set of connectors between function and events.

Similar to the definition of syntactically correct Workflow net (Definition 3.2), a syntactically correct EPC must fulfil a set of requirements. It must have a unique start event (as a Workflow net requires a unique input place) and a unique end event (as a Workflow net requires a unique output place). EPCs with multiple start and end
events can always be transformed to EPCs with a single start and a single end event by merging all the start events into a new start event via an OR-join, and all the end events into a new end event via an OR-split [MDA08]. A further requirement for the syntactic correctness is that all (C-)EPC nodes need to be on a directed path from the unique start to the unique end event (similar to the nodes of a Workflow net). In addition to Workflow nets, connectors can only be of type split or join (i.e., they cannot have only one incoming and one outgoing arc), events must have at most one incoming and one outgoing arc, while functions must have exactly one incoming and one outgoing arc. As functions can be triggered by events or be triggers to events, this order must always be retained, thus it is not possible to have a connector between two functions or two events. This also applies when merging start or end events, where a silent function needs to be added as shown in Figure 3.6.

![Figure 3.6: Merging of start (a) and end (b) events.](image)

**Definition 3.15 (Syntactically correct EPC)** Let $\Upsilon = (E,F,C,l,A)$ be an EPC and $A^*$ be the reflexive transitive closure of $A$. $\Upsilon$ is syntactically correct iff:

- events have at most one incoming and one outgoing arc, i.e. $\forall e \in E\ | \cdot e| \leq 1 \land |e\cdot| \leq 1$,
- functions have exactly one incoming and one outgoing arc, i.e. $\forall f \in F\ |\cdot f| = 1 \land |f\cdot| = 1$,
- $C_S$ and $C_J$ partition $C$, i.e. $C_S \cap C_J = \emptyset$ and $C_S \cup C_J = C$,
- $C_{EF}$ and $C_{FE}$ partition $C$, i.e. $C_{EF} \cap C_{FE} = \emptyset$ and $C_{EF} \cup C_{FE} = C$,
- there exists exactly one start event, i.e. $\exists e_S \in E\ e_S = \emptyset$,
- there exists exactly one end event, i.e. $\exists e_E \in E\ e_E = \emptyset$, and
- every node is on a directed path from start to end event, i.e. $\forall n \in E \cup F \cup C\ [(e_S, n) \in A^* \land (n, e_E) \in A^*]$.

Accordingly, the EPC of Figure 3.5 is syntactically correct.

### 3.4.2 EPCs: Semantics

There is currently no commonly agreed understanding of the EPC semantics with regards to the OR-join. The intended behavior of this connector is to wait until all “active” incoming branches complete. That is, the OR-join needs to know whether the incoming branches may eventually receive a token. If this is the case, the join should
3. Conceptual Foundation for Process Model Configuration

wait, otherwise it should fire by taking a token from each incoming branch that has completed. This feature is called *non-locality* since the state of all transitive predecessor nodes has to be computed [MA07].

Different proposals are available in literature to tackle this issue, e.g. [Rit00, LSW98, NR02, Kin06, MA07], but most of them impose syntactic restrictions in the EPC, e.g. no cycles, block-structureness [Rit00, LSW98], or have drawbacks [NR02, Kin06]. Therefore, in the following we provide a general definition of EPC soundness which abstracts from the subtle differences of the above approaches. In particular, this definition is based on the three requirements of option to complete, proper completion and no dead functions as in Workflow nets. These requirements, which are compatible with the above proposals, must always hold for an EPC to be sound. However we only provide an informal definition for reachable marking.

An EPC describes a process with an initial and a final state. A state of the execution of an EPC can be identified by the marking of its events with tokens. The marking $M_I$ represents the initial state where only $e_S$ holds a token, while the marking $M_O$ represents the final state where only $e_E$ holds a token. A sound process should always terminate properly and it should be possible to execute any function by following an appropriate route through the EPC. Formally:

**Definition 3.16 (Sound EPC)** Let $\Upsilon = (E, F, C, l, A)$ be a syntactically correct EPC and $M_I$ and $M_O$ be the initial and final markings. Then $\Upsilon$ is sound iff:

- option to complete: for every marking $M$ reachable from $M_I$, there exists a firing sequence leading from $M$ to $M_O$, and
- proper completion: $M_O$ is the only marking reachable from $M_I$ where only $e_E$ holds, and
- no dead functions: every function can be reached by $M_I$.

Accordingly, the EPC of Figure 3.5 is also sound. As in Workflow nets, syntactic correctness does not imply soundness. For instance, if we replaced the last XOR-join in the model of Figure 3.5 with an AND-join, the model would still be syntactically correct but not sound. In fact the final event will never be reached because the first XOR-split would trigger only one of the two outgoing paths while the AND-join would wait for the completion of all its incoming branches. In more complex EPCs such a mismatch between split and join connectors might be trickier to spot.

3.4.3 EPCs: Configuration

A configurable EPC is essentially an EPC enhanced with two kinds of variation points: configurable functions and configurable connectors. Both configurable functions and configurable connectors are integrated into an EPC as regular nodes with the only difference that they are marked as configurable. Thus to define a C-EPC we just need to mark a subset of these nodes as configurable. Although in the original C-EPC definition [RA07] AND connectors could be marked as configurable, their behavior could never be restricted, i.e. a configurable AND could only be configured to a regular AND. In the following we decided to forbid this option a priori by allowing only configurable OR and XOR connectors.
Definition 3.17 (Configurable EPC) A configurable EPC (C-EPC) is a seven-tuple \( \Gamma = (E, F, C, l, A, F^C, C^C) \) where:

- \((E, F, C, l, A)\) is an EPC,
- \(F^C \subseteq F\) is the set of configurable functions,
- \(C^C \subseteq C_{OR} \cup C_{XOR}\) is the set of configurable connectors.

Clearly, Definitions 3.15 and 3.16 hold for C-EPCs too.

Before presenting the notion of C-EPC configuration, a partial order \( \leq^C \) is introduced to order configurable connectors from specific to more generic ones. \( \leq^C \) forces a configurable OR to be configured as a regular OR, AND, XOR or as a sequence operator \( SEQ_n \) starting with node \( n \), while a configurable XOR to be configured as a regular XOR or as a sequence operator.

Definition 3.18 (C-EPC Partial Order) The partial order \( \leq^C \) is defined on \( CT \cup CTS \) where \( CT = \{ OR, XOR \} \) is the set of configurable connector types and \( CTS = \{ SEQ_n \mid n \in E \cup F \cup C \} \) is the set of sequence operators. \( \leq^C = \{(n, n) \mid n \in CT\} \cup \{(AND, OR), (XOR, OR)\} \cup CTS \times \{XOR, OR\} \).

For example \( SEQ_n \leq^C XOR \) implies that a configurable XOR can be mapped onto \( SEQ_n \), i.e., a choice may be replaced by a sequence but not the other way around.

A configuration is a function that maps a configurable node to an allowed value (i.e. variant) according to the node type. It also ensures that a sequence can be chosen as configuration value only if it is an outgoing branch of a configurable split, or an incoming branch of a configurable join.

Definition 3.19 (C-EPC Configuration) Let \( \Gamma = (E, F, C, l, A, F^C, C^C) \) be a syntactically correct C-EPC. The mapping \( C_\Gamma \in (F^C \rightarrow \{ ON, OPT, OFF \}) \cup (C^C \rightarrow CT \cup CTS) \) is a configuration of \( \Gamma \) iff for each \( c \in C^C \cap \text{dom}(C_\Gamma) \):

- \( C_\Gamma(c) \leq^C l(c) \),
- if \( c \in C_S \) and \( C_\Gamma(c) = SEQ_n \) for some \( n \in E \cup F \cup C \), then \( n \in c \),
- if \( c \in C_J \) and \( C_\Gamma(c) = SEQ_n \) for some \( n \in E \cup F \cup C \), then \( n \in c \).

By applying a configuration \( C_\Gamma \) to a C-EPC, we can derive a new EPC from the original net if \( C_\Gamma \) assigns a value to all configurable nodes, or a partly configured C-EPC if only some configurable nodes are set. This is done in five steps. Firstly, we change the type of all configurable connectors that have been restricted by configuration to their new types and remove all arcs to or from the connectors that are not permitted to be taken any longer. Secondly, we replace all functions \( f \) that have been configured as \( OFF \) with silent functions \( skip_f \) to reflect that the original function is not executed. In this way, the structure of the net does not change. Thirdly, for all functions \( f \) that have been configured as \( OPT \) we insert an XOR-split just before \( f \) and an XOR-join just after \( f \), and link these connectors with a new function \( skip_f \), to allow \( f \) to be optionally skipped.
For each sequence of four events, we remove all elements that are after the first two steps are no longer on a path between the start event and the end event. Finally, we replace all connectors with a single incoming and a single outgoing arc with one arc, as they are no longer required. The following definition formalizes this algorithm, which has been adapted from [RA07] to handle partial configurations. Also, in step 2 we simply replace a function switched OFF with a silent function instead of removing it as in the original algorithm. This allows us to easily preserve the strict alternation of events and functions as required by Definition 3.15. In fact, to preserve this rule, in the original algorithm a function can only be removed if it is between two events, with the two events being merged into one. In all other cases the function is replaced by a silent one.

**Definition 3.20 (Configured EPC)** Let $\Gamma = (E, F, C, I, A, F^C, C^C)$ be a syntactically correct C-EPC and let $C_T$ be one of its configurations. $\beta_T(\Gamma, C_T)$ defines a (C-)EPC $\Psi$ constructed as follows:

1. Map the configurable connectors $c \in C^C \cap \text{dom}(C_T)$ onto their concrete type and remove arcs not involving the selected sequence, i.e. $\Psi_1 = (E, F, C, I_1, A_1)$ with $l_1 = I \oplus \{(c, C_T(c)) \mid c \in C^C \cap \text{dom}(C_T)\}$ and $A_1 = A \setminus \{(c, n) \in C_S \times c \bullet \mid \exists n' \in c, n' \neq n [C_T(c) = \text{SEQ}_{n'}] \} \cup \{(n, c) \in c \times C_F \mid \exists n' \in c, n' \neq n [\text{SEQ}_{n'}]\}.$

2. For each $f \in F^C \cap \text{dom}(C_T)$ such that $C_T(f) = \text{OFF}$, replace $f$ with a new function $\text{skip}_f$, i.e. $\Psi_2 = (E, F_2, C, I_1, A_1)$ with $F_2 = F \oplus \{\text{skip}_f \mid f \in F^C \cap \text{dom}(C_T) \land C_T(f) = \text{OFF}\}.$

3. For each $f \in F^C \cap \text{dom}(C_T)$ such that $C_T(f) = \text{OPT}$, add a new function $\text{skip}_f$ and surround $f$ and $\text{skip}_f$ with an XOR-split and an XOR-join, i.e. $\Psi_3 = (E, F_3, C_3, I_3, A_3)$ with $F_3 = F_2 \cup \{\text{skip}_f\}, C_3 = C \cup \{\text{split}_f, \text{join}_f\}, l_3 = I_1 \cup \{(\text{split}_f, \text{XOR}), (\text{join}_f, \text{XOR})\}, A_3 = \{(n_1, n_2) \in A_1 \mid f \notin \{n_1, n_2\}\} \cup \{(\text{split}_f, f), (\text{split}_f, \text{skip}_f), (\text{join}_f, f), (\text{join}_f, \text{skip}_f), (f, \text{join}_f)\} \cup \{(n, \text{split}_f) \mid (n, f) \in A_1\} \cup \{(\text{join}_f, n) \mid (f, n) \in A_1\}. \text{ Repeat this step for all such } f \text{ and let } \Psi_3 \text{ be the resulting (C-)EPC.}$

4. Remove all nodes not on some path from the start event $e_S \in E$ to the end event $e_E \in E$. Let $\Psi_4 = \{E_4, F_4, C_4, I_4, A_4\}$ be the resulting (C-)EPC.

5. Remove all connectors with just one input and one output node, i.e. $\beta_T(\Gamma, C_T)$ = $\Psi = (E_4, F_4, C_5, I_5, A_5)$ with $C_5 = \{c \in C_4 \mid |c \bullet| > 1 \lor |c \cdot| > 1\}, I_5 = \{(c, x) \in I_4 \mid c \in C_5\}$ and $A_5 = \{(n_1, n_2) \in A_4 \mid \{n_1, n_2\} \cap (C_4 \setminus C_5) = \emptyset\} \cup \{(n_1, n_2) \mid \exists c \in C_4 \setminus C_5 \{((n_1, c), (c, n_2)) \in A_4\}\}$.

Figure 3.7 shows the EPC resulting from a configuration of the travel form approval C-EPC (Figure 3.5). Here the first configurable XOR-split has been configured to $\text{SEQ}_{f_1}$, allowing only international travels, the configurable XOR-split after $e_2$ and $e_3$ has been configured to $\text{SEQ}_{f_2}$, thus forbidding the secretary to prepare the travel form, and the configurable function $f_3$ has been switched OFF to deny the possibility to drop an application. All other configurable nodes have not been restricted.

As per the $\beta_T$ algorithm, the configuration of the top XOR-split leads to the removal of its outgoing arc to $f_2$ (step 1). In doing so, all nodes subsequent to $f_2$ until the last

---

4$\oplus$ is the override operator.
**Figure 3.7:** The configured EPC for international travels from the C-EPC of Figure 3.5.

XOR-join (excluded) are no longer reachable from $e_S$ and thus they are removed (step 4). Similarly, the configuration of the XOR-split after $e_2$ and $e_3$ makes $f_5$ no more reachable from $e_S$. As a result, $f_5$ is removed from the net (step 4) and the XOR-join after $f_5$ remains only with a single incoming arc and a single outgoing arc and is therefore removed too (step 5). Switching function $f_9$ OFF leads to its replacement with a silent function (step 2).

In [RA07] the authors prove the following theorem which states that the resulting (C-) EPC $\beta_T(\Gamma, C_T)$ is syntactically correct:

**Theorem 3.21** Let $\Gamma = (E, F, C, I, A, F^C, C^C)$ be a syntactically correct C-EPC and let $C_T$ be one of its configurations. $\beta_T(\Gamma, C_T)$ is a syntactically correct (C-) EPC.

Indeed only step 1 of $\beta_T$ may cause issues by disconnecting some branches. However these issues are always fixed by step 4 of the algorithm, which removes all nodes not on a path from $e_S$ to $e_E$. At most this operation may lead to an empty net if $e_S$ and/or $e_E$ become disconnected after step 1, i.e. if there no longer exists a path connecting these two events. The configured EPC in Figure 3.7 is syntactically correct.
In the next section we show that the configuration of C-EPC models corresponds to applying the hiding and blocking operators onto a Petri net that is induced by the C-EPC. To do that, we first present the mapping between C-EPCs and Petri nets.

3.5 Formal relation between C-EPCs and configurable Workflow nets

The mapping of EPCs to Petri nets has been discussed for more than a decade. The reason for this mapping has been to exploit the abundance of analysis techniques and tools available for Petri nets in the context of EPCs. However, while the mapping of XOR and AND connectors is rather trivial, the OR-join poses considerable challenges. In essence, the problem stems from the fact that its non-local semantics implies a recursive definition if there are multiple OR-joins in a loop. As shown in [Kin06], a unique fixed point is not guaranteed for evaluating such a definition. Nevertheless, the OR-join does not add behavior in EPCs [MDA08] – it only represents a process in a more compact way. Furthermore, it has been shown that a behavior-equivalent Petri net can always be constructed [MDA08] using the theory of regions [ER89, CKLY98], although the resulting model may be rather complex. This significant increase in the complexity also holds for OR-splits although their mapping is generally less challenging than the one of the OR-join. Since the OR connector does not increase the model’s expressiveness but complicates the mapping dramatically, we reuse the mapping proposed in [Aal99] which abstracts from this element.

To induce a Petri net from a C-EPC, we first create an expanded C-EPC in which we get rid of all chains of connectors. We do this by replacing any arc between two connectors by a silent function, a silent event and arcs to connect these new elements with the two connectors in question. In this way, connectors can only be linked with functions and events, and not with other connectors. These additional nodes do not correspond to observable behavior and thus do not change the overall behavior captured by the model, but are merely added to simplify the mapping to Petri nets. We base the following definitions on C-EPCs although they also hold for EPCs.

Definition 3.22 (Expanded C-EPC) Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct C-EPC. $\Gamma' = (E', F', C, l, A', F^C, C^C)$ is the expanded net of $\Gamma$ such that:

- $E' = E \cup \{e^a | a \in A \cap (C \times C)\}$,
- $F' = F \cup \{f^a | a \in A \cap (C \times C)\}$,
- $A' = A \setminus (C \times C) \cup \{(c, f(c,d)) | f(c,d) \in F' \wedge c \in C_{EF} \} \cup \{(c, e(c,d)) | e(c,d) \in E' \wedge c \in C_{EF} \} \cup \{(f(c,d), e(c,d)) | f(c,d) \in F' \wedge e(c,d) \in C_{EF} \} \cup \{(e(c,d), f(c,d)) | f(c,d) \in F' \wedge e(c,d) \in C_{FE} \} \cup \{(e(c,d), d) | e(c,d) \in E' \wedge c \in C_{EF} \} \cup \{(f(c,d), d) | f(c,d) \in F' \wedge e(c,d) \in C_{FE} \}.$

For simplicity, in the remainder we only consider expanded C-EPCs. A Petri net is constructed by mapping each event of a C-EPC onto a place and each function onto a transition. If events are directly connected to functions or vice versa, we can map the in-between arc onto a flow relation. In case of connectors we have to re-build their
splitting or joining behavior in Petri net. For this, we use the transformations depicted in Figure 3.8 which might require inserting additional silent transitions and places.

An AND-join between a set of events and one function is directly mapped to the synchronizing behavior of the transition corresponding to the function, with the events corresponding to the places in the function’s preset (Figure 3.8.a). Otherwise, if the AND-join is between a set of functions and one event, we need to add two silent places and one silent transition to capture the synchronizing behavior (Figure 3.8.b). Similarly, an XOR-join between a set of functions and one event is directly mapped to the merging behavior of the place corresponding to the event, with the functions corresponding to the transitions in the place’s preset (Figure 3.8.d). Otherwise, if the XOR-join is between a set of events and one function, we need to add two silent transitions and one silent place to capture the merging behavior (Figure 3.8.c). Similar considerations hold for the mapping of split connectors (Figure 3.8.e-h). As mentioned before, we assume the C-EPC has no OR connectors. The following definition, adapted from [Aal99], formally describes the induced Petri net.

**Definition 3.23 (Induced Petri net)** Let \( \Gamma = (E, F, C, I, A, F^C, C^C) \) be a syntactically correct C-EPC with \( C_{OR} = \emptyset \). \( \mathcal{N} (\Gamma) = (P^{PN}, T^{PN}, F^{PN}) \) is the Petri net induced
by $\Gamma'$ such that:

- $P^P_N = E \cup \bigcup_{c \in C} P^P_N$,
- $T^P_N = F \cup \bigcup_{c \in C} T^P_N$,
- $F^P_N = (A \cap ((E \times F) \cup (F \times E))) \cup \bigcup_{c \in C} F^P_N$,

where $P^P_N$, $T^P_N$, and $F^P_N$ are defined as per Table 3.1.

| $c \in C_{EF} \cap C_J \cap C_{AND}$ | $\emptyset$ | $\emptyset$ | $\{(x, y) \mid x \in \bullet c \land y \in \bullet c\}$ |
| $c \in C_{FE} \cap C_J \cap C_{AND}$ | $\{p^c_x \mid x \in \bullet c\}$ | $\{t^c\}$ | $\{(x, p^c_x) \mid x \in \bullet c\} \cup$ $\{(p^c_x, t^c) \mid x \in \bullet c\} \cup$ $\{(t^c, x) \mid x \in \bullet c\}$ |
| $c \in C_{EF} \cap C_J \cap C_{XOR}$ | $\{p^c\}$ | $\{t^c_x \mid x \in \bullet c\}$ | $\{(x, t^c_x) \mid x \in \bullet c\} \cup$ $\{(t^c_x, p^c) \mid x \in \bullet c\} \cup$ $\{(p^c, t^c_x) \mid x \in \bullet c\}$ |
| $c \in C_{FE} \cap C_J \cap C_{XOR}$ | $\emptyset$ | $\emptyset$ | $\{(x, y) \mid x \in \bullet c \land y \in \bullet c\}$ |
| $c \in C_{EF} \cap C_S \cap C_{AND}$ | $\{p^c_x \mid x \in \bullet c\}$ | $\{t^c\}$ | $\{(x, t^c) \mid x \in \bullet c\} \cup$ $\{(t^c_x, p^c) \mid x \in \bullet c\} \cup$ $\{(p^c_x, t^c) \mid x \in \bullet c\}$ |
| $c \in C_{FE} \cap C_S \cap C_{AND}$ | $\emptyset$ | $\emptyset$ | $\{(x, y) \mid x \in \bullet c \land y \in \bullet c\}$ |
| $c \in C_{FE} \cap C_S \cap C_{XOR}$ | $\{p^c\}$ | $\{t^c_x \mid x \in \bullet c\}$ | $\{(x, p^c) \mid x \in \bullet c\} \cup$ $\{(p^c_x, t^c) \mid x \in \bullet c\} \cup$ $\{(t^c_x, x) \mid x \in \bullet c\}$ |

Table 3.1: Mapping a C-EPC connector $c \in C$ to places, transitions and arcs (see Figure 3.8).

It is easy to see that for any syntactically correct C-EPC $\Gamma$, $\mathcal{N}(\Gamma) = (P^P_N, T^P_N, F^P_N)$ is a Petri net, since by definition $P^P_N \cap T^P_N = \emptyset$ and $F \subseteq (P^P_N \times T^P_N) \cup (T^P_N \times P^P_N)$. Moreover, the Petri net is a Workflow net fulfilling the free-choice property, as illustrated by the following lemma which extends a lemma in [Aal99].

**Lemma 3.24** Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct C-EPC with $C_{OR} = \emptyset$ and $\mathcal{N}(\Gamma)$ be its induced Petri net, then:

a) $\mathcal{N}(\Gamma)$ is a WF-net.
b) $\mathcal{N}(\Gamma)$ is FC.

Proof

a) $\mathcal{N}(\Gamma)$ is induced by a syntactically correct C-EPC where all nodes are on a directed path from start to end event, and the rules given in Table 3.1 do not violate this property.

b) See [Aal99]. □

Figure 3.9 depicts the expanded net for the C-EPC of Figure 3.5 and its induced Petri net, while Figure 3.10 depicts the configured EPC obtained from this expanded net using the example configuration, and its induced Petri net. Both these Petri nets are free-choice Workflow nets since all transitions that are preceded by a place with multiple outgoing arcs have only that place in their preset (Definition 3.10). In fact we can observe that since the C-EPC and its configured EPC are syntactically correct, an event cannot have more than one output arc (Definition 3.15), thus the only way to obtain a place with multiple output arcs is the mapping of XOR-split connectors onto Petri net constructs. But according to the rules given in Table 3.1, the transitions in these constructs can never synchronize, i.e. they only contain one place mapping the split in their preset.

The following proposition, proven in [Aal99], states that a C-EPC is sound if and only if its induced Workflow net is sound.

Proposition 3.25 Let $\Gamma = (E,F,C,1,A,F^C,C^C)$ be a C-EPC with $C_{OR} = \emptyset$, $\mathcal{N}(\Gamma)$ be its induced WF-net. $\Gamma$ is sound iff $\mathcal{N}(\Gamma)$ is sound.

The Workflow net of Figure 3.9.b is sound, as is its originating C-EPC. However, as mentioned before, if we changed the behavior of the last join in the C-EPC with an AND, the model would become unsound, and so would be its induced Workflow net. In fact this unbalance between XOR and AND translates into a so-called PT-handle [ES90] in Petri nets. A PT-handle is a construct in which a place is complemented by a transition or vice versa, and may lead to no proper completion or to a deadlock like in this case.

We can now formalize the relation between configurable C-EPCs and configurable Workflow nets through their mapping. We do this by deriving a WF-net configuration from a C-EPC configuration and projecting it onto the induced Workflow net. If a configurable function is switched OFF, the corresponding transition in the Workflow net is to be hidden. If a configurable XOR-split (-join) is restricted to a sequence, all the transitions in the postset of the place mapping the split (or in the preset of the place mapping the join) must be blocked, except the transition starting the sequence.

Definition 3.26 (Induced WF-net Configuration) Let $\Gamma = (E,F,C,1,A,F^C,C^C)$ be a syntactically correct C-EPC with $C_{OR} = \emptyset$, $\mathcal{C}_\Gamma$ be one of its configurations and $\mathcal{N}(\Gamma) = (P^{PN}, T^{PN}, F^{PN})$ be its induced WF-net. $\mathcal{C}^{\mathcal{N}}_{\Gamma} \in T^{PN} \rightarrow \{\text{allow, hide, block}\}$ is the configuration of $\mathcal{N}(\Gamma)$ induced by $\mathcal{C}_\Gamma$. For $t \in T^{PN}$:
Figure 3.9: a) The expanded net for the C-EPC of Figure 3.5 and b) its induced Petri net.

In the above definition we do not consider functions set to \( OPT \). Configuring a function to \( OPT \) corresponds to “optionally” hiding the respective transition in the induced Petri net. The idea of optional hiding and blocking was introduced in [GAJ07] and applied to LTSs. These operators are an extension of the hiding and blocking operators to allow one to defer the decision whether or not to hide (block) a transition till run-time. For example, to incorporate the optional hiding we need to extend Definition 3.6 such that when a transition \( t \) is configured as such, a silent transition \( \text{skip}_t \) is added and
connected to the places in the preset and postset of \( t \). In doing so, both the free-choice and the Workflow net properties are preserved. However, given the structural changes that are involved and given that essentially setting a function to \( \text{OPT} \) is just a deferral of the choice till run-time, we abstract from it for the moment to achieve a simpler configured WF-net. Later we will reason show that indeed this abstraction does not have any implications on the relation between C-EPCs and configurable WF-nets.

Figure 3.11.a shows the induced WF-net configuration applied to our example model (Figure 3.9.b), where \( f_2 \) and \( f_3 \) have been blocked and \( f_9 \) has been hidden. To determine which further nodes need to be removed in order to preserve the syntactic correctness of this model, we use the process constraint \( PC_N(\Gamma) \) and a SAT solver as illustrated in Section 3.3.1. In our example, \( PC_N(\Gamma) \) imposes to remove \( e_4 \) and \( f_5 \) because these nodes would no longer be on a directed path from the input to the output place after removing \( f_2 \). Figure 3.11.b depicts the configured WF-net, which we call \( \beta_N^*(N(\Gamma), C_N^x) \). This is the configured net obtained after applying \( C_N^x \) and removing all nodes that are not on a directed path from the input to the output place in order to fulfil \( PC_N(\Gamma) \).

We observe that in our example \( \beta_N^*(N(\Gamma), C_N^x) \) is equal to the Workflow net induced by the configured EPC, i.e. to \( N(\beta(\Gamma, C_{\bar{\eta}})) \) (see Figure 3.10.b). We conjecture this
equality relation between $\beta^*_N(N(\Gamma), C^\ell_N)$ and $N(\beta_T(\Gamma, C_T))$, depicted in Figure 3.12, holds in general for any syntactically correct C-EPC without OR connectors. In fact, we can also observe that configuring functions to OPT does not contradict this equality relation. This is because the Petri net construct induced by a function configured to OPT in the configured EPC ($\beta_T(\Gamma, C_T)$) is the same as the construct produced by applying the optional hiding operator to the Petri net induced by the C-EPC ($N(\Gamma)$), as shown in Figure 3.13.

$$
\begin{array}{c}
\Gamma \\
\downarrow C_R \\
N \\
\downarrow \beta_N(N(\Gamma), C^\ell_N) \\
\beta_T(\Gamma, C_T) \\
\downarrow N \\
N(\beta_T(\Gamma, C_T))
\end{array}
$$

Figure 3.12: The relation between configurable EPCs and configurable WF-nets.
Figure 3.13: Configuring a function to OPT in C-EPC means optionally hiding the corresponding transition in the induced Petri net.

We can now formulate the following proposition which shows that if we configure a sound C-EPC without OR connectors for which the equality relation between $\beta_N^*(N(\Gamma), C_{\Gamma}^T)$ and $N(\beta_T(\Gamma, C_T))$ holds, the resulting EPC is always sound.

Proposition 3.27 (Soundness-preserving C-EPC configuration) Let $\Gamma$ be a sound C-EPC with $C_{OR} = \emptyset$, $C_\Gamma$ be one of its configurations and $N(\Gamma)$ be its induced WF-net. Let also $C_{\Gamma}^T$ be the configuration of $N(\Gamma)$ induced by $C_\Gamma$ and $\beta_N^*(N(\Gamma), C_{\Gamma}^T)$ be the configured net in which all the nodes not on a directed path from the input to the output place have been removed to fulfil PC$_{N(\Gamma)}$. If $N(\beta_T(\Gamma, C_T))$ is equal to $\beta_N^*(N(\Gamma), C_{\Gamma}^T)$, then $\beta_T(\Gamma, C_T)$ is sound.

Proof We observe that: (i) $\Gamma$ is sound, hence its configured EPC $\beta_T(\Gamma, C_T)$ is syntactically correct (Theorem 3.21) and $N(\Gamma)$ is a FC WF-net (Lemma 3.24) and is sound (Proposition 3.25); (ii) since $\beta_T(\Gamma, C_T)$ is syntactically correct, its induced Petri net $N(\beta_T(\Gamma, C_T))$ is a FC WF-net (Lemma 3.24). Thus $\beta_N^*(N(\Gamma), C_{\Gamma}^T)$ is sound, since it is the configured FC WF-net of $N(\Gamma)$ which is sound (Theorem 3.12). If $N(\beta_T(\Gamma, C_T))$ is equal to $\beta_N^*(N(\Gamma), C_{\Gamma}^T)$, then $N(\beta_T(\Gamma, C_T))$ is sound. Hence $\beta_T(\Gamma, C_T)$ is sound (Proposition 3.25).

3.6 Correctness-Preserving Configuration of C-EPCs

In this section we first reason on the soundness issues that can be created while configuring a C-EPC that includes OR connectors. Next, we show how the technique for staged correctness-preserving configuration of Workflow nets can be extended to deal
with C-EPCs. Finally, we describe the implementation of this technique in the Synergia toolset.

### 3.6.1 Soundness Issues in C-EPC Configuration

The result of Proposition 3.27 is intuitive: without configurable OR connectors a sound C-EPC can never turn unsound. In fact, by configuring the model in Figure 3.5 the only issues that may be produced are syntactic ones, e.g. some nodes are no longer on a path from the start to the end event as a result of configuring an XOR to a sequence. However, these issues get then fixed by the $\beta_F$ algorithm which ensures the syntactic correctness of the individualized net (Theorem 3.21).

An OR connector is a potential source of behavioral issues because its configuration can lead to various types of mismatches between splits and joins. A simple example is provided by the C-EPC in Figure 3.14.a. Although this net is sound, if we configured the OR-split to an XOR and the OR-join to an AND (Figure 3.14.b), we would obtain a mismatch between these two connectors and thus the individualized net would be unsound, even if it is syntactically correct.

![Figure 3.14: Configuring a C-EPC with OR connectors may lead to unsound individualizations.](image)

In addition to the mismatches between AND and XOR, the configuration of the OR connector can generate two new mismatches: one between OR-split and XOR-join and another between OR-split and AND-join, as depicted in Figure 3.15. The first construct may lead to no-proper completion if the OR-split activates more than one incoming branch of the XOR-join. The second construct may lead to a deadlock if the OR-split does not activate all the incoming branches of the AND-join. Specifically, in C-EPC a mismatch is identified by two simple paths between a split and a join, where the paths only share these two nodes and the types of split and join belong to one of the four combinations shown in Figure 3.15.

As shown by the above example, in the presence of configurable OR connectors, a sound C-EPC may generate a syntactically correct EPC which is unsound. In the following, we argue that if joins are not configurable, a sound C-EPC always yields a sound configured EPC.

If joins cannot be configured, a configurable node can be either a function or a split. A configurable function which is configured to OFF simply becomes silent. Since the structure of the initial C-EPC does not change, the soundness of the configured EPC.
Figure 3.15: Possible split–join mismatches generated by configuring the OR connector.

is not affected. A configurable function which is configured to OPT yields only a local
structural change which, again, has no implications on the soundness of the configured
EPC.

For the configuration of splits, we need to make some observations. Let us consider
the OR-split. At run-time this connector can choose to behave as an AND-split if it
activates all outgoing branches, or as an XOR-split if it activates only one branch. This
means the set of markings produced by the OR-split transition includes those markings
produced by the XOR-split transition and the marking produced by the AND-split
transition, as illustrated in Figure 3.16. Since these choices can be made at run-time,
we can anticipate them at configuration-time without affecting the semantics of the
configured EPC.

Figure 3.16: The set of markings produced by the OR-split transition includes those
produced by the XOR-split transition and that produced by the AND-split transition.

Similarly, when configuring an OR-split or an XOR-split to one of their sequence values,
this corresponds to selecting one specific marking out of the ones that can be generated
at run-time by their respective transitions. Yet, this configuration leads to some nodes
being disconnected from the net, i.e. to structural issues. However, we have shown
that these issues are then fixed by step 4 of the $\beta_\Gamma$ algorithm which removes all nodes
not on some path from start to end event, leading to a syntactically correct configured
EPC (Theorem 3.21). Thus, we can also anticipate the choice of a specific sequence at
configuration-time without affecting the soundness of the configured EPC.

Therefore, we can conclude that since the configuration of both functions and split
connectors does not yield semantic issues, any sound C-EPC without configurable joins
that is configured with $\beta_\Gamma$ yields a sound EPC. As a matter of fact, Figure 3.17 shows
that if an OR-join is not configurable, it handles any configuration of the preceding
OR-split without generating semantic issues.

The C-EPC in Figure 3.18 is an example of a sound net where joins are not config-
urability. The OR-split can be freely set without affecting the soundness of the indi-
vidualized models, provided the syntactic correctness of the net is preserved. Therefore,
this restriction on the joins allows us to avoid semantical issues in the individualized
3. Conceptual Foundation for Process Model Configuration

Figure 3.17: A non-configurable OR-join handles all configurations of the preceding OR-split without generating semantic issues.

Figure 3.18: An example of sound C-EPC with no configurable joins. Any configured net obtained from this C-EPC will be sound so long as it will be syntactically correct.

We can argue that the restriction to C-EPCs without configurable joins is legitimate as follows. Firstly, any sound net resulting from the configuration of a join can always be obtained by configuring the splits instead of the joins. Thus, forbidding the configuration of joins does not reduce the configuration expressiveness. Secondly, in practice one would configure a C-EPC by restricting the routing behavior of the splits and not that of the joins, because the splits are the nodes of the process where business decisions are taken at run-time. Thirdly, if joins are configurable one must “match” their configuration according to that of the splits, to avoid the risk of generating behavioral issues. Finding this matching requires extra effort, above all if the net is unstructured [Aal99].
3.6.2 Deriving C-EPC Process Constraints

Although under the assumption that we have made the use of the $\beta_T$ algorithm is sufficient to ensure the soundness of the resulting models, there are some practical limitations stemming from the application of $\beta_T$. First of all, when we set a configurable function to $ON$ or $OPT$, there is no guarantee the function will remain in the model. In fact the function may belong to a branch that is removed after configuring an upstream split to a sequence. For instance, let us consider the C-EPC of Figure 3.18. Here, if we configure $f_5$ to $ON$, this function will still be removed from the model if $c_4$ is set to $SEQ_{f_6}$. Furthermore, if start and end event happen to be no longer connected by any path, $\beta_T$ will return an empty net by removing all nodes. In our example this can happen if $c_4$ is set to $SEQ_{f_5}$ and $c_5$ is set to $SEQ_{e_2}$.

These situations cannot be avoided unless we impose some restrictions on the values each configurable node can take, before applying $\beta_T$. Specifically, at each configuration step we need to check that: (i) there exists at least one directed path from $e_S$ to $e_E$ (to avoid the empty net), and (ii) any configurable function not switched OFF is on at least one such a path (to offer the function a chance to be executed). Similarly to Workflow nets, we can interactively check these conditions by enforcing the satisfiability of a system of propositional logic constraints inferred from a C-EPC. Specifically, we do not define a boolean variable for each variant of a variation point as we did for Workflow nets. When searching for a path in a C-EPC, it is sufficient to check the $SEQ_n$ values of the configurable splits because these are the only values that if selected can change the structure of the net (i.e. some nodes will get removed from the net). Configuring a split to any other of its values (e.g. configuring an $OR$ to an $XOR$) has no effect on the computation of a path. Accordingly, we define one variable $p_n^c$ for each value $SEQ_n$ of each configurable split $c$. The variable is three-valued: $true$, $false$, $unset$. Setting $p_n^c$ to $true$ indicates that $c$ has been configured with $SEQ_n$, thus the variables corresponding to the other outgoing sequences of $c$ are set to $false$.

In the computation of a path, when we reach a configurable split we set to $false$ all the variables for that split but the one corresponding to the sequence being taken. For example, when reaching $c_1$ in the model of Figure 3.18, we have to pick one of its outgoing arcs. If we take $SEQ_{e_2}$ we negate the variable for $SEQ_{c_1}$ or vice versa. We do not explicitly set $SEQ_{e_2}$ to $true$ because at run-time the path through it can be taken not only if $c_1$ is configured to $SEQ_{c_1}$, but also if the split is configured to any other of its variants except $SEQ_{c_2}$, e.g. to a regular $OR$.

To ensure the existence of at least one directed path from $e_S$ to $e_E$ we simply enforce the disjunction of all the paths from $e_S$ to $e_E$. To ensure each configurable function $f$ not switched OFF is on at least one such a path, we define a variable $p_{f,OFF}^c$ for the value $OFF$ of $f$ and impose that its negation implies the disjunction of all the paths traversing $f$ from $e_S$ to $e_E$. In doing so, we decompose the path into two sub-paths, one from $e_S$ to $e_E$, as we did for Workflow nets. We only need to use one variable for the value $OFF$ of a configurable function, because this variable being negated means the function can be configured to either $ON$ or $OPT$. For instance, for $f_2$ to be configured to $ON$ or $OPT$ in our example we need to forbid the value $SEQ_{c_1}$ for $c_1$, so that $f_2$ can be reached from $e_S$, and either the value $SEQ_{f_5}$ for $c_4$, so that $f_2$ can reach $e_E$ via $SEQ_{f_6}$, or the values $SEQ_{f_6}$ for $c_4$ and $SEQ_{e_2}$ for $c_5$, so that $f_2$ can reach $e_E$ via $SEQ_{e_2}$. Formally, the constraint will be: $¬p_{f_2,OFF}^c \Rightarrow ¬p_{f_5}^{c_4} \land (¬p_{f_6}^{c_4} \lor (¬p_{f_6}^{c_5} \land ¬p_{e_2}^{c_5}))$. This formula
also tells us when \( f_2 \) must be set to \( \text{OFF} \), e.g. if \( c_1 \) is configured to \( \text{SEQ} \).

Before presenting the formal definition of process constraint, we define the notation for acyclic paths in C-EPCs (as we did for Workflow nets), and two operators that are needed in the computation of an acyclic path.

**Notation 3.28 (C-EPC Acyclic Paths, Path Operators)** Let \( \Gamma = (E, F, C, l, A, F^C, C^C) \) be a syntactically correct C-EPC. We define the following notation:

- \( \phi = (n_1, n_2, ..., n_k) \) is an acyclic path of \( \Gamma \) such that \((n_i, n_{i+1}) \in A \) for \( 1 \leq i \leq k-1 \) and \( i \neq j \Rightarrow n_i \neq n_j \),

- \( \alpha(\phi) = \{n_1, n_2, ..., n_k\} \) is the alphabet of \( \phi \),

- \( \alpha^C(\phi) = \alpha(\phi) \cap C^C \) is the set of those elements of a path \( \phi \) that are directly preceded by a configurable connector,

- \( \Phi^C \) is the set of all acyclic paths of \( \Gamma \),

- for all \( n \in E \cup F \cup C \), \( AC_I(n) = \{\phi \in \Phi^C \mid \phi = (e_S, ..., n)\} \) is the set of all acyclic paths from \( e_S \) to \( n \),

- for all \( n \in E \cup F \cup C \), \( AC_O(n) = \{\phi \in \Phi^C \mid \phi = (n, ..., e_E)\} \) is the set of all acyclic paths from \( n \) to \( e_E \).

- Let \( n \in C^C \), then \( \xi(n) = \bigwedge_{p, x \in \bullet \setminus \{n\}} \neg p^x_c \), where \( c = \leftrightarrow (\bullet n) \). This operator accepts a node \( n \) in the postset of a configurable split and returns the negation of all the boolean variables encoding the sequence variants of that split, except the variable encoding \( \text{SEQ}_n \).

Now we are ready to define the process constraint \( PC^C \) for a C-EPC \( \Gamma \).

**Definition 3.29 (C-EPC Process Constraint)** Let \( \Gamma = (E, F, C, l, A, F^C, C^C) \) be a syntactically correct C-EPC with \( C^C \cap J = \emptyset \). The process constraint \( PC^C \) is a propositional logic formula over the boolean process variables of \( \Gamma \) given by the conjunction of the following expressions:

- \( \forall \phi \in AC_O(e_S) (\bigwedge_{n \in \alpha^C(\phi)} \xi(n)) \);

- \( \bigwedge_{f \in F^C} [\neg p^P_{OFF} \Rightarrow (\bigvee_{\phi \in AC_I(f)} (\bigwedge_{n \in \alpha^C(\phi)} \xi(n))) \wedge (\bigvee_{\phi \in AC_O(f)} (\bigwedge_{n \in \alpha^C(\phi)} \xi(n)))] \).

It is easy to see that any configured net derived through \( \beta^C \) from a configuration that satisfies \( PC^C \) is still a syntactically correct net. In fact \( PC^C \) only enforces stricter requirements on the configuration. Specifically, the first expression of Definition 3.29 ensures the existence of at least a directed path from \( e_S \) to \( e_E \), thus avoiding the empty net. The second expression ensures each function that is not switched \( OFF \) is on at least one such a path.

---

\( \xi_p(\{x\}) = x \), i.e. it converts a singleton set to the element in that set.
3.6.3 Tool Support

Based on Definition 3.29, we implemented an algorithm in the Synergia toolset to automatically infer $PC_\Gamma$ from a sound C-EPC model. The toolset also embeds an SBDD solver [Ray]. This solver builds an SBDD in memory from $PC_\Gamma$ and returns a polynomial representation of this canonical form in conjunctive normal form. If the negation of this formula is a tautology, the always-false formula is returned as output, meaning $PC_\Gamma$ is not satisfiable. At each configuration step the tool evaluates the satisfiability of the conjunction of the internal representation of $PC_\Gamma$ and the valuation provided by the user. If the result is not satisfiable it means the user’s configuration is not valid. In this case logical inference is used to determine the configuration of further variation points in order to keep the model correct, thus providing interactive support for staged configuration of C-EPC models.

Moreover, the toolset implements the $\beta_\Gamma$ algorithm as per Definition 3.20, which is an adaptation of the original algorithm [RA07] to handle partial configurations. When a configuration step satisfies $PC_\Gamma$, $\beta_\Gamma$ can be applied to commit the configuration onto the model and safely remove all disconnected nodes. The values of the variables in $PC_\Gamma$ are determined by imposing the exclusive disjunction of all the variants for each configurable node, such that when the user selects a variant for a given variation point, the variables that are not associated with that variant are set to false. The Synergia toolset is exposed in greater detail in Chapters 4 and 5, where we elaborate on the concept of staged configuration to incorporate domain constraints.

We conclude this section by showing an example of interactive configuration for the C-EPC in Figure 3.18. Assume we start the configuration by setting $f_2$ to OPT. This step satisfies $PC_\Gamma$ only if the variant $SEQ_{c_3}$ of $c_1$ is not taken. Therefore the respective boolean variable is disabled. Since now the configuration step complies with $PC_\Gamma$, $\beta_\Gamma$ is applied: function $skip_{f_2}$ and the two XOR connectors $split_{f_2}$ and $join_{f_2}$ are added (Figure 3.19.a).

Now assume we configure $c_1$ to $SEQ_{c_2}$. This step directly satisfies $PC_\Gamma$, thus $\beta_\Gamma$ is applied: $c_3$, $f_3$, $f_4$ and their arcs are removed from the model as well as $c_1$, $c_6$ and

![Figure 3.19: The staged configuration of the C-EPC in Figure 3.18.](image_url)
c7 since now they only have one incoming and one outgoing arc (Figure 3.19.b). The boolean variables associated with the variants of the remaining variation points are still undefined. Then we configure c5 to SEQe2: the variant SEQf6 of c4 becomes unavailable to ensure the end event can still be reached through SEQf6. βT is applied to remove c3, f7, c5 and c8. At this point we complete the model configuration by switching f5 OFF and setting c4 to a regular XOR. Figure 3.19.c shows the resulting EPC after applying βT one more time. The model is sound.

3.7 Summary and Discussion

This chapter exposed a conceptual foundation for business process model configuration based on the application of the hiding and blocking operators to Workflow nets. The objective was to formally define the concept of process model configuration and to use this formalization to analyze properties of configurable process models, particularly the correctness of the individualized models. The conceptualization led to the development of a technique for staged correctness-preserving configuration of process models. Assuming the initial model is correct, the technique guarantees that the individualized models are also correct at each stage of the configuration procedure. This is achieved by capturing the syntactic correctness constraints as a propositional logic formula. If a configuration step violates the constraints, a formula is derived to suggest ways of making the configuration step correctness-preserving. A cornerstone of this technique is a proof that, for free-choice process models, the enforcement of these syntactic constraints also ensures the preservation of semantic correctness.

Having established this formal foundation using Workflow nets, we illustrated the relation between the notion of hiding and blocking and the concept of configuration in a practical process modeling notation, namely EPCs. This was achieved by extending previous results on the mapping between Petri nets and EPCs. We also extended the technique for staged correctness-preserving configuration to deal with the specificities of EPCs. In particular, it was observed that configuring join connectors may cause semantic issues and that forbidding the configuration of these nodes does not reduce the expressiveness of a C-EPC. It was argued, but not formally proved, that if the initial C-EPC is sound and joins are not configurable, the syntactic correctness of an individualized net is sufficient to also ensure soundness. Then we showed that syntactic correctness can be interactively preserved by satisfying a set of boolean constraints in combination with the C-EPC individualization algorithm. These process constraints can be automatically inferred from a sound C-EPC without requiring additional effort from the user. Both the algorithm for inferring constraints and the individualization algorithm were implemented in the Synergia toolset. This toolset can be used by analysts and process modelers to efficiently configure C-EPC processes in a correctness-preserving manner.

In [GAJL08, GL08] we applied the conceptual foundation to the YAWL workflow language [AH05]. We extended the YAWL meta-model with a notion of variation point leading to Configurable YAWL (C-YAWL). In C-YAWL the configuration of a variation point is achieved by directly applying the hiding and blocking operators, since a YAWL model is an extension of a Workflow net. Moreover, we formulated an individualization algorithm to transform configured C-YAWL models into syntactically correct YAWL models, and adapted the technique for staged correctness-preserving configuration to
C-YAWL. Both the algorithm and the technique were implemented in the Synergia toolset to provide tool support for the configuration of C-YAWL models.

In this chapter we presented a technique to derive propositional logic constraints from process models. Similar techniques have been used for analyzing Petri nets [AIN04] and process graphs [SOS05]. However, the constraints we derived are specifically aimed to check that a configuration step preserves the structural properties of workflow nets. The concept of staged configuration is inspired by [CHE04], where the authors defined this concept in the context of Feature Diagrams. The concept of partial process configuration first appeared in [RA07], where the authors envisaged a multi-level configuration of C-EPC processes. Novel is the application of these concepts in an interactive, correctness-preserving manner, and their realization in a toolset for process model configuration.

In the next chapter we expose a questionnaire-based approach to configure process models which provides abstraction from the specific process modeling notation being used.
Chapter 4

Questionnaire-Based Approach for Variability Modeling

The second major shortcoming affecting existing approaches for configurable (reference) process modeling is the lack of support for the selection of configuration alternatives. We advocate that the selection of configuration alternatives should be driven by the domain of application, and not by the structure of process models. Indeed, while it is legitimate to assume that designers who produce process models are familiar with the notation in question, it is less realistic to assume this from stakeholders who provide input to configure these models (e.g., a logistics expert). Also, the increase in the model’s complexity brought by the configuration layer, makes the configuration of process models with a large number of variations points close to unmanageable, if a proper level of abstraction is not provided.

In light of the above, this chapter exposes a questionnaire-based approach to capture domain variability for the purpose of configuring process models. The approach proposes to capture this variability as a collection of interdependent choices. These choices, organized in a questionnaire model, can be linked to variation points in a process model to drive the configuration of the process model.

More generally, this approach is motivated by the need to proactively guide a user through the configuration of a generic system or model (i.e., not necessarily a process model) by focusing on the choices that need to be made. These choices should be presented at a suitable moment, and choices that may lead to invalid configurations should be avoided a priori. With respect to this objective, the approach supports the definition of order dependencies for posing questions, as well as domain constraints expressed in propositional logic to restrict the combinations of possible choices. Moreover, it includes a technique to generate interactive questionnaires from questionnaire models. These questionnaires guide the configuration process by posing only relevant questions, consistent with the order dependencies, and by preventing the user from entering incorrect answers that would violate the domain constraints. It is shown that simple well-formedness criteria are sufficient to ensure that no circular dependencies occur that may lead to deadlocks during configuration. Satisfiability solving techniques are used to guarantee the consistency of domain constraints and to incrementally prune the space of allowed answers at configuration time.

The remainder of this chapter is organized as follows. Section 4.1 outlines the approach by means of a working example. Next, Section 4.2 presents the formal definition
of questionnaire models while Section 4.3 describes the generation of interactive questionnaires from questionnaire models. Section 4.4 presents the two tools that have been implemented in Synergia to assist users in the creation and answering of questionnaires, and provides scalability results. Finally, Section 4.5 summarizes the content and compares the proposal with related ones. The application of the approach to the configuration of process models is discussed in the next chapter.

The work presented in this chapter has been published in [LADH09].

4.1 Overview of the Approach

We propose to explicitly capture the variability of an application domain by means of a questionnaire model. A questionnaire model encodes choices as a set of boolean variables, namely domain facts ("facts" for short), representing the possible answers to a set of questions. Questions and facts are formulated in natural language, thus can be answered solely requiring domain expertise. Order dependencies are used to determine a partial order in which questions should be asked to users, while domain constraints are used to capture the interplay among choices. As the questionnaire is answered, values are assigned to facts. These values can be propagated to the derivation of an individualized system or model by associating facts with variation points of the system or model to be configured. Figure 4.1 provides an overview of the approach.

![Diagram](image)

Figure 4.1: A questionnaire model is composed of questions and domain facts.

Making a choice corresponds to setting a fact within a question. Facts are simply statements such as “Shipping via DHL” or features such as “Return-merchandise Claim”. Initially, each fact is unset, while at configuration time its value can be set to true or false. For example, setting “Shipping via DHL” to false would mean that we are not interested in using DHL for shipping, whilst “Return-merchandise Claim” = true would mean that we want to support this particular type of claim. Each fact has a default value (true or false) which is provided as a suggestion, e.g. it may correspond to the most common choice in that domain. Moreover, a fact can be marked as ‘mandatory’
if it needs to be explicitly set by the user (e.g., it can be used to refer to an important aspect of the domain that cannot be overlooked).

Facts are grouped into questions according to their content, so that all the facts of the same group can be set at once. For example, the facts “Return-merchandise Claim” and “Loss or Damage Claim” can be grouped under the question “Which Claims have to be handled?”. Questions are thus a structuring mechanism to aid users assign values to facts, and are organized in a partial order such that users are not posed all questions at the same time.

A fact can appear in multiple questions. For example, in a screen business project, the fact “DVD” (true if the project releases on DVD, false otherwise) can be seen as a finish format as well as a distribution channel. It can thus be included in the questions “What finish formats have to be supported?” and “What are the distribution channels?”, with the purpose of facilitating the configuration process. In fact, since questions are organized in a partial order, users can set the same fact by following different answering paths (i.e. by answering different questions), according to their preferences. The value of a fact can only be set the first time, and is preserved in all the subsequent questions that contain it. However, it is possible to change a decision by rolling back an answered question.

A valuation of facts is any combination of facts’ values where all the facts have been set, either explicitly by answering questions or by using default values.

4.1.1 Working Example

To illustrate these concepts, we consider the decisions that have to be taken to configure an order fulfillment process model to the needs of an organization. This process is inspired by the Voluntary Interindustry Commerce Solutions (VICS) reference model [VICS]. VICS is an industry standard endorsed and used by various large companies that interact with suppliers and logistics providers by means of Electronic Data Interchange transactions. Specifically, we will not focus on the way the variation points are represented in the process model, but rather on the choices that a logistics expert has to face with to configure the model. A representation of the order fulfilment configurable process model in the C-YAWL notation is reported in [GL08].

The order fulfillment process may involve up to three roles: Supplier, Buyer and Carrier, and may support one or more business functions among Product Merchandising, Ordering, Logistics and Payment. Logistics may comprise one or more sub-phases among Freight Tender, Carrier Appointment, Freight in Transit and Freight Delivered. These phases span the whole logistics sub-process, from making an offer to a Carrier (Freight Tender), through agreeing on the freight pick-up and delivery details (Carrier Appointment) and on the messages to be exchanged during the shipment (Freight in Transit), to the types of claims to be supported after the delivery (Freight Delivered). The planned usage of a Carrier’s supplied trailer can also be decided upon based on the size of the freight being shipped. It can be “Truckload” (TL) for full usage, “Less-than Truckload” (LTL) for partial usage, or “Small Package” (SP) when just single packages are to be shipped. This choice has a strong influence on subsequent decisions. For TL- or LTL-shipments, the roles responsible for fixing the Pickup and the Delivery appointments can be decided, provided Carrier Appointment is included in
Logistics. For the pickup, this role can be played by either the Supplier or the Carrier; for the delivery, by either the Buyer or the Carrier. The appointment negotiation is not allowed in case of SP shipments, as the dates of pickup and delivery are imposed by the Carrier.

The Carrier’s usage also affects the type of notifications to be sent during the transit if Freight in Transit is included in Logistics. For TL or LTL, a Supplier’s or Buyer’s inquiry to the Carrier is followed by a shipment-status message for each parcel of the freight, whilst for SP the inquiry is followed only by one package-status message. Also, only in case of TL or LTL, and if Payment is selected, the Carrier can support a module for charging incidental costs that may be incurred during the transit.

Finally, in Freight Delivered, Claims support can be configured, in order to handle a Merchandise Return and/or cases of Freight Lost or Damaged. If the latter type of claim is selected, the Claim Manager is to be chosen between the Supplier and the Buyer.

Let us see how we can capture the choices behind the configuration of this process in terms of questions and facts. Figure 4.2 shows the questionnaire model for this example, where questions and facts are assigned a unique identifier and a description. For instance, facts \( f_1 \) to \( f_4 \) refer to the four business functions the process can implement, which have been grouped under question \( q_1 \). Question \( q_2 \) groups the facts relating to the expected Carrier’s usage. Since this choice is rather important as it affects the overall process, these facts are mandatory (labeled with an \( \text{M} \) in the picture). This means they have to be explicitly set to true or false when answering \( q_2 \). Other questions allow one to decide on the roles responsible for Pickup and Delivery (\( q_6, q_7 \)), the Claims to be handled (\( q_4 \)) and the Manager for Loss or Damage Claims (\( q_5 \)). Default values have been assigned to the facts of this model (a \( \text{T} \) indicates a fact whose default=\text{true}, while no symbol means that default=\text{false}). Selecting the default values leads to a process that implements all the business functions (\( f_1, f_2, f_3, f_4 = \text{true} \)) and all the Logistics’ sub-phases (\( f_8, f_9, f_{10}, f_{11} = \text{true} \)), and that supports TL shipments (\( f_5 = \text{true}, f_6, f_7 = \text{false} \)). In this type of shipment, the Supplier is usually responsible for organizing and scheduling the Pickup (so \( f_{16} = \text{true} \) and \( f_{17} = \text{false} \)) while the Buyer is responsible for organizing and scheduling the Delivery (\( f_{18} = \text{true}, f_{19} = \text{false} \)). The process handles only Loss or Damage Claims (thus \( f_{12} = \text{false} \) and \( f_{13} = \text{true} \)), that are managed by the Supplier which acts as intermediary between the Buyer and the Carrier (\( f_{14} = \text{true}, f_{15} = \text{false} \)).

### 4.1.2 Order Dependencies and Domain Constraints

Order dependencies (“dependencies” for short) can be introduced to enforce a partial ordering on facts and on questions. Let us first consider the ordering of facts. For example, we can use dependencies to impose that the role responsible for Pickup (either \( f_{16} \) or \( f_{17} \)) is to be chosen only after deciding on the Carrier Appointment (\( f_9 \)), because the latter includes the pickup details. We express such dependencies by associating a set of alternative preconditions with a fact \( x \), where a precondition is a group of facts that all need to be set before \( x \). Only one precondition needs to be satisfied for a dependency to be fulfilled. Therefore, fact \( x \) can be set only if at least all the facts in one of its preconditions have already been set. We say a fact partially depends on another fact if the latter belongs to at least one of its preconditions. On the other hand,
a fact fully depends on another one if the latter belongs to all its preconditions. A full dependency subsumes a partial dependency.

A partial dependency is represented in Figure 4.2 by a dashed arrow connecting a fact to its dependent fact, while a full dependency is depicted by a solid arrow. Accordingly, $f_{16}$ and $f_{17}$ fully depend on $f_9$, i.e. they can be set only after $f_9$, as they have one precondition containing only $f_9$. However, if a fact $x$ partially depends on another fact $y$, it is still possible that $x$ is set before $y$ if a precondition not including $y$ is satisfied.

 Dependencies over facts affect the order in which questions are posed to users because questions “inherit” the dependencies defined on their facts. In our example, since $f_{16}$ in $q_6$ depends on $f_9$ in $q_3$, $q_6$ automatically depends on $q_3$, although this dependency is not explicitly shown. Similarly, $q_7$ depends on $q_3$ and $q_5$ on $q_4$.

Sometimes though, it may be more natural to express dependencies directly at the level of questions. This is allowed so long as the dependencies defined at the level of questions do not contradict those defined at the level of facts. In Figure 4.2, $q_4$ has a (direct) full dependency on $q_3$ and its facts have no dependencies on other facts, whilst $q_2$ has a (direct) partial dependency on $q_1$ and $q_3$, so it can be answered after at least one of $q_1$ and $q_3$ has been answered. Figure 4.3 shows the final structure that defines the partial order in which the questions of Figure 4.2 will be be posed to users. From the diagram we can see that $q_5$, $q_6$ and $q_7$ have inherited their facts’ dependencies. Circular dependencies among questions and facts are prevented by means of simple well-formedness rules (further details will be given in Section 4.2).

Dependencies provide a means for ordering questions but do not affect facts’ values. For example, with a dependency we cannot capture the restriction on the Carrier’s
Usage, which implies that only one type of shipment can be supported at a time. This corresponds to saying that exactly one fact among $f_5$, $f_6$ and $f_7$ can be true in $q_2$. Moreover, answering a question may restrict the allowed answers to subsequent questions, and not all combinations of answers may lead to valid valuations of facts. Indeed, if SP ($f_7$) is asserted in $q_2$, no appointment negotiation is allowed for Pickup and Delivery, i.e. $f_{16}$ and $f_{17}$ have to be negated in $q_6$ and $f_{18}$ and $f_{19}$ have to be negated in $q_7$.

We model these domain constraints as propositional logic expressions over facts. From an analysis of the order fulfilment process, we have identified the following constraints among the facts of our questionnaire model:

\[ DC_1: f_1 \lor f_2 \lor f_3 \lor f_4 \]
\[ DC_3: \text{xor}(f_5, f_6, f_7) \iff (f_1 \lor f_6 \lor f_{10}) \]
\[ DC_5: \neg(f_5 \lor f_6 \lor f_7) \iff \neg(f_4 \lor f_9 \lor f_{10}) \]
\[ DC_7: (f_9 \land \neg f_7) \iff (\text{xor}(f_{16}, f_{17}) \land \text{xor}(f_{18}, f_{19})) \]
\[ DC_9: \neg(f_9 \lor \neg f_7) \iff \neg(f_{16} \lor f_{17} \lor f_{18} \lor f_{19}) \]

$DC_1$ ensures that at least one business function is chosen in $q_1$. $DC_3$ and $DC_5$ ensure exactly one type of shipment is to be selected as Carrier’s usage in $q_2$, if and only if at least one phase among Payment, Carrier Appointment and Freight in Transit is selected in $q_3$, otherwise no shipment type can be chosen. Indeed, as mentioned before, TL, LTL and SP affect the above Logistics phases, so it makes no sense to decide on the shipment type unless a phase that is affected by the Carrier’s Usage is selected. Likewise, as per $DC_7$ and $DC_9$, exactly one role between Supplier and Carrier is to be responsible for Pickup ($q_6$), and exactly one role between Buyer and Carrier is to be responsible for Delivery ($q_7$), if and only if Carrier Appointment is selected and one of TL and LTL is true. This is because the Pickup and Delivery appointments are handled during the Carrier Appointment phase of the order fulfilment process and only in case of TL- or LTL-shipments.

A domain configuration is therefore any valuation of facts that complies with the domain constraints. Constraints can also be defined over questions (e.g., an OR question is a question whose facts are all in an OR relation). However in the end they need to be traced back to the level of facts. From the above list of constraints it is easy to derive that $q_1$ is an OR question, while $q_3$ and $q_4$ are OR questions and $q_2$, $q_5$, $q_6$ and $q_7$ are XOR questions. $q_8$ is a question whose facts are all in an XOR relation.

\[ \text{xor} \] indicates the exclusive disjunction, i.e. $\text{xor}(f_1, f_2) \iff (f_1 \lor f_2) \land (f_1 \neq f_2)$.
\( q_7 \) are XOR questions, provided some conditions are met. For example, \( q_5 \) is an XOR question as exactly one Manager is to be chosen for Loss or Damage Claim, provided Loss or Damage Claim has been set to true in \( q_4 \).

Dependencies and constraints complement each other. An example is shown by \( DC_4: (f_{12} \lor f_{13}) \Rightarrow f_{11} \) and the full dependency that \( q_4 \) has on \( q_3 \). Here the behavior we want to capture is that Claims can be handled only if Freight Delivered ‘has been’ selected, viz., \( f_{12} \) and \( f_{13} \) can be set to true only if \( f_{11} \) has been set to true before. Similarly, due to \( DC_6: f_{13} \iff \text{xor}(f_{14}, f_{15}) \) and \( q_5 \) which indirectly depends on \( q_4 \), exactly one Manager for Loss of Damage Claim is to be selected in \( q_5 \), but only after Loss or Damage Claim (\( f_{13} \)) ‘has been’ set to true in \( q_4 \).

Because of the domain constraints, answering certain questions can lead to other questions becoming constrained or irrelevant. This suggests that questions should be ordered such that the most discriminating questions are asked first. This way, the configuration space is pruned and the total number of questions that need to be answered is minimized. For example, \( q_1 \) and \( q_3 \) are highly discriminating questions so it makes sense that they are posed before the other ones. However, dependencies can be defined based on other considerations. For example, in some contexts, dependencies may be defined based on the role of the user(s) that will configure the system. For instance, a logistics expert can be first posed questions related to Logistics, while a merchandise expert can be first asked questions related to Product Merchandising. Alternatively, each user can be assigned a subset of questions according to their expertise (distributed configuration).

The approach relies on boolean encodings of the space of possible answers. This encoding enables the use of efficient techniques to incrementally prune the space of answers, so that questions that become constrained or irrelevant due to answers given to previous questions, can be simplified or skipped. It is also possible to define questions with non-enumerated types as their space of possible answers. For example, one can define a configuration parameter “number of carriers?” of type integer and use this as the answer to a question. However, this parameter cannot be used in the domain constraints. Alternatively, if a discretization can be defined (e.g. ”one carrier”, ”between 2 and 5 carriers”, and ”more than 5 carriers”), each of these discretized values can then be mapped to a fact that can be used to express domain constraints.

The following section formalizes the notions discussed above. The formalization allows us to convey the ideas in an unambiguous way and will be used as a basis for the implementation presented in Section 4.4.

### 4.2 Questionnaire Models

We use the concept of questionnaire model (QM) to directly capture choices in terms of facts, questions and their dependencies. Given a questionnaire model, a domain configuration is the result of assigning values to each fact by answering questions. We can view a configuration as a valuation of facts that complies with the domain constraints. Below we formally capture this intuition.

**Definition 4.1 (Questionnaire Model)** A questionnaire model is a ten-tuple \( QM = (F, F_D, F_M, Q, \text{map}_{QF}, \text{pre}_F, \text{pre}_Q, CS) \) where:
• $F$ is a finite, non-empty set of facts,
• $F_D \subseteq F$ is the default valuation, i.e. the set of facts whose default is true,
• $F_M \subseteq F$ is the set of mandatory facts,
• $Q$ is a finite (non-empty) set of questions,
• $\text{map}_{QF} \in Q \rightarrow \mathcal{P}(F) \setminus \{\emptyset\}$\textsuperscript{2} is a function mapping a question onto a set of facts, such that $\bigcup_{q \in Q} \text{map}_{QF}(q) = F$,
• $\text{pre}_F \in F \rightarrow \mathcal{P}(\mathcal{P}(F)) \setminus \{\emptyset\}$ is a function mapping a fact onto a set of sets of facts, where for any $f \in F$, $\text{pre}_F(f)$ is the set of preconditions of $f$, i.e. each $F' \in \text{pre}_F(f)$ represents a precondition. A precondition is satisfied if all its facts are set. Only one precondition needs to be satisfied to set $f$. There is always at least one precondition ($\text{pre}_F(f) \neq \emptyset$), hence a situation with no order dependencies is represented by the empty precondition ($\text{pre}_F(f) = \{\emptyset\}$). Moreover, function $\text{pre}_F$ needs to satisfy the following well-formedness rules:

1. $\forall f \in F \ \forall r, p \in \text{pre}_F(f) \ (r \subseteq p \Rightarrow r = p)$, i.e. no redundancies,
2. $\forall f \in F \ \forall r, p \in \text{pre}_F(f) \ (r \subseteq p \Rightarrow r = p)$, i.e. no undesired circular dependencies,

• $\text{pre}_Q \in Q \rightarrow \mathcal{P}(\mathcal{P}(Q)) \setminus \{\emptyset\}$ is a function mapping a question onto a set of sets of questions. For any $q \in Q$, $\text{pre}_Q(q)$ is the set of preconditions of $q$. Each precondition corresponds to a set of questions that need to be answered before $q$ is answered, but it is sufficient to satisfy at least one precondition. There is always at least one precondition, hence a situation with no order dependencies is represented by the empty precondition. Moreover, function $\text{pre}_Q$ needs to satisfy the following well-formedness rules:

1. $\forall q \in Q \ \forall r, p \in \text{pre}_Q(q) \ (r \subseteq p \Rightarrow r = p)$, i.e. no redundancies,
2. $\forall q \in Q \ \forall q' \in \text{pre}_Q(q) \ Q' \cap G = \emptyset$, i.e. no undesired circular dependencies,
3. $\forall q \in Q \ \forall q' \in \text{pre}_Q(q) \ \forall f \in \text{map}_{QF}(q) \ \forall f' \in \text{pre}_F(f) \ F' \subseteq \bigcup_{q' \in Q'} \text{map}_{QF}(q')$, i.e. facts dependencies must be preserved at the level of questions,

• $CS \subseteq \mathcal{P}(F)$ is the set of the allowed valuations of the facts in $F$, such that $F_D \in CS$, i.e. the default valuation is always allowed.

Elements of $CS$ are those valuations of facts that satisfy all the domain constraints, where only the facts asserted are present in each element. Hence, if a fact is not contained in a clause of $CS$, it follows that the fact is negated in that valuation. For example, if $F = \{f_1, f_2, f_3, f_4\}$ and $\{f_1, f_2, f_3\} \in CS$ is a valuation of facts, in the latter all the facts but $f_3$ are set to true. Since the default valuation must always be allowed $CS$ is non-empty. Moreover, $CS = \mathcal{P}(F)$ indicates that no constraints are defined, $CS = \{F\}$ indicates that all facts must be asserted (upper-bound case), and $CS = \{\emptyset\}$ indicates that all facts must be negated (lower-bound case).

\textsuperscript{2}$\mathcal{P}$ indicates the power set, i.e. each question is mapped onto a non-empty set of facts.
We say a fact $f$ is meaningful if it truly represents a variation, i.e. if it is allowed by the constraints to assume both values true and false. Formally, if there exist $F'_1, F'_2 \in CS$ such that $f \in F'_1$ and $f \notin F'_2$. Since a questionnaire model does not incorporate elements of commonalities, i.e. those aspects of a domain that do not vary and hence upon which choices cannot be taken, a non-meaningful fact should not be included in $QM$.

The set of preconditions for facts and questions are used to specify the order dependencies as follows.

**Definition 4.2 (Order Dependencies)** Let $QM = (F, F_D, F_M, Q, map_QF, \text{pre}_F, \text{pre}_Q, CS)$ be a questionnaire model and $f, f'$ and $q, q'$ pairs of facts, respectively questions:

- $f$ partially depends on $f'$ iff $\exists F' \in \text{pre}_F(f) \ f' \in F'$,
- $f$ fully depends on $f'$ iff $\forall F' \in \text{pre}_F(f) \ f' \in F'$,
- $q$ partially depends on $q'$ iff $\exists Q' \in \text{pre}_Q(q) \ q' \in Q'$,
- $q$ fully depends on $q'$ iff $\forall Q' \in \text{pre}_Q(q) \ q' \in Q'$.

The set of preconditions represents for a fact and for a question the disjunction of preconditions being conjunctions of the dependencies. In other words, a fact can be set only if at least all the facts in one of its preconditions have already been set. Likewise, a question can be answered if all the questions in one of its preconditions have already been answered. Thus facts or questions in the same precondition are in an AND relation, while preconditions are in an OR relation.

**Example 4.3** Let $\text{pre}_F(f_1) = \{\{f_2, f_3\}, \{f_2, f_4\}\}$ be the set of preconditions of fact $f_1$. Then either $f_2$ and $f_3$ or $f_2$ and $f_4$ have to be set before $f_1$ can be set. We can observe that $f_2$ must be set in any case before $f_1$, since it appears in all the clauses of $\text{pre}_F(f_1)$. This is a full dependency. On the other hand, $f_1$ partially depends on $f_3$ as well as on $f_4$, since these two facts do not belong to each clause of $\text{pre}_F(f_1)$.

As per Definition 4.1, for any fact $f$ and question $q$, both $\text{pre}_F(f)$ and $\text{pre}_Q(q)$ are not the empty set. Thus, if we want to model a situation where no dependencies are defined for a fact $f$ or question $q$, $\text{pre}_F(f)$ or $\text{pre}_Q(q)$ should contain only the empty set.

The first well-formedness rule on $\text{pre}_F$ and $\text{pre}_Q$ (see Definition 4.1) is used to avoid redundancies among preconditions. Accordingly, if a precondition contains the empty set it cannot contain other sets, since all the sets would include the empty one.

**Example 4.4** A situation where $\text{pre}_F(f_1) = \{\{f_2\}, \{f_2, f_3\}\}$ is not allowed because the first clause is a subset of the second one. Since all the preconditions are in an OR relation, it does not make sense for $f_1$ to depend on $f_2$ OR on $(f_2$ AND $f_3)$, as the latter set of dependencies implies the former. In such cases only one clause should be selected.

The second well-formedness rule on $\text{pre}_F$ and $\text{pre}_Q$ avoids ‘undesirable circular dependencies’. These occur whenever for each fact or question of a given set, all their preconditions contain at least one element of the set itself.
Example 4.5 A case where $\text{pre}_F(f_1) = \{\{f_2\}\}$, $\text{pre}_F(f_2) = \{\{f_3\}\}$ and $\text{pre}_F(f_3) = \{\{f_1\}\}$ (Figure 4.4.a), or a case where $\text{pre}_F(f_1) = \{\{f_2\}\}$, $\text{pre}_F(f_2) = \{\{f_3\}\}$ and $\text{pre}_F(f_3) = \{\{f_1\}, \{f_2\}\}$ (Figure 4.4.b) are not allowed according to Definition 4.1. The reason is that there exists a $G = \{f_1, f_2, f_3\} \subseteq F$ such that for all $f \in G$, all the clauses in $\text{pre}_F(f)$ contain at least a fact in $G$. This violates the second well-formedness rule on $\text{pre}_F$ because of an undesirable cycle. Such undesirable cycles can be caused by both partial and full dependencies.

Not all circular dependencies are undesirable, though. For example, a loop created by a set $G$ of facts can be allowed if there exists an entry point to the loop, i.e. an element of the given set which satisfies all the preconditions one by one. This entry point is a fact with at least one precondition that contains only elements not in this set $G$. Such considerations hold for questions as well.

Example 4.6 A combination where $\text{pre}_F(f_1) = \{\{f_2\}, \{f_3\}, \{f_4\}\}$, $\text{pre}_F(f_2) = \{\{f_1\}, \{f_3\}\}$, $\text{pre}_F(f_3) = \{\{f_1\}, \{f_2\}\}$, $\text{pre}_F(f_4) = \emptyset$ (Figure 4.4.c) is allowed as $f_4$ does not have dependencies on the set $\{f_1, f_2, f_3\}$ and thus it first enables $f_1$ and then $f_2$ and $f_3$ in any order. In fact we cannot find a $G \subseteq F$ such that the second well-formedness rule on preconditions does not hold.

The only difference between the definitions of $\text{pre}_F$ and $\text{pre}_Q$ is the addition of a third well-formedness rule to the latter, so as to move dependencies over facts to the level of questions without violating them. Given a question $q$, the formula checks for the existence of preconditions $F'$ on the facts of $q$. If these exist, the formula forces each precondition $Q'$ of $q$ to contain a set of questions whose facts cover all the facts in all the preconditions $F'$. These dependencies that $q$ inherits from its facts can be extended by adding further dependencies directly at the granularity of questions, provided they comply with the first two conditions. This is possible since $\bigcup_{q' \in Q'} \text{map}_{QF}(q')$ is defined as a superset of all preconditions $F'$.

Example 4.7 Consider a situation where $\text{map}_{QF}(q_1) = \{f_1, f_2\}$, $\text{map}_{QF}(q_2) = \{f_3, f_4\}$, $\text{map}_{QF}(q_3) = \{f_3\}$, $\text{pre}_F(f_1) = \{\{f_3\}\}$ and $\text{pre}_F(f_4) = \{\{f_2\}\}$ (Figure 4.4.d). Here $f_3$ is a shared fact between $q_2$ and $q_3$. If we lift dependencies among facts to the level of questions, we observe that: i) $q_3$ does not inherit any dependencies as it is only mapped to $f_3$, and ii) $q_2$ fully depends on $q_1$ by means of $f_4$. Also, we can observe that there are four possible sets of preconditions for $q_1$, i.e. $\text{pre}_Q(q_1) = \{\{q_2\}, \{q_3\}\}$ or...
\{\{q_3\}\} or \{\{q_2, q_3\}\} or \{\{q_2\}\}. All these sets meet the third well-formedness rule as \(f_3\) – the only fact \(f_1\) depends on – is contained in at least one question \(q' \in Q'\) for each \(Q' \in \text{pre}_Q(q_1)\). However for the second well-formedness rule, only the first two alternatives are valid, as they do not create undesirable circular dependencies between \(q_1\) and \(q_2\).

### 4.3 Generation of Interactive Questionnaires

This section completes the formal description of the approach presented so far by defining the process of configuring a questionnaire model by answering questions. This way we provide executable semantics for \(QM\) defined in Definition 4.1. In a configuration process questions are dynamically posed to users according to the order dependencies, and answers can be given only if these comply with the constraints.

We first define some concepts to work with valuations of facts, such as set of valuations of facts, answer, state and state space. These concepts are needed to specify when a question can be posed to users. In particular, an answer is any valuation of facts where only a subset of facts (the ones that relate to a question) are set, while a state of \(QM\) is identified by a valuation of facts and a set of answered questions.

**Definition 4.8 (Set of Valuations of Facts, Answer, State, State space)** Let \(QM = (F,F_D,F_M,Q,map_QF,\text{pre}_F,\text{pre}_Q,CS)\) be a questionnaire model:

- \(V = F \to \{\text{true}, \text{false}, \text{unset}\}\) is the set of all valuations of facts, independently of set \(CS\);
- \(a \in V\) is an answer, i.e. a valuation of facts where all \(f \in F\) for which \(a(f) \neq \text{unset}\) are set;
- \(s = (vs, qs)\) is a state of \(QM\) if and only if \(vs \in V\) and \(qs \subseteq Q\), where \(qs\) is the set of questions answered and \(vs\) is the valuation of the facts thus far;
- \(S_{QM} = V \times \mathcal{P}(Q)\) is the state space of \(QM\).

Elements of \(V\) are valuations of facts, i.e. “answers” \((a)\) as well as “parts of state” \((vs)\). Hereafter \(S_{QM}\) is shortened to \(S\) whenever the configuration context is clear.

In order to perform operations on valuations of facts, we define the following notation.

**Notation 4.9 (Valuation of Facts)** Let \(QM = (F,F_D,F_M,Q,map_QF,\text{pre}_F,\text{pre}_Q,CS)\) be a questionnaire model and let \(s = (vs, qs) \in S\) be a state of \(QM\) and \(a \in V\) an answer:

- \(t(s) = t(vs) = \{f \in F \mid vs(f) = \text{true}\}\) is the set of facts that are true in state \(s\),
- \(f(s) = f(vs) = \{f \in F \mid vs(f) = \text{false}\}\) is the set of facts that are false in state \(s\),
- \(u(s) = u(vs) = \{f \in F \mid vs(f) = \text{unset}\} = F \setminus (t(s) \cup f(s))\) is the set of facts that are unset in state \(s\). \(t(vs), f(vs)\) and \(u(vs)\) can be applied to any valuation \(vs \in V\), thus to any answer \(a \in V\):
• $t(a) = \{ f \in F \mid a(f) = true \}$, is the set of facts set to true by answer $a$,
• $f(a) = \{ f \in F \mid a(f) = false \}$, is the set of facts set to false by answer $a$,
• $u(a) = F \setminus (t(a) \cup f(a))$, is the set of facts left unset by answer $a$,
• $\text{compl}(s) = \text{compl}(u_s) = \{ f \in F \mid u_s(f) = true \lor (f \in \mathcal{F}_D \land u_s(f) \neq false) \}$ is the set of facts set to true through answers, merged with those facts left unset which are true by default,

For each state a set of valid questions is presented to the user. For a question to be valid in a state ($valid(q,s)$), two conditions must hold: i) the question has not been answered yet, and ii) at least one of its preconditions is satisfied.

Users can answer one valid question at a time. An answer to a question in a certain state is valid ($valid(a,q,s)$) if and only if all the facts of that question are set and the outcome of the answer ($outcome(a,q,s)$) results in a valid state ($valid(s)$), i.e. a state whose valuation of facts complies with the constraints on facts. Also, since facts can appear in more than one question, those of them already set in previous questions (if they exist) must keep their values in the answer, i.e. it is possible to reconfirm answers.

**Definition 4.10 (Valid answer)** Let $QM = (F, \mathcal{F}_D, \mathcal{F}_M, Q, \text{map}_C, \text{pre}_F, \text{pre}_Q, CS)$ be a questionnaire model and let $s = (vs, qs) \in S$ be a state of $QM$, $q \in Q$ a question, and $a \in V$ an answer:

- $valid(q,s) = q \not\in qs \land \exists_{Q' \in \text{pre}_{Q}(q)} Q' \subseteq qs$, i.e. question $q$ may be asked if it has not been answered yet and at least a group of preceding questions has been answered,
- $outcome(a,q,s) = (vs \oplus a, qs \cup \{q\})$, i.e. the state resulting after answering $a$ to question $q$ in state $s$,
- $valid(s) = \exists_{F' \in CS} (t(s) \subseteq F' \land f(s) \cap F' = \emptyset)$, i.e. the valuation of facts of a state has to comply with the constraints on facts,
- $valid(a,q,s) = valid(q,s) \land t(a) \cup f(a) = \text{map}_C(q) \land \forall_{f \in \text{map}_C(q) \setminus u(a)} a(f) = vs(f) \land valid(outcome(a,q,s))$, i.e. a valid answer to a valid question has to set all the facts of the question without changing the value of the facts already set, and the given valuation must result in a valid state.

The valuation resulting from an answer has to be checked against set $CS$, so as to verify if it complies with the constraints defined on facts’ values. In this way we ensure it is always possible to complete the current valuation of facts by setting any remaining fact still unset.

By joining the possible states of a configuration process, we can now build a labeled transition system ($LTS$) on top of $QM$. This is later used to formally define the concept of configuration.
Definition 4.11 (Labeled Transition System of QM) Let \( QM = (F, F_D, F_M, Q, map_{QF}, pre_F, pre_Q, CS) \) be a questionnaire model and let \( S \) be the state space of \( QM \) and \( V \) the set of valuations of facts. The labeled transition system of \( QM \) is a five-tuple \( LTS = (S_v, L, T, s_{init}, S_F) \) where:

- \( S_v = \{ s \in S \mid valid(s) \} \) is the set of states of \( LTS \), corresponding to the valid states of \( QM \),
- \( L = \{(a, q) \in V \times Q \mid t(a) \cup f(a) = map_{QF}(q) \} \) is the set of transition labels of \( LTS \), where each element of \( L \) is a pair composed of an answer and a question of \( QM \),
- \( T = \{(s, (a, q), s') \in S_v \times L \times S_v \mid valid(a, q, s) \land s' = outcome(a, q, s) \} \) is the set of transitions of \( LTS \), where for each \( t = (s, (a, q), s') \in T \) \( source(t) = s \) and \( target(t) = s' \),
- \( s_{init} = \{ ((f, unset) \mid f \in F}, \emptyset) \} \in S_v \) is the initial state of \( LTS \), i.e. the state in which all the facts are unset and all the questions are unanswered,\(^3\)
- \( S_F = \{(vs, qs) \in S_v \mid (f \in F_M \Rightarrow vs(f) \neq unset) \land valid(s^*) \} \) is the set of final states of \( LTS \), where \( s^* = (vs^*, qs) \in S \) with \( t(vs^*) = compl(vs) \) and \( f(vs^*) = F \setminus t(vs^*) \).

A configuration process always starts from an initial state where no questions are answered and all the facts are unset, and terminates in a final state where all the questions have been answered, or all the mandatory facts have been set and the remaining unset facts can take their defaults. As shown in the definition of final state of the labeled transition system, this is possible only if the valuation of facts that results after applying the defaults complies with the constraints on facts’ values, i.e. if it does not violate the configuration process so far.

Example 4.12 Consider a questionnaire model where \( map_{QF}(q_1) = \{ f_1 \}, map_{QF}(q_2) = \{ f_2, f_3, f_4, f_5 \}, F_D = \{ f_2, f_3 \}, F_M = \{ f_1 \}, \) and the constraint \( f_1 \Rightarrow xor((f_2 \land f_3), (f_2 \land f_3)) \). It follows that \( CS = \{ \{ f_1, f_2, f_3 \}, \{ f_1, f_3, f_5 \}, \ldots \} \), where the remaining elements of \( CS \) are the elements of \( \mathcal{P}(\{ f_2, f_3, f_4, f_5 \}) \), thus including \( F_D \). If \( f_1 \) is set to true by answering \( q_1 \), although all the mandatory facts have been set, the default valuation cannot be applied for the remaining unset facts in \( q_2 \), since only either \( f_2 \) and \( f_4 \) or \( f_3 \) and \( f_5 \) can assume value true. Hence, we cannot find an \( F' \in CS \) such that \( \{ f_1, f_2, f_3 \} \subseteq F' \). On the other hand, if we set \( f_1 \) to false we reach a final state straightforward, where all the mandatory facts have been set and the remaining ones can take their default value.

A configuration trace of \( QM \) is a sequence of transitions of \( LTS \) linking the initial state to a final state.

Definition 4.13 (Configuration Trace of QM) Let \( QM = (F, F_D, F_M, Q, map_{QF}, pre_F, pre_Q, CS) \) be a questionnaire model, \( V \) the set of valuations of facts, \( S \) the state space of \( QM \) and let \( LTS_{QM} = (S_v, L, T, s_{init}, S_F) \) be its labeled transition system:

\(^3\)\( s_{init} \) is valid by definition, since \( t(s_{init}) = f(s_{init}) = \emptyset \).
σ = (t₁, ..., tₙ) ∈ T⁺ is a trace of LTS if \( \text{target}(tᵢ) = \text{source}(tᵢ₊₁) \) for each \( 1 ≤ i ≤ n - 1 \), where firstₜ(σ) = source(t₁) and lastₜ(σ) = target(tₙ),

- valid(σ) = (firstₛ(σ) = sᵢₐₗᵢᵦ \( ∧ \) lastₛ(σ) ∈ S_F), i.e. a trace is valid if it joins the initial state with a final state. Each valid trace is a configuration trace of QM.

A domain configuration, or simply, a configuration of QM, is the result of any configuration trace of QM, i.e. the valuation of facts reached with the last state of a configuration trace, completed with default values. Therefore a configuration always complies with the constraints.

**Definition 4.14 (Configuration of QM, Configuration Space of QM)** Let QM = \( (F, F_D, F_M, Q, \text{map}_Q, \text{pre}_Q, \text{pre}_Q, CS) \) be a questionnaire model, \( V \) the set of valuations of facts, \( S \) the state space of QM, LTS\(_{QM} \) = \( (S_v, L, T, sᵢₐₖᵢᵦ, S_F) \) its labeled transition system, and let \( σ \in T⁺ \) be a configuration trace of QM:

- \( \text{cf}_σ ∈ V \) is a configuration of QM resulting from \( σ \), if \( t(\text{cf}_σ) = \text{compl} (\text{last}_s(σ)) \) and \( f(\text{cf}_σ) = F \setminus t(\text{cf}_σ) \),

- \( \text{Cf}_{QM} = \{ \text{cf}_σ ∈ V \mid (σ ∈ T⁺) \land \text{valid}(σ) \} \) is the configuration space of QM, i.e. the set of all the possible configurations of QM.

We now show that a configuration process can always terminate in a final state, since: i) the state space is finite, and ii) for all the valid non-final states, there always exists at least one valid question whose answer leads to another valid state, taking the process closer to a final state. In particular, the following theorem proves that the definition of \( \text{pre}_Q \) and \( CS \) are sufficient to avoid any deadlock during the configuration process. This is because undesirable circular dependencies are excluded a priori in \( \text{pre}_Q \), and only those valuations of facts that comply with the constraints are represented in \( CS \).

The theorem is followed by a corollary that shows the application of the result.

Before presenting the theorem we introduce a shorthand notation.

**Notation 4.15 (Trace)** Let QM = \( (F, F_D, F_M, Q, \text{map}_Q, \text{pre}_Q, \text{pre}_Q, CS) \) be a questionnaire model, \( V \) the set of valuations of facts, \( S \) the state space of QM and let LTS\(_{QM} \) = \( (S_v, L, T, sᵢₐₖᵦ, S_F) \) be its labeled transition system. Given two valid states of LTS \( s \) and \( s' \), we write \( s \xrightarrow{τ} s' \) if \( σ ∈ T⁺ \) is a trace of LTS such that firstₜ(σ) = s and lastₜ(σ) = s'.

**Theorem 4.16** Let QM = \( (F, F_D, F_M, Q, \text{map}_Q, \text{pre}_Q, \text{pre}_Q, CS) \) be a questionnaire model, \( V \) the set of valuations of facts, \( S \) the state space of QM and let LTS\(_{QM} \) = \( (S_v, L, T, sᵢₐₖᵦ, S_F) \) be its labeled transition system. For any \( s ∈ S_v \), either \( s ∈ S_F \) or \( \exists q ∈ Q \, \exists a ∈ V \, \exists s' ∈ S_v \, s \xrightarrow{(s, (a, q), s')} s' \) \( [(s, (a, q), s') ∈ T] \).

**Proof** We prove the theorem in two steps: i) we show that for all valid non-final states there always exists at least one valid question; ii) we show that for all valid questions in a valid state there always exists at least one valid answer.
Valid question \([\forall s \in S_v \setminus S_F \exists q \in Q \ valid(q, s)]\) Let \(s = (vs, qs) \in s_v \setminus s_F\). Let \(G = Q \setminus qs\), then \(G \not= \emptyset\) as \(s \notin s_F\). According to the 2\(^{nd}\) well-formedness rule on \(pre_Q\), there is a \(q \in G\) and a \(Q' \in pre_Q(q)\) such that \(G \cap Q' = \emptyset\).

- \([q \notin qs]\). True by definition of \(G\) and \(pre_Q\).
- \([Q' \subseteq qs]\). \(G \cap Q' = \emptyset\), that is \((Q \setminus qs) \cap Q' = \emptyset\), thus \((Q \cap Q') \setminus qs = \emptyset\), \((Q' \subseteq Q)\)

\(Q' \setminus qs = \emptyset\), hence \(Q' \subseteq qs\).

Hence \(valid(q, s)\).

Valid answer \([\forall s \in S_v \setminus S_F \forall q \in Q, valid(q, s) \exists a \in V \ valid(a, q, s)]\) Let \(s = (vs, qs) \in s_v \setminus s_F\). Since \(s \in s_v\), we can find \(F' \in CS\) such that \(t(s) \subseteq F'\) and \(f(s) \cap F' = \emptyset\). Let \(q \in Q\) such that \(valid(q, s)\). We define \(t_s(q) = \{f \in map_QF(q) \mid vs(f) = true\}\), \(f_s(q) = \{f \in map_QF(q) \mid vs(f) = false\}\), \(t_a(q) = (F' \cap map_QF(q)) \setminus t_s(q)\) and \(f_a(q) = map_QF(q) \setminus (F' \cup t_a(q))\). We choose \(a = \{(f, true) \mid f \in t_s(q) \cup t_a(q)\} \cup \{(f, false) \mid f \in f_s(q) \cup f_a(q)\} \cup \{(f, unset) \mid f \in F \setminus map_QF(q)\}\), then \(a \in V\).

- \([valid(q, s)]\). True by assumption.
- \([t(a) \cup f(a) = map_QF(q)\]. \(t(a) \cup f(a) = t_s(q) \cup t_a(q) \cup f_s(q) \cup f_a(q)\).
  - \([\subseteq]\) Let \(f \in map_QF(q)\),
    1) if \(vs(f) = true\), then \(f \in t_s(q)\),
    2) if \(vs(f) = false\), then \(f \in f_s(q)\),
    3) if \(vs(f) = unset\),
      a) if \(f \in F'\), then \(f \in t_a(q)\) as \(f \notin t_s(q)\),
      b) if \(f \notin F'\), then \(f \in f_a(q)\) as \(f \notin t_s(q)\),
    hence \(f \in t_s(q) \cup t_a(q) \cup f_s(q) \cup f_a(q)\).
  - \([\supseteq]\) Follows from the definitions of \(t_s(q), t_a(q), f_s(q)\) and \(f_a(q)\).
- \([\forall f \in map_QF(q) \cup a(s) \ a(f) = vs(f)\]. Let \(f \in map_QF(q)\) and \(f \notin a(s)\), then \(f \in t_s(q)\) or \(f \in f_s(q)\), hence (definition of \(a\)) \(a(f) = true\) and \(f \in t_s(q)\) or \(a(f) = false\) and \(f \in f_s(q)\), hence (definitions of \(t_s(q)\) and \(f_s(q)\)) \(a(f) = true\) and \(vs(f) = true\) or \(a(f) = false\) and \(vs(f) = false\), hence \(a(f) = vs(f)\).
- \([valid(outcome(a, q, s))]. Let \(s' = outcome(a, q, s) = (vs \oplus a, qs \cup \{q\})\).
  - \([t(s') \subseteq F']\). \(t(s') = \{f \in F \mid a(f) = true \lor (vs(f) = true \land a(f) = unset)\}\) (definition of \(x \oplus y(f)\)). Let \(f \in t(s')\),
    1) if \(a(f) = true\), then \(f \in t_s(q) \cup t_a(q)\), hence \(f \in F'\) given that \(t_s(q) \subseteq F'\) and \(t_a(q) \subseteq F'\).
    2) if \(vs(f) = true\) and \(a(f) = unset\), then \(f \in t(s)\) and \(f \in F \setminus map_QF(q)\), hence \(f \in F'\) as \(t(s) \subseteq F'\).
  - \([f(s') \cap F' = \emptyset] f(s') = \{f \in F \mid a(f) = false \lor (vs(f) = false \land a(f) = unset)\}\) (definition of \(x \oplus y(f)\)). Let \(f \in f(s')\),
    1) if \(a(f) = false\), then \(f \in f_s(q) \cup f_a(q)\), hence \(f \notin F' = \emptyset\) given that \(f_s(q) \cap F' = \emptyset\) and \(f_a(q) \cap F' = \emptyset\).
Corollary 4.17 (Configuration processes always terminate) For any questionnaire model \( QM = (F, F_D, F_M, Q, map_QF, pre_F, pre_Q, CS) \) and its \( LTS_{QM} = (S_v, L, T, s_{init}, S_F) \), and for any state \( s \in S_v \setminus S_F \) for which there exists a trace \( \sigma \in T^+ \) such that \( s_{init} \xrightarrow{\sigma} s \), there exists a \( t \in T^+ \) and an \( s' \in S_F \) such that \( s \xrightarrow{t} s' \), i.e. each configuration process can reach a final state.

Let us now consider in which situations questions can be skipped. Although a fact is meaningful at the beginning, once the configuration process has begun, at a certain state it may turn out from the constraints that such a fact can only take one value of the two. In this case users do not have the freedom to choose, as the value to be given is inferred by the constraints. We call this type of fact forceable.

When this situation occurs for all the facts of a question, the question can have only one answer. Moreover, since a fact can appear in multiple questions, it may happen at a certain state that all the facts of a valid question have already been answered. Again, such a question can take only one possible answer. We call these questions skippable, as they can be automatically answered and thus should not be presented to the user.

Definition 4.18 (Forceable Fact, Skippable Question) Let \( QM = (F, F_D, F_M, Q, map_QF, pre_F, pre_Q, CS) \) be a questionnaire model, and let \( s \in S \) be a valid state of \( QM \), \( f \in F \) a fact and \( q \in Q \) a question:

- \( forceable(f, s) = f \in u(s) \land \forall F_1, F_2 \in CS \left[ (t(s) \subseteq F_1 \cap F_2 \land f(s) \cap (F_1 \cup F_2) = \emptyset) \Rightarrow F_1(f) = F_2(f) \right] \), i.e. \( f \) assumes the same value in all the valuations of facts still possible,
- \( skippable(q, s) = valid(q, s) \land \forall f \in map_QF(q) \left[ f \notin F_M \land (forceable(f, s) \lor f \notin u(s)) \right] \), i.e. a question can be skipped iff none of its facts is mandatory, and all its unset facts can have exactly one value or all its facts have been previously set.

If a question is skippable the only possible answer is valid, since this valuation always complies with the constraints. In fact the forceability of a fact is determined by the set \( CS \), whereas if all the facts have been previously set, the answer is already included in the last state \( s \), which is valid by assumption.

4.4 Tool Support

To establish the practical feasibility of the questionnaire-based approach, we have implemented an editor for the design of questionnaire models, namely Questionnaire Designer, and a tool for the dynamic generation of interactive questionnaires, namely Quaestio. Below we outline the features of these two applications which are part of the
Synergia toolset. Next, we demonstrate the use of Quaestio to configure the questionnaire model of Section 4.1.1. Finally, we describe a set of performance measurements to assess the scalability of Quaestio. An overview of Synergia showing how all the applications fit together is provided in the next chapter.

Questionnaire Designer (QD) is a Java application which has been realized with the Eclipse’s Graphical Modeling Framework [ECb]. QD is available in two releases: as a pluggable rich client platform and as a plugin of the Eclipse IDE [ECa]. QD allows users to visually create questionnaire models that can be later imported into Quaestio. The format of a questionnaire model (.qml) is described by an XML schema encoding the sets defined in Definition 4.1. The only exception is set $CS$. Instead, a string is used to store domain constraints directly as the conjunction of their boolean formulae (e.g., as the formulae shown in Section 4.1.2). A diagram file (.qml_diagram) is created from the XML format of the questionnaire model to provide a graphical representation of the model. Users can work on both the formats which are synchronized. A screenshot of the tool showing the questionnaire model for the order fulfilment process is depicted in Figure 4.5.

![Figure 4.5: Questionnaire Designer – showing the model in Figure 4.2.](image)

All the elements of a questionnaire model, such as facts, questions and preconditions, can be created through the palette or directly drawn from the canvas via a context-aware menu. A properties view allow the specification of domain constraints and offer specific options for facts and questions. For example, we can associate a descriptive guideline with each fact or question to aid the user in the configuration process. Via
this properties view we can also set a fact as mandatory, specify its default value or assign it a non-boolean type, i.e. integer, float or string, although no constraints can be defined over these types.

Theorem 4.16 proves that a few syntactic checks on the dependencies between facts and questions are enough to ensure deadlock-free configuration processes. This result ultimately enables the implementation of an efficient validation feature in QD to spot undesired circular dependencies that may cause deadlocks. This feature also detects inconsistencies in the model, e.g. a fact that is not associated with any question, a question without facts, a fact or question with a single partial dependency. Moreover, QD embeds an SBDD solver [Ray] to check the satisfiability of domain constraints and to identify non-meaningful facts which should be excluded from the model. An example of the validation feature is shown in Figure 4.6, where a cycle was found among four facts.

![Figure 4.6: Questionnaire Designer - detection of a cyclic dependency among 4 facts.](image)

Quaestio is a Java application that guides users through a set of questions given a questionnaire model as input. The graphical interface, shown in Figure 4.7, comprises a main window showing a list of Valid Questions, a list of Answered Questions and a Question Inspector. When a question is picked from one of these lists, the Question Inspector shows the question’s details, such as facts, dependencies on other questions, and guidelines to aid users answer the question. In a separate window, a Fact Inspector shows detailed information for each fact, such as its default value, whether it is mandatory, specific guidelines for the fact and the constraints that bind the fact.

Once a model is loaded, Quaestio shows the set of initial valid questions. Next, for each answer that users give, the tool dynamically calculates the next valid state and
updates the lists of valid and answered questions. The configuration process completes when all questions have been answered, or at least all mandatory facts have been set and the remaining ones can take their default value without violating the constraints. A (partial) domain configuration can be exported to an XML file (.dcl). This file keeps track of the facts that have been set and whether they deviate from their defaults.

For efficiency reasons, Quaestio does not build the state space of the questionnaire in memory (as defined in Section 4.3). Instead, we opted for a dynamic generation of the state space. For each answer given, the next state is calculated by scanning only those valid questions that are still unanswered. For each of them, we check if at least one precondition can be satisfied (we know there will be at least one such a question). If so, a question is put into the Valid Questions list if it is not skippable, otherwise it is added straight to the Answered Questions list.

Quaestio embeds the same SBDD solver used by QD to check the satisfiability of the domain constraints before starting the configuration process. At configuration time, the solver is used to construct an SBDD from the conjunction of the domain constraints and each potential answer given by the user. This generates a reduced formula. So long as this formula yields \textit{false}, the answer is not valid and the Answer button is kept disabled. In this way Quaestio prevents users from entering responses which would violate the constraints. For each valid answer that is fed into the tool, a new SBDD is constructed by updating the formula with the values of the facts that have been set. This avoids to compute the canonical form of the constraints every time from scratch, improving the overall response-time. After an answer, some facts may become forceable and thus some questions may be skippable. This condition is tested against the generated formula. For example, a formula that implies the negation of a fact indicates that the fact is forceable.
to false. In a similar way, Quaestio determines the type for a question (e.g. an XOR question).

Quaestio also offers an automatic completion feature: upon request the tool can automatically complete the configuration process whenever all the mandatory facts have been answered and default values can be used for the remaining facts. Furthermore, each answered question can be rolled back to the state before their answer. This feature was implemented with the purpose of preserving the answering order. When a question is rolled back, the current state is set to the one before answering the question, and hence, all the questions that were answered thereafter are rolled back too. Moreover, a fact occurring in multiple questions is kept forced to the value it was set the first time, until all its questions are rolled back.

In principle, it is possible to implement a selective rollback that ignores the answering order by suppressing the dependencies of the answered questions. In this way we could, e.g., roll back \( q_4 \) without rolling back \( q_5 \), although the latter has a dependency on the former. The new state, with \( q_5 \) in \( qs \) but not \( q_4 \), is still a valid state, since \( q_5 \) has no dependencies anymore. If a question being selectively rolled back had an interplay with some question already answered, its facts might be forced to assume an exact value. For example, if \( q_5 \) is answered with at least one of its facts set to \( \text{true} \) (i.e. by choosing a Manager for Loss or Damage Claim), \( q_4 \) will be rolled back with \( f_{13} \) forced to \( \text{true} \), as per \( DC_6 \).

### 4.4.1 Sample Configuration Process

We now demonstrate how Quaestio can be used to interactively configure the questionnaire for the order fulfillment process. For convenience, we introduce the notation \( a_m^q \) to indicate the valuation that is given by answer \( m \) to the facts of question \( n \) (where the facts that are not set by the answer are left out). Symbols \( T \) and \( F \) are used as shorts for a \( \text{true} \), respectively for a \( \text{false} \), valuation.

Assume, for example, that we want to handle SP shipments and support only Loss or Damage Claims managed by the Supplier, and that we are not interested in the Payment phase of the process as it will be outsourced. These can be common choices among the stakeholders of an organization interested in implementing the VICS specification.

Once the corresponding questionnaire model has been loaded into Quaestio, the valid questions are shown in the Valid Questions list. These are \( q_1 \) and \( q_3 \), since they have no dependencies (see Figure 4.8). The initial state is \( s_1 \) where no answers have been given, i.e. \( qs(s_1) = \emptyset \). We decide for example to answer \( q_3 \) – Which Logistics phases have to be implemented? – with its default answer. This corresponds to giving answer \( a_1^{q_3} = \{(f_8, T), (f_9, T), (f_{10}, T), (f_{11}, T)\} \), since all the facts of \( q_3 \) are \( \text{true} \) by default (shown by a green \( \square \) next to the fact’s description).

With \( a_1 \) we reach state \( s_2 \) with \( qs(s_2) = \{q_3\} \). \( q_2 \) is added to the valid questions due to its partial dependency on \( q_1 \) or \( q_3 \). Assume we choose \( q_1 \) from the Valid Questions. From the Question Inspector we can see that \( f_3 \) has been forced to \( \text{true} \) and has been grayed out (see Figure 4.9). The system has reacted to \( a_1 \) by setting \( f_3 \) in order to comply with \( DC_2 \). We answer \( q_1 \) with \( a_2^{q_1} = \{(f_1, T), (f_2, T), (f_3, F), (f_4, F)\} \) so as to exclude Payment.

After \( a_2 \), we reach \( s_3 \) with \( qs(s_3) = \{q_3, q_1\} \). Questions \( q_4, q_6 \) and \( q_7 \) are added to the
Figure 4.8: State $s_1$: the only valid questions are $q_1$ and $q_3$.

valid ones as they depend on $q_3$. Assume we now pick $q_2$ – *What is the expected Carrier’s Usage?*. Due to $DC_3$ and to the answers given so far, this question can only be answered if exactly one of its facts is set to *true* (the answer button is disabled). Also, this question needs to be explicitly answered as all its facts are mandatory (indicated by a red $\mathbf{X}$ next to the fact’s description). We select Single Package and $a_3^{q_2} = \{(f_5, F), (f_6, F), (f_7, T)\}$ is given.

The next state is $s_4$ with $qs(s_4) = \{q_3, q_1, q_2\}$. Although no questions depend on $q_2$, after answering $a_3$ both $q_6$ and $q_7$ become skippable, since all their facts can take only value *false* due to $DC_9$. Thus $a_4^{q_6} = \{(f_{16}, F), (f_{17}, F)\}$ and $a_5^{q_7} = \{(f_{18}, F), (f_{19}, F)\}$ are automatically given by the system, which moves from $s_4$ to $s_5$ with $a_5$, and from $s_5$ to $s_6$ with $a_6$. $q_6$ and $q_7$ are added to the set of answered ones (shown in gray in Figure 4.10) and $qs(s_6) = \{q_3, q_1, q_2, q_6, q_7\}$. Next we answer the only valid question remaining, $q_4$ – *Which claims have to be handled?*, with its default answer $a_6^{q_4} = \{(f_{12}, F), (f_{13}, T)\}$ which complies with the constraints.

After $a_6$ we reach $s_7$ with $qs(s_7) = \{q_3, q_1, q_2, q_6, q_7, q_4\}$. $q_5$ – *Which role has to act as Manager for Loss or Damage Claims?* is now valid as it depends on $q_4$. $s_7$ is a final state as all the mandatory facts have been set and the remaining ones still unset ($f_{14}$ and $f_{15}$) can take their defaults without violating the constraints. $q_5$ can thus be answered automatically with defaults. At this point we can decide whether to continue or to complete the configuration automatically. We decide to use the automatic completion feature and answer $a_7^{q_5} = \{(f_{14}, T), (f_{15}, F)\}$ is given by the system.
State $s_8$ is the next state with $qs(s_8) = \{q_3, q_1, q_2, q_6, q_7, q_4, q_5\}$. Assume that now we want to change $q_4$ in order to support only Return-merchandise Claims. In this case we can rollback $q_4$ and re-answer it. The system restores the current state to $s_6$, i.e. the state before answering $q_4$. We then answer $a_{q_4}^s = \{(f_{12}, T), (f_{13}, F)\}$ and reach $s_7$ again. This time, though, $q_5$ is skippable since a Manager can be chosen only for Loss or Damage Claims. The only valid answer is $a_{q_5}^s = \{(f_{14}, F), (f_{15}, F)\}$. With this we reach $s_8$ and complete.

The corresponding configuration trace is $\sigma = \{(s_1, (a_1, q_3), s_2), (s_2, (a_2, q_1), s_3), (s_3, (a_3, q_2), s_4), (s_4, (a_4, q_6), s_5), (s_5, (a_5, q_7), s_6), (s_6, (a_6, q_4), s_7), (s_7, (a_7, q_5), s_8)\}$ and the resulting configuration is $cf_\sigma = \{(f_1, T), (f_2, T), (f_3, T), (f_4, F), (f_5, F), (f_6, F), (f_7, T), (f_8, T), (f_9, T), (f_{10}, T), (f_{11}, T), (f_{12}, T), (f_{13}, F), (f_{14}, F), (f_{15}, F), (f_{16}, F), (f_{17}, F), (f_{18}, F), (f_{19}, F)\}$.

### 4.4.2 Performance Experiments

To demonstrate the scalability of Quaestio, we measured the time ($t_0$) taken by the embedded SBDD solver to transform the boolean function corresponding to the conjunction of all the domain constraints, into a canonical form that is then used for subsequent steps. This was found to be the critical factor to ensure the scalability of the tool when the number of facts and constraints increases.

The complexity of a boolean function can be determined by the number of nodes.
of the corresponding SBDD constructed in memory. As the SBDD size increases, the complexity of the function increases. This in turn entails that the solver will take more time to reduce and manipulate the function. The boolean function from the above case study, which ranges over 148 facts, was transformed to an SBDD of 2048 nodes with a response time of 28.3 ms (on average). This is the time needed by the solver to check the satisfiability of the function, and it represents the upper bound time of any (subsequent) evaluation of the same function. In fact, as users enter answers to the tool, the configuration space is pruned and thus the complexity of the boolean function decreases, lowering down the response time. In other words, users have to wait at most a time equal to \( t_0 \) to see if an answer is correct. Therefore, \( t_0 \) is an important yardstick of the efficiency of a pruning technique.

To simulate complex functions, we measured \( t_0 \) as we doubled the number of facts starting from 25, and randomly replicated the constraints over the new facts, yielding a proportional increase of the SBDD size. The experiments were conducted on an Intel Pentium M processor (2.0 GHz, 533 MHz FSB, 2 MB L2 cache), 1GB RAM (DDR2 533), 100GB HDD Serial ATA, running Microsoft Windows XP SP2. The results are shown in Table 4.1.

Although \( t_0 \) increases exponentially, we can observe that the SBDD size must be over 21,000 nodes for performance to start degrading significantly (\( t_0 \) is around 24 s). This size corresponds to 1,600 facts, which represents a configuration space of considerable size [Bat05].
In the above experiments, pseudo-constraints were generated with an interplay granularity of 25–50 facts. Although this can seem as a limitation, it reflects the fact that in practical scenarios, most configurable variables interact with a limited number of other variables. This allowed us to increase the size of the SBDD proportionally to the increase of the number of facts, and to draw conclusions based on the latter.

### 4.5 Summary and Discussion

This chapter presented a formal approach for capturing domain variability. The approach relies on questionnaire models composed of questions and domain facts, which encode possible answers to questions. These answers can be linked to variation points in a generic model or system to derive an individualized version of the latter. Questionnaire models support the definition of dependencies, both at the level of questions and at the level of facts, as well as domain constraints expressed in propositional logic. Simple well-formedness criteria ensure that no circular dependencies may occur, while satisfiability solving techniques are used to ensure the consistency of domain constraints and to incrementally prune the space of allowed answers at configuration time.

While the development of the approach was motivated by the need to support the configuration of business process models, the approach may be applied to support the configuration of other types of models (e.g. data models) or software artifacts in general, so long as an appropriate mapping is defined to match the notation(s) used. The approach was illustrated using a standard reference model for collaborative B2B processes.

An embodiment of this approach was presented in the form of an editor to create and validate questionnaire models and a tool to generate interactive questionnaires. The latter tool guides users through a set of questions in an order consistent with the dependencies between questions and facts, and in such a way that violations of domain constraints are preemptively avoided. Performance measurements have shown that this tool is stable and can efficiently scale with complex configuration scenarios.

The approach relies on boolean encodings of the space of possible answers. This encoding enables the use of efficient techniques to incrementally prune the space of answers, so that questions that become constrained or irrelevant due to answers given to previous questions, can be simplified or skipped. On the other hand, this boolean encoding can be a limitation in some scenarios. While enumerated types can be encoded by a collection of boolean values (and this encoding can be made transparent with appropriate tool support), this is not applicable for non-enumerated types (e.g. integers, strings). As a tradeoff, our approach also supports the definition of configuration parameters with non-enumerated types, but these cannot be used in the domain constraints unless a discretization of these types can be defined. One can envisage extensions of the

<table>
<thead>
<tr>
<th>Facts</th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
<th>1600</th>
<th>3200</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBDD Nodes</td>
<td>338</td>
<td>679</td>
<td>1361</td>
<td>2725</td>
<td>5453</td>
<td>10909</td>
<td>21821</td>
<td>43645</td>
</tr>
<tr>
<td>$t_0$ (ms)</td>
<td>3.2</td>
<td>12.3</td>
<td>22</td>
<td>67.2</td>
<td>457.8</td>
<td>3040.6</td>
<td>23859.4</td>
<td>164400</td>
</tr>
</tbody>
</table>

Table 4.1: Performance measurements.
approach by introducing linear programming constraints into the questionnaire model.

Questionnaire models such as the one presented in this chapter, can be seen as particular types of decision models [Alt80]. Indeed, questionnaire models use order dependencies to offer decision support in an interactive manner. Moreover, they help users to evaluate and compare alternative answers by means of guidelines and constraints. In this respect, our proposal shares commonalities with the CML2 language which was designed to capture configuration processes for the Linux kernel [Ray00]. Like Quaestio, CML2 supports the definition of validity constraints based on propositional formulas over so-called symbols (which may be three-valued in CML2). A questionnaire model in CML2 is composed of questions which lead to a given symbol being given a value. Questions can be grouped into menus which are arranged in a hierarchy. In CML2, questions within a menu are arranged sequentially while menus are visited from top to bottom. This is in contrast with our approach where questions (and facts) can be arranged in any partial order. Also, questions in CML2 only lead to one symbol being set, while our questions can be used to set multiple inter-related facts at once.

Questionnaire models are just one way among others to capture domain knowledge. We have chosen it because questionnaires provide a direct link between domain concepts and decisions that need to be made in the context of business process model configuration. There are other ways of capturing domain models, which one could be consider using instead of, or in addition to questionnaires. A common approach to capture domain knowledge is by means of ontologies. Ontologies are generally used to reason about the semantic relationships between domain concepts, e.g. for the disambiguation of terms. They have found wide applicability in the Semantic Web community [AH08], where languages such as OWL (Web Ontology Language) [W3C04] have been proposed to describe Web resources in order to enable their discovery and invocation. However ontologies are more suitable to categorize and to link domain concepts, while the focus of questionnaire models is on capturing decision points and alternatives.

In Chapter 2 we described an approach for process model configuration based on model projection [BDD+04, BDK07]. This approach provides a means to capture the variability of the application domain as a set of logical configuration parameters, similar to domain facts in our approach. These parameters can then be linked to process model elements to drive the configuration of the latter. However, only local constraints among parameters can be defined (e.g. “A includes B” or “A excludes B”), but no method is provided to check for model-wide consistency that could deny parameter settings that are practically unfeasible. Moreover, no control is given over the order in which parameters are presented to users.

We also outlined two research streams in the area of SPLE, namely Software Configuration Management (SCM) and Feature Diagrams (FDs), that have studied techniques for modeling variability in large software systems. While our questionnaire models only capture variability, SCM and FDs capture both variability and commonality in the same model. As a result, in these approaches there is no clear separation between what can vary and what is always stable. For example, in a feature diagram features can be mandatory if they must be included in the system or model, or optional if they can be excluded. Although the latter always represents a variability, the former does not always represent a commonality. In fact, if a mandatory feature or any of its parents has, e.g., an XOR relation with other features, it can still be excluded depending on the configuration of the other features. Similarly, when a mandatory feature is the child
of an optional feature, it can still be excluded if its parent is excluded. This lack of separation between stable and variable aspects can increase the model complexity and hinder the communication of variability to the relevant stakeholders [PM07].

In our proposal, variability and commonality are captured separately: questionnaire models focus on resolving variability, while the commonalities are captured in the generic model or system to be configured (e.g. a non-configurable function in a C-EPC process model). In this respect, our approach is closer to the principles of Orthogonal Variability Models (OVMs) [PBL05, PM07]. An OVM captures the variability of a product line in a dedicated tree-structure diagram by means of so-called variation points. The variants for each variation point are linked to a separate conceptual model representing the product line to be configured, where both variability and commonalities are modeled. In our approach we explicitly model this relation by attaching domain facts to variation points (cf. Chapter 5), so as to reflect the effects of a configuration on the system or model to be configured. Our facts can be compared to variants, and questions to variation points, i.e. answering a question would allow one to determine which variants are chosen for a given variation point. Having said that, a questionnaire model offers more flexibility than a tree-like structure, as we can express non-hierarchical dependencies among features, and questions can refer to multiple variation points/features. Moreover, our approach describes the process through which a system or model is to be configured and for this reason, it can be used to complement a FD or OVM. The relation among our approach, SCM, FD and OVM is depicted in Figure 4.11.

![Figure 4.11: Comparison among SCM, FDs, OVM and questionnaire models (QMs).](image)

Feature modeling languages have been embodied in a number of tools. Some of them rely on a boolean encoding of features. This is the case of the guids1 module in AHEAD [Bat] and of Pure::Variants [P-S08]. Other tools can check constraints over non-boolean variables but validate a configuration only a posteriori [AC04, BEL, BLSI, HHPK05]. In contrast to Quaestio, none of these tools provides (fine-grained) control over the order in which choices are presented to users at configuration time. In particular, an early version of FeaturePlugin [AC04] provided a wizard to traverse a feature model in a predetermined order (depth-first), but did not support other orderings. However this feature was later removed from the tool. Moreover, the above tools are not able to incrementally prune the configuration space. In other words, these tools are not able to preemptively stop the user from entering configuration values that would lead to inconsistent configurations. For example Pure::variants only includes an auto-resolving function that, given a constraint “A requires B”, if feature A is selected then feature B becomes automatically selected. However, Pure::variants does not support arbitrary constraints between features (i.e. “facts” in our framework). An exception is provided by guids1 which, like Quaestio, relies on a SAT solver. The
possibility of using SAT solvers to incrementally prune the configuration space has been raised by Batory [Bat05], but to the best of our knowledge, Quaestio provides the first realization of this idea in an interactive manner.

Our tool is also related to questionnaire systems. A range of commercial products, such as Vanguard Software’s Vista [VSC], support the definition of online questionnaires and the collection and analysis of responses. Such systems rely on the notion of question flows, wherein questions are related by a fixed precedence order, while branching operators are used to capture conditional questions. This paradigm is procedural: the developer of the questionnaire needs to determine in advance the points where branching occurs. Additionally, constraints are expressed at the granularity of questions and only used to skip questions. This makes it difficult to capture scenarios where questions can be (partially) answered on the basis of previous answers.

The next chapter shows how the questionnaire-based approach can be applied to the configuration of process models.
Chapter 5

Questionnaire-Driven Staged Process Configuration

This chapter completes the discussion on domain-driven configuration by illustrating how the questionnaire-based approach presented in Chapter 4 can be applied to the configuration of process models. In doing so, we exploit the results on process model correctness exposed in Chapter 3.

The idea is to capture the variability of a configurable process model in terms of domain decisions that need to be taken. These decisions are encoded by domain facts and their interdependencies are captured via a set of domain constraints. Domain facts and their constraints are structured in a questionnaire model. On the other hand, we encode each variant of a variation point in the configurable process model by a boolean variable, namely a process fact. We use the technique exposed in Chapter 3 to infer a set of process constraints over these process facts that guarantee the semantic correctness of the individualized process models. Then we link domain constraints and process constraints by mapping each process fact to a boolean formula over domain facts. By satisfying this unified set of constraints we ensure that the individualized process models are both consistent with the domain configuration and semantically correct. In this way a process model can be configured by subject-matter experts at the domain level, by answering the interactive questionnaire generated from the questionnaire model. From the answers (i.e. from a valuation of domain facts) a valuation of process facts is automatically derived. Via the mapping this valuation is applied to the process model, and via an individualization algorithm an individualized process model is generated which is semantically correct. An overview of the approach is shown in Figure 5.1.

The main innovation of this proposal is that process configuration is not driven directly by a (configurable) process model, but rather by a model that captures the variability of the application domain. This model masks the complexity of the underlying process model. An advantage of this approach is that process model correctness can be ensured by reasoning only with process constraints. As shown in Chapter 3, these constraints can be automatically induced from a configurable process model, thus requiring minimal effort from the modeler who has to create the process models and the mapping between these and the questionnaire models. Moreover, defining domain constraints over business choices is more intuitive than doing the same over the nodes of a process model. Consequently, domain experts can directly provide input for the creation of questionnaire models.
5. Questionnaire-Driven Staged Process Configuration

This chapter demonstrates the applicability of the proposal to the C-EPC notation (cf. Chapter 3). Also, it shows how the Synergia toolset can be used to provide tool support to the approach. We have tested this approach on several examples, particularly in the area of screen post-production. In this chapter we reuse the reference process model for picture post-production introduced in Chapter 2. The applicability of the approach to the C-YAWL notation is discussed in [LGDA08].

The remainder of this chapter is structured as follows. Section 5.1 describes a working example in C-EPC. This prepares the ground for the presentation of the approach in Section 5.2. Next, Section 5.3 discusses a methodology for the construction of the mapping between questionnaire models and process models and shows some desirable properties of the mapping. Finally, Section 5.4 presents the architecture of the Synergia toolset and shows how this toolset can be used for the questionnaire-driven configuration of process models. Section 5.5 concludes the chapter with a summary and a discussion on related work.

This chapter presents and expands upon work published in [LLS+07, LGDA08, LD08].

5.1 Working Example

Let us consider again the reference process model for picture post-production in C-EPC that was introduced in Chapter 2. Figure 5.2 depicts this process model without configurable join connectors and their requirements. Since this model is sound and joins are not configurable, we can apply the technique exposed in Chapter 3 (cf. Definition 3.29) to derive a set of process constraints that, if satisfied by a process configuration, ensure the preservation of the soundness in any individualized EPC.

In order to link this C-EPC model to a questionnaire model, we first need to capture each variant of each variation point with a process fact, which will be set to true if the specific variant is selected during configuration, and to false otherwise. Since the process
Figure 5.2: The configurable process model for picture post-production in C-EPC, without configurable joins.

model features three configurable OR-splits and four configurable functions, we derive a total of 27 process facts to cover all the possible variants. The process facts for each of the seven variation points in this C-EPC model are presented below.

\[ c_1 \rightarrow p_{ON}^{c_1}, p_{AND}^{c_1}, p_{XOR}^{c_1}, p_{SEQ}^{e_1}, p_{SEQ}^{c_2} \]
\[ c_2 \rightarrow p_{ON}^{c_2}, p_{AND}^{c_2}, p_{XOR}^{c_2}, p_{SEQ}^{e_2} \]
\[ c_3 \rightarrow p_{ON}^{c_3}, p_{AND}^{c_3}, p_{XOR}^{c_3}, p_{SEQ}^{e_3}, p_{SEQ}^{e_4} \]
\[ c_4 \rightarrow p_{ON}^{c_4}, p_{AND}^{c_4}, p_{XOR}^{c_4}, p_{SEQ}^{e_4}, p_{SEQ}^{c_5} \]
\[ c_5 \rightarrow p_{ON}^{c_5}, p_{AND}^{c_5}, p_{XOR}^{c_5}, p_{SEQ}^{c_6}, p_{SEQ}^{e_6} \]
\[ n_1 \rightarrow p_{ON}^{n_1}, p_{OPT}^{n_1}, p_{OFF}^{n_1} \]
\[ n_2 \rightarrow p_{ON}^{n_2}, p_{OPT}^{n_2}, p_{OFF}^{n_2} \]
\[ n_3 \rightarrow p_{ON}^{n_3}, p_{OPT}^{n_3}, p_{OFF}^{n_3} \]
\[ n_4 \rightarrow p_{ON}^{n_4}, p_{OPT}^{n_4}, p_{OFF}^{n_4} \]
The second step is to infer a set of process constraints over the above process facts, in order to avoid structural issues that might lead to unsoundness. In fact, as argued in Chapter 3 (cf. Section 3.6), it is sufficient to enforce minimal structural requirements to guarantee the semantic correctness of any individualized EPC. The process constraints for the C-EPC example are presented below.

\[
PC_1: \quad (\neg p_{SEQ_1}^1 \lor \neg p_{SEQ_1}^2) \land (\neg p_{SEQ_4}^3 \lor \neg p_{SEQ_4}^4) \land (\neg p_{SEQ_4}^5 \lor \neg p_{SEQ_5}^6) \\
PC_2: \quad \neg p_{OFF}^2 \Rightarrow (\neg p_{SEQ_1}^1 \lor \neg p_{SEQ_4}^2) \land (\neg p_{SEQ_4}^3 \lor \neg p_{SEQ_4}^6) \land \neg p_{SEQ_5}^6 \\
PC_3: \quad \neg p_{OFF}^2 \Rightarrow (\neg p_{SEQ_2}^1 \lor \neg p_{SEQ_4}^4) \land (\neg p_{SEQ_4}^3 \lor \neg p_{SEQ_4}^5) \land \neg p_{SEQ_5}^6 \\
PC_4: \quad \neg p_{OFF}^2 \Rightarrow (\neg p_{SEQ_2}^1 \lor \neg p_{SEQ_4}^4) \land (\neg p_{SEQ_4}^3 \lor \neg p_{SEQ_4}^5) \land \neg p_{SEQ_5}^6 \\
PC_5: \quad \neg p_{OFF}^2 \Rightarrow (\neg p_{SEQ_2}^1 \lor \neg p_{SEQ_4}^4) \land (\neg p_{SEQ_4}^3 \lor \neg p_{SEQ_4}^5) \land (\neg p_{SEQ_4}^6 \lor \neg p_{SEQ_5}^6). \\
\]

\(PC_1\) ensures the existence of at least a directed path between the start and the end event, while the other constraints ensure that each configurable function that is not switched \(OFF\) be on at least one such a path.

Let us now construct a questionnaire model to capture the variability of this domain. The choices that need to be made before starting a picture post-production project relate to the shooting media (Tape, Film) the type of picture cut (Online, Negmatching) and the delivery (Tape, Film, New medium). We capture each of these choices via domain facts and group them in questions. The questionnaire model is depicted in Figure 5.3.

Question \(q_3\) groups the facts referring to the shooting media, \(q_1\) groups the facts for the type of cut and \(q_2\) those for the expected deliverables. Besides these choices, we define two more questions, \(q_1\) and \(q_2\), which capture two critical decisions for post-production: on the estimated budget and on the distribution channels. The facts of \(q_1\) capture the three typical budget levels for a screen project (Low, Medium and High) while the facts of \(q_2\) capture the available distribution channels (e.g. Cinema, TV, Home). These decisions have a strong influence on the other choices, and as such they are usually taken first. Accordingly, we have set order dependencies such that the choices on the shooting media, picture cut and delivery can only be taken after answering either \(q_1\) or \(q_2\).

The interdependencies among these choices have been captured by the following set of domain constraints:

\[
DC_1: \quad \text{xor}(f_1, f_2, f_3) \quad DC_2: \quad f_1 \Rightarrow \neg(f_{10} \lor f_{14}) \quad DC_3: \quad f_2 \Rightarrow \neg f_{10} \\
DC_4: \quad f_4 \lor f_5 \lor f_6 \lor f_7 \lor f_8 \quad DC_5: \quad f_4 \Rightarrow f_{14} \quad DC_6: \quad f_5 \Rightarrow f_{13} \\
DC_7: \quad f_6 \Rightarrow (f_{13} \lor f_{15}) \quad DC_8: \quad (f_7 \lor f_8) \Rightarrow f_{15} \quad DC_9: \quad f_9 \lor f_{10} \\
DC_{10}: \quad f_{11} \lor f_{12} \quad DC_{11}: \quad \neg f_{10} \Rightarrow \neg f_{12} \quad DC_{12}: \quad f_{13} \lor f_{14} \lor f_{15} \\
DC_{13}: \quad f_{12} \Rightarrow f_{14}. \\
\]

For example, since a project must shoot at least on tape or film, the facts of \(q_3\) are bound by a logical \(OR\) (\(DC_9\)). Similarly, the facts capturing the budget ranges in \(q_1\) are in an \(XOR\) relation (\(DC_1\)) since exactly one of them can be chosen at a time. The influence that \(q_1\) and \(q_2\) exercise on the other choices is captured by the constraints over their facts (\(DC_1\) to \(DC_8\)), which determine a number of restrictions on the answers of the other questions. For example, since shooting on film (\(f_{10}\)) results in a more costly operation due to the special treatments that are required for making it visible and
permanent, this option is not available in low/medium budget projects ($DC_2$, $DC_3$). In this case, Negmatching ($f_{12}$) cannot be chosen in $q_4$, because this activity can only be performed if the project is shot on film ($DC_{11}$). Moreover, given the costs involved, Negmatching is worthwhile only if the project finishes on film ($DC_{13}$), but the latter is not allowed in low budget projects ($DC_{2}$).

As an illustration, Figure 5.4 shows the propagation of the domain constraints on the questionnaire model if we chose low budget in $q_1$. Some facts would be forceable to true (e.g. $f_9$) and others to false (e.g. $f_{10}$) and consequently some questions would become irrelevant (i.e. $q_3$ and $q_4$) and thus skippable. At this point it would be sufficient to answer $q_2$ and $q_5$ to complete the configuration.

Figure 5.3: The questionnaire model for picture post-production.

Figure 5.4: Choosing low budget has a number of implications on other decisions.
We can observe some correspondence between the above choices and the variation points in the C-EPC process model. For example, choosing to shoot only on tape (i.e. setting \( f_9 \) to \( \text{true} \) and \( f_{10} \) to \( \text{false} \)) would correspond to configuring the first \( OR \)-split to \( SEQ_{e_1} \), so that function Prepare film for editing is dropped from the model, and vice versa, choosing to shoot only on film would correspond to configuring the split to \( SEQ_{e_2} \), to avoid function Prepare tape for editing. In the next section we show how this link between process facts and domain facts is achieved.

5.2 Linking Domain and Process Variability

We link the domain variability captured by a questionnaire model with the process variability captured by a configurable process model by equating each process fact with a propositional logic formula over domain facts. For example, let us consider the configurable \( OR \)-split \( c_1 \) in the example C-EPC process model. We equate its process fact \( p_{SEQ_{e_1}}^{c_1} \) with the formula \( \neg f_{10} \) (only tape), i.e. \( p_{SEQ_{e_1}}^{c_1} \Leftrightarrow \neg f_{10} \). We also equate \( p_{SEQ_{e_2}}^{c_1} \) with \( \neg f_9 \) (only film) and \( p_{AND}^{c_1} \) with \( f_9 \land f_{10} \) (both tape and film), while we equate \( p_{OR}^{c_1} \) and \( p_{XOR}^{c_1} \) with \( \text{false} \), since we do not want to defer the choice on the shooting media until run-time. The conjunction of all these equivalence formulae is a \( c \)-mapping (“\( MC \)” for short).

Via \( MC \) we can configure a variation point in the process model by checking which formula of the ones that are equated with the process facts for that variation point holds against a domain configuration. For example, according to the above example, if we set both \( f_9 \) and \( f_{10} \) to \( \text{true} \) in the questionnaire, \( c_1 \) gets configured to its \( AND \) variant in the C-EPC model. Specifically, to determine a process configuration from a domain configuration, we enforce the conjunction of the domain constraint (\( DC \)), the process constraint (\( PC \)) and the \( c \)-mapping (\( MC \)). By satisfying this unified constraint (\( UC \)), we are sure that any process configuration derived from a domain configuration will be semantically correct and consistent with the domain choices. An overview of the approach is illustrated in Figure 5.5.

![Diagram showing the mapping between questionnaire models and process models](image)

Figure 5.5: The mapping between questionnaire models and process models is achieved by equating each process fact with a boolean expression over domain facts.

The above concepts are formalized in the following definition.

**Definition 5.1 (C-Mapping, Unified Constraint, Solution Spaces)** Let \( F \) be the set of domain facts and \( P \) be the set of process facts inferred from a configurable process model defined in a suitable notation (e.g. C-EPC). Then:
- \( DC \in \mathbb{B}_F \) is a propositional logic formula over \( F \) denoting the domain constraint;
- \( PC \in \mathbb{B}_P \) is a propositional logic formula over \( P \) denoting the process constraint;
- let \( M \in P \rightarrow \mathbb{B}_F \), then \( MC = \bigwedge_{p \in P} (p \iff M(p)) \) is a propositional logic formula over the union of \( F \) and \( P \) denoting the c-mapping;
- \( UC = DC \land PC \land MC \) is a propositional logic formula over the union of \( F \) and \( P \) denoting the unified constraint;
- \( V_F = \{true, false\} \) is the set of final valuations of \( F \) where no domain fact is left unset;
- \( V_P = \{true, false\} \) is the set of final valuations of \( P \) where no process fact is left unset;
- \( S_{DC} = \{v \in V_F \mid v \models DC\} \) is the set of domain configurations, i.e. the solution space of \( DC \);
- \( S_{PC} = \{v \in V_P \mid v \models PC\} \) is the set of process configurations, i.e. the solution space of \( PC \);
- \( S_{UC} = \{v \in V_F \cup V_P \mid v \models UC\} \) is the set of unified configurations, i.e. the solution space of \( UC \).

The complete c-mapping between the questionnaire model of Figure 5.3 and the C-EPC process model of Figure 5.2 is given by the conjunction of the following formulae.

\[
\begin{align*}
MC_1 &: p_{or}^1 \iff false \\
MC_2 &: p_{and}^1 \iff f_9 \land f_{10} \\
MC_3 &: p_{xor}^1 \iff false \\
MC_4 &: p_{seq_1}^1 \iff \neg f_{10} \\
MC_5 &: p_{seq_2}^1 \iff \neg f_9 \\
MC_6 &: p_{or}^3 \iff false \\
MC_7 &: p_{and}^3 \iff f_{11} \land f_{12} \\
MC_8 &: p_{xor}^3 \iff false \\
MC_9 &: p_{seq_3}^1 \iff \neg f_{12} \\
MC_{10}:& p_{seq_4}^1 \iff \neg f_{11} \\
MC_{11}:& p_{or}^5 \iff false \\
MC_{12}:& p_{and}^5 \iff (f_{12} \land \neg f_{13} \land f_{15}) \lor (f_{13} \land f_{14}) \\
MC_{13}:& p_{xor}^5 \iff false \\
MC_{14}:& p_{seq_5}^3 \iff (f_{13} \lor f_{15}) \land \neg f_{14} \\
MC_{15}:& p_{seq_6}^3 \iff (\neg f_{12} \lor \neg f_{15}) \land \neg f_{13} \land f_{14} \\
MC_{16}:& p_{on_1}^4 \iff \neg f_{11} \land (f_{13} \lor f_{15}) \\
MC_{17}:& p_{opt_1}^4 \iff false \\
MC_{18}:& p_{off_1}^4 \iff \neg (\neg f_{11} \land (f_{13} \lor f_{15})) \\
MC_{19}:& p_{on_2}^4 \iff \neg f_{12} \land f_{14} \\
MC_{20}:& p_{opt_2}^4 \iff false \\
MC_{21}:& p_{off_2}^4 \iff (\neg f_{12} \land \neg f_{14}) \\
MC_{22}:& p_{on_3}^7 \iff f_{13} \\
MC_{23}:& p_{opt_3}^7 \iff false \\
MC_{24}:& p_{off_3}^7 \iff \neg f_{13} \\
MC_{25}:& p_{on_4}^7 \iff f_{15} \\
MC_{26}:& p_{opt_4}^7 \iff false \\
MC_{27}:& p_{off_4}^7 \iff \neg f_{15}
\end{align*}
\]

In order for a process configuration to be effective, we must ensure that \( UC \) implies the exclusive disjunction of all the process facts for a variation point. This condition is needed to guarantee that a domain configuration will lead to exactly one process fact per variation point being asserted and all the other process facts being negated. If this condition holds, we say that \( MC \) is unambiguous. In an unambiguous c-mapping,
exactly one variant is selected per variation point, i.e., for example, a domain configuration should not require that at the same time a C-EPC function is turned ON and OFF.

The c-mapping presented above is unambiguous. In fact, the boolean formulae for the variants of $c_1$ ($MC_1$ to $MC_5$) are in mutual exclusion because of the restrictions imposed by $DC_9$. Therefore, only one variant of $c_1$ can be selected at a time. Similar considerations hold for the other variation points of the C-EPC example.

Answering a question may determine the configuration of one or multiple variation points. Consider for example $q_4$ which inquires the type of cut. Setting its facts $f_{11}$ (Online) and $f_{12}$ (Negmatching) determines only the configuration of the OR-split $c_3$ (see $MC_7$, $MC_9$ and $MC_{10}$). On the other hand, answering $q_1$, which inquires the budget level, has a wide impact on the process model. If, e.g., we set $f_1$ (low budget) to true, a number of variation points will be configured at once, even if $f_1$ does not explicitly appear in $MC$. This is because of the domain constraints that bind this fact to various domain facts (cf. Figure 5.4), and consequently, to various process facts. In particular, all the variation points whose process facts have been equated with $\neg f_{10}$, $\neg f_{12}$ and $\neg f_{14}$, i.e. $c_1$, $c_3$ and $c_5$, would be configured in order to remove all the branches of the process involving film. In general, the more impact a domain fact has on other facts, the more variation points are likely to be affected in the process model. This depends on how the domain constraints and the c-mapping have been set.

In other cases, we need to answer more than one question to determine the configuration of a variation point. This is the case of configurable function Transfer in telecine ($n_1$), whose configuration depends on the answer given to $q_4$ and $q_5$. In fact, a transfer in Telecine, which converts the edited film to tape, is required if the cut is only done in Negmatching (i.e. if $f_{11}$ is negated in $q_4$) and the project is finished on tape or on a new medium (i.e. if $f_{13}$ or $f_{15}$ is asserted in $q_5$), as per $MC_{16}$. Similarly, we can determine the configuration of function Record Digital Film Master ($n_2$) by answering $q_4$ and $q_5$. In fact, a Digital Film Master, which transfers the editing results to film, is required if the cut is only done online and the project is finished on film, as per $MC_{19}$.

By representing the domain variability in a separate model, we can avoid capturing the interdependencies of the domain in the configurable process model, since these are directly captured over business choices and propagated to the process model via an unambiguous c-mapping. Therefore, we do not need to set up configuration requirements in a C-EPC model, such as the ones shown in Figure 5.2. In fact, Requirements 5 and 6 are captured by $MC_{16}$ and $MC_{19}$ respectively, Requirement 1 is captured by $DC_{11}$ and Requirement 4 is captured by $DC_{13}$. Requirements on the process model have thus to deal only with the preservation of the model correctness, and can be automatically inferred from the structure of the process model. Likewise, we do not need to specify configuration guidelines as logical expressions over the variation points of a C-EPC, as these guidelines can be defined in natural language over the business choices of a questionnaire model. For example, we can express Guideline 1 – which suggests performing only a negmatching if the project is shot on film – in terms of budget, creativity or other business-related choices. These are all aspects that cannot be captured by the variation points of a C-EPC.

Let us now see how process constraints and domain constraints interact with each other when we configure a process model through a questionnaire. Assume we want to configure the C-EPC model for picture post-production for a low budget project.
releasing on TV and on the Internet. Assume we start the questionnaire by answering \( q_2 \) with TV \( (f_5) \) and Internet \( (f_8) \). TV requires to finish on tape \( (DC_6) \), while Internet requires to finish on a new medium \( (DC_8) \). This determines a partial answer for \( q_5 \) with \( f_{13} \) and \( f_{15} \) becoming true, as shown in Figure 5.6. We propagate this choice to the process model via the c-mapping. This results in configuring functions Finish on tape \( (n_3) \) and Finish on new medium \( (n_4) \) to ON (as per \( MC_{22} \) and \( MC_{25} \)). Since \( n_3 \) is ON, the process constraints forbid the variant \( SEQ_{e_6} \) of \( c_5 \) \( (PC_4) \), thus keeping the process model sound. The resulting model after applying the \( \beta_I \) algorithm is shown in Figure 5.6.

Figure 5.6: The effects of choosing TV and Internet as distribution channels on the process model.

At this point it is sufficient to answer low budget in \( q_1 \) to conclude the configuration. We have seen that this choice affects a number of domain facts by denying all the options involving film. Consequently, the project must shoot on tape, cut online and release on tape or new medium. This complies with our previous choice of TV and Internet in \( q_2 \). As shown in Figure 5.4, we can now fully determine the answers to questions \( q_3, q_4 \) and \( q_5 \). One more time, we propagate this choice to the process model. This results in configuring all the OR-splits to their left-hand side sequence to avoid the film variant, and to configure function Telecine transfer \( (n_1) \) to OFF since the cut is only performed online. Given this process configuration, the process constraints impose to configure function Record Digital Film Master \( (n_2) \) to OFF since this function is now unreachable \( (PC_3) \). The resulting model after applying the individualization algorithm is shown in Figure 5.7.

Given that UC must be satisfiable and that MC must be unambiguous in order to configure a process model through a questionnaire model, we can decide whether to apply the individualization algorithm at each configuration step, as we did in the
example above, or once we have completed the domain configuration. In the first case, we obtain a valuation of process facts for each answer we give and use the individualization algorithm to commit this partial process configuration to the process model. In the second case, we configure the questionnaire model first and then use the individualization algorithm to commit the final process configuration to the process model. In any case, the result is ensured to be the same by $UC$.

In the next section we propose a methodology to constructively define a $c$-mapping between a questionnaire model and a process model and we discuss some desirable properties of the mapping.
5.3 Constructing Mappings between Questionnaire Models and Process Models

The first step towards the construction of a c-mapping is to capture the variability of a given domain by means of a questionnaire model. This task should be conducted by the modeler in close collaboration with domain experts (in our example, a post-production supervisor or a producer). Similarly, the modeler should represent the process(es) in the application domain via configurable process models defined in a suitable notation. Process facts should be identified with the variants of the process and process constraints should be defined to preserve the model correctness (depending on the notation adopted, process facts and their constraints can be automatically derived from the model).

As a rule of thumb, a fact is meant to represent a variant of the domain or process model. Thus a fact should be considered as such only if it can be freely set before starting the configuration; otherwise it would represent a commonality in the domain or process model, and should be left out.

Once domain facts, process facts and their constraints have been identified, it is possible to define the c-mapping by means of a two-way impact analysis:

- **from domain to process**: e.g., given a domain fact, we need to estimate what are the implications of setting such a fact to true/false in the process model;
- **from process to domain**: e.g., given a variation point, we need to consider which domain facts are impacted by configuring such a point with one of its variants.

As per Definition 5.1, the solution space yielded by UC ($S_{UC}$) is given by the intersection of the two solution spaces of the domain and of the process via MC. Even if UC is satisfiable (i.e. $S_{UC}$ is not empty) and MC is unambiguous, some process configurations might no longer be available, i.e. $DC \land MC$ might reduce the solution space of PC ($S_{PC}$). A simple example is shown by the c-mapping presented in Section 5.2, where some process facts are mapped to the always-false formula. As a result, all the process configurations where these variants are selected would be denied. If PC is not restricted by $DC \land MC$, it means that $S_{PC}$ is a subset of the solution space $S_{PC}$. In this case we say that MC is process-neutral.

Similarly, some domain configurations might no longer be feasible after applying the c-mapping, i.e. $PC \land MC$ might reduce the solution space of DC ($S_{DC}$). This situation may lead to problems when it comes to communicating these restrictions to the stakeholders, and as such should be avoided. In these cases, either the c-mapping or the process model should be revised, assuming the questionnaire model is correct. If DC is not restricted by $PC \land MC$, $S_{DC}$ is a subset of $S_{UC}$. In this case we say that MC is domain-neutral.

It is desirable to have a c-mapping which is domain-neutral but not necessarily process-neutral. The first property guarantees that all the domain options are feasible. The second property denies all those process configurations that contradict the domain dependencies even if they lead to semantically correct individualized process models.

The c-mapping presented in Section 5.2 is domain-neutral but not process-neutral. In the following we formally define the properties of domain-neutrality and process-neutrality.
Definition 5.2 (C-Mapping Properties) Let $F$ be the set of domain facts, $DC$ the domain constraint and $S_{DC}$ its solution space; $P$ the set of process facts, $PC$ the process constraint and $S_{PC}$ its solution space. Let also $MC$ be a c-mapping between process facts and domain facts, $UC$ be the unified constraint and $S_{UC}$ its solution space. Then:

- $MC$ is domain-neutral iff for each valuation $v \in S_{DC}$ there exists a valuation $v' \in S_{UC}$ such that $v'|_P = v$;
- $MC$ is process-neutral iff for each valuation $v \in S_{PC}$ there exists a valuation $v' \in S_{UC}$ such that $v'|_P = v$.

If $MC$ is domain-neutral, we can configure the questionnaire model independently of the c-mapping, i.e. we only need to impose $DC$ while answering the questionnaire. Once we have completed the domain configuration, we can use $MC$ to derive the corresponding process configuration and enforce $PC$ so that this leads to a sound individualization. In fact $PC$ cannot affect any domain fact, so it can be safely enforced after answering the questionnaire.

Finally, if $UC$ is not satisfiable while both $DC$ and $PC$ are satisfiable, it means that the combination of domain constraints and the c-mapping are so restrictive that no correct process model is allowed at all.\(^1\) In these cases, process facts and their constraints can be used to evaluate the feasibility of the domain constraints and of the c-mapping, and to suggest their revision.

In the next section we show how the Synergia toolset supports the creation of c-mappings and the questionnaire-driven configuration of process models.

### 5.4 Tool Support

Synergia is an open-source toolset which comprises six interrelated tools to provide end-to-end support for process model configuration. Its tools assist domain experts and process modelers in creating questionnaire models and configurable process models, in mapping questionnaire models to process models, in answering interactive questionnaires and in individualizing process models according to the answers given. The toolset currently supports process models defined in the C-EPC and C-YAWL notations.

From a software perspective each tool is a standalone rich client application responsible for a specific task in the configuration process. Figure 5.8 provides an overview of the software architecture showing the various tools and how they interact with each other via their input/output formats.

The Questionnaire Designer and Quaestio have already been described in Chapter 4. The former is used to create questionnaire models (.qml) while the latter is used to generate interactive questionnaires that can be answered by users.

Questionnaire models can be imported into the Mapper tool for the purpose of defining c-mappings (.cmap) with configurable process models. The tool can import C-EPC models specified in the EPML 1.2 format (.epml) [MN06], and C-YAWL models specified in the YAWL 2.0 format (.yawl) [YF]. The latter incorporates elements to represent and configure variation points in C-YAWL, which we described in [GAJL08].

\(^1\)We observe that if the process model is sound, the process constraints are always satisfiable.
C-EPC models can be visually created in the *C-EPC Designer*. This tool is an extension of the EPC Tools [CK] to support the creation of variation points on top of EPC models. EPC Tools is an open-source plugin for Eclipse which also offers verification capabilities. The C-EPC Designer is presented in detail in Chapter 6.

Once the serialization of a process model has been imported into the Mapper, the tool derives a process fact from each variant of the model, and builds a set of process constraints over these facts according to the process model notation adopted (cf. Definition 3.29 in Chapter 3 for C-EPCs and [LGDA08] for C-YAWL). Process constraints are stored in the c-mapping file as well.

A c-mapping between process facts and domain facts can be created by assigning each process fact a boolean formula over the domain facts. The Mapper relies on the same SBDD solver [Ray] embedded in Quaestio and in Questionnaire Designer. The solver is used to check the satisfiability of the set of unified constraints $UC$ obtained by the conjunction of domain constraints, process constraints and the c-mapping. Given $UC$, the tool can also verify the properties of a c-mapping, i.e. unambiguity, domain-neutrality and process-neutrality. Unambiguity is verified by checking if $UC$ implies the disjunction of all the variants for each variation point. Domain-neutrality and process-neutrality are verified by exploiting properties that are specific to the SBDD representation (cf. the *constrain* operator in [Ray]). In essence, the tool compares the reduced formula obtained by $DC$ with that obtained by evaluating the impact of $MC \land PC$ onto $DC$. If the latter imposes further constraints, it means that the solution space of $DC$ is reduced by $MC \land PC$, i.e. the c-mapping is not domain-neutral. Process-neutrality can be verified in a similar way.

An interesting feature of the Mapper tool is the possibility of creating multiple mappings between one questionnaire model and several process models in a single file. For example, this can be used when the same process model is defined in both the C-EPC and the C-YAWL notations.

The next tool, *Process Configurator*, configures a process model according to the answers of a questionnaire. It accepts a domain configuration (.dcl) generated by Quaestio, the serialization of a process model in C-EPC or C-YAWL, and the c-mapping between the latter and the questionnaire model of the domain configuration. The tool
5. Questionnaire-Driven Staged Process Configuration

uses the c-mapping to configure those variation points in the process model that are affected by the domain configuration. Since it is possible to export a partial domain configuration from Quaestio, not all the variation points in the process model might be impacted by this domain configuration. In this case the result will be a partial process configuration. The c-mapping is also used to enforce the satisfiability of the process constraints, which are read from the c-mapping file. This might lead to further variation points being restricted. The output represents a (partially) configured C-EPC or C-YAWL process model where (some) variation points are tagged as configured and assigned to one of their variants.

The last tool of the configuration chain is the Process Individualizer. This tool implements the C-EPC individualization algorithm (cf. Definition 3.20 in Chapter 3) and the C-YAWL individualization algorithm [GAJL08], to generate an individualized process model from a (partially) configured process model. This is done by transforming each configured variation point to the variant it has been assigned to, and by removing those process fragments that are no longer required. The resulting model is guaranteed to be semantically correct provided the configurable process model is as such.

The toolset and the files for the application example can be downloaded from the Process Configuration web-site [LaR].

5.5 Summary and Discussion

This chapter concluded the discussion on domain-driven process configuration by showing how questionnaire models can be linked to configurable process models and used to drive their configuration. To achieve this link, we proposed to encode each variation point in a process model by a set of process facts, and to infer a set of process constraints over these facts, in order to rule out those process configurations that are not semantically correct. Meanwhile, we proposed to encode the domain choices that drive the configuration of a process model by domain facts, and to structure them in a questionnaire model. Similar to process constraints, domain constraints are used to rule out those domain configurations that contradict business requirements.

The link between domain constraints and process constraints was achieved by equating each process fact with a boolean formula over domain facts. By satisfying the unified set of constraints, the individualized process models are both domain-compliant and semantically correct.

We described a methodology for constructing c-mappings between questionnaire models and process models, and discussed desirable properties for such mappings. By means of an example taken from the screen business domain, we demonstrated the applicability of the approach to the C-EPC modeling notation. In separate work [LGDA08], not discussed in this thesis, we showed that the same approach can be applied to C-YAWL. We showed how a domain decision can impact the configuration of the process model in terms of variation points being directly or indirectly affected, and how process constraints can trigger the configuration of further variation points to ensure the model remains correct. Finally, we discussed the implementation of the approach in the Synergia toolset.

The main innovation of the proposal is that process configuration is not driven directly by a (configurable) process model, but rather by a model that captures the
variability of the application domain. This model masks the complexity of the underlying process model. In fact, the more variation points are added to a process model, the more likely the configuration complexity is going to increase and, thus, configuring the model without any means of abstraction may get close to unmanageable. Instead, by representing the domain variability in a separate model, we need not to capture the interdependencies of the domain in the configurable process model, as these are captured by the domain constraints and propagated to the process model via the c-mapping. Constraints over process facts have thus to deal only with the preservation of the model correctness. Depending on the notation adopted, these can be derived from the process model (as in C-EPCs), thus requiring minimal effort from the modeler. Moreover, defining domain constraints over business choices is more intuitive than doing the same over the nodes of a process model.

An assumption of the approach is that the questions of a questionnaire model have a finite or discretized domain of possible answers. This assumption is reasonable, given that the number of configuration alternatives in a process model (e.g. in a C-EPC) is also finite. This assumption allows the models to be efficiently analyzed to prevent the user from entering inconsistent answers that would lead to inconsistent process model configurations.

In the next chapter we expose a meta-model to extend process configuration to resources and business objects that participate in a process.
Chapter 6

Holistic Process Model Configuration

The third major shortcoming affecting existing approaches for configurable (reference) process modeling is the lack of suitable mechanisms for representing process variability beyond the control-flow perspective. Since business processes are meant to provide an integrated view over an organization, neglecting equally important aspects such as resources and business objects, leads to limited expressive power when it comes to configure these processes. Firstly, process modelers are unable to capture the unavailability of given entities in the process, e.g. a missing resource. Secondly, it is not possible for analysts to estimate the impact of such variations on the process model, e.g. estimating the impact of removing a business object on the remaining tasks. Yet, these situations occur quite often as processes vary from setting to setting.

Against this background, this chapter presents a meta-model for holistic process model configuration. The meta-model extends the expressive power of configurable process models by incorporating resources and business objects. Moreover, it provides mechanisms to represent a range of variations in the way resources and objects are associated with tasks. This meta-model allows us to explore interplays that occur among the control-flow, object flow and resource modeling perspectives during individualization. We propose to encode these interdependencies via domain constraints and meanwhile to capture structural requirements via process constraints. In this way we can use the questionnaire-based approach for staged process configuration presented in Chapters 4 and 5, to deal with the overhead induced by the integration of resources, objects and their variations to the modeling lifecycle.

We decided to define this meta-model as an extension of the EPC notation, namely Configurable integrated-EPC (C-iEPC). Four reasons underpin this choice. Firstly, EPCs are widely used by modelers and analysts for reference process modeling (cf. the SAP R/3 reference model). Hence, this can foster the adoption of the proposed extension. Secondly, eEPCs provide basic features for associating objects and resources to tasks, which we extend in this chapter. Thirdly, this choice allows us to build on top of the C-EPC notation, thus exploiting the available mechanisms for the configuration of the control-flow, and the results on correctness illustrated in Chapter 3. Finally, this choice facilitates the extension of the Synergia toolset to provide support for C-iEPCs, given that the toolset supports C-EPCs. Nonetheless, we present our extensions in an abstract and formal manner to make them applicable beyond the scope of EPCs.
From C-EPCs we also reuse the $\beta_T$ algorithm presented in Chapter 3, which we extend to deal with the individualization of configurable process models containing roles and objects. We prove that by construction this algorithm ensures the preservation of the syntactic correctness, as the original $\beta_T$ algorithm does with normal C-EPCs. The proposal has been applied to a comprehensive reference process model for audio post-production, that we built with domain experts from the Australian Film Television & Radio School (AFTRS) [AFTRS]. An extract of this model is used as a working example throughout this chapter.

The remainder of this chapter is structured as follows. Section 6.1 illustrates the meta-model that extends EPCs with resources and object flow modeling by means of the working example. Next, Section 6.2 explores the configuration of process models along the resource and object flow perspectives. Section 6.3 presents the formal model of C-iEPCs which leads to the definition of the individualization algorithm for deriving configured EPCs. Section 6.4 describes the tool support for C-iEPCs while Section 6.5 reports the case study conducted with the AFTRS. Finally, Section 6.6 concludes the chapter with a summary and a discussion on related work.

This chapter presents and expands upon work published in [LDH+08, LDH+07, MLH08, LHRS08].

### 6.1 Integrated Business Process Model

An integrated EPC (iEPC) extends an EPC by associating roles and objects to EPC functions. A role, depicted on a function’s left hand, captures a class of organizational resources that is able to perform that function. For example, the role Producer associated with function Picture editing in Figure 6.1, captures the set of all the persons with this role in a given screen project. At run-time a role is dynamically bound to one concrete resource (e.g., Producer will be bound to Steven Spielberg).

Resources, depicted on a function’s left hand side, can be human or non-human, such as an information system or a robot. The SAP system associated with function Process Invoice in Figure 6.1 is an example of non-human role. Therefore we can distinguish among manual functions (those performed by human roles), automated functions (those performed by non-human roles) and semi-automated functions (those performed by both).

An object, depicted on a function’s right hand side, captures an information artifact (e.g. file) or a physical artifact (e.g. paper document or production material) of an enterprise that is used (input object) or produced (output object) by a function. For example, Edit notes is an input object for Picture editing in Figure 6.1, while Credit note is an output object for Process invoice. Each object in the process model is statically bound to a concrete artifact. Therefore if two objects in a model bring the same label they are treated as being the same artifact.

#### 6.1.1 Working Example

The associations shown in Figure 6.1 are two examples of basic task-role and object-role associations. To illustrate more complex associations that can be captured in iEPC, we use the working example in Figure 6.2. This model is an exemplification of a reference
process model on audio editing for screen post-production, which was developed and validated in collaboration with subject-matter experts of the AFTRS. We chose this case study for the high level of creativity, and thus of variability, that characterizes the screen business. For example, the number of personnel involved depends greatly on the type of project. In an animation movie, dialogs are often recorded before the characters are finalized and scenes are complete. In a small budget feature film, the design and editing begins once the picture editing is complete, and is carried out by a single sound designer. On a high budget feature film, instead, the sound department can be made up of 30 or more roles. More information on the case study is reported in Section 6.5.

The first function of the model is Spotting session starting once the shooting has been completed. Roles and objects can be linked to functions either directly or via a connector. A connector allows one to specify a logical condition among a set of roles or objects. For example, the OR-join between Composer and Sound Designer indicates that at least one of these roles is required to perform this activity. Composer is needed if the project features music, Sound Designer is needed if the project features sound. Sound is a composition of dialogs, effects and/or atmospheres (atmos). Based on the screening of the Picture cut, Composer and Sound Designer hold a Spotting session to decide what music and sound should be added and at which point of time in the Picture cut. This information is stored in the cues (e.g. Music cues for music). Picture cut is thus an input object while music cues, dialog cues, etc., are output objects. These are connected via an OR-split which imposes that at least one set of cues be produced as a result of the Spotting session. This choice depends on the type of project. For example, a documentary would typically have no effects.

A spotting session may be supervised by at least two roles among Producer, Director and Assistant Director that have creative authority in the project. These roles are linked together by a range connector. This connector indicates the upper bound and lower bound for a number of elements (roles or objects) that are required. The parameter \( k \) for a range connector refers to the outdegree for a split or to the indegree for a join. In this case \( k = 3 \). A range connector subsumes the routing behavior of the common logical connectors of OR (equivalent to a range of \( 1 : k \)), AND (\( k : k \)) and XOR (\( 1 : 1 \)). Therefore we consider all connectors involving roles and objects as being range connectors, although we maintain the standard EPC notation for OR (\( \lor \)), XOR (\( \times \)) and AND (\( \land \)). All the associations we can capture through the range connector are illustrated in Figure 6.3 (the example of roles is shown). A function associated with more than one human role captures teamwork, i.e. a collaborative activity where each person contributes different skills. For example, function Spotting session captures the teamwork between the Composer, the Sound Designer and two roles among Producer, Director and Assistant Director.

Once the cues are ready the design of music and sound starts. In Music design, the
Composer records the project’s Music tracks (an output) following the Music cues and using the Picture cut as a reference (an AND-join connects these two inputs). A Temp music file may also be produced at this stage. This object is linked to the function via a dashed arc which indicates that an object or a role is optional, whereas a full arc indicates mandatoriness. The optionality of a group of roles/objects linked by a range connector is modeled by making the connector optional (see Figure 6.3). Sound design is usually more complex than Music design as it involves the recording of the Dialog, Effects and/or Atmos tracks, according to the respective cues on the Picture cut. The Editor or the Sound Designer are responsible for this task. Similarly to Music design,
a Temp sound file may also be produced.

Afterwards, the Composer and/or the Sound Designer provide the Director and usually the Producer with an update on the work-in-progress. Producer is an optional role. At least one mandatory role is to be assigned to each function to ensure its execution. Temp files may be used by the Composer and by the Sound Designer as a guide for the Progress update (the \textit{OR}-join between these two objects is thus optional). Generally, the result of this task is a set of notes describing the changes required; sometimes, however, the Composer or the Sound Designer may prefer not to take notes.
If changes are needed, the Music and Sound design can be repeated as specified by the loop in the model. In this case, the notes can be used as input to these tasks.

Upon completion of the design phase, the Mixer and the Composer mix the Music tracks into a Music premix if the project has music, while the Mixer and the Sound Designer mix the Sound tracks into a Sound premix if the project has sound. The Producer may supervise both mixings. In Picture editing, the Picture cut is edited by an Editor, while a Negcutter is required if the cut is on Film. A cross below an object, like the cross below ‘Picture cut’ in Figure 6.2, indicates that the object is consumed by the function and is no longer available afterwards.

The process ends with Final mixing, where the Mixer, who might be helped by the Sound Designer and/or the Composer, releases a Final mix using the available Premixes. A Deliverable may also be released by overlaying the premixes onto the Edited picture, should a demo of the video with the integrated audio be required.

Besides the process model, we use a hierarchy model to represent all the roles and objects referred to by the nodes of the process model. For example, in the editing process there are five nodes labelled ‘Producer’ and four labelled ‘Picture cut’. A hierarchy model also captures the specializations that can be associated with a role or object, by means of a specialization relation. Figure 6.4 shows the hierarchy models for the roles and objects of the editing process, where the specialization relation is depicted by an empty arrow linking a special role (object) to its generalization. Typically, for a role this relation represents a separation of tasks among its specializations (e.g., Executive Producer, Line Producer and Co-Producer share the Producer’s duties). For an object, it represents a set of subtypes (e.g., 16mm and 35mm are two types of Film). The specializations in the hierarchy models will be used later on for configuration.

In the next section we show how to represent variability on top of an iEPC model.

Figure 6.4: The role-hierarchy model and the object-hierarchy model for the post-production process model.
6.2 Exploring Integrated Process Configuration

A Configurable iEPC (C-iEPC) extends C-EPCs by widening the spectrum of variation points beyond functions and control-flow connectors to include roles, objects and range connectors. The configurable version of the reference process model for audio post-production is shown in Figure 6.5, where variation points are indicated with a thicker border as in the C-EPC notation.

A major novelty with respect to C-EPCs is that configurable roles, objects and range connectors have multiple configuration dimensions. This means they can take one configuration value for each dimension. Configurable roles and configurable objects...
have two dimensions: *optionality* and *specialization*. If a configurable role (object) is ‘optional’ (*OPT*), it can be restricted to ‘mandatory’ (*MND*), or switched *OFF* to be removed from the process. If it is ‘mandatory’ it can only be switched *OFF*. For example, if a project does not feature music, the participation of the Composer and the production of Music cues can be excluded from the Spotting session.

Configurable roles and objects for which there exists a specialization in the hierarchy model can be restricted to any of their specializations. As per the hierarchy model of Figure 6.4, Picture cut can be specialized to Tape if the project does not support an editing on Film. Also, the Producer associated with Progress update can be specialized to Line Producer and made mandatory, should the Director need creative support in this phase. The availability of a specialization for a role or object is depicted with a small pyramid on the node’s right-hand side.

*Configurable input objects* have a further configuration dimension, namely *usage*, such that those inputs that are ‘consumed’ (*CNS*) can be restricted to ‘used’ (*USE*). For instance, we can restrict Picture cut to *used* if its specialization is Tape. This is because a Picture cut is only physically destroyed if it is on Film.

*Configurable range connectors* have two configuration dimensions: *optionality* and *range restriction*. The same rules for roles and objects govern the possible changes of optionality values for range connectors. For example, the optional *OR*-join connecting the temp files in Progress update, can be made mandatory if the temp files are always used by this function. The range restriction allows one to restrict the routing behavior of the connector at configuration time, i.e. before the actual execution of the process. This is achieved by increasing the lower bound and/or decreasing the upper bound. Moreover, a choice can be made for a single node (role or object) to be associated with the function linked to the connector, effectively removing the connector altogether. This latter option is similar to configuring a control-flow connector to a sequence of nodes, and is allowed if the lower bound is 1 and the node is in the connector’s postset in case of a split, or in its preset in case of a join. For example, the configurable range connector $2 : k$ associated with Spotting session can be restricted to $3 : k$ – all the supervisors have to partake in the Spotting session – or to $2 : 2$ – exactly two of them have to partake – but not to a single role.

The configuration of range connectors is consistent with the configuration of control-flow connectors, since as mentioned before the range connector subsumes all connector types. In fact, a $1 : k$ range connector is equivalent to an *OR* and can thus be restricted to an *XOR* ($1 : 1$), to an *AND* ($k : k$) and to a single node, but also to any other reduced range (e.g. $2 : k$). A range $1 : 1$ can only be restricted to a single node while a range $k : k$ cannot be restricted.

### 6.2.1 Configuration Interplays

As argued in Chapter 3, domain requirements may prevent a configuration node from being freely set, or more generally, may impose a set of configurable nodes to take only certain combinations of values. For example, in Figure 6.5, the role Editor associated with Sound design cannot be specialized to Video Editor due to the capabilities required by the associated function. For the same reasons, if Editor is associated with Picture editing it cannot be specialized to Sound Editor.
An example of more intricate interdependency is that involving the role Negcutter. This role is required only if the project is edited and delivered on Film. Thus, if it is configured to MND, all the occurrences of objects Picture cut, Edited picture and Deliverable must be specialized to Film. Furthermore, in this case the Picture cut, which is an input object of Picture editing, must be restricted to CNS because it will be physically destroyed during the editing.

Similarly, switching OFF function Progress update implies the restriction of the subsequent XOR-split to the sequence starting with event Design finished. This is because at run-time the decision whether or not to repeat the design phase is determined by the outcome of Progress update.

Interdependencies may also involve range connectors. For instance, the two OR-joins for the roles and the input objects of Progress update must be configured the same way. The configuration of the first join allows the restriction of the run-time choice of which role is to partake in Progress update, while the configuration of the second join allows the restriction of which temp files have to be used. Although the second connector is optional (i.e. no temp file may be used), a configuration where, e.g., the first OR is restricted to AND and the second one is restricted to a mandatory XOR must be denied. This is because if temp files are available, these need to be linked to the roles Composer and Sound Designer that will actually use them. The Composer will use the Temp music files, while the Sound Designer will use the Temp sound files.

Besides domain requirements, structural and semantic requirements may restrict the configuration space as well to avoid the generation of incorrect individualizations. For example, we mentioned earlier that a function needs to have at least a mandatory role that can perform it. Thus, if a function is no longer associated with any role after configuration, it needs to be dropped from the model, because there are no resources that can execute it. This is the example of function Music design, which needs to be removed from the model if the role Composer is not available in the project. We discuss these correctness requirements in Section 6.3, where we provide the definition of correct iEPC.

We propose to use the questionnaire-based approach for staged process configuration exposed in Chapters 4 and 5, to configure C-iEPC process models. Accordingly, we can encode domain requirements and correctness requirements like the above via domain constraints and process constraints. In this way we can cope with the overhead in the modeling lifecycle induced by the introduction of new configurable elements.

A simple example of configured iEPC is provided in Figure 6.6. This model describes the audio editing process that was followed by Bill Bennett to direct the feature film “Kiss or Kill” [Ben97]. It is the result of configuring and individualizing the reference process model of Figure 6.5 for editing a feature movie without music on tape. Accordingly, functions Music design and Music premixing have been switched OFF. Progress update has been excluded and thus the subsequent XOR-split configured so as to remove the loopback sequence. The Editor in Picture editing has been specialized to Video Editor. Furthermore, since the editing is on Tape, all instances of Picture cut, Edited picture and Deliverable have been specialized to Tape, the Picture cut in input to Picture editing has been set to ‘used’ and Negcutter has been switched OFF.
Figure 6.6: The audio editing process model individualized for a project without music.

6.3 Correctness and Configuration of Integrated Process Models

In this section we formalize the C-iEPC meta-model to provide a precise characterization of a C-iEPC configuration and discuss the requirements that need to be fulfilled to yield a syntactically correct individualized iEPC. First, we define the notion of iEPC and syntactically correct iEPC. Next, we provide the definition of C-iEPC and configuration and show an extended version of the $\beta_1$ algorithm presented in Chapter 3, to configure C-iEPCs. Finally, we prove that the algorithm preserves the syntactic correctness of the models. For example, this algorithm is able to generate the model shown in Figure 6.6 from the model of Figure 6.2 given a configuration.

6.3.1 Integrated Business Process Model

In order to formally define the concepts of iEPC and correct iEPC, we first need to have a formal definition of role and object-hierarchies. A role hierarchy is essentially a set of roles with a specialization relation. Similarly, an object hierarchy is a set of objects with a specialization relation.

**Definition 6.1 (Role-hierarchy Model)** A role-hierarchy model is a tuple $Rh = (R, \rightarrow^R)$, where:

- $R$ is a finite, non-empty set of roles,
• $\leftarrowtriangle R \subseteq R \times R$ is the specialization relation on $R$ ($\leftarrowtriangle R$ is transitive, reflexive, acyclic\(^1\)).

**Definition 6.2 (Object-hierarchy Model)** An object-hierarchy model is a tuple $Oh = (O, \leftarrowtriangle O)$, where:

- $O$ is a finite, non-empty set of objects, i.e. physical or information artifacts,
- $\leftarrowtriangle O \subseteq O \times O$ is the specialization relation on $O$ ($\leftarrowtriangle O$ is transitive, reflexive, acyclic).

If $x_1 \leftarrowtriangle O x_2$, we say $x_1$ is a generalization of $x_2$ and $x_2$ is a specialization of $x_1$ ($x_1 \neq x_2$). For example, Dialog Editor is a specialization of Editor.

The definition of iEPC given below extends that of EPC from Chapter 3, which focuses on the control-flow only. Specifically, iEPCs add a precise representation of roles and objects participating in the process. These roles and objects stem from the hierarchy-models defined above. In an iEPC each node represents an instance of a function, role or object.

The range connector is modeled by a pair of natural numbers: lower bound ($n$) and upper bound ($m$). AND, OR and XOR correspond to a range connector respectively with $n = m = k$, with $n = 1, m = k$ and with $n = m = 1$. So we do not need to model the logical operators with separate connectors for roles and objects, although they can be graphically represented with the traditional EPC notation, as in Figure 6.2. For the sake of keeping the model consistent with previous EPC formalizations, the range connector is not allowed in the control-flow, although a minimal effort would be required to add this construct. The optionality of roles, objects and range connectors, shown in Figure 6.2 as a dashed arc that links a node with a function, is modeled in iEPC as an attribute of the node being optional. The consumption of input objects is modeled in the same way.

**Definition 6.3 (iEPC)** Let $F$ be a set of functions, $Rh = (R, \leftarrowtriangle R)$ be a role-hierarchy model and $Oh = (O, \leftarrowtriangle O)$ be an object-hierarchy model. An integrated EPC over $F, Rh, Oh$ is a tuple $\iota F, Rh, Oh = (E, F_N, R_N, O_N, nm, C, A, L)$, where:

- $E$ is a finite, non-empty set of events;
- $F_N$ is a finite, non-empty set of function nodes for the process;
- $R_N$ is a finite, non-empty set of role nodes for the process;
- $O_N$ is a finite set of object nodes for the process;
- $nm = nf \cup nr \cup no$, where:
  - $nf \in F_N \rightarrow F$ assigns each function node to a function;
  - $nr \in R_N \rightarrow R$ assigns each role node to a role;
  - $no \in O_N \rightarrow O$ assigns each object node to an object;
- $C = C_{C\text{r}} \cup C_{r} \cup C_{in} \cup C_{out}$ is a finite set of logical connectors, where:

\(^1\)no cycles of length greater than one.
6. Holistic Process Model Configuration

- \( C_{cf} \) is the set of control-flow connectors,
- \( C_r \) is the set of range connectors for role nodes (role connectors),
- \( C_{in} \) is the set of range connectors for input nodes (input connectors),
- \( C_{out} \) is the set of range connectors for output nodes (output connectors),

where \( C_{cf}, C_r, C_{in} \) and \( C_{out} \) are mutually disjoint;

- \( A = A_{cf} \cup A_r \cup A_{in} \cup A_{out} \) is a set of arcs, where:
  - \( A_{cf} \subseteq (E \times F_N) \cup (F_N \times E) \cup (E \times C_{cf}) \cup (C_{cf} \times E) \cup (F_N \times C_{cf}) \cup (C_{cf} \times F_N) \cup (C_{cf} \times C_{cf}) \) is the set of control-flow arcs,
  - \( A_r \subseteq (R_N \times F_N) \cup (R_N \times C_r) \cup (C_r \times F_N) \) is the set of role arcs,
  - \( A_{in} \subseteq (O_N \times F_N) \cup (O_N \times C_{in}) \cup (C_{in} \times F_N) \) is the set of input arcs,
  - \( A_{out} \subseteq (F_N \times O_N) \cup (F_N \times C_{out}) \cup (C_{out} \times O_N) \) is the set of output arcs,
  where \( A_r, A_{in} \) and \( A_{out} \) are intransitive relations;

- \( L = l^r \cup l^c \cup l^w \cup l^m \cup l^w_o \cup l^m_o \) is a set of label assignments, where:
  - \( l^r \in C_{cf} \to \{\text{AND, OR, XOR}\} \) specifies the type of control-flow connector,
  - \( l^c \in (C_r \cup C_{in} \cup C_{out}) \to \mathbb{N} \times (\mathbb{N} \cup \{k\}) \cup \{(k, k)\} \), specifies lower bound and upper bound of the range connector,
  - \( l^w \in (C_r \cup C_{in} \cup C_{out}) \to \{\text{MND, OPT}\} \) specifies if a role connector, an input connector or an output connector is mandatory or optional,
  - \( l^m \in R_N \to \{\text{MND, OPT}\} \) specifies if a role node is mandatory or optional,
  - \( l^w_o \in O_N \to \{\text{MND, OPT}\} \) specifies if an object node is mandatory or optional,
  - \( l^v_o \in O^*_N \to \{\text{USE, CNS}\} \) specifies if an input object node is used or consumed, where \( O^*_N = \text{dom}(A_{in}) \cap O_N \).

Given a connector \( c \), let \( l^c_r(c) = (n, m) \) for all \( c \in C \setminus C_{cf} \). Then we use \( \text{lbh}(c) = n \) and \( \text{upb}(c) = m \) to refer to lower bound and upper bound of \( c \). Moreover, if \( F, Rh \) and \( Oh \) are clear from the context, we drop the subscript from \( i\Upsilon \). Also, we call all the function nodes, role nodes and object nodes simply as functions, roles and objects, wherever this does not lead to confusion.

Before defining a syntactically correct iEPC, we introduce the following subsets of nodes, functions and predicates to allow a more concise characterization of iEPCs.

**Notation 6.4 (Auxiliary sets, functions and predicates)** Let \( F \) be a set of functions, \( Rh \) be a role-hierarchy model, \( Oh \) be an object-hierarchy model and \( i\Upsilon = (E, F_N, R_N, O_N, nm, C, A, L) \) be an iEPC. Then:

- \( N_{cf} = F \cup F_N \cup C_{cf}, \) as its set of control-flow nodes;
- \( N_r = F_N \cup R_N \cup C_r, \) as its set of role nodes;
- \( N_{in} = F_N \cup O_N \cup C_{in}, \) as its set of input nodes;
- \( N_{out} = F_N \cup O^*_N \cup C_{out}, \) as its set of output nodes, where \( O^*_N = \text{dom}(A_{out}) \cap O_N \).

\[ N = N_{CF} \cup N_r \cup N_{IN} \cup N_{OUT}, \text{ as its set of nodes;} \]

\[ \forall n \in N_r, n = \{ x \in N_\sigma \mid (x, n) \in A_\sigma \}, \text{ as the } \sigma\text{-preset of } n, \sigma \in \{ CF, R, IN, OUT \}; \]

\[ \forall n \in N_r, n = \{ x \in N_\sigma \mid (n, x) \in A_\sigma \}, \text{ as the } \sigma\text{-postset of } n, \sigma \in \{ CF, R, IN, OUT \}; \]

\[ E_s = \{ e \in E \mid |e^{CF}| = 0 \land |e^{CF}| = 1 \} \text{ as the set of start events;} \]

\[ E_e = \{ e \in E \mid |e^{CF}| = 1 \land |e^{CF}| = 0 \} \text{ as the set of end events;} \]

\[ C_{s, CF} = \{ c \in C_{CF} \mid |c^{CF}| = 1 \land |c^{CF}| > 1 \} \text{ as the set of control-flow split connectors;} \]

\[ C_{j, CF} = \{ c \in C_{CF} \mid |c^{CF}| > 1 \land |c^{CF}| = 1 \} \text{ as the set of control-flow join connectors;} \]

\[ \text{link}^\sigma(x, y) = \begin{cases} (y, x) & \text{if } \sigma = R, \text{ returns the role arc from } y \text{ to } x, \\ (y, x) & \text{if } \sigma = IN, \text{ returns the input arc from } y \text{ to } x, \\ (x, y) & \text{if } \sigma = OUT, \text{ returns the output arc from } x \text{ to } y; \end{cases} \]

\[ \text{degree}(x) = \begin{cases} |^R x| & \text{if } x \in C_r, \text{ returns the indegree of a role connector,} \\ |^IN x| & \text{if } x \in C_{IN}, \text{ returns the indegree of an input connector,} \\ |x^{OUT}| & \text{if } x \in C_{OUT}, \text{ returns the outdegree of an output connector;} \end{cases} \]

\[ \phi = \langle n_1, n_2, \ldots, n_k \rangle \text{ is a control-flow path such that } (n_i, n_{i+1}) \in A_{CF} \text{ for } 1 \leq i \leq k - 1. \text{ For short, we indicate that } \phi \text{ is a path from } n_1 \text{ to } n_k \text{ as } \phi : n_1 \mapsto n_k. \text{ Also,} \]

\[ \alpha(\phi) = \{ n_1, \ldots, n_k \} \text{ indicates the alphabet of } \phi. \]

It follows that for all \( f \in F_N \mid f^{R} = 0, \mid f^{IN} = 0 \text{ and } \mid^{OUT} f = 0; \text{ for all } r \in R_N \mid r = 0; \text{ for all } o \in O_N \mid o = 0 \text{ and } \mid^{OUT} o = 0. \]

We can now define a syntactically correct iEPC. This definition extends that of syntactically correct EPC (cf. Chapter 3) by adding additional requirements for roles, objects and range connectors. For example, it specifies that range connectors associated with roles and input objects must be of type join, while range connectors associated with output objects must be of type split. Also, it imposes that functions be associated at least with a mandatory role or mandatory role connector, and that roles and objects linked with range connectors be mandatory since the optionality of a group of roles/objects is modeled by making the connector optional.

Furthermore, this definition relaxes the constraint of single start and single end events and the strict alternation of events and functions defined for syntactically correct EPCs (i.e. trivial events can be omitted).

**Definition 6.5 (Syntactically Correct iEPC)** Let \( F \) be a set of functions, \( Rh \) be a role-hierarchy model, \( Oh \) be an object-hierarchy model and \( i\Upsilon_{F,Rh,Oh} = (E, F_N, R_N, O_N, nm, C, A, L) \) be an iEPC. \( i\Upsilon \) is syntactically correct if it fulfills the following requirements:

1. Every control-flow node is on a control-flow path from a start to an end event:
   \[ \forall n \in N_{CF} \exists e_s \in E_s, e_e \in E_e \mid (e_s, n) \in A^e_{CF} \land (n, e_e) \in A^e_{CF}. \]
2. There is at least one start event and one end event in \( i \mathcal{Y} \): \( |E_s| > 0 \) and \( |E_e| > 0 \).

3. Events have at most one incoming and one outgoing control-flow arc:
\[ \forall e \in E \ [ | e \overset{CF}{\bullet} | \leq 1 \land | e \overset{CF}{\bullet} | \leq 1]. \]

4. Functions have exactly one incoming and one outgoing control-flow arc:
\[ \forall f \in F \ [ | f \overset{CF}{\bullet} | = | f \overset{CF}{\bullet} | = 1]. \]

5. Control-flow connectors have one incoming and multiple outgoing arcs or vice versa:
\[ \forall c \in C_{CF} \ [ ((| c \overset{CF}{\bullet} | = 1 \land | c \overset{CF}{\bullet} | > 1) \lor (| c \overset{CF}{\bullet} | > 1 \land | c \overset{CF}{\bullet} | = 1)), \] (split, join),
Role connectors have multiple incoming arcs and exactly one outgoing arc:
\[ \forall r \in R \ [ | r \overset{IN}{\bullet} | > 1 \land | r \overset{IN}{\bullet} | = 1], \) (join),
Input connectors have multiple incoming arcs and exactly one outgoing arc:
\[ \forall o \in O \ [ | o \overset{IN}{\bullet} | > 1 \land | o \overset{IN}{\bullet} | = 1], \) (join),
Output connectors have exactly one incoming arc and multiple outgoing arcs:
\[ \forall c \in C_{OUT} \ [ | c \overset{OUT}{\bullet} | > 1 \land | c \overset{OUT}{\bullet} | = 1], \) (split).

6. Roles have exactly one outgoing arc: \( \forall r \in R \ [ | r \overset{OUT}{\bullet} | = 1]. \)

7. Objects have exactly one outgoing input arc or one incoming output arc:
\[ \forall o \in O \ [ | o \overset{IN}{\bullet} | = 1 \land | o \overset{OUT}{\bullet} | = 0) \lor (| o \overset{IN}{\bullet} | = 0 \land | o \overset{OUT}{\bullet} | = 1]. \]

8. Functions are linked to at least a mandatory role or a mandatory role connector:
\[ \forall f \in F \ [ \exists c \in C \ [ l_n^M(r) = MND] \lor \exists c \in C \ [ l_n^M(o) = MND)], \] it follows that \( | f \overset{IN}{\bullet} | > 0. \)

9. Roles and objects linked to connectors are mandatory:
\[ \forall r \in R \ [ r \in \text{dom}((R \times C) \cap A_R) \Rightarrow l^M_n(r) = MND], \]
\[ \forall o \in O \ [ o \in \text{dom}((O \times C) \cap A_O) \Rightarrow l^M_n(o) = MND], \]
\[ \forall o \in O \ [ o \in \text{dom}((C \times O) \cap A_O) \Rightarrow l^M_n(o) = MND]. \]

10. Upper bound and lower bound of range connectors are restricted as follows:
\[ \forall c \in C \cup C \cup C \ [ 1 \leq l_b(c) \leq u_b(c) \land (l_b(c) \leq d(c) \lor u_b(c) = k)], \] where \( n \leq m \) iff \( (n \leq m) \lor (m = k) \lor (n = m = k). \)

The post-production process model of Figure 6.2 is syntactically correct.

The behavior of an iEPC has to take into account the routing rules of the control-flow, the availability of the resources and the existence of the objects participating in the process. A state of the execution of an iEPC can be identified by a marking of tokens for the control-flow (cf. Chapter 3), plus a variable for each role indicating the availability of the respective resource, and a variable for each object indicating its existence.

A function is enabled and can fire if it receives control, if at least all its mandatory roles are available and all its mandatory input objects exist. The state of roles and objects is evaluated directly or via the respective range connectors. During a function’s execution, the associated roles become unavailable and once the execution is concluded, the output objects are created (i.e. they become existent), and those ones that are indicated as consumed, are destroyed. Initial process objects, i.e. those ones that are used by a function that follows a start event (e.g. the Picture cut), exist before the execution starts. A function does not wait for an optional role to become available.
However, if such a role is available before the function is executed, it is treated as a mandatory role.

The iEPC semantics is not formally addressed in this thesis. For a formal definition, we refer to separate work [MLH08].

6.3.2 Integrated Process Configuration

A C-iEPC is an extension of an iEPC where a subset of its functions, connectors, roles and objects is identified as configurable. As per a C-EPC, control-flow connectors of type AND are not configurable.

Definition 6.6 (Configurable iEPC) A configurable iϒ is a tuple $i\Gamma = (E, F_N, R_N, O_N, nm, C, A, L, F^c_N, R^c_N, O^c_N, C^c)$, where:

- $E, F_N, R_N, O_N, nm, C, A, L$ refer to the elements of a syntactically correct $i\Upsilon$,
- $F^c_N \subseteq F_N$ is the set of configurable functions,
- $R^c_N \subseteq R_N$ is the set of configurable roles,
- $O^c_N \subseteq O_N$ is the set of configurable objects,
- $C^c \subseteq (C \setminus C_{AND})$ is the set of configurable connectors, where $C_{AND}$ is the set of AND connectors.

All the auxiliary sets of Definition 6.4 are also defined for $i\Gamma$. For example, $N^c = F^c_N \cup R^c_N \cup O^c_N \cup C^c$.

We define a C-iEPC valuation as an assignment of values to each configurable node according to the node type. This valuation then constrained to achieve a configuration.

Definition 6.7 (C-iEPC Valuation) Let $i\Gamma = (E, F_N, R_N, O_N, nm, C, A, L, F^c_N, R^c_N, O^c_N, C^c)$ be a syntactically correct C-iEPC. Let also $M = \{MND, OPT, OFF\}$ be the set of optionality attributes, $U = \{USE, CNS\}$ the set of usage attributes, $CT = \{OR, XOR\}$ the set of configurable control-flow connector types and $CTS_{cf} = \{SEQ_n \mid n \in N_{cf}\}$ the set of sequence operators for the control-flow. A valuation of $i\Gamma$ is defined as $C_{i\Gamma} = (C_F, C_R, C_O, C_C)$, where:

- $C_F \in F^c_N \mapsto \{ON, OPT, OFF\}$;
- $C_R \in R^c_N \mapsto M \times R$, $(M$ is used for optionality and $R$ for role specialization$)$;
- $C_O = C_{IN} \cup C_{OUT}$, where:
  - $C_{IN} \in O^c_{IN} \mapsto M \times O \times U$, $(O$ is used for object specialization and $U$ for usage$)$;
  - $C_{OUT} \in O^c_{OUT} \mapsto M \times O$;
- $C_C = C_{CF} \cup C_{R} \cup C_{IN} \cup C_{OUT}$, where:
  - $C_{CF} \in C^c_{CF} \mapsto CT \cup CTS_{cf}$, $(CT$ is used for the connector’s type and $CTS_{CF}$ is used to configure the connector to a sequence of nodes$)$;
- \( \mathcal{C}_{C_R} \in C^C_R \rightarrow M \times ((N \times N) \cup R_N), (N \text{ and } N \text{ are used for lower bound increment and upper bound decrement, } R_N \text{ is used to configure a role connector to a single role});\)
- \( \mathcal{C}_{C_{IN}} \in C^C_{IN} \rightarrow M \times ((N \times N) \cup O^N_N), (O^N_N \text{ is used to configure an input connector to a single input object});\)
- \( \mathcal{C}_{C_{OUT}} \in C^C_{OUT} \rightarrow M \times ((N \times N) \cup O^N_OUT), (O^N_OUT \text{ is used to configure an output connector to a single output object}).\)

We define the following projections over the codomain of \( \mathcal{C}_{i\Gamma} \) to address each configuration value in a more compact way.

**Notation 6.8 (C-iEPC Projections)** Let \( i\Gamma = (E, F_N, R_N, O_N, nm, C, A, L, F^C, R^C_N, O^C_N, C^C) \) be a syntactically correct C-iEPC and \( \mathcal{C}_{i\Gamma} = (\mathcal{C}_F, \mathcal{C}_R, \mathcal{C}_O, \mathcal{C}_C) \) be a valuation of \( i\Gamma \):

- let \( x \in R^C_N \cup O^{OUT^C}_N \) \( , \sigma \in \{R, OUT\} \) and \( \mathcal{C}_\sigma(x) = (m, s), \) then \( \pi^M(x) = m \) and \( \pi^S(x) = s; \)
- let \( x \in O^{IN^C}_N \) and \( \mathcal{C}_{IN}(x) = (m, s, u), \) then \( \pi^M(x) = m, \pi^S(x) = s \) and \( \pi^U(x) = u; \)
- let \( x \in C^C_R \cup C^C_{IN} \cup C^C_{OUT} \) \( \text{and } \sigma \in \{R, IN, OUT\} \) \( , \text{then if } \mathcal{C}_{C_\sigma}(x) = (m, (p, q)), \) then \( \pi^M(x) = m, \pi^I(x) = p \) and \( \pi^d(x) = q, \) otherwise if \( \mathcal{C}_{C_\sigma}(x) = (m, y), \) then \( \pi^M(x) = m \) and \( \pi^N(x) = y. \)

The restrictions on the values each configurable node can take are captured by the following partial orders. These are used to constrain the possible values a C-iEPC valuation can take in order to achieve a valid valuation, i.e. a configuration of C-iEPC. For example, the partial order on the optionality dimension prevents a ‘mandatory’ node from being configured to ‘optional’ while it allows the contrary.

**Definition 6.9 (C-iEPC Partial Orders)** Let \( M, U, CT \) and \( CTS_{cf} \) be as in Definition 6.7. The partial orders for the configuration of a C-iEPC are defined as follows:

- \( \preceq^M = \{MND, OFF\} \times \{MND\} \cup M \times \{OPT\} \) (on optionality),
- \( \preceq^U = \{(n,n) \mid n \in U\} \cup \{(USE, CNS)\} \) (on usage),
- \( \preceq^{CF} = \{(n,n) \mid n \in CT\} \cup \{XOR, AND\} \times \{OR\} \cup CTS_{cf} \times \{XOR, OR\} \) (on the type of control-flow connectors).

With these elements we are now ready to define the notion of C-iEPC configuration, which formalizes the concepts presented in Section 6.2.

**Definition 6.10 (C-iEPC Configuration)** Let \( i\Gamma = (E, F_N, R_N, O_N, nm, C, A, L, F^C, R^C_N, O^C_N, C^C) \) be a syntactically correct C-iEPC and \( \mathcal{C}_{i\Gamma} \) be a valuation of \( i\Gamma \). Then \( \mathcal{C}_{i\Gamma} \) is a configuration of \( i\Gamma \) iff it fulfills the following requirements for any configurable node:

1. Roles and objects can be restricted to MND or OFF if they are OPT, or to OFF if they are MND (\( \sigma \in \{R, O\} \)):
\begin{align*}
\forall x \in R^C_N \cup O^C_N \quad |\mathcal{C}_{i\Gamma}(x)| \leq M \quad l^M_i(x).
\end{align*}
2. Roles and objects can be restricted to any of their specialization:
   \[ \forall x \in R^G \cup O^G \ [\pi^G(x) \leftarrow nm(x)]. \]

3. Input objects that are CNS can be restricted to USE:
   \[ \forall x \in O_{\text{IN}}^G \ [\pi^G(x) \preceq^u l^G_0(x)]. \]

4. Control-flow OR connectors can be restricted to XOR, AND or to SEQ^m; control-flow XOR connectors can be restricted to SEQ^m:
   \[ \forall x \in C_{\text{CF}}^G \cdot n \in N_{\text{CF}} \cdot (C_{\text{CF}}^G(x) \preceq^{cr} l^G_{n}(x) \land (C_{\text{CF}}^G(x) = \text{SEQ}^m_n \Rightarrow \left( (x \in C_{\text{CF}}^G \land (x,n) \in A_{\text{CF}} \right) \lor (x \in C_{\text{CF}}^G \land (n,x) \in A_{\text{CF}}))) \] (the sequence must be in the connector’s postset in case of split or in the connector’s preset in case of join).

5. Range connectors can be restricted to MND or OFF if they are OPT, or to OFF if they are MND:
   \[ \forall x \in R^G \cup O^G \cup C_{\text{OUT}}^{\text{m}} \ [\pi^G(x) \preceq^M l^G_0(x)]. \]

6. Range connectors can be restricted to a smaller range or to a single node (i.e. a role or object):
   - **Range**: \( \forall x \in C^G \cup C_{\text{IN}}^G \cup C_{\text{OUT}}^G \):
     \[
     \pi^G(x) = \pi^d(x) = 0, \text{ if } lwb(x) = upb(x) = k \ (\text{the AND case cannot be restricted}),
     \]
     \[
     lwb(x) + \pi^G(x) \leq \begin{cases} upb(x) - \pi^d(x), & \text{if } upb(x) \in \mathbb{N}, \\ degree(x) - \pi^d(x), & \text{if } lwb(x) \in \mathbb{N} \text{ and } upb(x) = k; \end{cases}
     \]
   - **Node** (\( \sigma \in \{R, IN, OUT\} \)):
     \[ \forall x \in C^G \cup C_{\text{IN}}^G \cup C_{\text{OUT}}^G \ [\pi^G(x) = y \Rightarrow (\text{link}^G(x,y) \land lwb(x) = 1)] \] (the node must be in the connector’s postset in case of split or in the connector’s preset in case of join, and the lower bound must be 1).

The individualization algorithm \( \beta_\Gamma \) is an extension of the \( \beta_\Gamma \) algorithm (cf. Chapter 3) to deal with C-iEPCs. It applies a (partial) configuration to a syntactically correct C-iEPC, to generate a syntactically correct (C-)iEPC. Each step of the algorithm operates over a different type of element in a C-iEPC. The order of these steps has been chosen in such a way that no unnecessary operations are applied. For example, the control-flow connectors are configured first, as this operation may lead to skipping certain paths of the process model including connectors, events and functions. Then all the roles, objects and range connectors that are associated with functions no longer existing are removed as well. Finally, the remaining roles, objects and range connectors are configured.

The algorithm also removes all functions not associated with a mandatory role or mandatory role connector, and aligns the range of connectors with possible changes in degree resulting from switching some role or object OFF. Figure 6.7 illustrates this situation for the join 2 : k associated with Spotting session in Figure 6.5. Although we configure this connector to 3 : k (\( \pi^G(c) = 1, \pi^d(c) = 0 \)), if at least one of the two configurable roles linked by it is switched OFF, its range needs to be restricted. In order to guarantee syntactic correctness, this rule also applies to non-configurable range connectors. For example, the AND join associated with Sound premixing must be dropped if Sound Designer is switched OFF.

Since we relaxed the strict alternation of functions and events in Definition 6.5, when a function is switched OFF the algorithm removes the function altogether instead...
of replacing it with a silent function. Similarly, when a function is set to OPT, the algorithm bypasses it by adding only an XOR-split, an XOR-join and an arc in-between.

Before presenting the algorithm, we define some operators which allow us to perform the model transformations that are needed to configure a C-iEPC, e.g. removing a function or removing all the roles and objects associated with a function.

**Notation 6.11 (iEPC Operators)** Let $i^T = (E, F_N, R_N, O_N, nm, C, A, L, C^c, R^c_N, O^c_N, C^c)$ be a syntactically correct C-iEPC. We define the following operators:

- **Remove-Operator** $\delta$ to delete the nodes in set $X$ and their arcs:

  $\delta(i^T, X)$ is an iEPC such for all sets $Y^\delta \in \delta(i^T, X)$, $Y^\delta = Y \setminus X$, except $A^\delta = A \setminus \{(x, y) \in A \mid x \in X \lor y \in X\}$, and for all functions $\psi^\delta \in \delta(i^T, X)$, $\psi^\delta = \psi|_{\text{dom}(\psi)}$.

- **Replace-Operator** $\varrho$ to delete the nodes in set $X$ and connect their preset and postset elements:

  $\varrho(i^T, X) = \delta(i^T, X)$ except $A^\varrho = A^\delta \cup \{a, b \mid \exists x \in X \mid a \in \text{CF} x \land b \in \text{CF} x\}$.

- **Bypass-Operator** $\varphi$ to insert two XOR connectors to bypass optional functions:

  Let $C_X = \{x^b \mid x \in X\} \cup \{x^a \mid x \in X\}$ be the set of new control-flow connectors that will be placed before and after the optional function. Then $\varphi(i^T, X) = i^T$ except $C^\varphi = \varphi|_{\text{CF}} \cup C_{\text{in}} \cup C_{\text{out}}$ where $C^\varphi = C^\varphi \cup C_X$, and $A^\varphi = (A \setminus (N \times X \cup X \times N)) \cup \{(x^b, x) \mid x \in X\} \cup \{(x^a, b) \mid x \in X\} \cup \{(y, x^b) \mid (y, x) \in A\} \cup \{(x^a, y) \mid (x, y) \in A\}$.

- **Events-Operator** $\Lambda_{\text{e}}$ to remove sequences of events and add new arcs:

  Let $X = \{e \in E \mid \text{CF} e \cap E = \emptyset\}$ be the set of events to be deleted and $A_X = \{(e, x) \in (E \setminus X) \times (N_{\text{CF}} \setminus X) \mid \exists e' \in X \exists \phi \in \text{CF}(X \cup \{e\}) \mid \phi(e', x) \in A \land \phi : e \leftarrow e'\}$ be the set of arcs to be added. Then $\Lambda_{\text{e}}(i^T) = \delta(i^T, X)$ except $A^\Lambda_{\text{e}} = A^\delta \cup A_X$.

- **Connectors-Operator** $\Lambda_{\text{c}}$ to remove sequences of control-flow connectors and add new arcs:

  Let $X = \{c \in C_{\text{CF}} \mid |\text{CF} c| = |c^{\text{CF}}| = 1\}$ be the set of control-flow connectors to be removed, and let $A_X = \{(x, y) \in (N_{\text{CF}} \setminus X) \times (N_{\text{CF}} \setminus X) \mid \exists \phi \in \text{CF}(X \cup \{x, y\}) \mid \phi : x \leftarrow y\}$ be the set of arcs to be added. Then $\Lambda_{\text{c}}(i^T) = \delta(i^T, X)$ except $A^\Lambda_{\text{c}} = A^\delta \cup A_X$. 

---

![Image](image.png)
• Functions-Operator $\Lambda_{\rho}$ to add an event for any two consecutive functions:
Let $E_X = \{(f,g) \mid (f,g) \in A \cap (F_X \times F_N)\}$ be the set of events to be added and $A_X = \{(f,e_{f,g}) \in F_X \times E_X \cup \{(f,g) \in E_X \times F_X\}$ be the set of arcs to be added. Then $\Lambda_{\rho}(i\Upsilon) = i\Upsilon$ except $E^\Lambda_{\rho} = E \cup E_X$ and $A^\Lambda_{\rho} = (A \cup A_X) \setminus (F_X \times F_N)$.

• Corona-Operator $\Omega$ to identify roles, objects and range connectors of functions:
$\Omega(i\Upsilon, X) = ((X \cup X) \cup *(X \cup X) \cup ((C \setminus C^c) \cup R \cup O)$.

Definition 6.12 (Configured C-iEPC) Let $i\Gamma = (E, F_N, R_N, O_N, nm, C, A, L, F^c_N, R^c_N, O^c_N, C^c)$ be a syntactically correct C-iEPC and $C_{i\Gamma}$ be one of its configurations.

$\beta_{A}(i\Gamma, C_{i\Gamma})$ defines a (C-)iEPC $i\Psi$ obtained as follows:

1. Populate all sets of $i\Psi_1$ with the respective sets of $i\Gamma$.

2. Apply control-flow connector configuration and remove arcs not involving $SEQ_n$:
   $i\Psi_2 = i\Psi_1$, except
   $I^c_{i\Psi_2} = I^c_{i\Psi_1} \cup \{(c, C^c_{\phi_{CF}}(c)) \mid c \in C^c_{\phi_{CF}} \land C_{\phi_{CF}}(c) \in CT\}$
   $A_2 = A_1 \setminus \{(c, n) \in C_{\phi_{CF}} \times C_{\phi_{CF}} \mid \exists n' \in C_{\phi_{CF}}, \pi_{CT} \neq n' \land C_{\phi_{CF}}(c) = SEQ_{\phi_{CT}}\} \cup \{(n, c) \in C_{\phi_{CF}} \mid C_{\phi_{CF}}(c) = SEQ_{\phi_{CT}}\}$.

3. Remove nodes not on some path from an original start event to an original end event:
   Let $e_s \in E_s, e_e \in E_e$ and let $N_X = \{n \in N_2 \mid \exists \phi \in N_{C^c_{\phi_{CF}}} \phi_e \leftarrow e_e [n \in \alpha(\phi)]\}$. Then $i\Psi_3 = \delta(i\Psi_2, N_X \cup \Omega(i\Psi_2, N_X))$.

4. Replace functions switched OFF with an arc and remove their roles, objects and range connectors:
   Let $F_X = \{f \in F_N \mid C_{\phi_{DF}}(f) = OFF\}$. Then $i\Psi_4 = \delta(\Lambda_{\rho}(i\Psi_3, F_X), \Omega(i\Psi_3, F_X))$.

5. Remove range connectors switched OFF, together with their roles and objects:
   Let $C_X = \{c \in C_4 \setminus C_{\phi_{DF}} \mid \pi^M(c) = OFF\}$ and $RO_X = \{(R_{N_4}) \cup (C_X \cup C_X) \cap (\bullet C_X \cup X)\}$. Then $i\Psi_5 = \delta(i\Psi_4, C_X \cup RO_X)$.

6. Remove roles and objects switched OFF:
   $i\Psi_6 = \delta(i\Psi_5, \{ro \in R_{N,5} \cup O_{N,5} \mid \pi^M(ro) = OFF\})$.

7. Remove range connectors no longer linked to roles and objects:
   $i\Psi_7 = \delta(i\Psi_6, \{c \in C_6 \setminus C_{\phi_{DF}} \mid degree^c\phi_e(c) = \emptyset\})$.

8. Replace all range connectors with a degree of one with arcs:
   $i\Psi_8 = \theta(i\Psi_7, \{c \in C_7 \setminus C_{\phi_{DF}} \mid degree^c\phi_e(c) = \emptyset\})$.

9. Increment lower bound and decrement upper bound of configured range connectors:
   $i\Psi_9 = i\Psi_8$ except
   $I^c_{i\Psi_9} = I^c_{i\Psi_8} \cup \{(c, (\uparrow_{b}b_{s_8}(c) + \pi^c(c), up_{b_8}(c) - \pi^d(c))) \mid c \in C_{\pi^c(c)} \setminus C_{\phi_{DF}} \land \pi^d(c) \neq k\} \cup \{(c, (\uparrow_{b}b_{s_8}(c) + \pi^c(c), degree(c) - \pi^d(c))) \mid x \in C_{\pi^c(c)} \setminus C_{\phi_{DF}} \land \uparrow_{b}b_{s_8} \neq k\}$.
10. **Align lower and upper bound of range connectors with potential change in degree:**
   \[ i\Psi_{10} = i\Psi_{9} \text{ except} \]
   \[ l_{\text{C}}^{\text{ lower}} = l_{\text{C}}^{\text{ upper}} \cup \{(c, (\text{degree}_g(c), \text{upb}_g(c))) \mid c \in C_9 \land C_{\text{CF}} \land \text{lwb}_g(c) > \text{degree}_g(c) \land \text{upb}_g(c) \leq \text{degree}_g(c) \lor \text{upb}_g(c) = k\} \]
   \[ l_{\text{C}}^{\text{ upper}} = l_{\text{C}}^{\text{ lower}} \cup \{(c, (\text{degree}_g(c), \text{degree}_g(c))) \mid c \in C_9 \land C_{\text{CF}} \land \text{lwb}_g(c) \leq \text{degree}_g(c) \land \text{upb}_g(c) > \text{degree}_g(c) \land \text{upb}_g(c) \neq k\} \]

11. **Apply configuration to roles, objects and range connectors not switched OFF:**
   Let \( \sigma \in \{C, R, O\} \), then \( i\Psi_{11} = i\Psi_{10} \) except
   \[ l_{\text{C}}^{\text{ lower}} = l_{\text{C}}^{\text{ lower}} \cup \{(x, \pi^{\text{ lower}}(x)) \mid x \in N_{10} \land \text{dom}(C_{\sigma}) \cup \text{dom}(C_{\sigma}) \cup \text{dom}(C_{\sigma}) \}\]
   \[ l_{\text{C}}^{\text{ upper}} = l_{\text{C}}^{\text{ upper}} \cup \{(x, \pi^{\text{ upper}}(x)) \mid x \in N_{10} \land \text{dom}(C_{\sigma}) \}\]
   \[ nm_{11} = nm_{10} \cup \{(r, \pi^{\text{ lower}}(r)) \mid r \in N_{10} \land \text{dom}(C_{\sigma}) \cup \text{dom}(C_{\sigma}) \}\]

12. **Remove functions without mandatory role assignment:**
   Let \( F_{X} = \{ f \in F_{X,11} \mid \exists R \in R \land l_{\text{C}}^{\text{ lower}}(f) = \text{MND} \land \exists \hat{R} \in \hat{R} \land l_{\text{C}}^{\text{ lower}}(c) = \text{MND} \}\). Then
   \[ i\Psi_{12} = \delta(\Lambda_{\text{F}}(\rho(i\Psi_{11}, F_{X})), \Omega(i\Psi_{11}, F_{X})) \]

13. **Replace one-input-one-output connectors with arcs:**
   \[ i\Psi_{13} = \Lambda_{\text{F}}(\Lambda_{\text{C}}(i\Psi_{12})) \]

14. **Insert connectors to bypass optional functions:**
   \[ \beta_{\text{I}}(i\Gamma, C_{i\Gamma}) = i\Psi_{14} = \varphi(i\Psi_{13}, \{ f \in F_{13} \mid C_{F}(f) = \text{OPT} \}) \]

As per the \( \beta_{\text{I}} \) algorithm, \( \beta_{\text{I}} \) may return an empty net if after step 2 there is no longer a path from an original start event to an original end event. We address this issue by enforcing the satisfiability of a set of process constraints that we infer from the control-flow of an iEPC by using the technique exposed in Chapter 3 (cf. Section 3.6.2). In fact the same considerations that we did for C-EPCs hold for the control-flow of a C-iEPC.

We do not need to generate extra constraints for configurable roles, objects and range connectors to ensure the syntactic correctness of an iEPC. This is because the only structural issues that might arise from the configuration of these nodes, i.e. a function with no mandatory role associations or a range connector with an inconsistent range, do not require any decision from the user to be addressed. In fact, we give priority to the configuration of roles over the configuration of functions. Likewise, we give priority to the configuration of roles and objects over the configuration of range connectors. Therefore we use \( \beta_{\text{I}} \) to automatically resolve these issues once we have ascertained that the control-flow configuration satisfies the process constraints.

The following theorem states that the resulting (C-)iEPC \( \beta_{\text{I}}(i\Gamma, C_{i\Gamma}) \) is syntactically correct.

**Theorem 6.13** Let \( i\Gamma = (E, F_{\text{E}}, R_{\text{E}}, O_{\text{E}}, nm, C, A, L, F_{\text{C}}, R_{\text{C}}, O_{\text{C}}, C_{\text{C}}) \) be a syntactically correct C-iEPC and \( C_{\text{I}} \) be one of its configurations. Then \( \beta_{\text{I}}(i\Gamma, C_{\text{I}}) \) is a syntactically correct (C-)EPC.

**Proof** We show that the properties of Definition 6.5 hold for \( \beta_{\text{I}}(i\Gamma, C_{i\Gamma}) \) by discussing the different steps. We index intermediate syntax issues by \( I_{s,r} \) where \( r \) refers to a syntactical requirement and \( s \) to the step where it occurs.
1. The definition of $\Gamma$ (Definition 6.6) implies that $i\mathcal{Y}_1$ is structurally correct.

2. Changing connector labels according to configuration values does not affect syntactic correctness. Removing non-SEQ$_n$ arcs potentially results in nodes that are no more on a path from an original start event to an original end event giving rise to issues ($I_{2.1}$). Furthermore, this step may lead to connectors with one input and one output arc ($I_{2.5}$).

3. This step resolves $I_{2.1}$ by removing all nodes that are not on a path from a start to an end event. Furthermore, by also removing $\Omega(i\mathcal{Y}_2, N)$ all requirements related to roles and objects are fulfilled. Still, connectors may loose arcs and this may lead to connectors with one input and one output ($I_{2.5}$).

4. Since deleting the $\Omega(i\mathcal{Y}_3, F)$ nodes of all skipped functions and replacing them with an arc does not affect syntactic correctness, $i\mathcal{Y}_4$ inherits issues $I_{2.5}$ and $I_{3.5}$ from $i\mathcal{Y}_3$. Possible event sequences are merged by the $\Lambda_e$ operator. Furthermore, this step may add issues with connector cardinality if alternative branches are merged due to skipping functions ($I_{4.5}$).

5. Removing skipped range connectors and their role and object nodes may create problems with the requirement that functions should have at least a mandatory role ($I_{5.8}$).

6. Removing role and object nodes that are switched OFF can potentially create syntactic issues regarding range connector cardinality ($I_{6.5}$), mandatory roles of functions ($I_{6.8}$), and range connector bounds ($I_{6.10}$). Furthermore, $i\mathcal{Y}_6$ inherits issue $I_{2.5} - I_{4.5}$ and $I_{5.8}$.

7. This step resolves those cases related to $I_{6.5}$ where the degree of the range connector is 0, but still there may be range connectors with a degree of 1.

8. This step resolves $I_{6.5}$. The resulting net $i\mathcal{Y}_8$ inherits issues $I_{2.5} - I_{4.5}$, $I_{5.8}$, $I_{6.8}$ and $I_{6.10}$.

9. This step may resolve misalignment issues with upper bounds if, after the decrement, the new upper bound is equal to or below the degree of a range connector. Still, misalignments of lower bounds cannot be resolved by this step. Therefore, the resulting net $i\mathcal{Y}_9$ inherits issues $I_{2.5} - I_{4.5}$, $I_{5.8}$, $I_{6.8}$, and $I_{6.10}$.

10. Since the bounds are aligned with a potential change in degree, issue $I_{6.10}$ is resolved. $i\mathcal{Y}_{10}$ inherits issues $I_{2.5} - I_{4.5}$, $I_{5.8}$ and $I_{6.8}$.

11. This step does not affect syntactic correctness, since role and object nodes become mandatory which may resolve some issues with requirement 8.

12. This step resolves $I_{5.8}$ and $I_{6.8}$, but it may result in control-flow connectors with one input and one output, if alternative branches are merged due to functions switched OFF ($I_{12.5}$). Accordingly, $i\mathcal{Y}_{12}$ keeps issues $I_{2.5} - I_{4.5}$ and adds $I_{12.5}$.

13. This step resolves the remaining issues $I_{2.5} - I_{4.5}$ and $I_{12.5}$. Potentially created sequences of events are merged using $\Lambda_e$, and sequential functions are separated by a new event by $\Lambda_r$. Therefore $i\mathcal{Y}_{13}$ is syntactically correct.
14. This step does not bring new issues. Hence the algorithm returns a syntactically correct \((C-)iEPC\) 
\[ \Gamma_{14} = \beta_\Gamma(i\Gamma, C\Gamma) \].

### 6.4 Tool Support

We enhanced the Synergia toolset to provide tool support for the design and questionnaire-driven configuration of C-iEPC process models. Specifically, we extended the EPML 1.1 specification to provide an XML serialization for roles, objects, range connectors and their variants, as well as for hierarchy models. In this new version – EPML 2.0 [MNL] – we also fixed several bugs from the original specification and provided a more concise serialization of some elements.

We also extended the C-EPC Designer plugin for Eclipse to support EPML 2.0 so that users can graphically create and edit C-iEPC models. A screenshot of the tool showing the configurable model for audio post-production is depicted in Figure 6.8. The C-iEPC Designer offers a Properties view to specify the attributes of a node, such as the range of a connector or the optionality of a role. Furthermore, we can use this view to configure a configurable node by assigning a value for each configurable dimension of its. For example, Figure 6.8 shows the Properties view for the configurable range split associated with the output objects of function Spotting session, where the range has been restricted to the single node Music cues. The tool enforces the configuration requirements specified in Definition 6.10. So, for example, it is not possible to restrict a mandatory role to optional or to restrict a range connector to a node which is not in its preset of postset. Accordingly, the range split in Figure 6.8 can only be restricted to one of Music cues, Dialog cues, FX cues and Atmos cues, as shown by the drop-down menu in the properties view (property ‘Goto’).

In order to use the questionnaire-based approach for the configuration of C-iEPCs, we also extended the Mapper tool, the Process Configurator and the Process Individualizer tools. The Mapper was enhanced to import C-iEPC models defined in EPML 2.0. Upon import, the tool derives process facts from the variation points of the C-iEPC model, so that these can be mapped to domain facts from a questionnaire model. For example, an optional role would yield three process facts (\(OPT, MND\) and \(OFF\)), while an optional range connector would also yield one process fact for each range restriction that is allowed. The Mapper also infers a set of process constraints over the process facts extracted from the control-flow of the C-iEPC, which are used to guarantee the semantic correctness of the control-flow during configuration.

The Process Configurator was extended to be able to configure a C-iEPC model (in EPML) according to the answers of a questionnaire. Finally, the \(\beta_\Gamma\) algorithm was implemented in the Process Individualizer as an extension of the \(\beta_\Gamma\) algorithm, to commit the required model transformations and generate a syntactically correct individualized net.

The next section describes the case study conducted with the AFTRS, in which we applied the C-iEPC notation and the questionnaire-based approach to capture variability in process models for the film industry.
The working example used in this chapter is an exemplification of a reference process model for post-production that we created in collaboration with members of the AFTRS, over a period of nine months (the complete case study is reported in [LHR08]).

The AFTRS is an Australian training and research facility for Graduate Diploma, Masters courses and short courses in Film and TV production. This school has engaged, with other stakeholders, in an initiative to capture business process models in the film industry. However, it was quickly noted that process models in this industry have a high degree of variability. Basically, each production project works different from the others.

In this case study, we focused specifically on process models for the post-production phase. Post-production starts after the shooting phase and deals with the design and edit of the picture, music and sound of a screen project. Creativity is a distinguishing feature of this domain: a single decision made by the director, such as that of not having any music, can radically change the whole post-production. This necessarily leads to a great deal of variability: for this reason, the domain in question was deemed suitable to evaluate our framework.

A number of configurable process models were defined to describe the overall post-
production phase, which comprises picture and audio post-production, and its variations. In this phase, we received input from a producer and two sound editors. Given the involvement of the sound editors, we were also able to capture detailed information regarding roles and business objects for the audio post-production process. For this reason we used the C-iEPC notation to model this process, while we used the C-EPC notation to model the process for picture post-production. An extract of the latter model was presented in Chapters 2 and 5. The complete reference process model consists of 792 process elements (spread across different diagrams), of which 183 are variation points (the 23% of the total), each allowing a number of process variants for a total of around 310,000 valid individualizations.

Having defined the process model for post-production, we identified a set of domain facts to capture the choices that need to be taken to configure the reference model, and we grouped them into suitable questions. Firstly, we defined one fact for each factor that yields a high number of process variations. Such factors, like the budget level, correspond to domain decisions which are usually not captured in the process model. Secondly, we encoded each fine-grained decision with one fact. Such decisions have little or no impact on the rest of the system. For example, the type of editing suite only affects the medium format, while both the type of suite and the format are determined by the available budget. Thirdly, we defined a system of constraints to encode the interplay among these facts, and we used the Questionnaire Designer to check the satisfiability. With the help of the tool, we also realized that some fine-grained facts were redundant, as the variants they captured could be determined by the configuration of other facts.

Order dependencies were set in a way to pose the most discriminating questions first, and then fine-tuned according to the indications of the domain experts, to better respond to their needs. We added contextual information to the majority of questions and facts in the form of textual guidelines to aid users during the configuration process. These guidelines were derived from the constraints and enriched by information taken from the literature. The complete questionnaire model consists of three sub-questionnaires (picture editing, sound editing and screen composition), plus one introductory questionnaire which links them together for a total of 53 questions and 162 domain facts. An excerpt of the questionnaire model for picture editing was used in Chapter 5 to illustrate the mapping with configurable process models.

The questionnaire was used to configure the reference process model to the requirements of student projects at the AFTRS. This showed that depending on the context, a customized process model can be generated by domain experts in a straightforward manner. The generated models can then be used by the members of the school to guide the planning and the actual execution of a screen project. Also, they can find employment in the learning environment to teach students who aspire to become editors, sound designers and screen composers about the stages of the post-production process. Furthermore, they can be useful for producing and directing students to clearly understand the relations among the various drivers behind post-production, such as budget and schedule. For these reasons, the models produced during this case study and the Synergia toolset are planned to be introduced in the AFTRS syllabus of production and editing courses in 2009.

In conclusion, this experience demonstrated that the C-iEPC notation is able to capture complex variability requirements in a process involving roles, business objects and their associations with tasks. It also demonstrated that questionnaire models are
able to cope with practical variability scenarios involving numerous facts, dependencies and constraints, and that, at least in some domains, the restriction to boolean variables is not a major impediment. In this scenario, it was not necessary to introduce questions with non-enumerated domains.

6.6 Summary and Discussion

This chapter addressed a major shortcoming in existing configurable process modeling notations: their lack of support for the object and resource perspectives. In doing so, we presented a meta-model for capturing advanced role-task and object-task associations, that while embodied in the EPC notation, was defined in an abstract and formal manner to be transposed to other notations. This meta-model extends the expressive power of configurable process models by providing mechanisms to represent a range of variations in the way resources and objects are associated with tasks. The study highlighted the intricacies that configurable process modeling across multiple perspectives brings by exploring the interplays that may exist between perspectives.

We proposed to encode these interdependencies via domain constraints over business choices, and meanwhile to capture the structural requirements of a C-iEPC's control-flow via process constraints. In this way we can use the questionnaire-driven approach for staged process configuration to deal with the increase in complexity induced by configurable resources and objects in the process model. Not only does this approach allow us to achieve abstraction from the modeling notation, but also guarantees that the individualized process models are domain-compliant and their control-flow is semantically correct. We also proved that syntactic correctness beyond the control-flow is ensured by the application of the $\beta_1$ algorithm. The latter extends the $\beta_1$ algorithm for C-EPCs to deal with the individualization of roles, objects and their connectors. On the other hand, this approach does not prevent users from creating inconsistencies between object-flow dependencies and control-flow dependencies that may lead to semantic issues beyond the control-flow. As a result, the overall semantic correctness of an iEPC is not guaranteed to be preserved during configuration.

We applied the proposal to a comprehensive reference process model for audio post-production as part of a case study conducted with the AFTRS. The validation was supported by the Synergia toolset, which we extended to cater for the questionnaire-driven staged configuration of C-iEPCs. Specifically, we defined the EPML 2.0 specification to encode an iEPC and its variation points, we extended the Mapper tool to import C-iEPC process models and to link them to questionnaire models, and extended the C-EPC Designer to provide visual support for the creation of C-iEPC models.

In this chapter we presented a process meta-model to incorporate participating resources and objects. A common approach to capturing resources in process models is to associate a role, a capability and/or an organizational group to each task [AH02]. Similarly, the flow of data and physical artifacts is generally captured by associating objects to tasks. These principles are followed by languages such as UML ADs, BPMN and eEPCs. Our meta-model defines more sophisticated features for role-based resource modeling and object-flow modeling which go beyond those found in these languages. Also, it layers configuration capabilities on top of these features.

In Chapter 2 we outlined two approaches that support the configuration of resources
or objects: the model projection approach of Becker et al. [BDK07] and the hierarchical variability modeling approach of Razavian et al. [RK08]. These approaches are penalized by the simple types of role-task and object-task associations that are provided by the adopted meta-models (eEPCs and UML ADs, respectively). As a result, they offer only basic features to configure resources and objects (e.g. dropping a role from an eEPC function). On the contrary, our enriched EPC meta-model allows us to achieve fine-grained configuration of task-role and task-object associations.

Resource modeling for business processes has also been studied from the perspective of role-based access control. Russell et al. [RAHE05] identify a set of resource patterns describing the various ways in which resources are represented and utilized in process models. Ferraiolo et al. [FSG+01] outline a reference model of well-accepted mechanisms for role-based access control, while Bertino et al. formalize role authorization constraints [BFA99]. Meanwhile, languages for executable process modeling, such as BPEL, YAWL or ADEPT, generally rely on global variables to capture data-flow and are concerned with the definition of data mappings between global variables and task input/output parameters. Such mappings are also used in formal system specifications based on Colored Petri nets [Jen97]. Our aim is to define configurable process models for analysis and design. As such, our proposal does not cover aspects such as role-based access control, resource binding mechanisms and data mappings which are relevant at the implementation level and need to be enforced and supported by WFMSs.
Chapter 7

Conclusion

The thing can be many in one sense, but also can be one in another sense—
Al-Ghazali: Revival of Religious Sciences

This thesis proposed an integrated framework to address three shortcomings of existing approaches to capturing and managing variability in Process-Aware Information Systems: (i) error-prone configuration, (ii) lack of decision support and (iii) lack of expressiveness.

The first issue stems from the use of manual methods for configuring process models. These methods leave analysts with the burden of manually fixing errors to ensure the correctness of the individualized models. In addition, developers cannot directly derive executable specifications from incorrect individualizations, thus additional effort is required. To address this issue, this thesis devised a conceptual foundation for process model configuration. In particular, the objective was to formally define the notion of process model configuration independently of vendor-specific notations, and to use this notion as a tool to analyze correctness properties of configurable process models. For this purpose a class of Petri nets, namely Workflow nets, was adopted, which is specifically designed to formally represent business processes. Workflow nets were extended with a notion of variation point leading to configurable Workflow nets, and configuration was defined as a behavior-restriction operation based on the hiding and blocking operators.

This foundation paved the way to the development of a technique for staged correctness-preserving configuration of process models. Assuming the initial model is correct, the technique guarantees that the individualized models are also correct at each stage of the configuration procedure. This was achieved by capturing syntactic requirements as propositional logic constraints which can be automatically inferred from a Workflow net. If a configuration step violates these process constraints, a formula is derived to suggest ways of making the configuration step correctness-preserving. A cornerstone of this technique is a proof that, for a large class of process models, namely free-choice Petri nets, the enforcement of these syntactic constraints also ensures the preservation of semantic correctness.

The foundation was then applied to enable staged correctness-preserving configuration of Configurable Event-driven Process Chains (C-EPCs) – the configurable extension of EPCs. This notation was chosen because EPCs are widely used in practice for reference process modeling (e.g. the SAP R/3 reference model). Previous results on
the mapping between EPCs and Workflow nets were extended to show the formal relation between C-EPCs and configurable Workflow nets, thus confirming the foundational nature of the conceptualization. Next, the technique for staged correctness-preserving configuration defined on Workflow nets was adapted to C-EPCs. In particular, it was shown that configuring join connectors may cause semantic issues and it was observed that forbidding the configuration of these nodes does not reduce the expressiveness of a C-EPC. Further on, it was argued that if the initial C-EPC is semantically correct and joins are not configurable, the syntactic correctness of an individualized net is sufficient to ensure semantic correctness as well. It was shown that syntactic correctness can be interactively preserved by satisfying a set of boolean constraints in combination with the C-EPC individualization algorithm. As in Workflow nets, these constraints can be automatically inferred from a C-EPC without requiring additional effort from the modeler.

The application of the conceptual foundation to the realm of workflow languages was not discussed in this thesis and is reported in [GAJL08, GL08]. Here the YAWL metamodel was extended with a notion of variation point leading to Configurable YAWL (C-YAWL), and the technique for correctness-preserving configuration was adapted to fit the specificities of the YAWL language.

The second shortcoming affecting existing approaches for process model configuration is their lack of decision support for the selection of configuration alternatives. As a result of this shortcoming, stakeholders involved in the configuration are demanded to possess expertise both in the application domain and in the process modeling language employed. This assumption represents an adoption obstacle in domains where users are unfamiliar with modeling notations. Moreover, without proper means of abstraction, the configuration of complex process models can get close to unmanageable. To address this issue, this thesis devised a questionnaire-based approach for variability modeling. This approach abstracts from the notation used to model configurable processes, by capturing the variability of the application domain as a collection of interdependent choices. These choices are organized in a questionnaire model and used to drive the configuration of process models.

As part of this questionnaire-based approach, a technique was developed to generate interactive questionnaires from questionnaire models. These questionnaires guide the configuration process by posing relevant questions to users, consistent with the order dependencies established in the questionnaire model, and in a way that prevents users from entering incorrect answers that would violate the domain constraints captured in the questionnaire model. It was proven that simple well-formedness criteria are sufficient to ensure that no circular dependencies occur that may lead to deadlocks during configuration. Satisfiability solving techniques were used to guarantee the consistency of domain constraints and to incrementally prune the space of allowed answers at configuration time.

The link between configurable process models and questionnaire models was achieved by mapping each process variant to a condition over domain choices, such that when the latter holds, the specific variant is selected in the process model. In this way configuration can be achieved by answering a set of questions that mask the complexity of the underlying process model. Furthermore, the approach was combined with the results on process model correctness to guarantee that the individualized models are both consistent with the domain choices and semantically correct. The applica-
bility of the questionnaire-based approach was demonstrated on the C-EPC notation, however the approach was formalized in a generic manner so that it can be used for the configuration of other types of models (e.g. data models).

The last issue this thesis dealt with is the lack of suitable mechanisms to represent process variability beyond the process control-flow. Since business processes are meant to provide an integrated view over an organization, neglecting equally important aspects such as resources and business objects, leads to limited expressive power when it comes to configure these processes. For example, it is not possible to model the unavailability of a given resource in the process, nor to estimate the impact that this may have on the overall process. This issue was addressed by developing an integrated meta-model for process configuration, covering control-flow, data objects and resources. Specifically, this meta-model allows modelers to define complex role-tasks and object-tasks associations, and to capture a range of variations on top of these associations, thus extending the expressive power of configurable process models.

The study highlighted the intricacies that configurable process modeling across multiple perspectives brings. To cope with the induced overhead, it was proposed to use the questionnaire-based approach in combination with the technique for the preservation of model correctness. The meta-model was defined as an extension of the C-EPC notation, namely Configurable-integrated EPC (C-iEPC), to be usable by analysts and modelers in practice. An individualization algorithm was also formulated for C-iEPCs, and it was proved that by construction this algorithm ensures the preservation of syntactic correctness. Although embodied in C-iEPC, the meta-model was defined in an abstract manner so that it can be transposed to other notations.

The framework presented in this thesis was rigorously defined by means of formal methods. Moreover, a comprehensive, open-source toolset was implemented to establish its practical feasibility. This toolset comprises six applications supporting end-to-end process model configuration. These include a visual designer to create and validate questionnaire models and an interactive questionnaire tool to guide users through the configuration process; a visual designer for C-(i)EPC process models; a tool to create and validate mappings between questionnaire models and configurable process models, which implements an algorithm to infer process constraints from C-(i)EPC and C-YAWL process models; a tool to configure process models according to the answers of a questionnaire and a tool to individualize configured process models, which implements the C-(i)EPC and C-YAWL individualization algorithms.

The toolset was used to support the validation of the framework which was conducted using four scenarios drawn from different domains. For each scenario, a reference process model and a questionnaire model were built. Excerpts of these models were used as working examples throughout this thesis. One such scenario is related to screen post-production. This study was conducted in collaboration with domain experts from the Australian Film Television & Radio School (AFTRS), using the C-EPC and C-iEPC notations to capture the post-production processes. The screen industry was deemed suitable to evaluate the framework since post-production processes are not fixed, but vary from one project to another depending on a range of factors, e.g. creativity and budget, and the involved stakeholders have little modeling expertise. Another scenario was inspired by the VICS reference model for order management and modeled in C-YAWL. VICS is an industry standard endorsed and used by a number of large organizations, which describes a variety of options for interacting with suppliers and
logistics providers. A third scenario, reported in [GWJ+09], was drawn from the municipal domain. It provided insights on the use of C-YAWL and questionnaire models to capture variability across processes that are run at different Dutch municipalities. Finally, a fourth scenario, reported in [LM09], was drawn from the emergency management domain and modeled in C-EPC. In this domain there is a need for simple and self-explanatory support for process configuration, given the lack of modeling expertise of the people involved in recovery actions.

These experiences showed that the framework is able to cope with practical variability scenarios involving numerous choices, constraints and variation points. Meanwhile, performance measurements conducted on the interactive questionnaire tool showed that the framework can efficiently scale up to complex configuration scenarios.

In summary, given the uptake of reference process models in many reengineering, software development and software configuration projects, a framework that allows reference process models to be easily and effectively configured can significantly improve the state of the art.

There are several possible extensions to this body of research. One direction consists in extending the semantic correctness results of this thesis beyond the process control-flow. Specifically, this consists in investigating techniques for detecting semantic inconsistencies between control-flow and object-flow that can arise from configuration, e.g. object-flow dependencies that contradict control-flow dependencies. A possible starting point is to identify which assumptions need to be taken on the configurable model (e.g. captured in C-iEPC) to be able to avoid these inconsistencies through structural requirements. In this way (extended) process constraints, to be inferred from the model, can be used to enforce these requirements during configuration.

Another direction for future work consists in investigating techniques for automating the construction of configurable process models. A possible starting point is to collect a number of related process models from different (preferably successful) process design projects, and to merge them together. But how this merger can be facilitated is an open question. Since process models are usually represented as graphs, algorithms from the field of graph matching could prove beneficial [Bun00]. These algorithms could be employed to identify a common denominator among all models, and variants with respect to such a common denominator. These variants could then be captured as configurable nodes on top of this common denominator. However, further information would need to be added, especially domain information to constrain the possible combinations of configurable nodes. Another option to automate the construction of configurable process models would be to use process mining techniques. The idea of process mining is to take event logs related to a business process and to derive a process model that matches the event log in question. In [JAR06] the authors discuss extensions to existing process mining techniques that allow one to derive a C-EPC from a regular EPC and one or several logs. Further research is required to refine these techniques and to validate their applicability in practice.

The development of questionnaire models and configurable process models may be a costly endeavor and the assumption of stable, invariant models is unrealistic. Therefore an open research question is how to support the evolution of these models. Specifically, how to facilitate the incremental adaptation of a questionnaire model as a result of changes to the corresponding configurable process model. A possible starting point is to exploit the mapping between questionnaire models and configurable process models.
to estimate the impact of removing a process fact on the questionnaire model.

Finally, while anecdotal evidence of the usability of the framework and its toolset was gathered during the engagement with the AFTRS, further usability tests need to be conducted. In this regard, the post-production models and the toolset are planned to be introduced in the AFTRS learning environment in 2009. This will be a concrete opportunity to conduct further experiments with domain experts.
Bibliography


