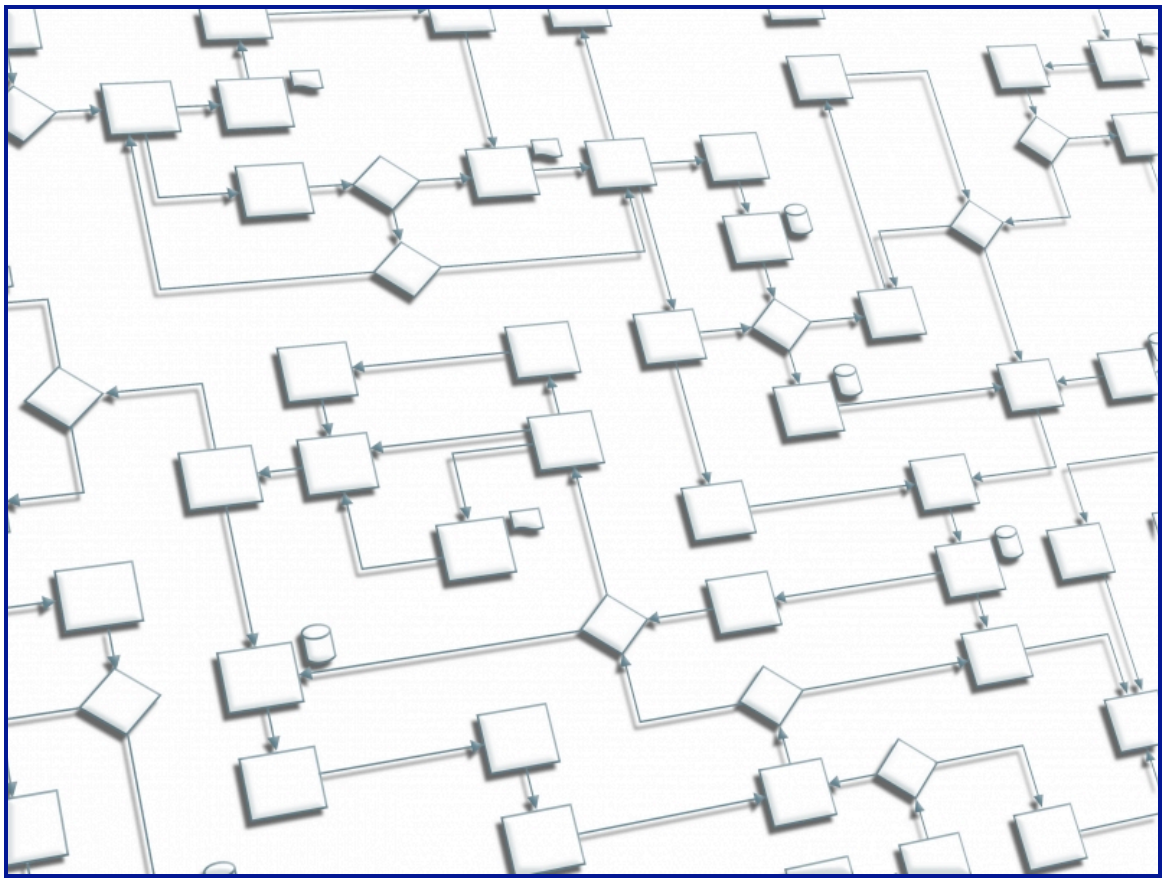
Modeling Business Process Variability

A search for innovative solutions to business process variability modeling problems



Mark Vervuurt

October 2007

University of Twente
Enschede - The Netherlands

Modeling Business Process Variability

A search for innovative solutions to business process variability modeling problems

Doctoral Thesis
Mark Vervuurt
Enschede, October 2007

Graduation committee
Dr. Manfred Reichert (first supervisor)
Dr. Ir. Bedir Tekinerdogan
Ir. Ingo Wassink

Chair
Information Systems

Department
Electrical Engineering, Mathematics and Computer Science

University
University of Twente

Management summary

This thesis is written at the University of Twente for the Information Systems department from the 1st of April 2007 to the 1st of November 2007. It presents all the research findings on business process variability modeling. The main goal of this research project is to analyze inherent problems of business process variability and solve them simply, innovatively and effectively.

To achieve this goal, process variability is defined by analyzing scientific literature, its main problems identified and is illustrated using a healthcare running example: process variability is classified into process variability within the domain space and over time. These two forms of process variability respectively lead to process variability modeling and process model evolution problems. After defining the main problems inherent to process variability, the focus of this research project is defined: solving process variability modeling problems.

First current business process modeling languages are evaluated to assess the effectiveness of their respective modeling concepts when modeling process variability, using a newly created set of evaluation criteria and the healthcare running example. The following business process modeling languages are evaluated: Event driven process chains (EPC), the Business Process Modeling Notation (BPMN) and Configurable EPC (C-EPC).

Business process variability modeling and Software product line engineering have similar problems. Therefore the variability modeling concepts developed by software product line engineering are analyzed. Feature diagrams and software configuration management are the main variability management concepts provided by software product line engineering. To apply these variability management concepts to model process variability meant combining them with existing business modeling languages. Riebisch feature diagrams are combined with C-EPC to form Feature-EPC. Applying software configuration management, meant merging Change Oriented Versioning with basic EPC to create COV-EPC, and merging the Proteus Configuration Language with basic EPC to design PCL-EPC. Finally these newly created business process modeling languages are also evaluated using the newly designed evaluation criteria and the healthcare running example.

EPC or BPMN are not suited to model business process variability within the domain space. C-EPC provide explicit means to model business process variability, however the process models tend to get big very fast. Furthermore the syntax, the contextual constraints and the semantics of the configuration requirements and guidelines used to configure the C-EPC process models are unclear. Feature-EPC improve C-EPC with domain modeling capability and clearly defined configuration rules: their syntax, contextual constraints and semantics have been clearly defined using a context free grammar in Backus-Naur form. Furthermore, consistent combinations of features and configuration rules are ensured using respectively constraints and a conflict resolution algorithm. However, Feature-EPC and C-EPC suffer from the same weakness: large configurable process models. In COV-EPC and PCL-EPC the problem of large configurable process models is solved. COV-EPC ensures consistent combinations of options and configuration rules using respectively validities and a conflict resolution algorithm. PCL-EPC guarantees consistent combinations of process fragments by means of a PCL specification.

Acknowledgements

I want to thank kindly my three supervisors for their time and effort: Manfred Reichert, Bedir Tekinerdogan and Ingo Wassink. The master thesis that was written by Noordhuizen provided valuable inspiration on how to write and structure this thesis. I would like to thank Maarten Fokkinga for helping me improve my formal set notations. I would also like to thank Riham Abdel Kader for providing useful comments on my first research proposal. I would like to thank Rogier Henrikez and my brother Sean Vervuurt for helping me improve the readability of this thesis. Finally I would like to thank my family and friends for their unconditional support!

Table of contents

Table of figures

List of Abbreviations

BPM	Business process management
BPML	Business process modeling language
BPMN	Business process modeling notation
C-EPC	Configurable event driven process chains
COV	Change oriented versioning
COV-EPC	Change oriented versioning event driven process chains
E-EPC	Extended event driven process chains
EPC	Event driven process chains
EPOS	Expert system for programming and system development
Feature-EPC	Feature event driven process chains
MIL	Module interconnection language
PCL	Proteus configuration language
PCL-EPC	Proteus configuration language event driven process chains
SCM	Software configuration management
SEI	Software engineering institute
SPL	Software product line
SPLE	Software product line engineering

Chapter 1 Introduction

1.1 Context information

Business process variability and software variability occur both within the domain space and over time. Business process variability is the source of many challenging problems. Current non-configurable business process modeling languages provide limited means to model business process variability within the domain space. Current configurable business process modeling languages are more suitable for the task but also have their weaknesses. To discover new solutions to the problems caused by business process variability, variability modeling concepts borrowed from software product line engineering (SPLE) shall be applied in business process variability modeling.

1.2 Problem focus

Business process variability problems can be reduced to the following two problems (Chapter 2):

- Business process variability modeling (domain space)
- Process model evolution (over time)

The focus throughout this research project shall be on business process variability modeling. Modeling simply and effectively business process variability needs to be achieved before process model evolution.

1.3 Research objective

The research objective is to model business process variability within the domain space using a business process modeling language and variability management concepts borrowed from software product line engineering like feature diagrams and software configuration management.

1.3.1 Main research question

How can business process variability within the domain space be modeled using existing business process modeling languages and variability management concepts borrowed from software product lines like feature diagrams and software configuration management?

1.3.2 Sub-research questions

1. What are major problems and challenges posed by business process variability?
    a. What is business process variability? When does it occur and why?
    b. What are challenges and problems caused by business process variability?
    c. Why is it interesting to solve the problems caused by business process variability?

2. What are current solutions to business process variability modeling problems? What are their strengths and limitations?

3. Are similar problems posed by business process variability modeling encountered in software product line engineering?

4. What solutions are offered by software product line engineering to the problems?

5. Can these solutions be applied or adapted to solve business process variability modeling problems? What are the strengths and limitations of the chosen approaches?

1.4 Research approach

A desk research strategy is used in this research project. Based on an extensive literature research and self-reflection, different theoretical concepts are combined to achieve the research objective and answer the research questions.

1.5 Thesis overview

Chapter 2 describes basic notions of business process modeling.

Chapter 3 provides the answer to the research question #1. Business process variability is defined. The origin of business process variability and the problems it causes are also described. A healthcare running example is introduced to illustrate the basic notions of business process variability. Furthermore it is discussed why these problems are worth solving.

Chapter 4 gives the answer to the research question #2. Using a new set of evaluation criteria the modeling concepts provided by current business process modeling languages are assessed when modeling business process variability within the domain space.

Chapter 5 answers the research questions #3 and #4. Business process variability modeling problems are also found in software product line engineering (SPLE). SPLE provides alternative solutions to business process variability modeling problems: mostly feature diagrams and software configuration management (SCM).

Chapter 6 provides the answer to the research question #5. The best alternative solutions are further explored and adapted to solve the business process variability modeling problems. Three innovative solutions have been contributed to the field of business process variability modeling: Feature-EPC, COV-EPC and PCL-EPC.

Chapter 7 presents prototypes of software environments for two of the three newly created solutions: Feature-EPC and PCL-EPC.

Chapter 2 Introduction to business process modeling

2.1 Introduction

Basic notions of business process modeling are introduced here such as business processes, business process management and business process modeling languages.

2.2 Business process modeling

According to Champy and Hammer [1], a business process is a "*collection of activities that takes one or more kinds of input and creates an output that is of value to the customer*". Some concrete examples of business processes are administrative, chemical or healthcare processes.

"*Business process management (BPM) is a systematic approach to improving an organization's business processes. BPM activities seek to make business processes more effective, more efficient, and more capable of adapting to an ever-changing environment [2]*".

Business process modeling can be described as the activity of representing or mapping business processes using diagrams or modeling concepts with the goal to describe, analyze or reengineer business processes. Business process modeling can be done using business process modeling languages or notations such as event driven process chains, the Business Process Modeling Notation, or Configurable EPC.

NB: in this thesis, the terms 'process' and 'business process' shall be used alternatively to refer to the term 'business process' as defined by Champy and Hammer.

2.3 Business process modeling languages

A distinction can be made between business process modeling languages (BPML) that are non-configurable and configurable. Currently most widely accepted BPML shall be illustrated in this subchapter. The following non-configurable BPML are illustrated here under: basic event driven process chains (EPC), extended event driven process chains (E-EPC) and the business process modeling notation (BPMN). Finally the following configurable BPML is described here: configurable event driven process chains (C-EPC).

NB: the BPML that are illustrated here (E-EPC, BPMN, C-EPC) were also selected and evaluated in Chapter 4.

2.3.1 Basic event driven process chains

Basic EPC are a business process modeling language whose basic concepts to describe business processes consist of: events, functions, logical operators and dynamic connectors.

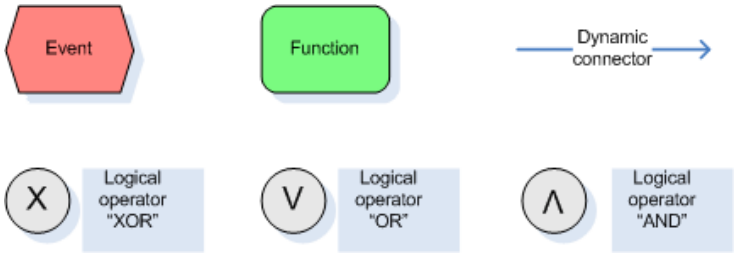


Figure 1: EPC basic modeling concepts

The healthcare running example described in Chapter 3 has been modeled using basic EPC. Furthermore basic EPC have been combined with variability modeling concepts borrowed from software configuration management in Chapter 6.

2.3.2 Extended event driven process chains

EPC were developed in 1992 in an R&D project with SAP AG at the Institute for Information Systems of the University of Saarland in Germany [3]. EPC are part of the ARIS Process Platform, which provides an integrated toolset for designing, implementing, and controlling business processes [3]. In the 1990s, and following the evolution of the ARIS toolset, the basic EPC notation has been extended with a number of symbols corresponding to various aspects of business modeling [3]: extended EPC (E-EPC). E-EPC allow the modeling of business processes using four different perspectives: organization, data, control, function and output [3].

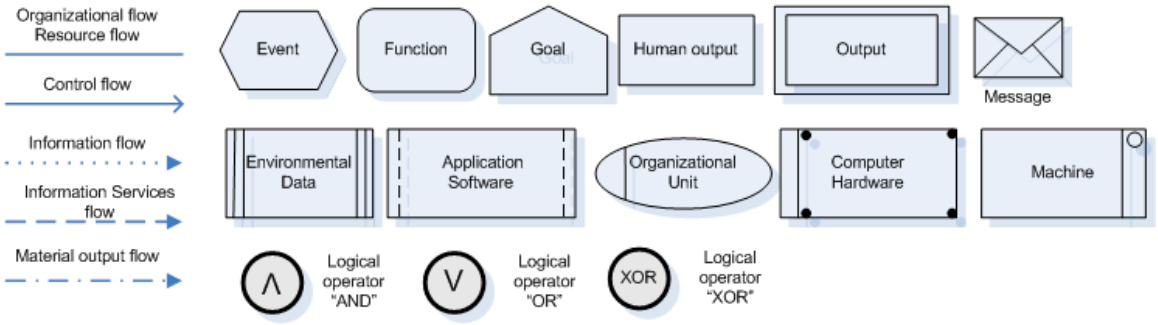


Figure 2: E-EPC modeling concepts (partially [3]).

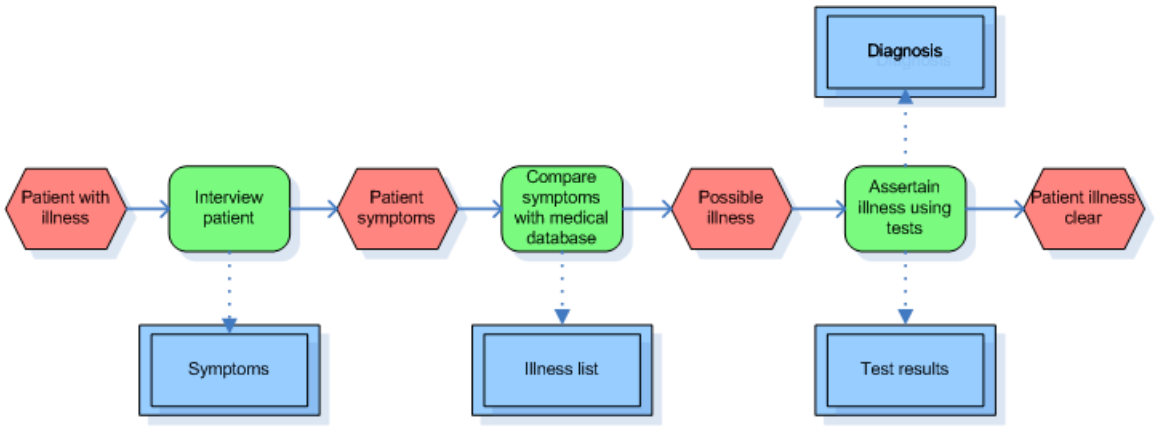


Figure 3: Example diagnosis process modeled using E-EPC

2.3.3 Configurable event driven process chains

In order to improve the configurability of Enterprise systems and reference models, C-EPC have been invented [4, 5]. C-EPC are basic EPC extended with configurable nodes and attributes to describe the configurable nodes. Configurable nodes can be:

- ON
- OFF
- OPT (conditionally skipped)

Furthermore configurable AND operators can only be reconfigured as logical AND operators. Configurable OR operators can be reconfigured as logical AND, OR, XOR operators and sequences. Lastly, configurable XOR operators can be reconfigured as logical XOR operators and sequences.

Figure 4: Configurable nodes [5]

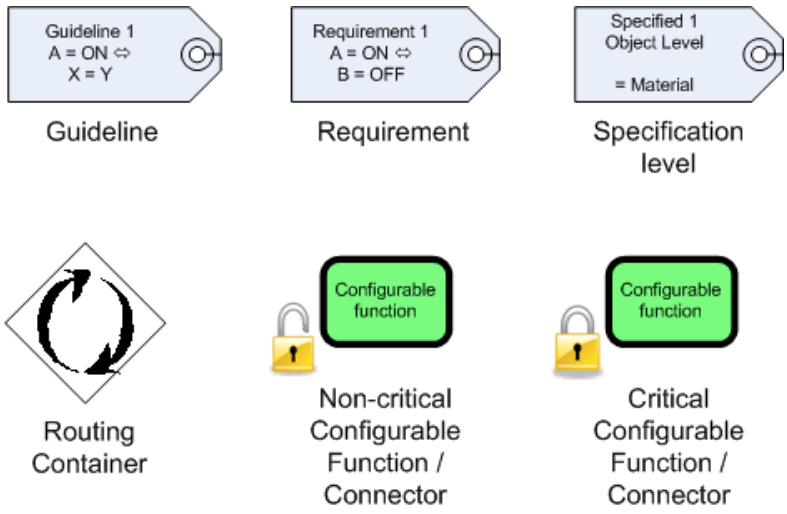


Figure 5: Configurable attributes [5]

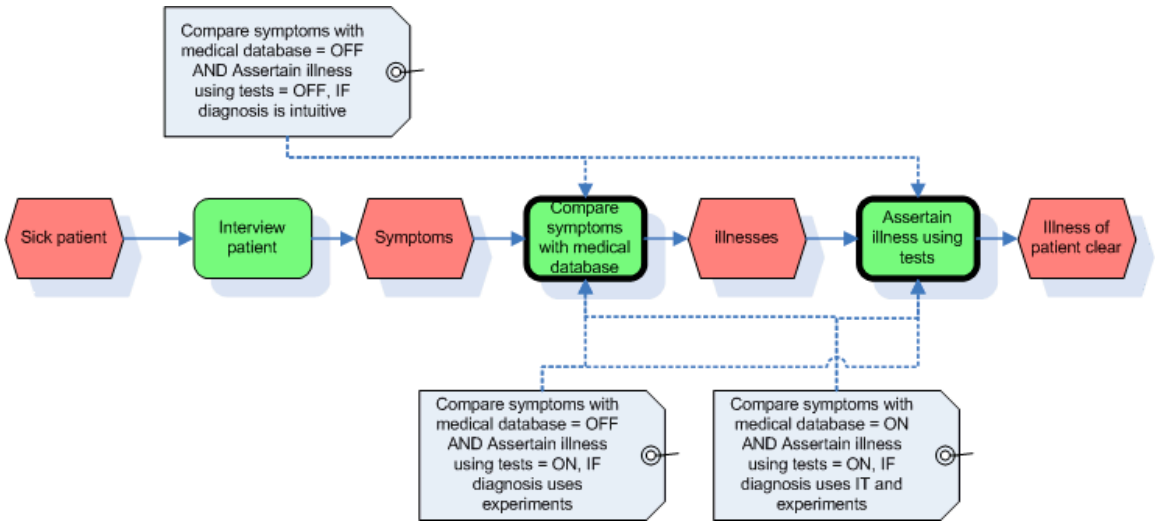


Figure 6: Example diagnosis process modeled using C-EPC

2.3.4 Business process modeling notation

BPMN was developed by the Business Process Management Institute (BPMI), which is since February 2006 an official standard maintained by the Object Management Group (OMG) [6]. BPMN is a very rich business process modeling notation, therefore only the basic modeling concepts (Figure 7) and concepts used to model the healthcare running example described in section 3.4 are illustrated (*Figure 8*). For a complete and more precise description of the BPMN modeling concepts, please view the OMG website[1] [7, 8] and especially the BPMN tutorial offered by the IBM Software Group[2] [6].

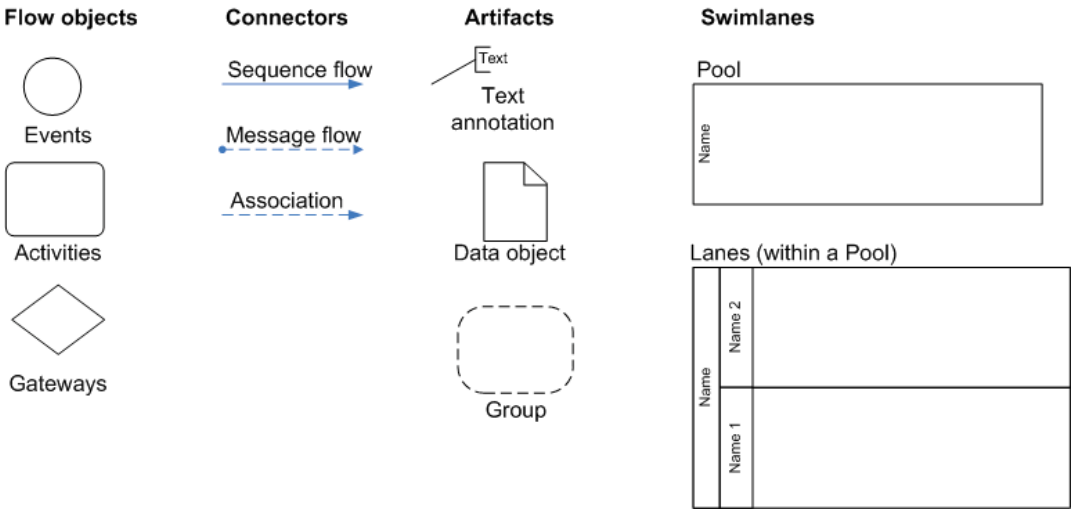


Figure 7: BPMN basic modeling concepts [6]

[1] http://www.bpmn.org/

[2] http://www.bpmn.org/Documents/OMG%20BPMN%20Tutorial.pdf

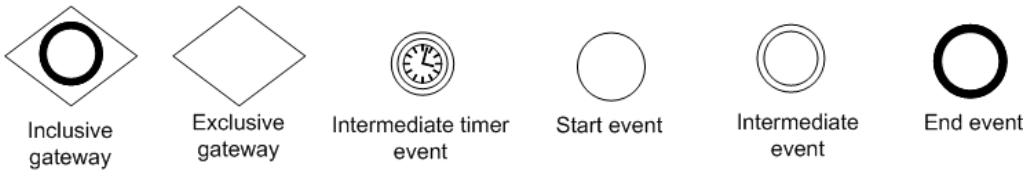


Figure 8: Additional modeling concepts used to model the healthcare running example

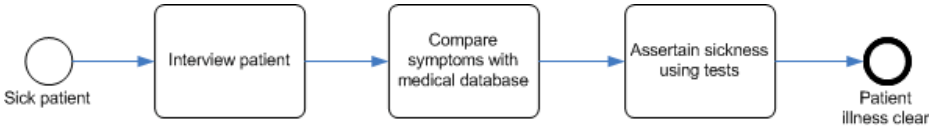


Figure 9: Simple diagnosis process modeled using BPMN

2.4 Conclusion

The basic notions that are needed to comprehend the content of this thesis have been introduced here. Basic EPC have been described here because most process models in this thesis will be illustrated using basic EPC: for example the healthcare running example described in Chapter 3. More importantly three widely known and accepted business process modeling languages have been introduced and illustrated here: E-EPC, BPMN and C-EPC.

Chapter 3 Understanding problems and challenges of business process variability

3.1 Introduction

Problems caused by business process variability have yet to be solved. The healthcare running example described in section 3.4 illustrates business process variability and its main problems. To understand these problems, it will be important to define first the notion of process variability. Secondly the origin of process variability will be determined. Thirdly the main problems of process variability will be analyzed and described. Finally the rewards for solving these problems are described as well as a problem focus.

NB: in this thesis, the terms 'process' and 'business process' shall be used alternatively to refer to the term 'business process' as defined by Champy and Hammer.

3.2 Business process variability description and definition

Pentland [9], asserts that the variability or variety of business processes can be described along three dimensions:

- "*Variety in the range of tasks performed (task variety)*"
- "*Variety in the order that these tasks are performed in (sequential variety)*"
- "*Variety in the inputs and outputs of the process (content variety)*"

The concept of process variability has many variants in the literature: process variety, process variance, process flexibility, process agility, process change, process adaptability, process evolution, etc. After an analysis of the literature, three main definitions of the concept of process variability have been found:

1. The concepts of "mean" and "variance" as often used to define the variability of a process in the field of operational research [10]:

    i. The mean

    "*The mean of a random process is the average of all realizations of that process.*"

    ii. The variance

    "*Now that we have an idea about the average value or values that a random process takes, we are often interested in seeing just how spread out the different random values might be. To do this, we look at the variance which is a measure of this spread.*"

    This type of process variability is left out of scope. Process variability shall not be analyzed from a statistical perspective in this research project but rather from a process modeling and change management perspective.

2. The concepts of process evolution, process agility, process change, and process adaptability are fairly equal in the literature and are all related to process changes as a result of a response to environmental changes [11-15]. These environmental changes can furthermore be classified into two categories [16-18]: ad hoc or evolutionary changes.

    i. Ad hoc changes are rare events, exceptions, etc.

    ii. Evolutionary changes are the consequences of "*reengineering efforts*".

    Using the healthcare running example (section 3.4), this type of process variability can be illustrated using the following example: as a result of an evolutionary change or reengineering effort the healthcare organizational process described in Figure 20 is being improved with digital reporting as described by Figure 21. These processes are labeled as "process revisions".

3. Process variability can also be a consequence of variability occurring within the application domain of the process. This is the case of manufacturing processes where process variability is a consequence of product variety. In manufacturing literature, the concept of process flexibility is relatively equivalent to the chosen definition of process variability:

    i. "*Process variety refers to the diversity of variations of the manufacturing processes for producing the product variants in the product family* [19]".

    ii. "*Design changes related to product variety usually result in frequent process variations (referred to as process variety)* [20]".

    This type of process variability also occurs in other application domains such as the healthcare application domain. In the healthcare running example of section 3.4, the healthcare organizational process has tree variants within the application domain:
    - The healthcare organizational process handles a patient without disabilities (Figure 17).
    - The healthcare organizational process handles a blind patient (Figure 18).
    - The healthcare organizational process handles a paralyzed patient (Figure 19).

    These processes will be labeled process variants.

The field of software engineering suggests that software can vary in time and space [21]; business processes also vary along these two dimensions:
- Process variability over time can thus be defined as process variability as a response to environmental changes as described in definition #2 here above.
- Process variability within the domain space can thus be defined as process variability within the application domain space at any point in time as described in definition #3 here above.

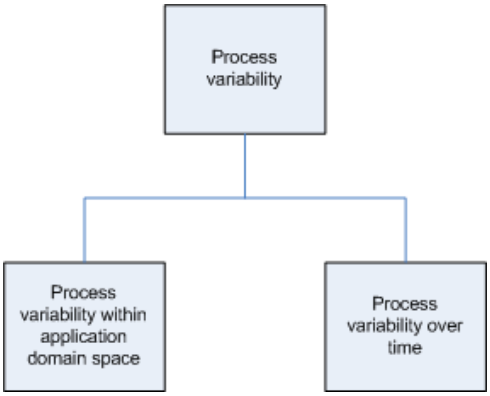


Figure 10: Process variability illustrated using a feature diagram

The focus of this chapter shall be on the analysis of the problems caused by process variability within the application domain space and over time. Finally process variability can occur in virtually any application domain. However what the origin is of process variability is rather unclear.

3.3 The origin of business process variability

Business process variability occurs within the domain space and over time. The origin of these two types of process variability shall be identified here.

3.3.1 The origin of business process variability over time

Business process variability over time or process model evolution is the result of process changes over time, which are the result of environmental changes having an influence on the process [22]:

"Usually, certain events such as the introduction of a new software development technology in a development team (e.g., new testing support tools and techniques), a new/updated process engineering technology (e.g., a new process modeling technique), new/updated standards/guidelines for software development or process engineering, new/updated regulatory constraints, or new/updated best practices emerging from community experience generate issues that must be resolved by performing changes to the software process models."

As was said previously in section 3.2, these changes can be categorized into ad-hoc or evolutionary changes.

NB: see section 3.2 for an illustration of evolutionary changes. Modeling ad hoc changes or exceptions falls out of the scope of this research project.

3.3.2 The origin of business process variability within the domain space

Business process variability occurs in application domains that display some variability themselves. The greater the variability or complexity of the environment of a business process, the greater the variability of the process shall be. Simply said process variability occurs when the environment or domain (events, resources, goals, products, etc) of the process is also variable.

NB: Process models shall vary only on those aspects that are being modeled by the process modeling language. For example, a process modeling language that does not model events cannot vary on those points.

Events
According to Scheer [23], an event can be defined as: "*Event characterize pinpointed activities containing facts (what) that occur at a certain point in time (when). What and when coincide in time events (such as 6 PM)*". Events can be "*start events*", states or triggers. The more events are to be modeled or integrated into a process model the more variability the process will display. Especially the start and end events have an influence on the variability of processes. The greater the set of start and end events, the greater the variability of a process. For example, curing a patient with cancer or a patient with diabetes shall necessitate different treatment plans. Here we have two different start events "*cure patient with cancer*" and "*cure patient with diabetes*" that lead to different treatment plans [24].

Organizational units
Organizational units are departments, groups, roles, etc [3]. Their variability can also induce process variability. Different departments can achieve the same goal using different work processes. The same reasoning holds for roles. To illustrate this type of variability, a clinical diagnostic process is taken as an example. Although the outcome of the diagnosis process is the same: a diagnosis. The process followed by a medical expert to reach this diagnosis is quite personal and unique [25]:

"*Every individual doctor has her own, idiosyncratic mode of diagnostic reasoning. What is even worse is that only a few physicians are aware of how they achieve their diagnosis. Usually a diagnosis seems to happen, just as much as a dream or a headache does.*"

The phenomenon of process variability thus also occurs here. The diagnostic process used by a medical expert is variable. The variability is thus introduced by the resource involved in the process: the medical expert.

NB: in this example, variability can be introduced by other resources such as the patient, the disease of the patient, the equipment or diagnostic techniques used, etc.

Resources
According to Dumas, van der Aalst and ter Hofstede [3], "*Resources include all kinds of objects that are necessary to perform a workflow or a task*". Resources are for example assembly lines, bricks, etc. Note that organizational units can also be considered resources [3]. A good example to illustrate process variability caused by resources is looking at the construction process of a building. Using concrete or bricks leads to different construction processes. Resources can also be considered inputs of a business process.

Goals
"*Goal states describe a future required state that the process should achieve, maintain or avoid* [26]". It is clear that different goals require different processes. A goal requiring building a house and a goal requiring cleaning a house shall lead to different processes. However different processes can lead to the achievement of the same goal. For example the sequence of processes or operations that lead to a clean hospital can be variable as demonstrated by Figure 11 and Figure 12.

Figure 11: Hospital cleaning process variant #1

Figure 12: Hospital cleaning process variant #2

Inputs and outputs

Process variability also occurs when a process has variable inputs and outputs such as customized products. Inputs of a process can als be considered resources used by a process. Production processes supporting the mass customization paradigm have variable products as outputs. Many authors argue that process variability also described as process flexibility is a key enabler to implement the mass customization paradigm.

Pine, Peppers and Rogers (1995) [27] "*asserted that to be successful, mass customizers must employ a production/delivery strategy that incorporates modularity into components and processes.*"

Finally Da Silveira, Borenstein and Fogliatto assert the following [28]: "*the broad, visionary concept was first coined by Davis and promotes mass customization as the ability to provide individually designed products and services to every customer through high process agility, flexibility and integration.*"

Here under is given a simple example to illustrate process variability. An assembly line that supports the mass customization paradigm must be capable of producing a car that can be slightly customized. Suppose this car has three variants:

- Blue car
- Red car
- Green car

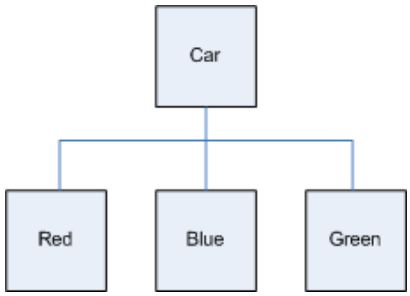


Figure 13: Colored variants of the original car

To enable the production of the variants of all these cars, the production process must integrate all these variations. Analyzing this simple example, the following statements can be inferred. The assembly line of the car has to integrate into its production process, the following processes:

- Paint car using blue paint
- Paint car using red paint
- Paint car using green paint

Figure 14: Simplified production process model a blue painted car

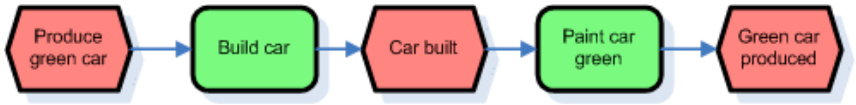


Figure 15: Simplified production process model a green painted car

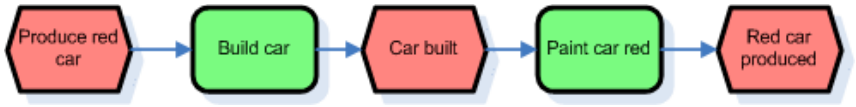


Figure 16: Simplified production process model a red painted car

The more variable the environment of a business process with respects to events, resources, goals, and products, the more variable the business process will become. We have determined the possible causes of process variability. However process variability introduces its shares of problems: rising costs, modeling problems, change management problems, difficult standardization, etc.

3.4 Healthcare running example

To make things more explicit, a running example of a healthcare process is used to illustrate important concepts related to business process variability. This example healthcare process is borrowed from Richard Lenz and Manfred Reichert's paper titled "*IT support for healthcare processes – premises, challenges, perspectives*" [24]: it is an example organizational process of a radiology department with order entry and reporting. This organizational process is described in Figure 17. To illustrate important aspects of process variability several hypothetical variants and one revision of this process model have been constructed. A distinction is made here between process variability within the domain space and over time.

3.4.1 Business process variability within the domain space

Business process variability can occur within the domain space. To illustrate business process variability within the domain space, the process models described by Figure 17, Figure 18 and Figure 19 are used. The healthcare organizational process is adapted to suit the needs of patients with specific needs: every adaptation of the healthcare organizational process results in a unique process variant.

The organizational process model as described in Figure 17, considers the case of a generic patient without disabilities. However, for example handling patients with disabilities requires the adaptation of this process model. Two hypothetical process variants of the organizational healthcare process model have been constructed:

- The healthcare organizational process handles the case of a blind patient (Figure 18), which requires escorting the patient in and out of the radiology department.

- The healthcare organizational process handles the case of a paralyzed patient (Figure 19), which requires the scheduling of an ambulance, escorting the patient and assisting the patient throughout the medical examination.

However modeling process variability within the domain space can be done in several ways. The healthcare running example is modeled using three distinct process models for each process variant:

- A patient without disabilities needs radiology
- A blind patient needs radiology
- A paralyzed patient needs radiology

The healthcare organizational process models described in Figure 17, Figure 18 and Figure 19 can and has also been modeled using one big process model: This model is described in Figure 20. The reason behind this modeling choice shall be explained latter on in Chapter 4.

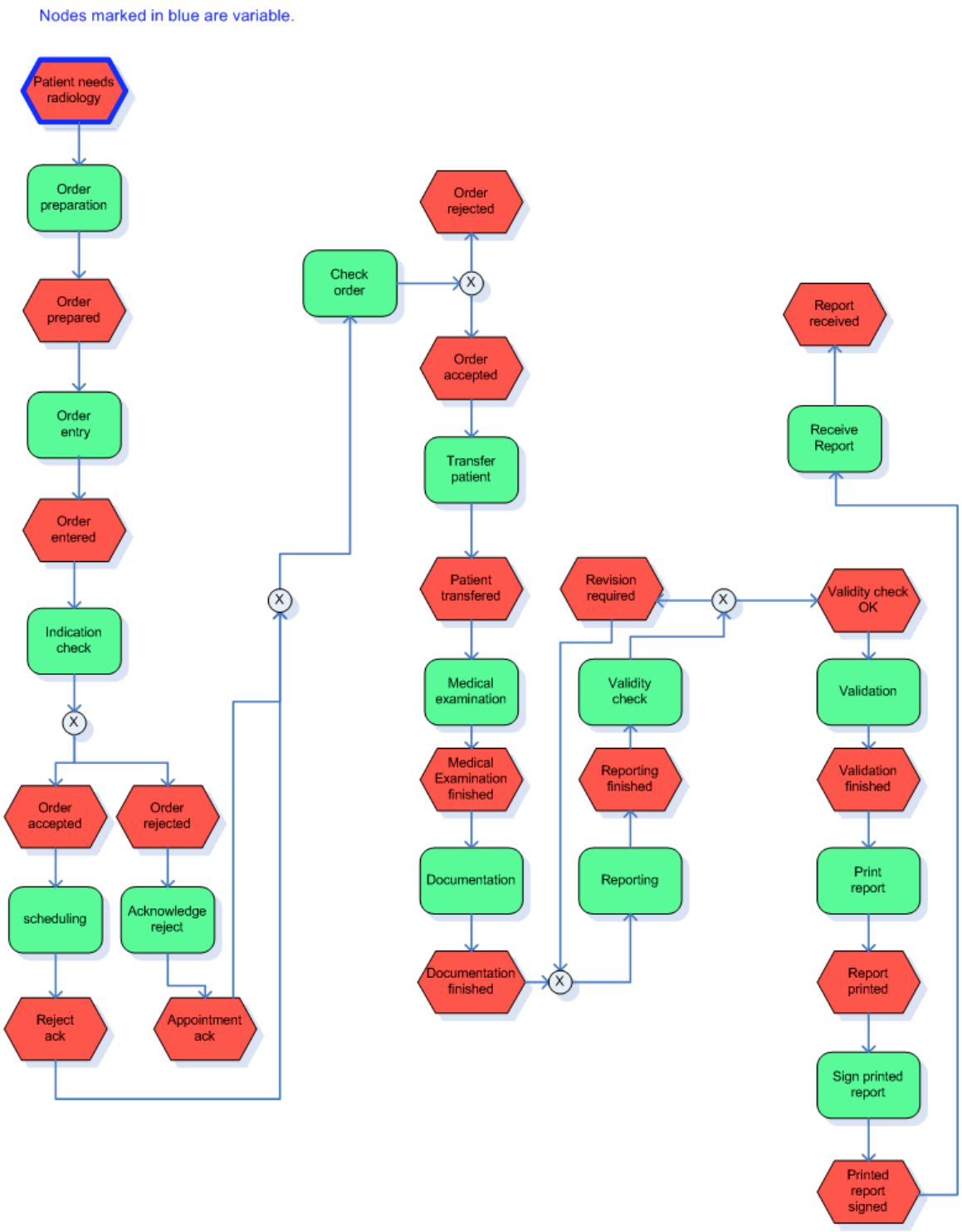


Figure 17: Example healthcare organizational process with patient without disabilities [24]

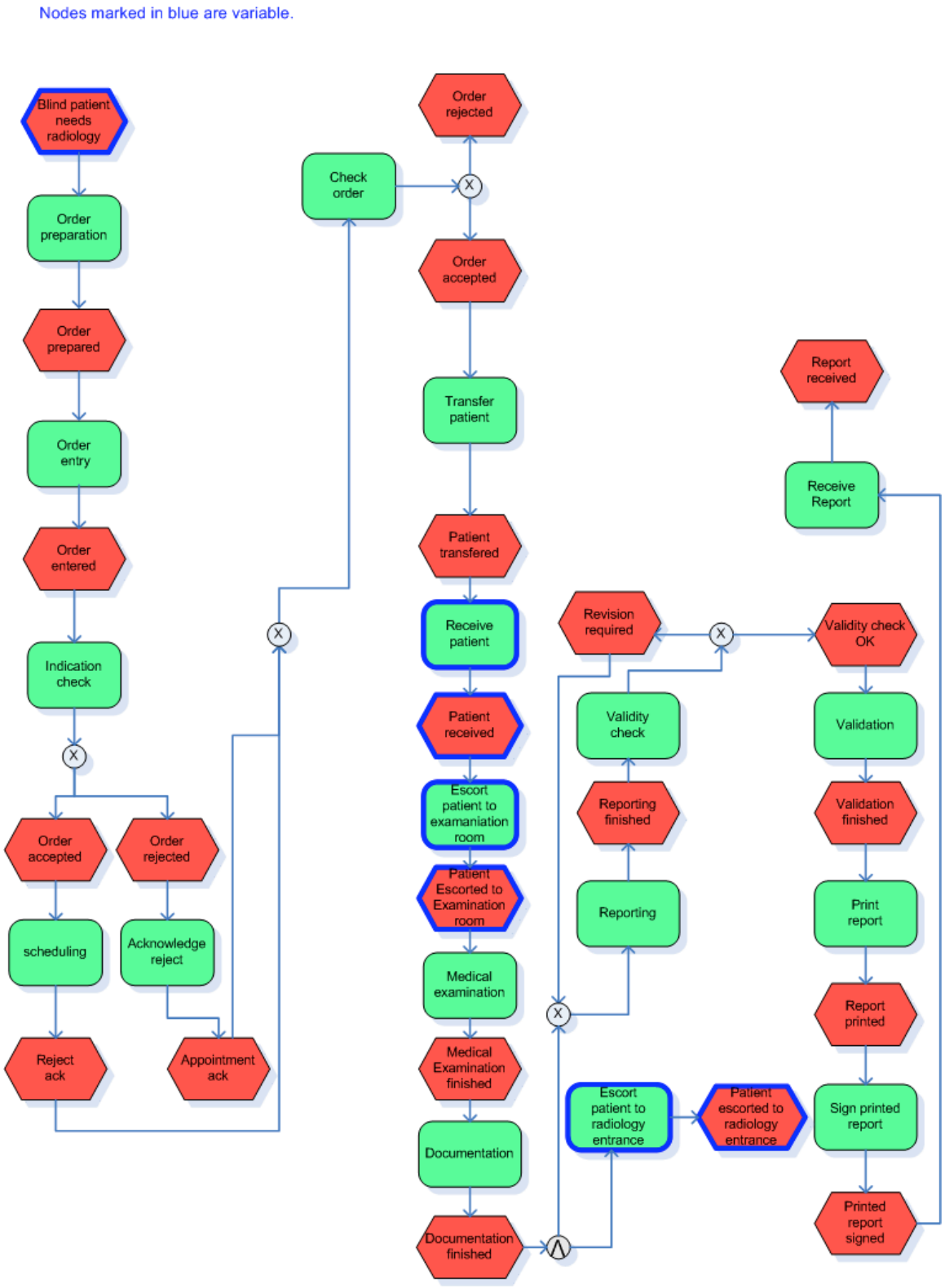


Figure 18: Example healthcare organizational process with blind patient

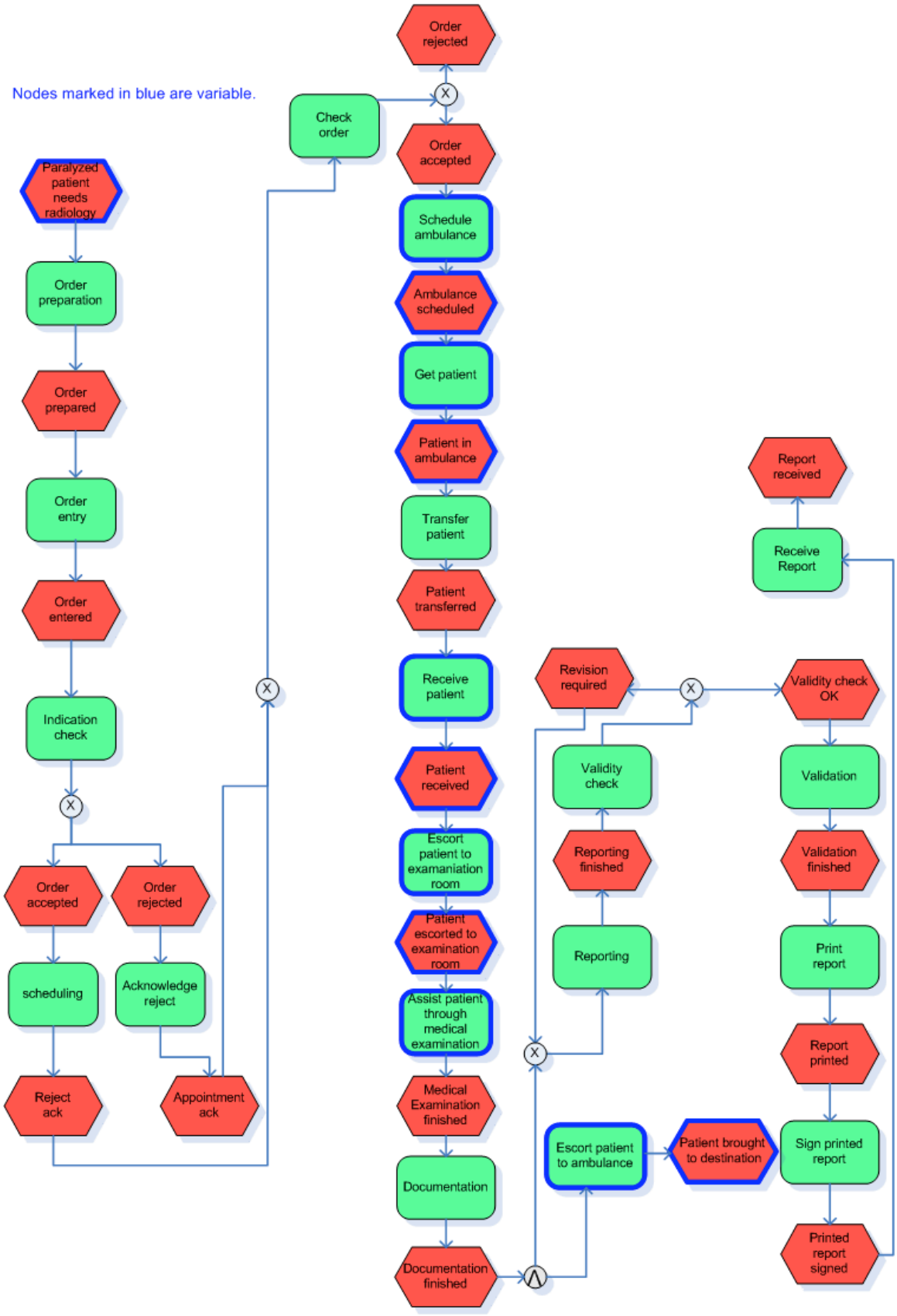


Figure 19: Example healthcare organizational process with paralyzed patient

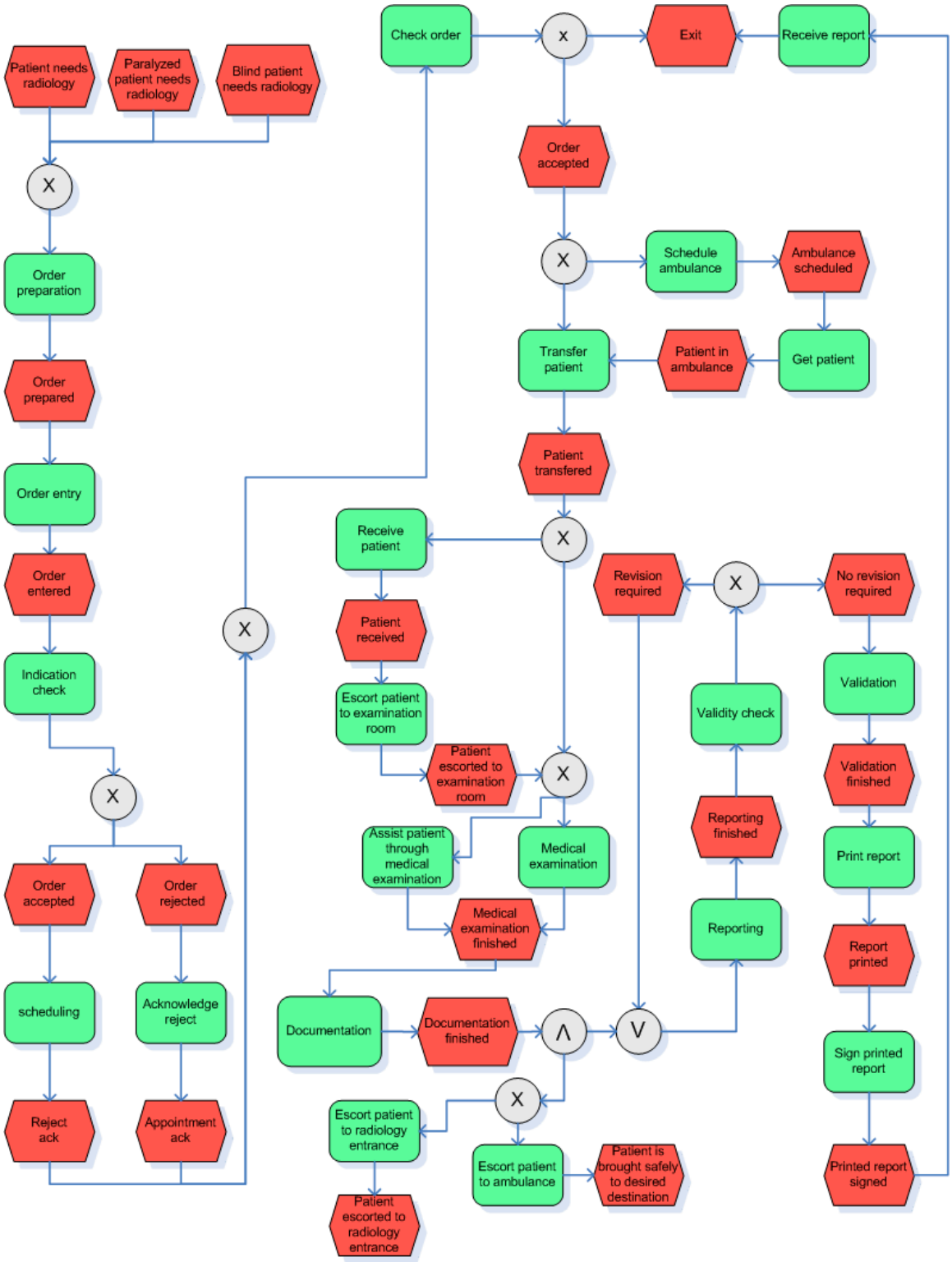


Figure 20: Figure 17, Figure 18 and Figure 19 modeled into one big process model

3.4.2 Business process variability over time

To illustrate business process variability over time, the process models described by Figure 17 and Figure 21 shall be used. The healthcare organizational process described in Figure 17 has been reengineered to improve its reporting process: the resulting or improved healthcare organizational process is described in Figure 21. These process models are called process revisions.

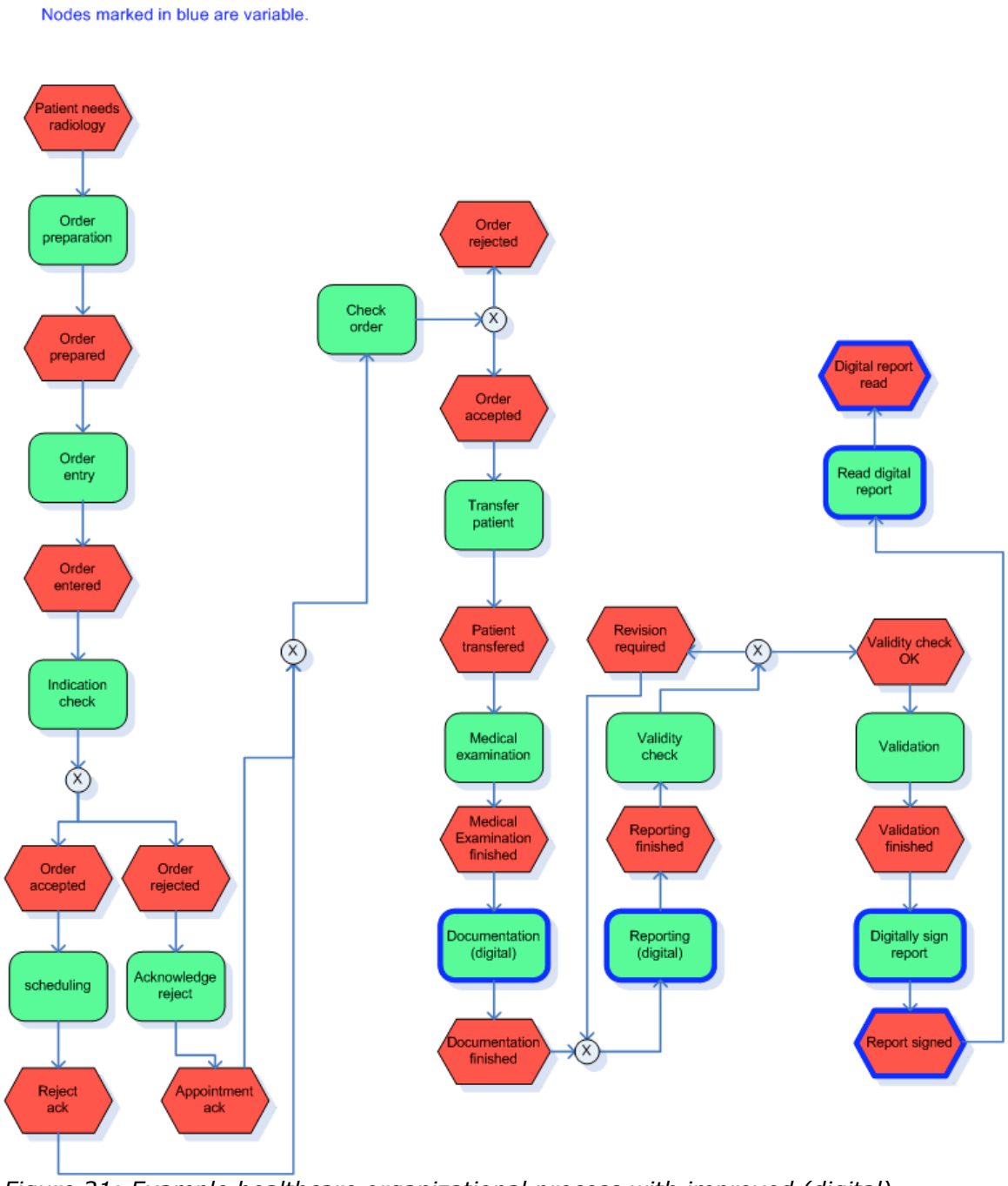


Figure 21: Example healthcare organizational process with improved (digital) reporting

3.5 Identifying main problems of process variability

Two main sources of business process variability have been identified previously as illustrated by Figure *10*: Process variability within the application domain space and over time. The healthcare running example (section 3.4) is used to illustrate process variability as well as the problems that come with it. The problems caused by process variability can be described using the example organizational and radiology process. This radiology process has several process variants of which three are illustrated respectively in Figure 17, Figure 18 and Figure 19. The differences between these three radiology processes can be depicted as process variability within the domain space. In this case, process variability is the consequence of variability within the application domain space of the process at any fixed point in time [21]: the radiology process is adapted to suit the needs of different patients (normal, blind, paralyzed, etc). Modeling simply and effectively, process variability within the application domain space is still a challenging problem. Furthermore the syntactic and semantic correctness of the models need to be ensured [29, 30]. Finally the storage and retrieval of all the process models must be possible.

In Figure 17 and Figure 21 is illustrated process variability over time. Reporting radiology results has been improved and is now done digitally as illustrated in Figure 21, instead of using paper as presented in Figure 17. More importantly, this process improvement also requires the update of the process models presented in Figure 18 and Figure 19. The propagation of the process changes (Figure 21) to the other process models (Figure 18, Figure 19) can be done manually if few process models need to be updated. However if a great amount or number of process models need to be updated then semi-automatically or automatically propagating these changes becomes necessary. Process changes over time also result in several versions of the same process model. An overview of these changes and different versions of the process models needs to be maintained. Managing simply and effectively variability over time of a business process is thus another challenging problem. Nevertheless must not be forgotten that syntactic and semantic correctness of the process models must also be guaranteed before and after their changes [29, 30]. Finally all these process models must be stored and be retrievable.

Simply trying to model all the process variants presented in the application domain space into one process model is difficult and should not be attempted: the resulting process model would be a big process model, which is hard to understand and maintain. In Figure 20, the three radiology process models (Figure 17, Figure 18, Figure 19) are merged into one big process model: this model is indeed harder to comprehend, lacks precise semantics and is also harder to maintain or modify over time.

Secondly, modeling all the process variants of the radiology process into separate process models could be a better solution (Figure 17, Figure 18, Figure 19): all the process models should then be understandable. However, maintaining a large number of process models can become problematic because all their changes and versions need to be tracked. Moreover when a large number of process models is being maintained, a change over time could affect a subset of the process models. Modifying all these models manually is then not a viable solution. Mechanisms are thus required:

- To determine the impact of changes on the set of process models.
- To propagate these changes semi-automatically or automatically.
- To preserve the correctness of the process models after applying these changes.

Thirdly, a promising approach to model process variability within the application domain space can be achieved by using a reference process model from which process variants are derived [31]. The reference process model is a generic process model that captures the similarities or commonalities of all the process variants within the application domain space. It remains unclear how the commonality of process variants should be modeled and also on the basis of what criteria. Additionally, it is also unclear how the process variants should be derived from the reference process model. Process variants can be modeled as options, extensions, specializations, change patterns, etc, of this common reference process model. Current implementations of reference process models from which process variants can be derived use configurable process models: questionnaire based reference process models [32-34], configurable reference process models [4, 35, 36], etc. Maintaining reference process models is not straightforward: Keeping track of changes and different versions of the reference process model and its respective options can be done as a whole or separately. Naturally the syntactic and semantic correctness of the process models must be guaranteed [29, 30]. Additionally updating a reference process model and its options over time implies the following:

- Changes only have an impact on the reference process model. This requires the modification of the reference process model. However it is not clear how the options are affected by the modification of the reference process model.

- Changes only have an impact on a subset of the options. The options thus need to be modified. However it is unclear whether the reference process model shall need to be modified to support the changes of the affected options. Furthermore, if the reference process model needs to be modified how does it affect the remaining unchanged options.

- Changes have an impact on the reference process model and a subset of the options. It is clear that both the reference process model and the affected options must be modified. The set of modifications could result in conflicting changes: changes that cannot occur at the same time. Finally it is unclear whether the set of unaffected options shall also need to be modified to acquire those changes.

Finally the reference process model and its respective options must be stored and be easily retrievable.

The main problems of process variability can thus be summarized by the following problems:

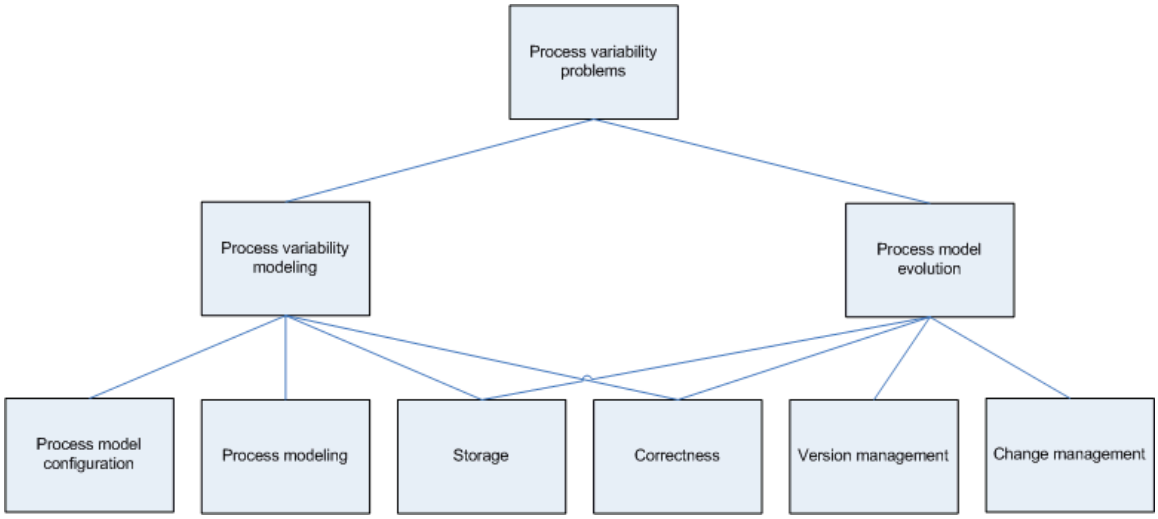


Figure 22: Process variability problems illustrated using a feature diagram

Moreover these two problems are closely related because process variability modeling has an impact on process model evolution. Process variability modeling should thus be done with the goal to enable the simple and effective evolution of the process models.

3.5.1 Business process variability modeling issues

Process model configuration

In this field, research still needs to be done. One of the purposes of this research project is to unravel how process model configuration can best be done. This purpose shall be achieved by borrowing concepts from other relevant fields such as software product lines and software configuration management. The purpose of process model configuration is to automatically adapt or configure process models for specific circumstances.

Business process modeling

The first important issue is that not all business processes can be modeled or are easy to model. Business processes can be classified according to their degree of structure [3]:

- Ad hoc processes
  "*An ad hoc process is one in which there is no a priori identifiable pattern for moving information and routing tasks among people; for example, a product documentation process or a process for preparing a response to a complex tender.*"

- Administrative processes
  "*Administrative processes, on the other hand, involve predictable processes with relatively simple task coordination rules.*"

- Production processes
  "*production processes involve repetitive and predictable tasks with more or less complex but highly stable task coordination rules.*"

Therefore the focus throughout this research project shall be on administrative or production processes because they can be modeled and thus also their variants.

Traditional business process modeling languages fail to capture the variability of business processes [9]:

"*Traditional languages for describing business processes appear too rigid and formal to cater for the wide variety of ways of doing things found in local government in the UK and, unless 'one size fits all' solutions can be imposed by fiat, a richer language is needed to underpin discussion on process variety and best practice"*.

A solution is extending business process modeling languages with variability management features: "*Reference modeling languages must therefore be created so that they support model-variant management* [37]". An important issue is then choosing the right process modeling language. This process modeling language must be extendable with variability management concepts. Additionally there are only a few process modeling languages available that support the modeling of process variability [4, 36]. Improved business process modeling languages are thus necessary to model process variance [9] [37].

When modeling process variability the following issues arise: modeling all the variations of a business process in one process model or several process models using one or several views. The ARIS business process modeling language comprises four views [23]: a functional, organizational, data and output view. Furthermore the chosen process variability modeling technique must ease or at least enable the change management, maintenance or evolution of all the variable processes.

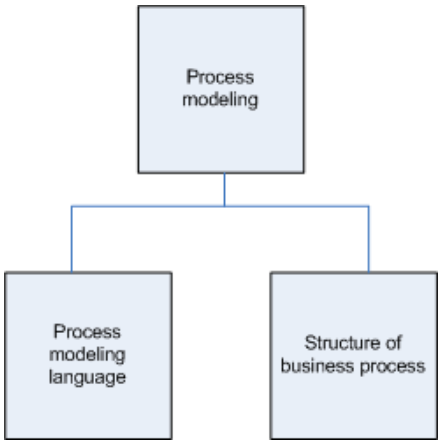


Figure 23: summary of process modeling issues

Storage of process models

Important elements of storage are related to database management systems because the storage facilities must enable not only the storage but also the proper retrieval and classification of the process models: "*A DBMS is a complex set of software programs that controls the organization, storage and retrieval of data in a database* [38]". Furthermore the storage facilities must guarantee the integrity and security of the stored process models.

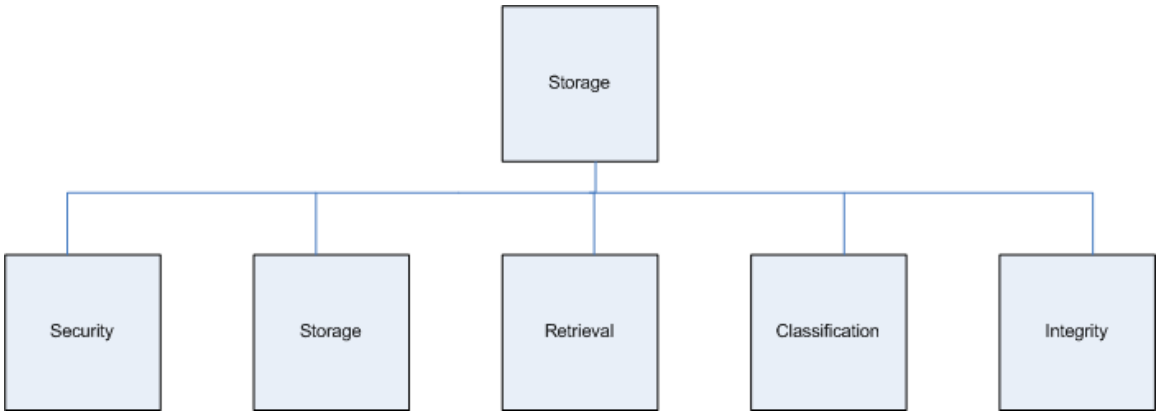


Figure 24: Summary of storage issues

Correctness of process models

Finally another important issue about process variability modeling is verifying and ensuring the correctness of all the process models before and after their changes: this requires the specification of appropriate correctness criteria [29]. Changes leading to incorrect process models should not be accepted. However checking the syntactical correctness of a process (deadlocks, bad input parameters, etc) is not enough [30]. Semantic errors can still occur as illustrated by the following example:

"*Assume that, due to suddenly arising headache, the drug Aspirin is administered to patient Smith. This is achieved by inserting activity Administer Aspirin into instance I in an ad-hoc manner by, for example, a nurse at her workplace. […]. However in this treatment process, the drug Marcumar, which is not compatible to Aspirin, is already administered some activities ahead (semantic conflict). Even if the process change is syntactically correct, it is not semantically*".

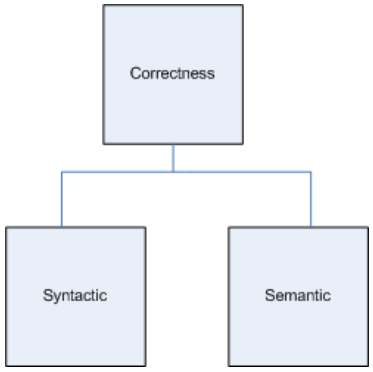


Figure 25: Summary of correctness issues

3.5.2 Process model evolution issues

Version management of process models

Maintaining an overview of all the changes over time of a process model requires the management of all the different versions of that process model:

- The changes must be logged.
- The different versions must be stored.

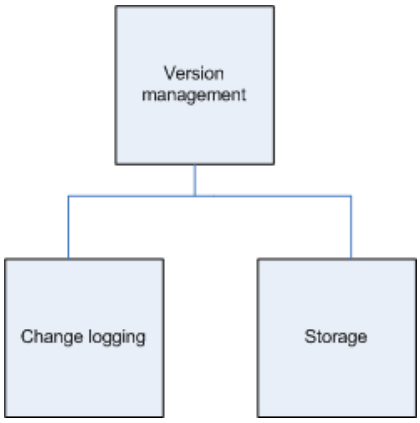


Figure 26: Summary of version management issues

Change management of process models

Assessing the impact, tracking and propagating changes must be done simply and effectively. This essential problem description is fairly similar to the problem description stated by Jaccheri and Conradi [39]: "*Solving the problem of process model evolution requires an answer to the following questions: which process model fragments should be changed, how and when? And how to analyze and guide change?*".

Process changes can be classified into two categories as was suggested in the subchapter "Process variance description and definition (3.2)":

- Ad hoc changes (exceptional events).
- Changes required because of the reengineering of the business process.

They must thus be handled differently.

Furthermore many authors make the distinction between process changes happening at the instance or process type (Schema [29]) level:

Weber, Rinderle, Wild and Reichert suggest the following: "*On the other hand, it provides support for adaptive processes at both the process instance and the process type level. Changes at the instance level may affect single process instances and be performed in an ad-hoc manner, e.g., to deal with exceptional or unanticipated situations. Process type changes, in turn, can be applied to adapt the PAIS to business process changes. In this context, concurrent migration of hundreds up to thousands of process instances to the new process schema may become necessary* [40]".

Chou and Chen state the following: "*Generally, process evolution research can be roughly classified into two categories: (1) those collecting and analyzing data, and then evolving processes according to analysis results; and (2) those supporting process program evolution during enactment* [41]."

However when modeling process variability and thereby also enabling its simple and effective change management, making the distinction between process changes at the instance and schema level complicates the achievement of this task unnecessarily. It is easier to concentrate first only on modeling process variability and afterwards its effective change management without making the distinction between process changes at the instance or process type level. Once the research goal has been achieved, it becomes very interesting to apply these change management schemes at the instance and process type level.

Secure change management falls out of the scope of this research project [14]: access control shall not be researched in this research project.

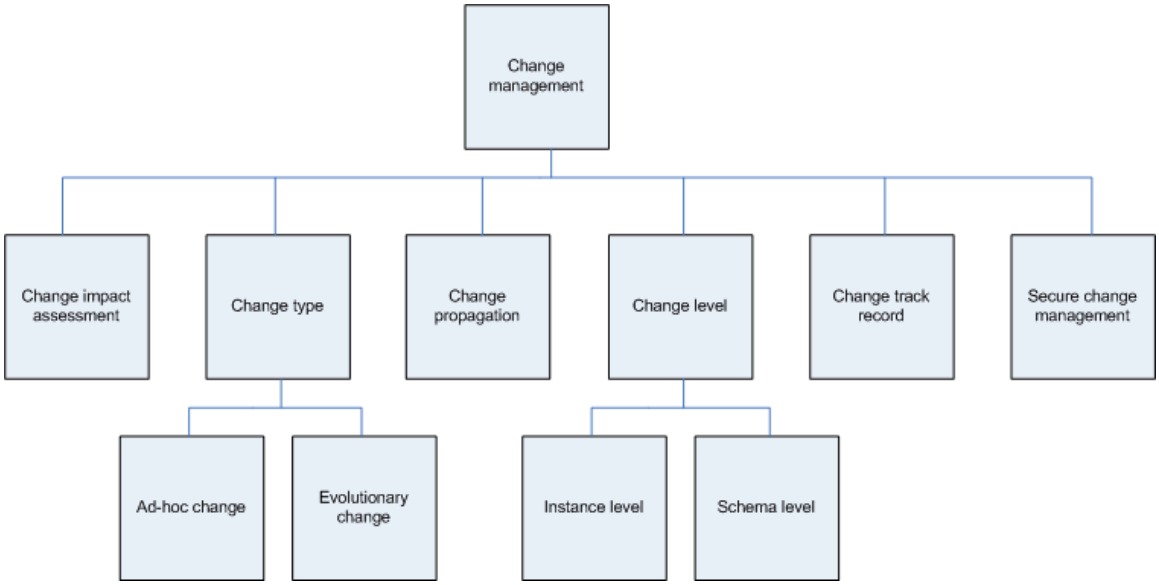


Figure 27: Summary of change management issues

Storage of process models

All the process models must be stored, retrievable and classified. Furthermore the security and integrity of the process models must be guaranteed. This issue has already been described previously in more detail.

NB: See process variability modeling issues for more information.

Correctness of process models

Finally changes must preserve the syntactical and semantic correctness of process models [29, 30]. This issue has already been described previously in more detail.

NB: See process variability modeling issues for more information.

3.6 The rewards for solving business process variability problems

The rewards for solving process variability problems have theoretical and practical implications. From a theoretical perspective, the reward for solving these problems is making a meaningful contribution to the academic world and resolving an open problem.

From a practical perspective, any industry or organization dealing with process variability or having problems managing process variability could benefit from the solutions to these problems:

- Improved control over the variability of a process.
- Improved process variability modeling.
- Improved automation of variable and changing processes.
- Improved process evolution.

For example, solving these problems would enable the design of mass customization systems with flexible processes [42], which would increase product variety and thereby greater customer satisfaction [43]: "*In reality customers are often willing to pay premium price for their unique requirements being satisfied, thus giving companies bonus profits (Roberts and Meyer, 1991). From an economic perspective, mass customization enables a better match between the producers' capabilities and customer needs*".

Furthermore, this could enable the creation and design of configurable and evolvable reference process models or enterprise resource management systems (ERP). Some authors have attempted their design [4, 32-36, 44].

Finally, process variability also occurs in medical guidelines or pathways. Usually formal procedures are specified in order to modify the medical pathway, if it cannot be followed. Being capable of modeling process variability could lead to the creation of improved medical pathways or clinical guidelines. For example, an MRI pathway could be improved by integrating process variants into the variance tracking record [45]. It could also lead to the effective modeling and generation of personalized treatment plans [24].

3.7 Problem focus

This research project is constrained by time and therefore requires narrowing its research focus. Furthermore choosing a research focus shall help raise the depth and the quality of the research results. As described in *Figure 22*, process variability problems can be reduced to the following problems:

- Process model evolution
- Process variability modeling

However process variability modeling must be done before process model evolution. Process model evolution cannot be implemented successfully, if there isn't a clear approach available to design process models for variable processes. Therefore the focus of this research project shall be on modeling process variability first, leaving out of scope their storage issues.

3.8 Conclusion

To resume this chapter, the notion of process variability has been defined. Two types of process variability have been identified:

- Process variability within the domain space
- Process variability over time

These two types of process variability respectively lead to the following main problems:

- Process variability modeling
- Process model evolution

The rewards for solving these problems are both theoretical and practical: enabling the mass customization paradigm, configurable and evolvable reference process models, personalized treatment plans, etc. The focus of this research project shall be on process variability modeling because designing process models for variable processes must be achieved first to enable their evolution. Current solutions to process variability modeling have yet to be assessed to determine their strength and weaknesses.

Chapter 4 Business process variability modeling evaluation

4.1 Introduction

Before crafting new solutions to the process variability modeling problems (Chapter 6), the limitations of current business process modeling languages (BPMLs) must be known. To assess the concepts provided by current BPMLs to model process variability a set of evaluation criteria will be needed. An overview of current available BPML will also be provided. Event driven process chains (EPC), the Business Process Modeling Notation (BPMN) and Configurable EPC (C-EPC) were the three BPMLs selected and evaluated in this.

4.2 Business process variability modeling evaluation criteria

Several different evaluation frameworks have been created to evaluate business process modeling languages [46-49]. However they are often evaluated on their capacity to implement or model workflow patterns [50-53]. After a broad literature analysis no framework or evaluation criteria have been found to evaluate the suitability of business process modeling languages with the goal to model process variability within the domain space.

Thus primarily the healthcare running example shall be used to assess the capacity of the BPML to model process variability within the domain space. Secondly a set of evaluation criteria shall be used to evaluate the modeling concepts provided by a BPML when modeling process variability.

4.2.1 Listing and description of evaluation criteria

Evaluating the capability of BPMLs to model process variability within the domain is challenging. Evaluating BPMLs with the goal to model business process variability within the domain can be done in several ways. The meta model of the BPML could be compared. However this is not a viable choice because the meta model very often just consists of a UML class diagram. Comparing UML class diagrams with each other will probably not lead to valuable insights.

The modeling concepts or language constructs provided by a BPML shall be evaluated to determine their effectiveness when modeling process variability. A BPML must thus be evaluated on its capacity to model or represent explicitly process variability.

As was shown in section 3.3.2, the origin of process variability within the domain space is caused by the variability of the domain space itself. It thus logical that to enable the simple and effective modeling of business process variability, a BPML must have the capacity:

- To model the domain of a business process.
- To specify the impact of the variability of the domain space on the business process model itself. Thus requiring the following:
    - Business process models must thus be adaptable or configurable in order to visualize the impact of the variable domain space (EC1, EC2, EC4, EC5, EC7)
    - Configuration rules are necessary to specify the impact of the domain space on business process models (EC3, EC6).
    - The consistency of the configuration rules as well as the correctness of the process models must be guaranteed (EC8, EC9).

These requirements lead to the evaluation criteria EC1, EC2, EC3, EC4, EC5, EC6, EC7, EC8 and EC9. Every BPML shall be evaluated in terms of *strengths* and *weaknesses* relatively to the evaluation criteria here under:

- **EC1**: The possibility to mark, distinguish or visualize variable elements of a process model.

- **EC2**: Variable elements of a process model must support a subset of the change patterns specified by Weber, Rinderle And Reichert [54]:
  - Adaptation patterns (AP)
    - **AP1**: Insert process fragment
    - **AP2**: Delete process fragment
    - **AP3**: Move process fragment
    - **AP4**: Replace process fragment
    - **AP5**: Swap process fragment
    - **AP6**: Extract sub process
    - **AP7**: Inline sub process
    - **AP8**: Embed process fragment in loop
    - **AP9**: Parallelize process fragment
    - **AP10**: Embed process fragment in conditional branch
    - **AP11**: Add control dependency
    - **AP12**: Remove control dependency
    - **AP13**: Update condition
  - Patterns for predefined changes (PP)
    - **PP1**: Late selection of process fragments
    - **PP2**: Late modeling of process fragments
    - **PP3**: Late composition of process fragments
    - **PP4**: Multi-Instance activity

  NB: "*A process fragment can either be an atomic activity, an encapsulated sub process or a process (sub) graph*". Furthermore Weber, Rinderle and Reichert specify that the change patterns happen at the instance or type level. In this research project this distinction is not relevant as was explained clearly in section 3.5.2.

- **EC3**: The possibility to specify conditions or configuration rules that guide the adaptation or configuration of a process model.

- **EC4**: The possibility to visualize conditions or configuration rules that guide the adaptation or configuration of the process model.

- **EC5**: The possibility to model and visualize how the domain affects the configuration of the process model.

- **EC6**: The possibility to specify conditions or configuration rules that guide the configuration of a process model based on the configuration of its domain space.

- **EC7**: The user has the possibility to display selectively the process models of process variants. This requires the adaptation of the process model either:
  - Automatically (software tool)
  - Manually

- **EC8**: Syntactic and semantic correctness of the process models must be ensured.

- **EC9**: Consistent domain and process model configurations must be ensured.

4.2.2 Summary of evaluation criteria

The framework of evaluation criteria used to assess the business process modeling languages are summarized here under:

- **EC1**: Mark variable elements
- **EC2**: Support of change patterns
- **EC3**: Configuration rules that adapt process model
- **EC4**: Visualization of configuration rules that adapt process models
- **EC5**: Domain visualization and process model configuration
- **EC6**: Domain and process configuration rules
- **EC7**: Selective display
- **EC8**: Correctness
- **EC9**: Consistency

4.3 Selection and evaluation of current business process modeling languages

Evaluating all the BPMLs available to determine how good they are at modeling process variability within the domain space is not feasible. A selection of the BPMLs to be evaluated must thus be made. Selecting the most successful and renowned BPMLs available for this assessment is a good first choice.

A distinction can be made between BPMLs that are non-configurable and configurable. Dumas, van der Aalst and ter Hofstede described three main non-configurable BPMLs in their book "Process-Aware Information Systems" [3]:

- UML Activity diagrams
- Event-driven process chains (EPC)
- Petri nets

However there are countless variants of Petri nets: Elementary Net System, workflow nets, business procedure nets, high level Petri nets (time, data, hierarchy), reconfigurable workflow nets (ad hoc changes), Object oriented Petri nets, Modular process nets, Higher-Order Object net, Business Process Petri nets, etc [55, 56]. Considering the heavy timing constraints of this project and their similarity with EPC, Petri nets shall not be evaluated in this research project. Furthermore again because of the timing constraints, only UML activity diagrams or EPC shall be evaluated. While UML activity diagrams are suitable for modeling business processes, EPC are meant to model business processes. Additionally EPC are widely used and accepted in practice: most likely it is their simplicity that lead to their successful adoption and wide use in practice [3]. EPC shall thus be evaluated in this research project and not UML activity diagrams.

The Business Process Modeling Notation (BPMN) is a non-configurable BPML developed by the Business Process Management Institute and is also a viable choice because of its newness and innovativeness. There are many other non-configurable BPMLs available (IDEF, CISMOSA, RAD, etc), however because of timing constraints they shall not be evaluated in this research project.

Main configurable BPMLs are:

- Configurable reference process models such as the Configurable Event-driven Process Chains (C-EPC) [4, 5, 35, 57-59].
- Questionnaire-driven configuration of reference process models [32-34].

However here again because of timing constraints only one of these configurable BPMLs shall be evaluated in this research project. The configurable reference process models shall be evaluated in this research project.

Miscellaneous other efforts have been developed to model process variability, however because of timing constraints they shall not be considered. A good example is VBP developed by King and Johnson [9], they model process variability and best practices using specialization, use cases, patterns and the unified modeling language (UML). Finally, to solve process variability modeling problems, Jiao, Zhang and Prasanna [31] suggest the use of generic process structures from which variants can be derived systematically. To achieve this end, they make use of object-oriented Petri nets, colored Petri nets and Petri nets with changeable structures.

In this research project shall thus be evaluated:

- Extended EPC one the most used and accepted BPML.
- BPMN because of their newness and innovativeness.
- C-EPC because of their configurable nature.

These three BPMLs form a good population to assess the current capability of BPMLs to model process variability within the domain space.

4.3.1 Business process variability modeling evaluation of extended event driven process chains (E-EPC)

The healthcare running example described in section 3.4 was already illustrated using basic EPC. However the healthcare running example shall now be illustrated using the extended EPC (E-EPC) notation. All the process models have been remodeled using E-EPC, except for the process model described in Figure 20; this was not done because it would simply result in a big and incomprehensible process model.

E-EPC provide no explicit means to model process variability. There are only two choices available, modeling the process variants using one big process model or distinct and separate process models (Figure 28, Figure 29 and Figure 30) using simple control flow patterns (split, merge, etc) [60].

It is clear that modeling these three different processes using one process model would result in a big and difficult to comprehend model: this can be observed in Figure 20. By modeling the three different processes into one process model, some determinism has also been lost:

- Observing Figure 20, it is not clear when an ambulance should be scheduled or not. In this process model, an ambulance could be scheduled for a patient without disabilities, which is unnecessary.
- Escorting a patient to the examination room should normally only be done for blind and paralyzed patients. However in this process model, it can be done indiscriminately for a patient with or without disabilities.
- Escorting a patient to the radiology entrance should normally only be done in the case of blind patients. Nevertheless, this function can be executed indiscriminately for patients without or with disabilities

- Escorting a patient back the ambulance must only be done for paralyzed patients. Nonetheless, this function can be executed for all patients in this process model.

The reason behind the loss of determinism or accuracy in this process model is a design choice to simplify the process model: adding more detail to this process model would have made it even more complex and unreadable.

There is therefore only one viable choice when modeling process variability using E-EPC: modeling the process variants using distinct and separate process models. For each type of patient, a respective process model has been designed:

- Without disability (Figure 28)
- Blind (Figure 29)
- Paralyzed (Figure 30)

Following this modeling choice, the process models will probably be easier to comprehend. However, in the case of a large number of process variants, process model evolution becomes challenging because a large number of process models needs to be maintained. The main difference between these three process models can be described in terms of change patterns: events, functions, outputs or organizational units have been added, deleted or modified. Here process variability within the domain space has been illustrated along only 1-dimension of variability: three different patients (with disabilities, blind, paralyzed) leading to three different process models. Considering for example a process with i inputs and o outputs, this process varies along 2-dimensions and has a maximum of i*o process variants.

Modeling the three different healthcare processes into one process model results in design choices that usually diminish the precision and determinism of the process models in return for simple and understandable process models: this was illustrated in Figure 20. Modeling the process variants using separate and distinct process models increases the understandability, simplicity and accuracy of the process models. However in a case of a large number of process variants, process model evolution becomes challenging because a large number of process models need to be evolved and maintained.

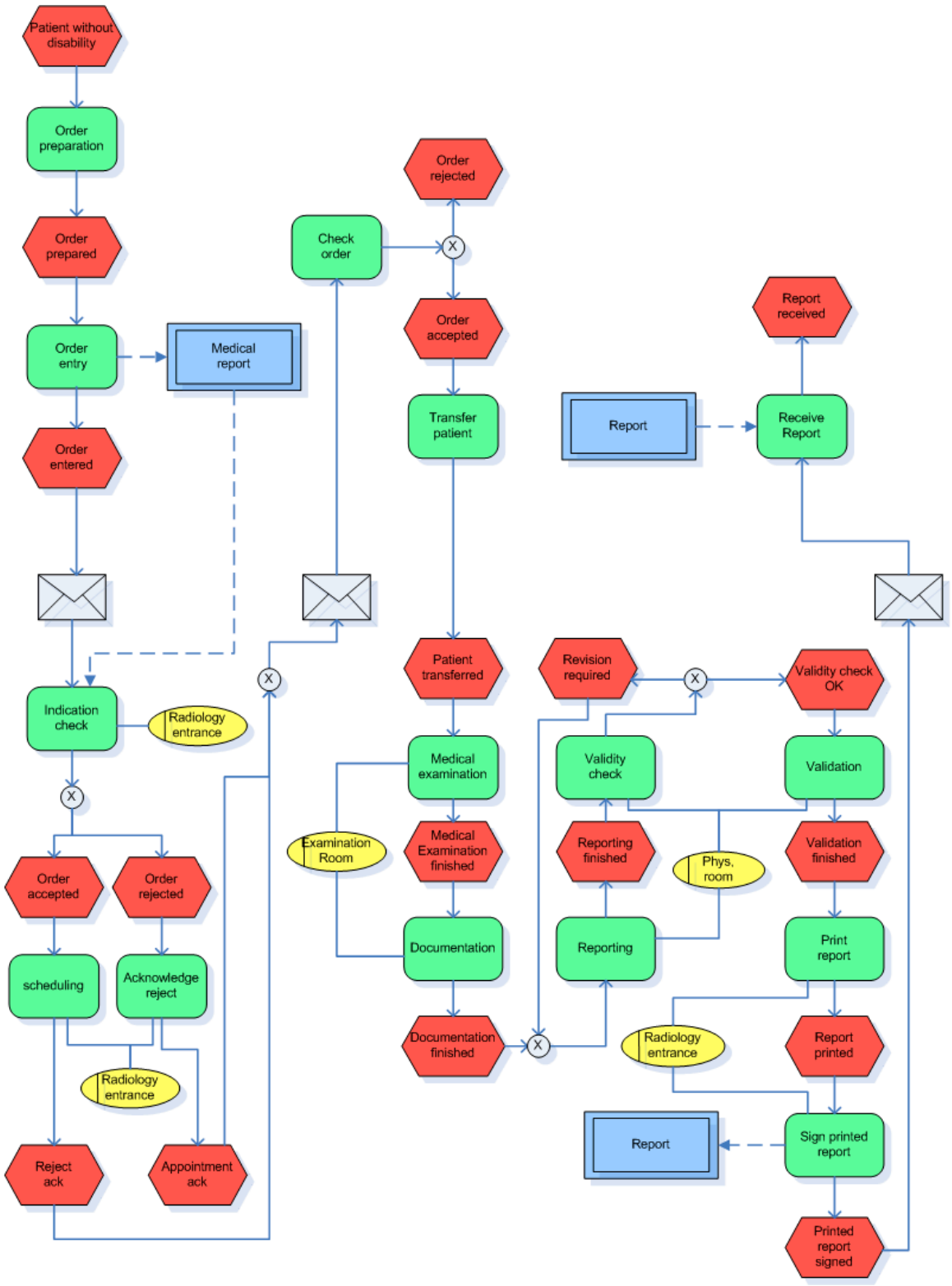


Figure 28: Patient without disabilities needs radiology

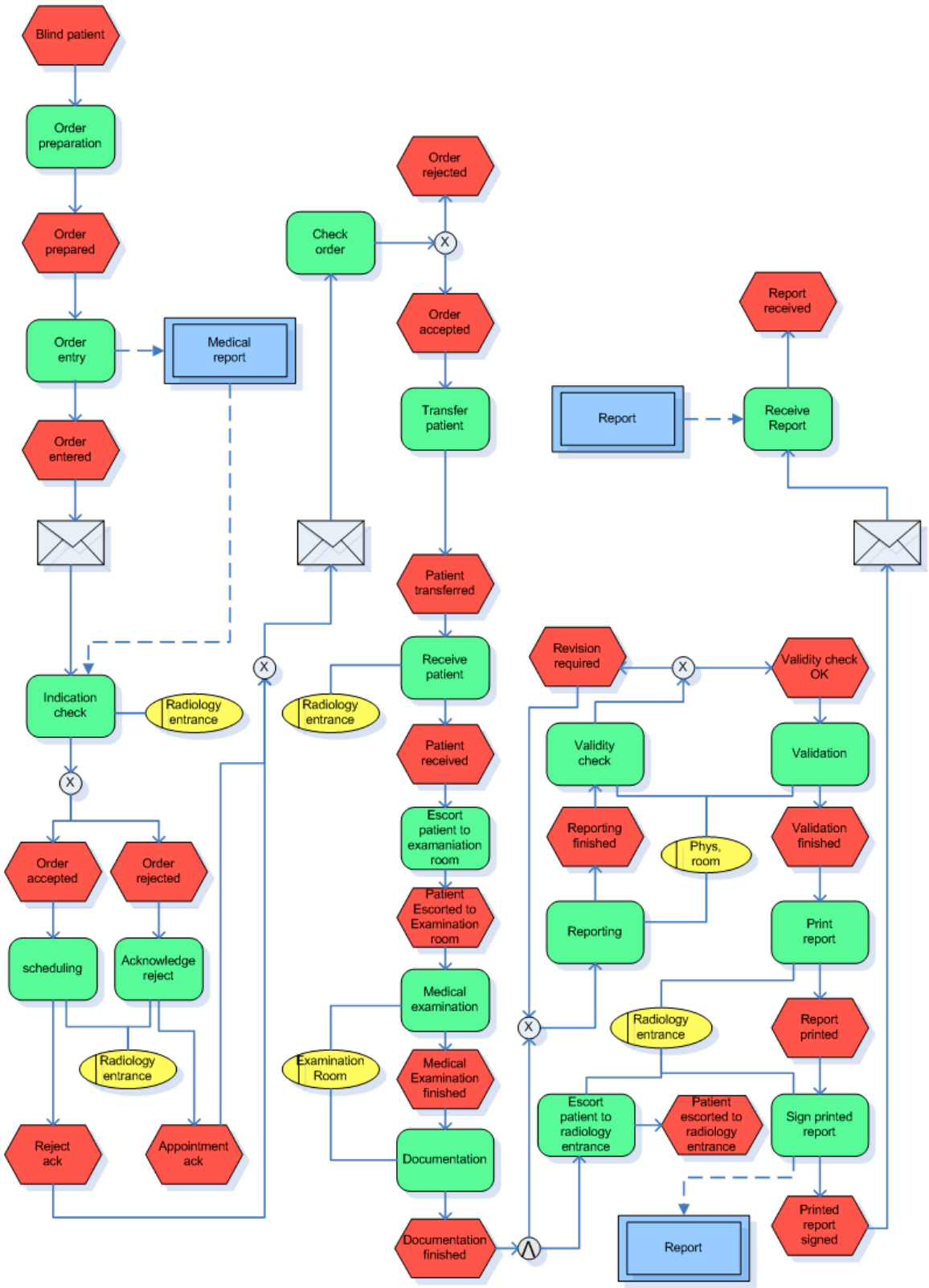


Figure 29: Blind patient needs radiology

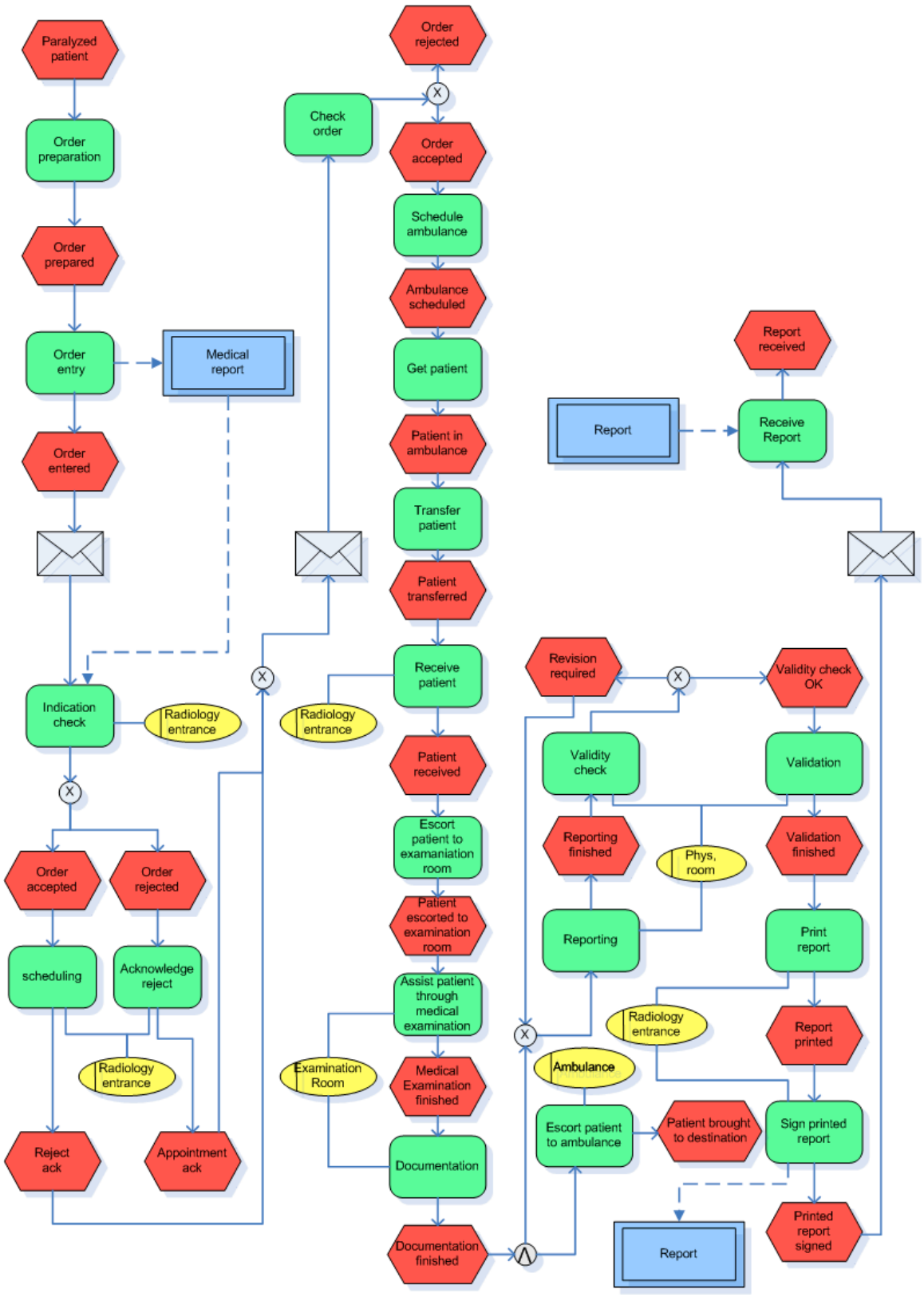


Figure 30: Paralyzed patient needs radiology

EC1: Mark variable elements
N/A

Strengths
N/A

Weaknesses
Not being able to mark variable elements of E-EPC is a serious weakness.

EC2: Support of change patterns
N/A

Strengths
N/A

Weaknesses
Not supporting any change patterns is a serious weakness of E-EPC.

EC3: Configuration rules that adapt process model
N/A

Strengths
N/A

Weaknesses
Not supporting the specification of configuration rules that adapt the E-EPC process models is a serious weakness.

EC4: Visualization of configuration rules that adapt process models
N/A

Strengths
N/A

Weaknesses
N/A

EC5: Domain visualization and process model configuration
Some elements of the domain (organizational unit, output, machine, etc) can be modeled directly into the E-EPC process models. However it is not possible to specify how these elements of the domain have an impact on the configuration of the process model.

Strengths
Some aspects of the domain can be modeled.

Weaknesses
Using E-EPC it cannot be modeled or visualized how domain impacts the configuration of the process model.

EC6: Domain and process configuration rules
N/A

Strengths
N/A

Weaknesses
No domain and process configuration rules can be specified.

EC7: Selective display
N/A

Strengths
N/A

Weaknesses
Selective display is only possible when modeling process variants using distinct and separate process models. To view the process model of the desired process variant, the appropriate process model needs to be selected and viewed.

EC8: Correctness
Eleven and simple design rules can be followed to model correct control flow and avoid problematic behavior such as deadlocks [3]. E-EPC can also be extended with formal concepts (Petri-Nets [61-64]) to ensure the syntactic or semantic correctness of the process models [65, 66].

Strengths
EPC can be extended with formal concepts to ensure and verify the correctness of EPC process models. Furthermore simple design rules can be followed to ensure the correctness of process models.

Weaknesses
Only the syntactic correctness can be verified using the earlier mentioned eleven design rules; this is time consuming, especially in the case of big process models.

EC9: Consistency
N/A

Strengths
N/A

Weaknesses
N/A

4.3.2 **Business process variability modeling evaluation of Business Process Modeling Notation (BPMN)**

Using BPMN process variability can only be modeled using one process model or separate and distinct process models. As shall be shown in a little later in the evaluation, BPMN offers almost no means to model process variability.

Using BPMN, modeling the healthcare running example using one process model will result in a big and incomprehensible process model. Thankfully by making creative use of swimlanes and using sub-processes this big process model can be made more comprehensible. The resulting process model is a big process model separated into clear and distinct sub process models as illustrated by Figure 31: this modeling choice is very similar to modeling the different cases using distinct and separate process models. In Figure 31, are not modeled or described the sub-processes. Another process modeling alternative available is again using swimlanes with only one process model (Figure 32): using this modeling alternative, it becomes visually clear what processes are generic or are specific to a case. Finally, modeling the healthcare running example using separate and distinct process model is always a good choice.

The healthcare running example can also be modeled using distinct and separate process models (Figure 33, Figure 34, Figure 35). This results in more detailed process models. Sadly this approach significantly increases maintenance overhead of the process models, especially in the case of a great number of process variants.

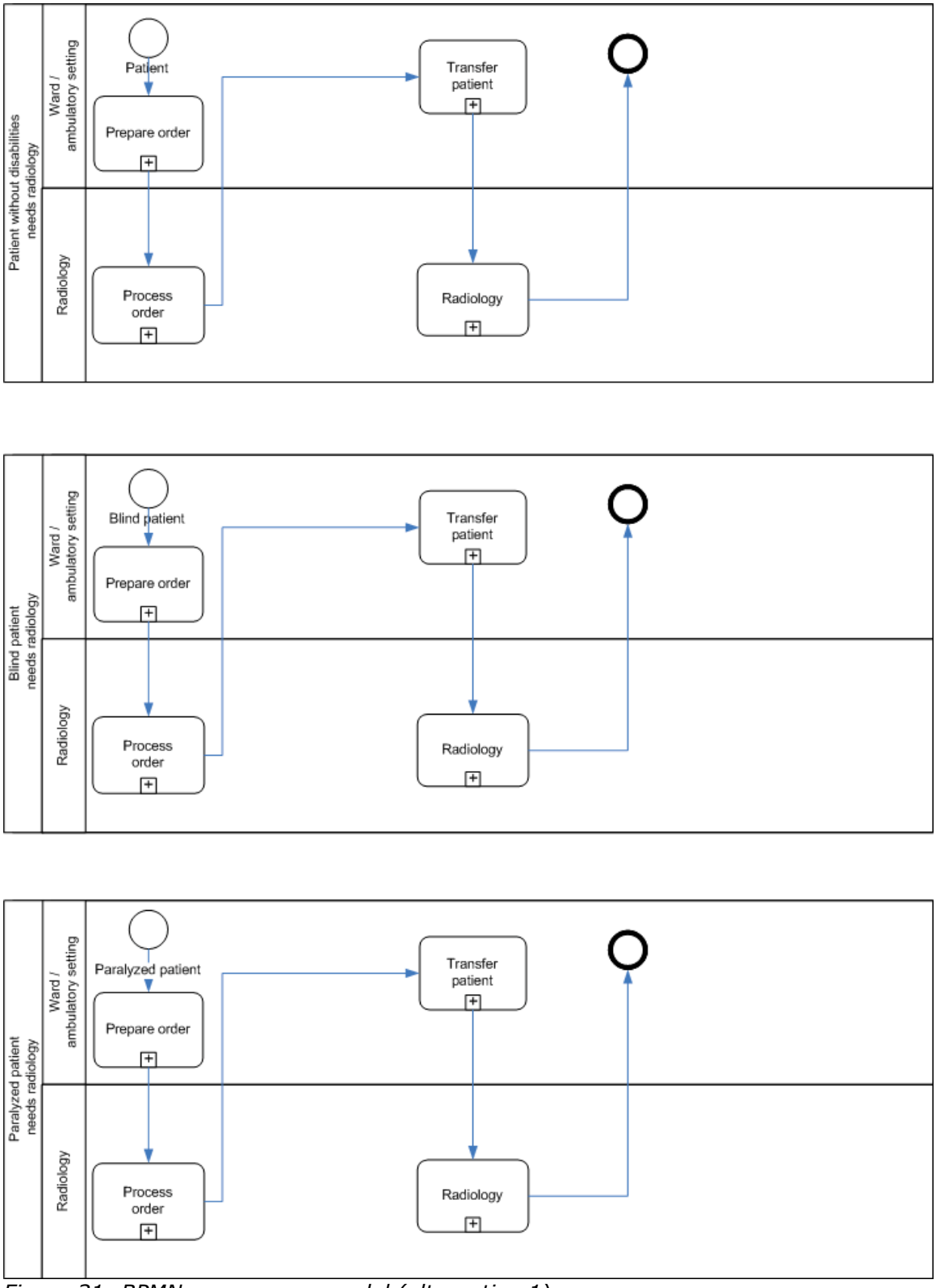


Figure 31: BPMN one process model (alternative 1)

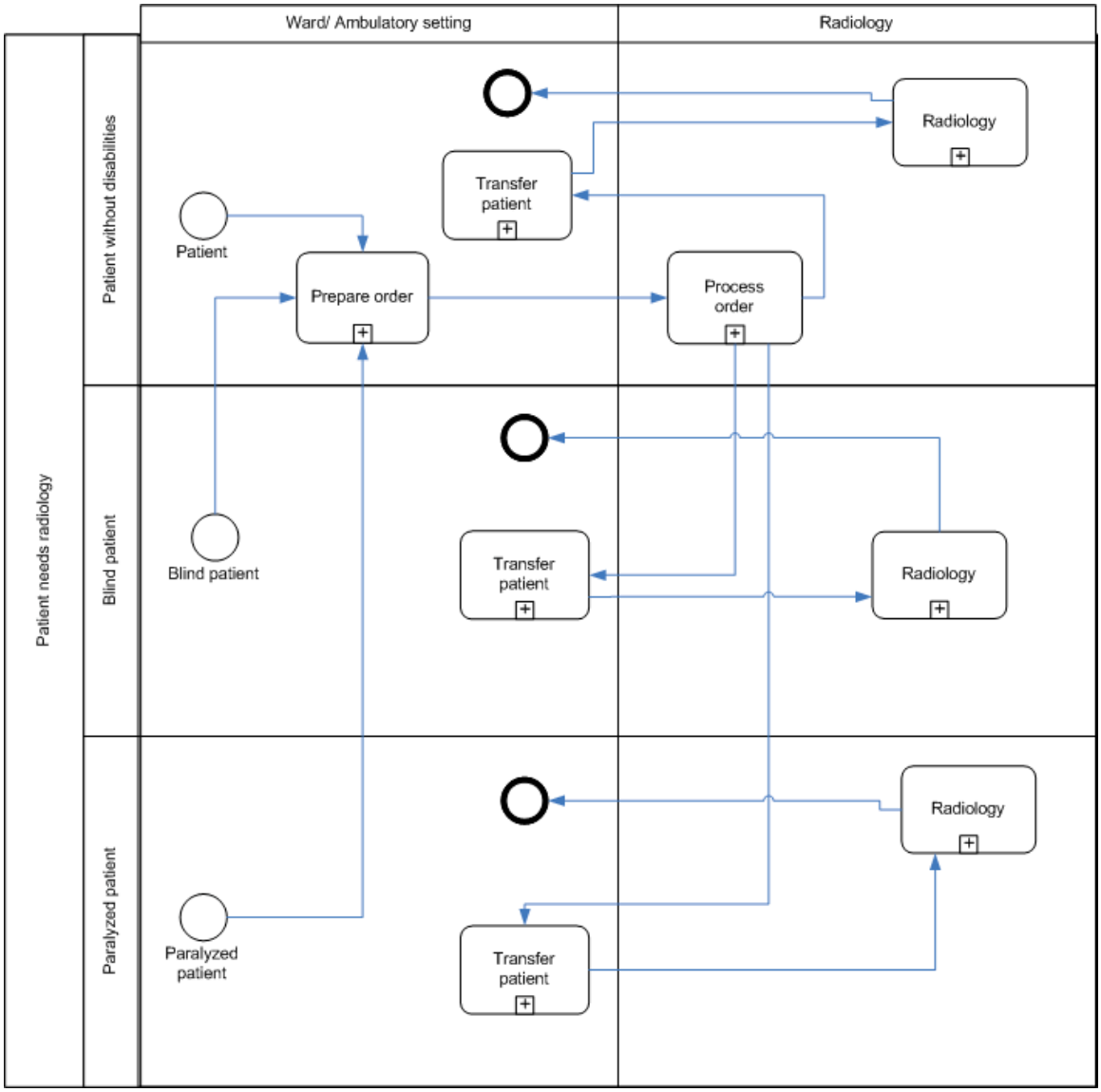


Figure 32: BPMN one process model (alternative 2)

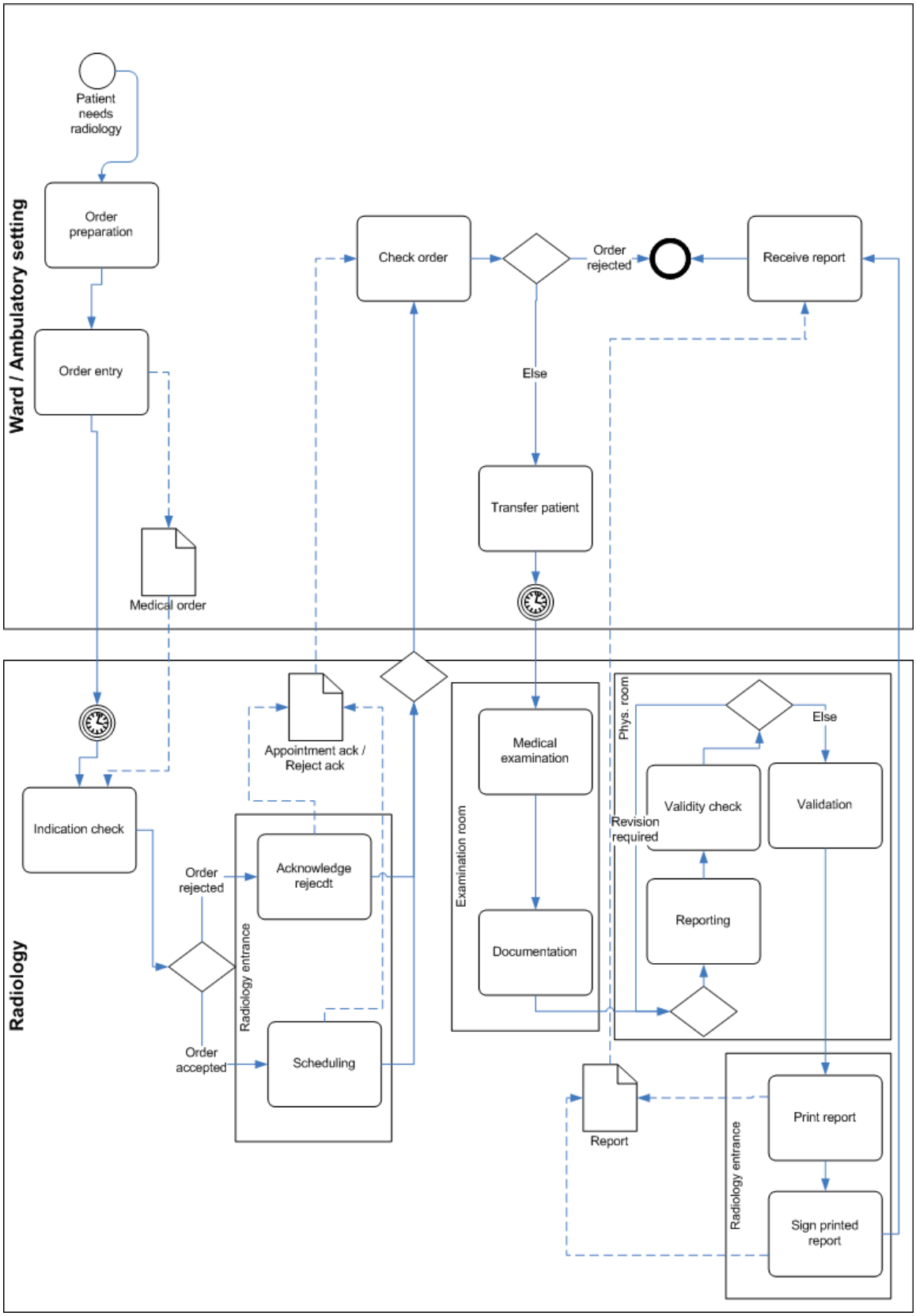


Figure 33: Radiology process with patient without disabilities

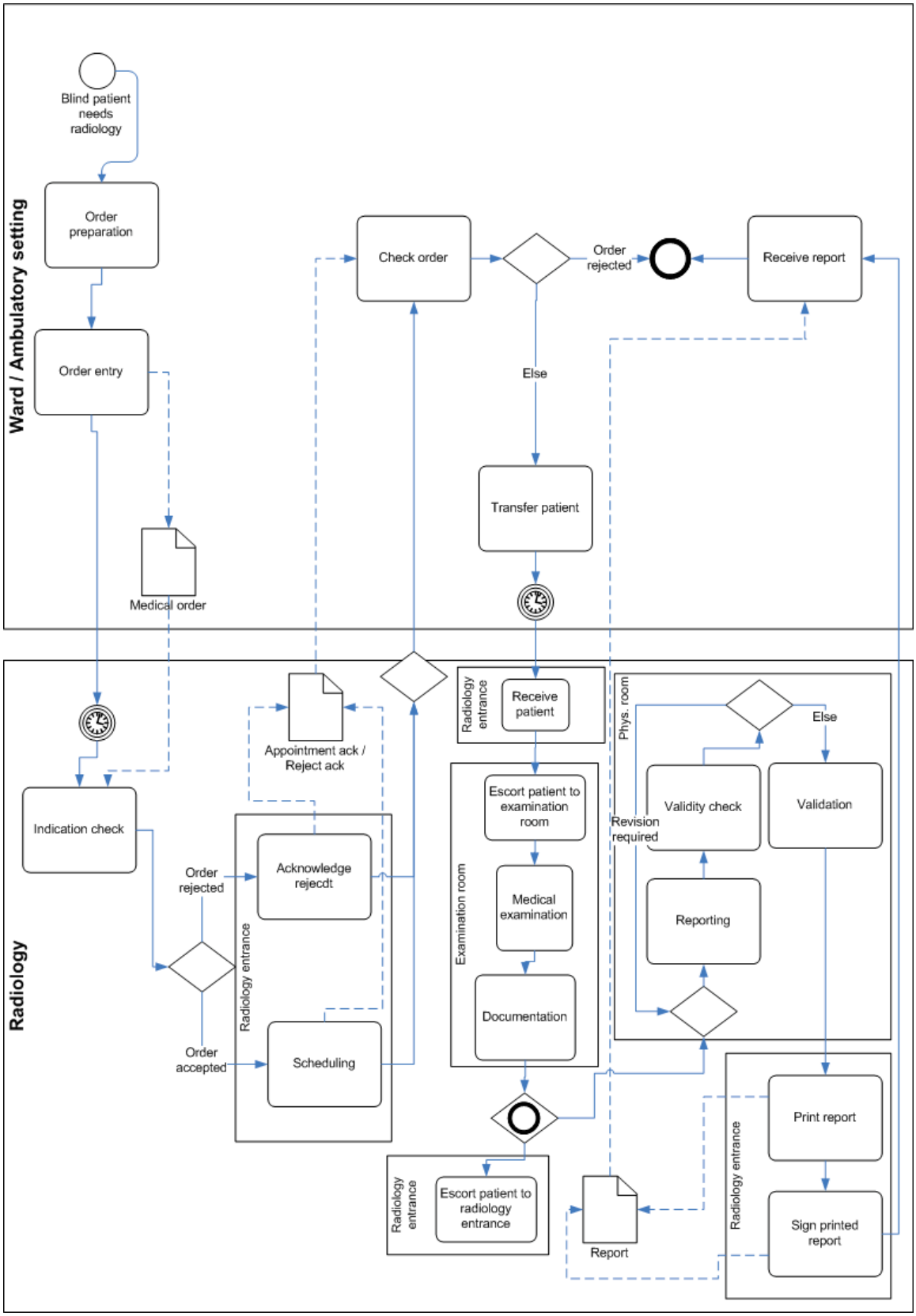


Figure 34: Radiology process with blind patient

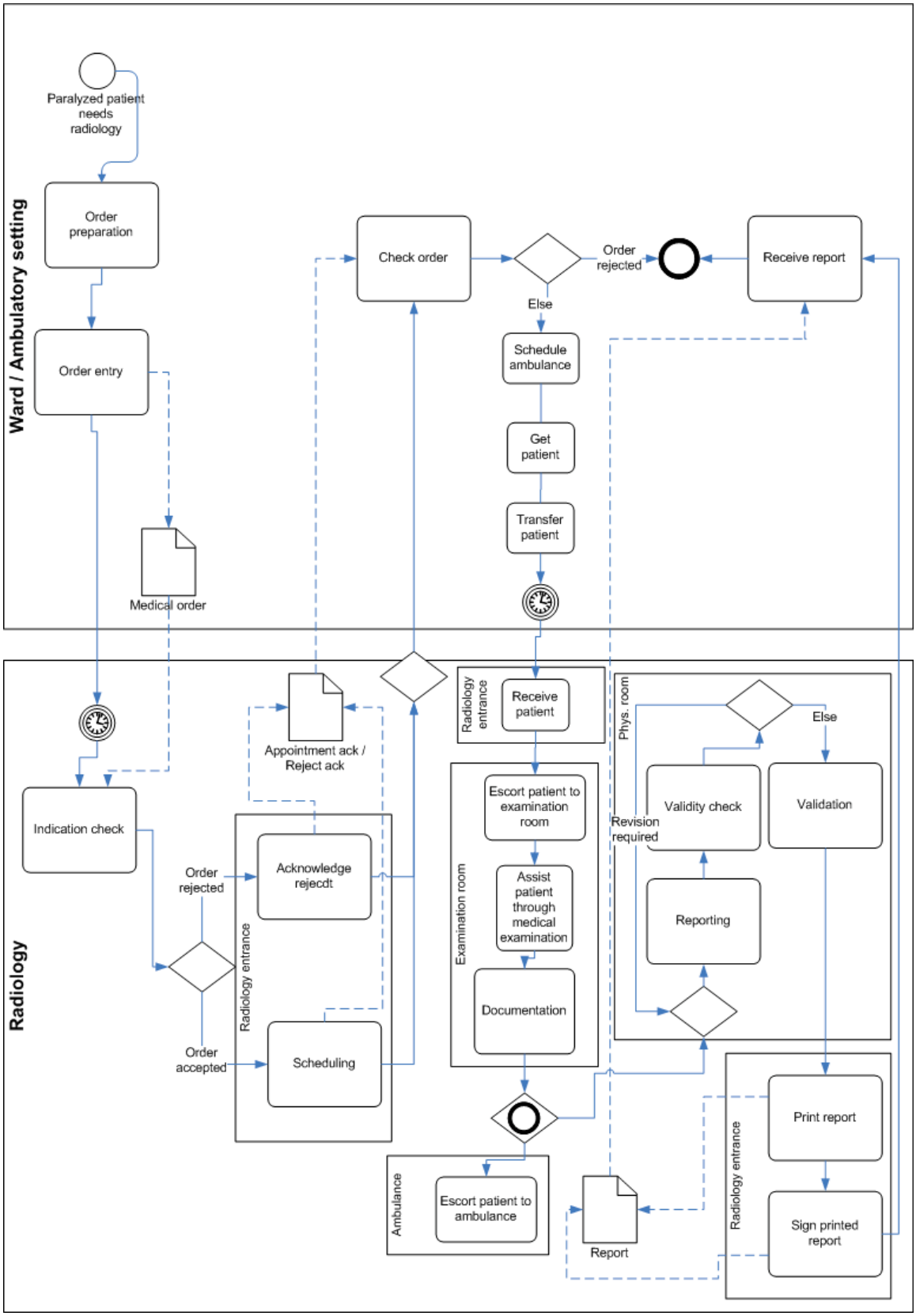


Figure 35: Radiology process with paralyzed patient

EC1: Mark variable elements
N/A

Strengths
N/A

Weaknesses
BPMN does not support the marking of variable elements.

EC2: Support of change patterns
N/A

Strengths
N/A

Weaknesses
BPMN does not support any change patterns.

EC3: Configuration rules that adapt process model
N/A

Strengths
N/A

Weaknesses
BPMN does not support the specification of configuration rules that adapt the process models.

EC4: Visualization of configuration rules that adapt process models
N/A

Strengths
N/A

Weaknesses
N/A

EC5: Domain visualization and process model configuration
Some aspects of the domain can be modeled using swimlanes.

Strengths
Using swimlanes creatively, aspects of the domain space with an impact on process variability can be modeled in BPMN.

Weaknesses
BPMN does not support the visualization of the domain and the respective process configurations.

EC6: Domain and process configuration rules
N/A

Strengths
N/A

Weaknesses
BPMN does not support the specification of configuration rules that dictate how the domain space of a business process affects the configuration of its process model.

EC7: Selective display
N/A

Strengths
N/A

Weaknesses
BPMN does not support explicitly selective display of process models. However process variants can be modeled using separate and distinct process models. Viewing the desired process model is equivalent to selecting the appropriate process model.

EC8: Correctness
BPMN does not provide any explicit means to verify the correctness of BPMN process models. However Dijkman, Dumas and Ouyang claim that by formalizing and mapping a subset of BPMN to Petri nets, the semantic correctness of these BPMN process models can be verified [67]: at least their soundness and liveliness. Raedts, Petkovic, Usenko, van der Werf, Groote and Somers claim that a subset of BPMN can indeed be translated into Petri nets and analyzed using tools such as Yasper, Woflan, Ina and Lola permitting the verification of the soundness of the BPMN process models, the detection of deadlocks, etc [68].

Strengths
The syntactic and semantic correctness of the BPMN process models can be verified.

Weaknesses
The syntactic and semantic correctness of the process models can only be verified using tools.

EC9: Consistency
N/A

Strengths
N/A

Weaknesses
N/A

4.3.3 Business process variability modeling evaluation of Configurable event driven process chains (C-EPC)

Using C-EPC, the healthcare running example is modeled into one process model (Figure 37, Figure 39, Figure 40). Modeling processes with C-EPC requires a significant amount of time, thinking and creativity. When business processes are modeled using C-EPC, two options are available:

- Modeling the healthcare running example with or without configuration guidelines and requirements. When modeling the healthcare running example without configuration guidelines and requirements, the resulting process model lacks determinism because it is not clear what parts of the process model should be hidden or blocked (Figure 37). Moreover the configurable process model allows process configurations that should not be allowed: a patient without disabilities does not need an ambulance (Figure 38). Configuration requirements are thus needed to avoid undesirable process model configurations. These configuration guidelines and requirements do have some drawbacks, there is no clear syntax to specify them. Although Rosemann and van der Aalst specify that the configuration requirements should be specified using logical expressions, it is not clear if first order logic or predicate logic should be used. Furthermore a C-EPC was illustrated with a guideline containing the following logical expression (Figure 36):

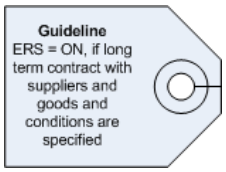


Figure 36: C-EPC configuration guideline example [5]

  The logical expression is extended here with a programming like syntax: a conditional if statement. In the healthcare running example, the start events guide the configuration process, they are included into the configuration requirements using conditional if statements.

- Modeling the healthcare running example with only configurable connectors and configuration requirements, or configurable nodes and configuration requirements. It has been possible to model the healthcare running example using only configurable connectors and configuration requirements (Figure 39). In Figure 40, the healthcare running example has been modeled using configurable nodes and configuration requirements. The configuration requirements have been used to guide the configuration process.

Modeling the healthcare running example using C-EPC leads to big, complex and configurable process models (Figure 39, Figure 40). The instances of these configurable process models result in process models that are similar to process models where the healthcare running example has been modeled using separate and distinct EPC process models (Figure 17, Figure 18, Figure 19).

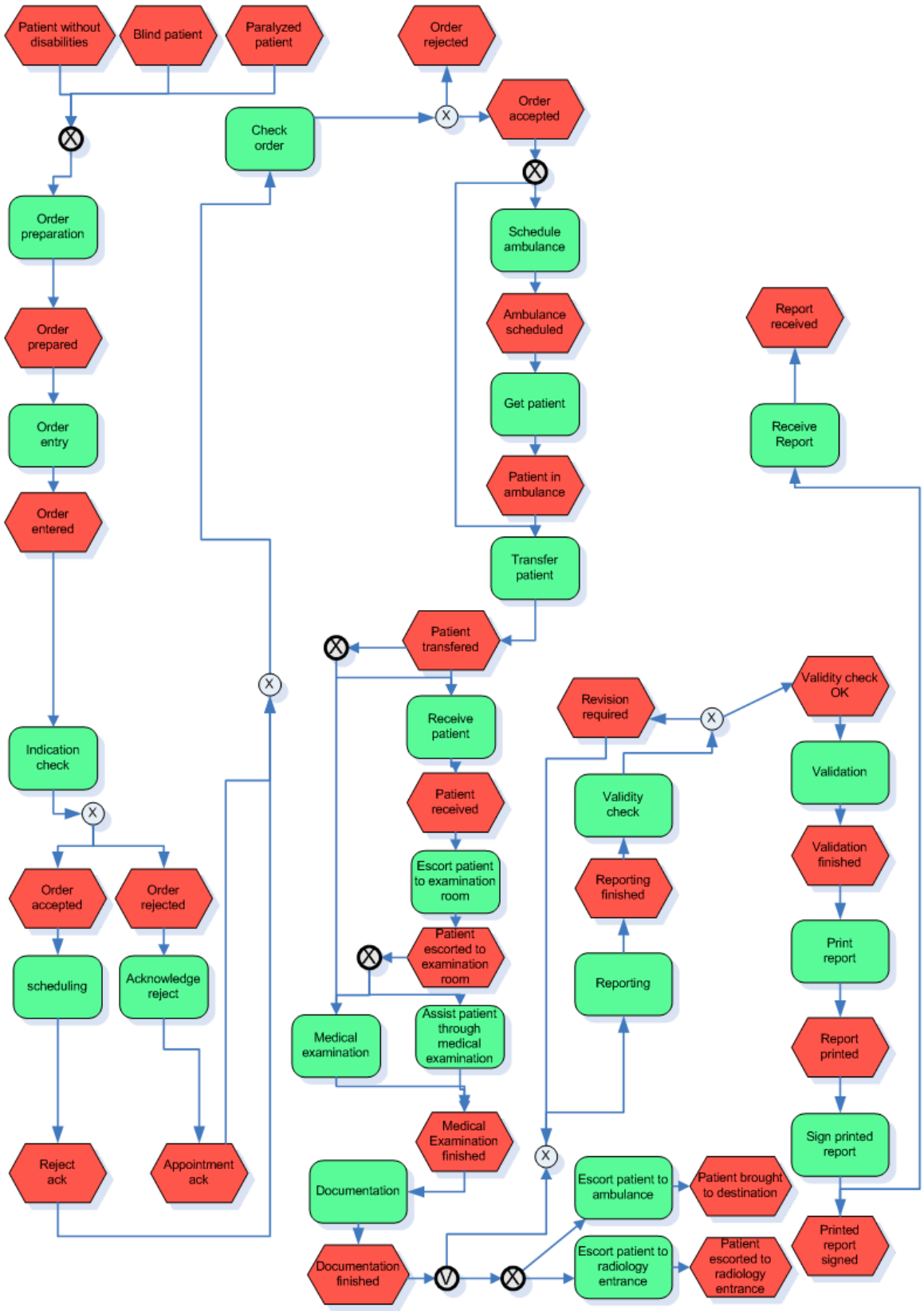


Figure 37: Healthcare running example modeled using C-EPC without configuration guidelines and requirements

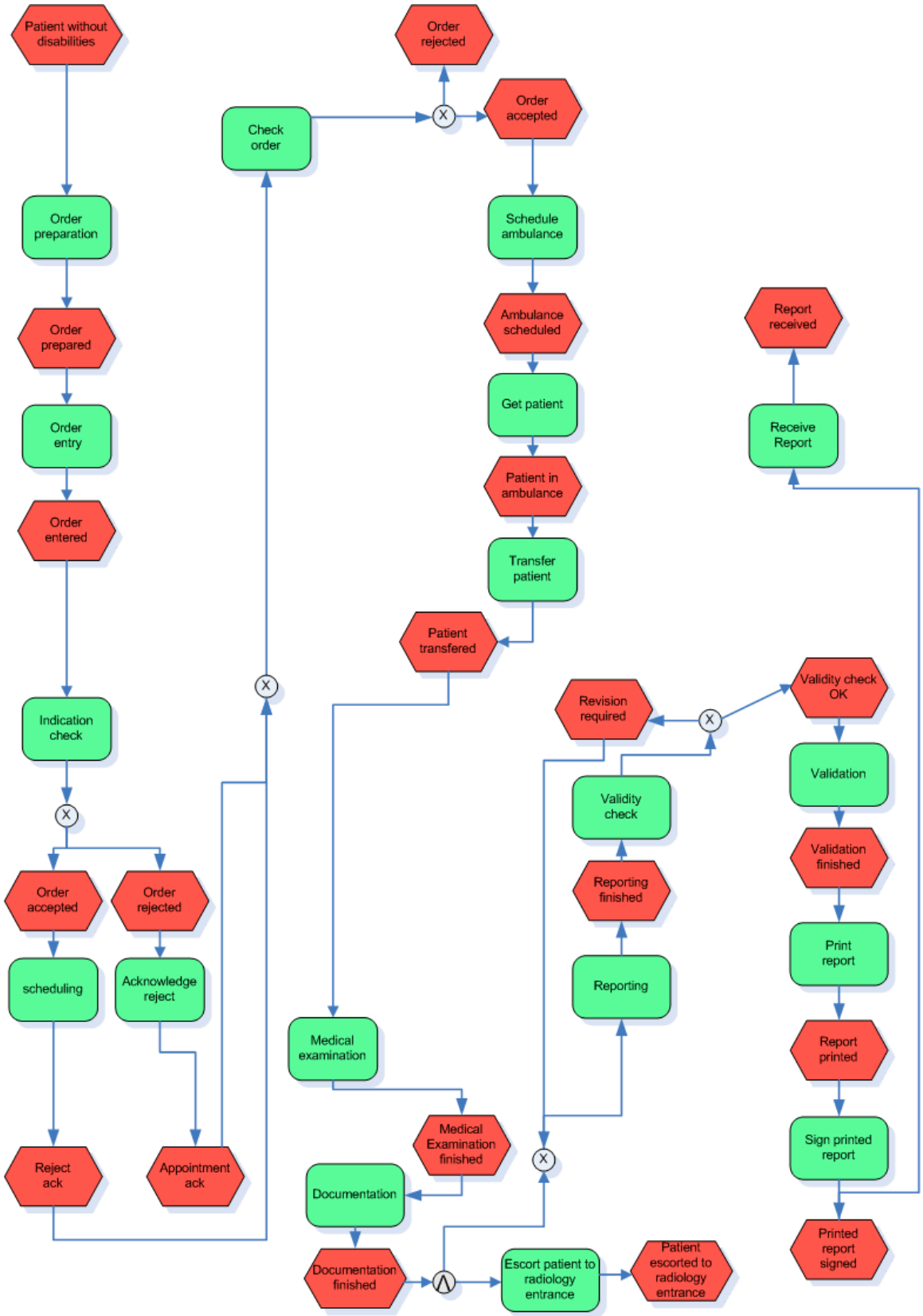


Figure 38: Invalid semantic process configuration of healthcare running example

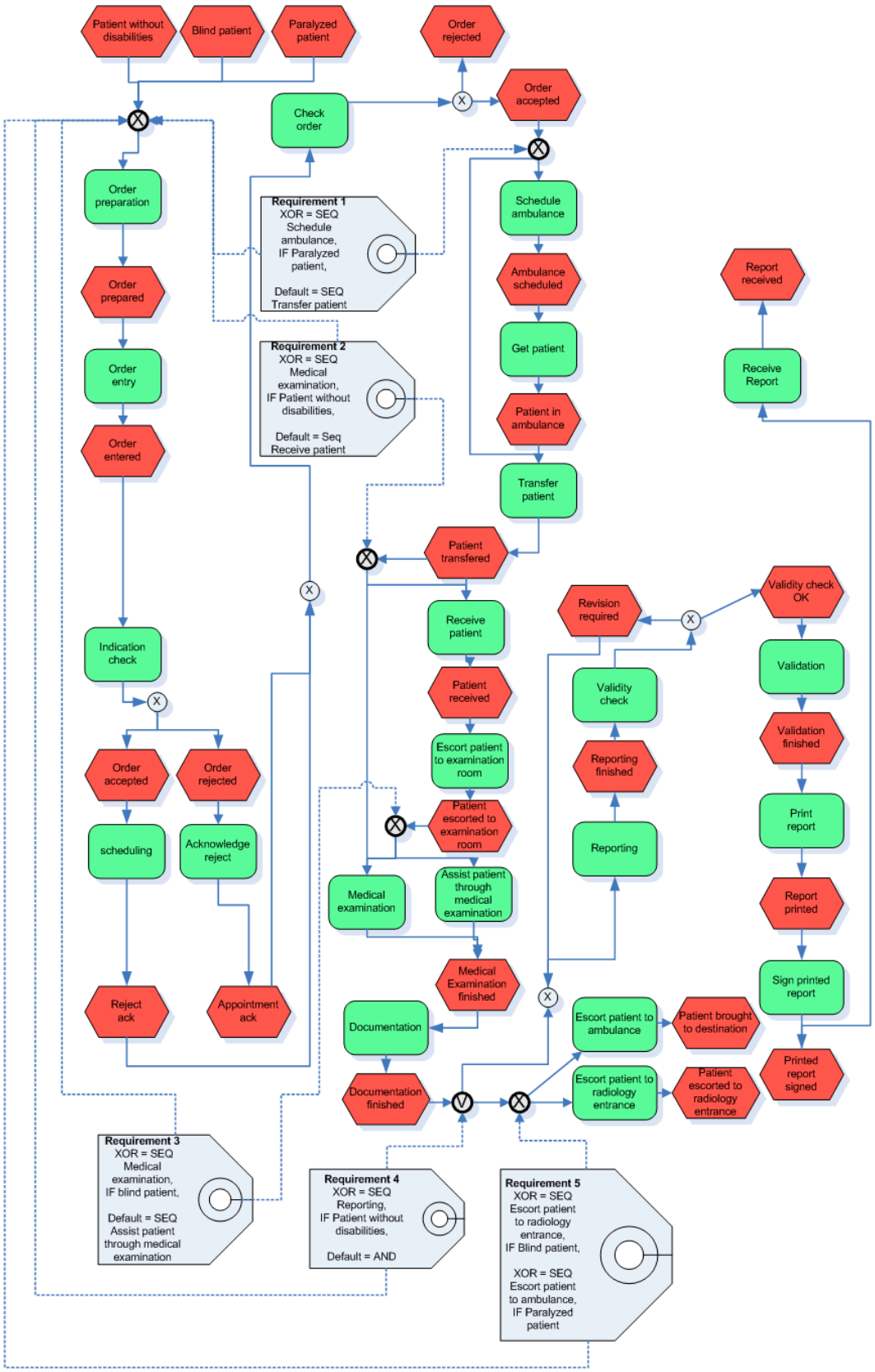


Figure 39: Healthcare running example modeled using C-EPC with only configurable nodes and configuration requirements

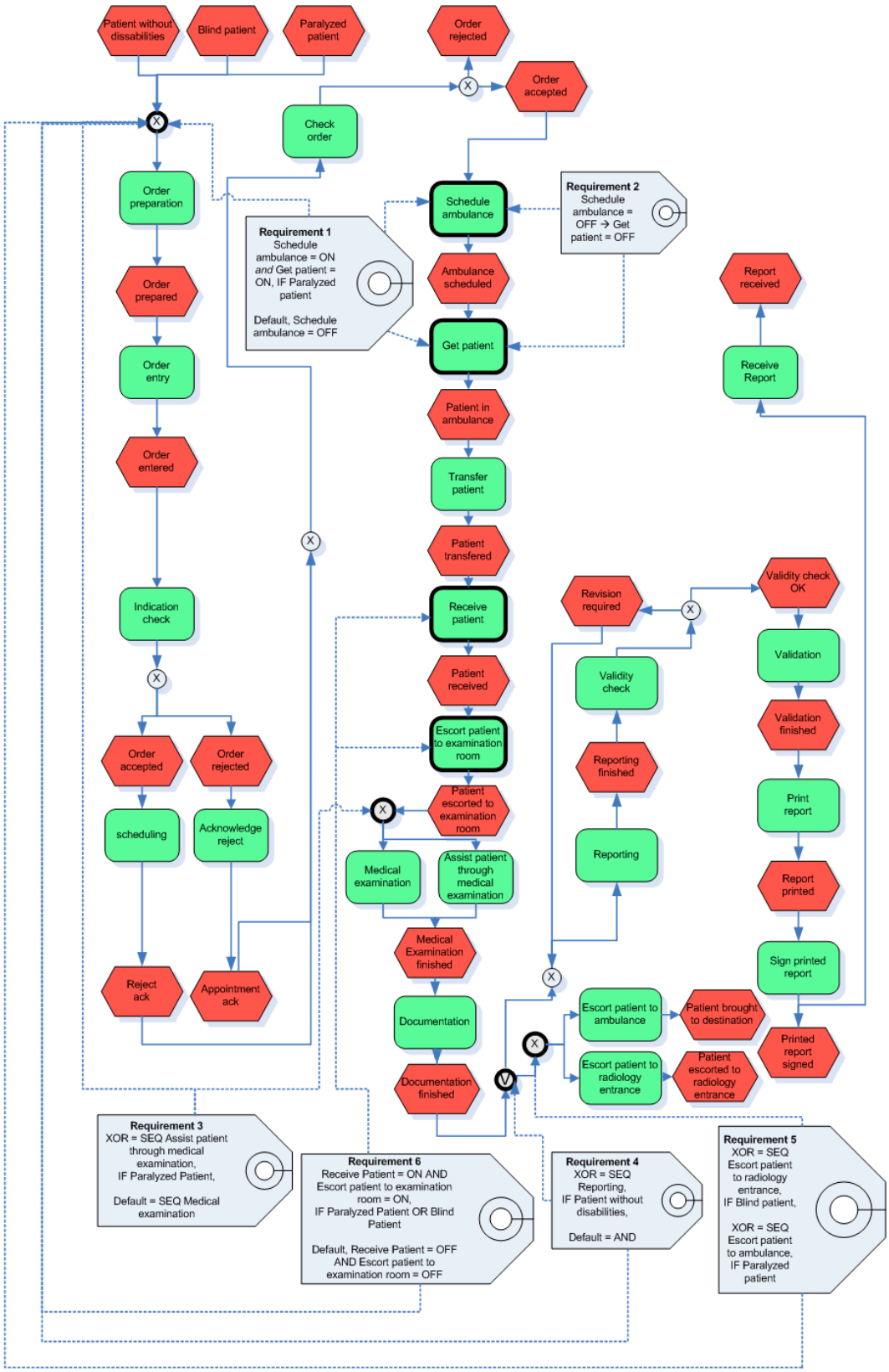


Figure 40: Healthcare running example modeled using configurable nodes and configuration requirements

EC1: Mark variable elements
C-EPC support the marking of variable nodes: configurable functions and connectors. Configurable nodes are marked with bold lines.

Strengths
Marking configurable nodes with bold lines is simple and effective.

Weaknesses
Without configuration guidelines, it is unclear how and when the configurable nodes should be configured.

EC2: Support of change patterns
Configurable functions support the delete process fragment (AP2) change pattern: a function marked as configurable can be deleted from the process model with its respective following events.

AND logical connectors marked as configurable are actually not configurable because they do not support any change patterns.

XOR logical connectors marked as configurable support the delete process fragment (AP2) change pattern: a configurable XOR can turn into a sequence by deleting process fragments or remain an XOR.

OR logical connectors marked as configurable support the delete process fragment (AP2) and replace process fragment (AP4) change patterns:
- A configurable OR can turn into a sequence by deleting process fragments.
- A configurable OR can be replaced by an OR, XOR or AND.

Strengths
Only two of the change patterns are supported by C-EPC simplifying their use and configuration:
- **AP2**: Delete process fragment
- **AP4**: Replace process fragment

Weaknesses
Supporting only the AP2 and AP4 change patterns results in big configurable process models: all the possible configurations must be added to the C-EPC process model, because configuring is most often done by deleting some of its parts. Big configurable process models are difficult to modify and maintain.

EC3: Configuration rules that adapt process model
C-EPC provide configuration requirements and configuration guidelines to specify configuration rules that adapt the C-EPC process models. These are specified using logical expressions.

Strengths
Default values and application levels can be additionally specified.

Weaknesses
The syntax of the logical expressions used to specify the configuration rules is unclear. Furthermore the distinction between configuration requirements and configuration guidelines is found unnecessary.

EC4: Visualization of configuration rules that adapt process models
The visualization of the configuration rules that adapt the C-EPC process models is done by tags and pointing dashed arrows from these tags to the configurable nodes.

Strengths
This approach is quite simple and effective.

Weaknesses
However a configuration requirement can have an impact on a great amount of configurable nodes, resulting in many pointing dashed arrows from this configuration requirement to the configurable nodes: all these dashed arrows result in less comprehensible process models. In the case of a great amount of configuration requirements or guidelines, the C-EPC process model diagram gets clouded with tags and additional dashed arrows.

EC5: Domain visualization and process model configuration
C-EPC does not explicitly support the visualization of the domain and its impact on the configuration of the C-EPC process models.

Strengths
Adding domain visualization to current C-EPC process models would render these process models incomprehensible because they would integrate too much information.

Weaknesses
N/A

EC6: Domain and process configuration rules
Aspects of the domain that have an impact on the configuration of the C-EPC process models can be captured by configuration guidelines or requirements. These are specified using logical expressions.

Strengths
N/A

Weaknesses
How logical expressions can be used to specify the configuration requirements or guidelines is unclear; nor what can be exactly specified using these logical expressions.

EC7: Selective display
The selective display of C-EPC process models can be done by manually configuring the C-EPC process models.

Strengths
Manually configuring the C-EPC process models leads to a better understanding of the C-EPC process models; This could possibly lead to the discovery and correction of errors.

Weaknesses
Manually configuring the C-EPC process models requires understanding the workings of the C-EPC process models. This approach is time-consuming and suffers from human mistakes.

EC8: Correctness
C-EPC can be represented using an XML based language such as the EPC Markup Language (EPML). Using the EPML representation of the C-EPC, syntactic correctness of the C-EPC can be verified using a tool [58, 69].

Strengths
The syntactic correctness of the C-EPC process models can be verified.

Weaknesses
Verifying the correctness of the C-EPC may require a tool.

EC9: Consistency
Consistent configurations of C-EPC process models can be ensured by specifying and following consistent configuration requirements.

Strengths
N/A

Weaknesses
Checking the consistency of the configuration requirements and C-EPC process model configurations needs to be done manually; this is furthermore time-consuming. The syntax used to specify the logical expressions is unclear, therefore automating this task is difficult.

4.4 Conclusion

Most importantly a useful set of evaluation criteria was crafted to assess the modeling concepts provided by BPMLs to model process variability within the domain space. It was found that only C-EPC provided explicit concepts to model process variability: configurable nodes, configuration requirements and guidelines. E-EPC and BPMN were not equipped with any explicit means to model process variability and were therefore found unsuitable to model process variability within the domain space.

Chapter 5 Finding alternative solutions to business process variability modeling problems

5.1 Introduction

Previously has been demonstrated that business process modeling languages (BPMLs) such as C-EPC can be used to model process variability within the domain space. Several authors have suggested that Software Product Lines (SPL) or Software Product families, where variability modeling has been intensively studied could provide helpful means to model process variability within the domain space [34]: the two main means to manage variability within SPL are feature diagrams and software configuration management (SCM).

5.2 Software and process variability

As was discussed in Chapter 3, process variability occurs within the domain space and over time. The same distinction can be made in the field of Software Product Line Engineering (SPLE) where variation management handles [21]:

- Variation in time
  "*refers to configuration management of the product line software as it varies over time*".

- Variation in space
  "*refers to managing differences among the individual products in the domain space of a product line at any fixed point in time*".

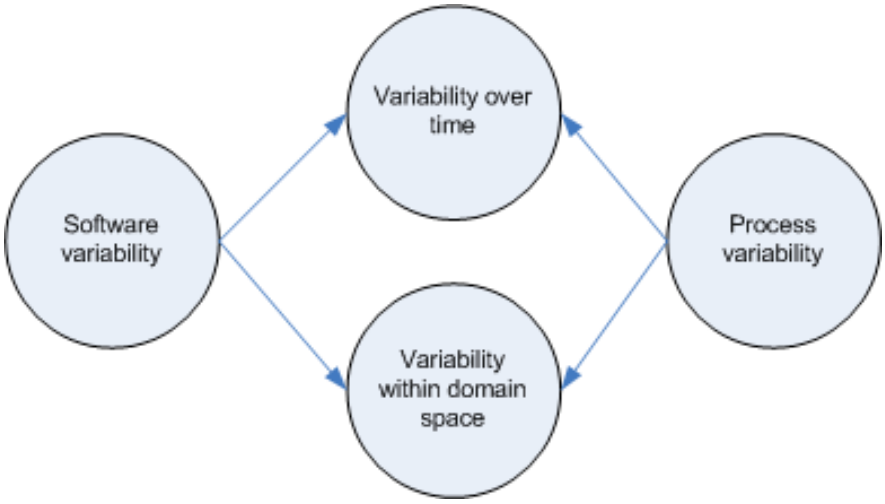


Figure 41: Software and process variability

Considering the similarities between software variability management and process variability management, it would be wise to try to apply these software variability management concepts to improve process variability modeling. However the focus of this research project is on modeling process variability within the domain space. The variability modeling or variation management concepts provided by SPLE shall thus only be applied and adapted to model process variability within the domain space.

5.3 Software product lines

According to Succi et al. [70], the most recent developments in the field of domain analysis and engineering are captured by the software product line as designed by the Software Engineering Institute (SEI). The SEI defines software product lines the following way:

"A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [71]*."*

Building successful software product lines implies managing the commonality and variability of software product families. According to Noordhuizen [72], "*commonality represents the functionality uniformly occurring across all products in the family, whereas variability represents those characteristics only occurring in some, but not all of the products."*

The following benefits have been attributed to using software product lines [73]: "*Advantages of applying product family engineering techniques are reduced development costs, a quicker time to market for the system variants, and typically a higher software quality"*. It is thus worthwhile to try to apply "*product family engineering techniques*" in order to improve process variability modeling.

There are many software variability management paradigms available. Thankfully a distinction can be made between, variability modeling and variability mechanisms [74]: "*variability modeling techniques model the variability that is provided by the product family artifacts, while variability mechanisms are commonly considered as ways to introduce or implement variability in those artifacts*". The focus of this research project is on modeling process variability, thus variability mechanisms such as GenVoca, Ahead, Frames, XVCL and Aspect oriented programming are thus left out of scope [74]. Furthermore a distinction can be made between variability modeling techniques based on features, use cases and other techniques such as ConIPF, COVAMOF, CONSUL/Pure::variants, GEARS, Koalish, OVM, VSL, etc.

Within software product line engineering, two main paradigms have emerged to address variability modeling [34]:

- Feature Oriented Domain Analysis (FODA)
- Software Configuration Management (SCM)

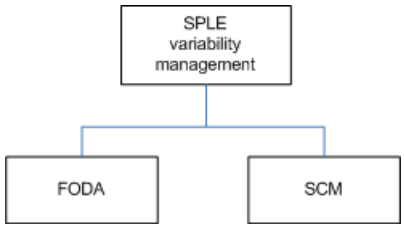


Figure 42: SPLE variability management

FODA is however only one of the many feature diagrams available. These shall be described and analyzed shortly in subchapter 5.4.

5.4 Feature diagrams

A great number of feature diagrams are available to model the domain of software product lines. Here shall be discussed how feature diagrams fit into domain engineering and which feature diagrams can best be applied or modified to support process variability modeling.

5.4.1 Domain engineering and modeling

Important software product lines activities, can be divided into two categories (two life cycle model) [72]:

- Domain Engineering
  "*Domain engineering then is the activity of collecting, organizing, and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets (i.e., reusable work products), as well as providing an adequate means for reusing these assets (i.e., retrieval, qualification, dissemination, adaptation, and so on) when building new systems*". Furthermore, Important activities of domain engineering are domain analysis, design and implementation.

- Application Engineering
  Application engineering is the activity concerned with building systems based on the "*results of domain engineering*".

The purpose of this research project is to solve process variability modeling problems using variability management concepts from software product lines. An important activity for the modeling of process variability is domain analysis. Domain analysis comprises the following main activities:

- Select and define the domain of focus.

- Collect the relevant domain information and integrate it into a coherent domain model: "*A domain model is an explicit representation of the common and variable properties of the systems in a domain, the semantics of the properties and domain concepts, and the dependencies between the variable properties*".

Domain modeling is mostly done using features models. Paul Noordhuizen identifies feature oriented domain analysis (FODA) or feature diagrams as the main means to model features thereby also make explicit the commonality and variability of a software product family [72].

5.4.2 Variation points and dependencies

"*Variation points are places in a design or implementation that identify the locations at which variation occurs* [75]". They are essential elements in managing variability. A variation point can occur at three abstraction layers within a software product family [75]:

- Features
- Architecture
- Component implementations

"*Dependencies in the context of variability are restrictions on the variant selection of one or more variation points, and are indicated as a primary concern in software product families* [75]".

There are several types of dependencies:

- Simple
- Complex

Simple dependencies are formulated in terms of "*requires*" and "*excludes*" relations. Complex dependencies affect a great number of variation points and cannot be stated easily formally.

5.4.3 Feature diagrams

Feature diagrams model variation points at the abstraction layer of features within software product families. Jean-Christophe Trigaux and Patrick Heymans describe elegantly, simply and effectively what feature diagrams are [76]:

"A feature diagram is a featural description of the individual instances of a concept. A feature diagram constitutes a tree composed of nodes and directed edges. The tree's root represents the concept which is refined using mandatory, optional, alternative (X-OR-features) and OR-features. In feature diagrams, mandatory features are features always included in every products. Common features are defined as all mandatory features whose direct and indirect parents are all recursively mandatory. The common features relationship is the transitive closure of the mandatory features relationship. Variable features are defined as all features except common features. Variation points are features (or concepts) that have at least one direct variable subfeature (or feature) [76]."

Jean-Christophe Trigaux and Paul Heymans provided a useful description and comparison of feature diagrams [76]: "*FODA's notation, FORM's notation, FeatuRSEB's notation, Bosch's notation and Riebisch's notation*". Czarnecki furthermore made a distinction between feature diagrams with and without edge decorations (filled, empty arcs) [77].

However only those feature diagrams useful in modeling simply and effectively variability shall be described and illustrated here. FODA and FORM are thus left out scope because of their respective inability to model OR variation points and unnecessary added complexity. Furthermore feature diagrams without edge decorations shall be left out of scope in this research project because they are considered less precise than feature diagrams with edge decorations.

FeatureRSEB's Notation

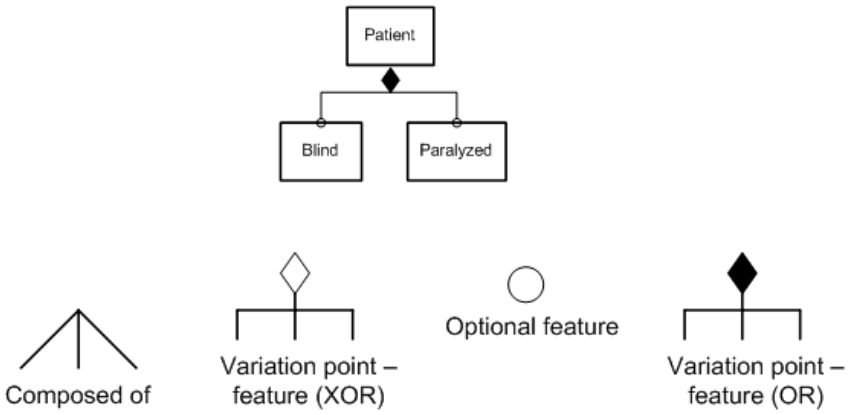


Figure 43: FeatureRSEB feature diagram and notation legend

Bosch's Notation

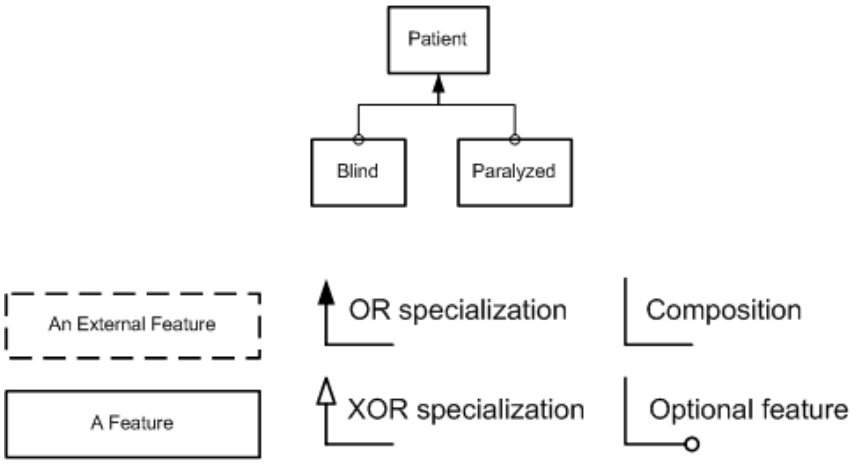


Figure 44: Bosch's feature diagram and notation legend

Riebisch's Notation

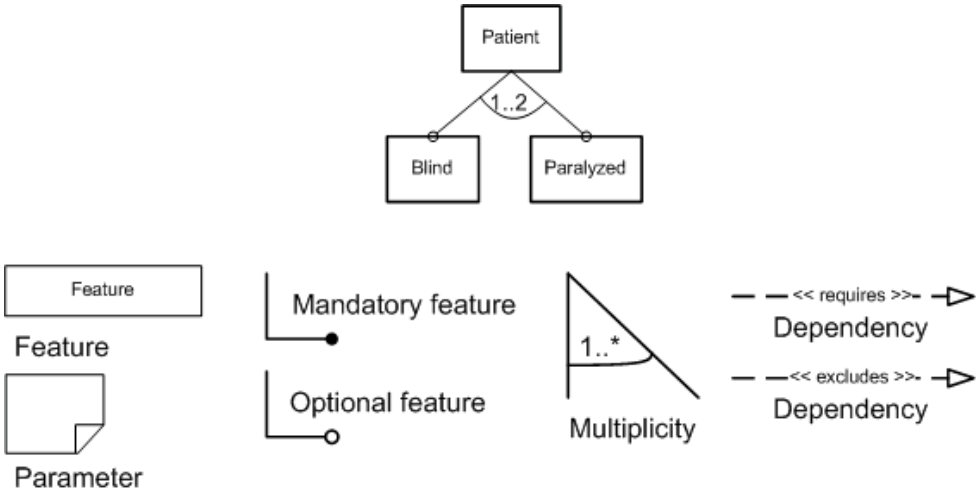


Figure 45: Riebisch's feature diagram and notation legend

Czarnecki's Notation

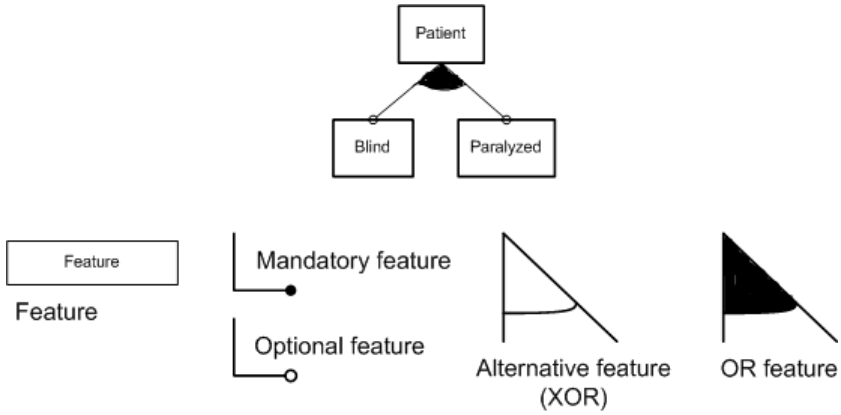


Figure 46:Czarnecki's feature diagram and notation legend

5.4.4 Feature diagram selection

Czarnecki's feature diagram, FeatureRSEB's feature diagram and Bosch's feature diagram are relatively similar. Thus one of these three feature diagrams or Riebisch's feature diagram shall be further applied to improve the modeling of process variability by combining it with an existing business process modeling language. The simplicity and clarity of the three previously cited feature diagrams is valued more than the additional modeling concepts provided by Riebisch's feature diagrams. However in Chapter 6, shall be discovered that specifying constraints between features is necessary to ensure consistent combinations of features. Riebisch's feature diagrams shall thus be applied in Chapter 6 to improve the current capacity of business process modeling languages to model process variability.

5.5 Software configuration management

There are disciplines of Software configuration management (SCM): the main two shall be identified in this section. Furthermore, a taxonomy that can be used to uniformly characterize SCM systems is presented here. A subset of this taxonomy shall be used to select two SCM systems that shall be further applied to improve process variability modeling in Chapter 6.

5.5.1 SCM disciplines

According to Estublier [78], "*SCM is the control of the evolution of complex systems*". SCM can be further classified into the following two disciplines [79, 80]:

- SCM as a management support discipline.
- SCM as a development support discipline.

SCM as a management support discipline consists of the following four activities [79]:

- Configuration identification
  "*Activities comprising determination of the product structure, selection of configuration items, documenting the configuration item's physical and functional characteristics including interfaces and subsequent changes, and allocating identification characters or numbers to the configuration items and their documents.*"

- Configuration control
  "*Activities comprising the control of changes to a configuration item after formal establishment of its configuration documents. Control includes evaluation, coordination, approval or disapproval, and implementation of changes.*"

- Configuration status accounting
  "*Formalized recording and reporting of the established configuration documents, the status of proposed changes and the status of the implementation of approved changes.*"

- Configuration audit
  "*Examination to determine whether a configuration item conforms to its configuration documents.*"

SCM as a development support discipline consists of the following six activities [79]:

- Version control
  "*The possibility to store, recreate and register the historical development of an item (document or source code) is a fundamental characteristic of a version control system.*"

- Build management
  "*Build management handles the problem of putting together and compile modules in order to create a running system. The description of dependencies and information about how to compile items is given in a system model, which is used to derive object code and to link it together.*"

- Workspace management
  "*Workspace management must provide functionality to create a workspace from a selected set of files from the repository.*"

- Concurrency control
  "*If we want to allow several developers to work on the system at the same time, we must also provide mechanisms to synchronize their work.*"

- Change management
  "*It includes tools and processes, which support the organization and tracking of changes from the origin of the change to the approval of the actually implemented source code.*"

- Release management
  "*Release management deals with both the formal aspects of releasing to the customer and the more informal aspects of releasing to the project.*"

The focus of this research project shall be on SCM as a development support discipline as this discipline of SCM provides means to manage software variability and evolution.

As was explained in section 3.5, process variability modeling has three main issues:
- Process modeling
- Process model configuration
- Preserving correctness

The sub-disciplines version control and build management of SCM as a development support discipline provide means to improve the configurability of business process modeling languages when modeling process variability. There are many SCM systems available and time constraints this research project: therefore only a few SCM systems can be applied to improve process configuration management.

NB: SCM systems can probably be used to model both process variability in the domain space and over time at the same time, as it can be done for software entities. However this task is considered future research.

5.5.2 SCM taxonomy

In the literature has been suggested that the Adele configuration manager, CoSMIC or the Proteus configuration language (PCL) could provide useful means to model process variability [34]. However the choice was made to select SCM systems based on the taxonomy provided by Conradi and Westfechtel (Figure 47) [80]. This taxonomy shall be described here briefly and evaluated to determine which of its elements are relevant when using SCM systems to improve business process variability modeling. Only a subset of the taxonomy shall be used to select those SCM systems that can be applied to improve process variability modeling (Figure 48).

Conradi and Westfechtel have written a comprehensive paper on the topic of software configuration management [80]. However this paper may be outdated because of its publication date being 1998. Thankfully, Estublier concluded that most of the needed improvements of SCM systems were on the support flexible processes [78]; practitioners had very few comments on the basic functionality of SCM systems such as versioning and merging. The assumption can thus be made safely that research in the domain of version models has not evolved that much: version models are an important element of the taxonomy that shall be used to select proper SCM systems that can possibly improve process variability modeling.

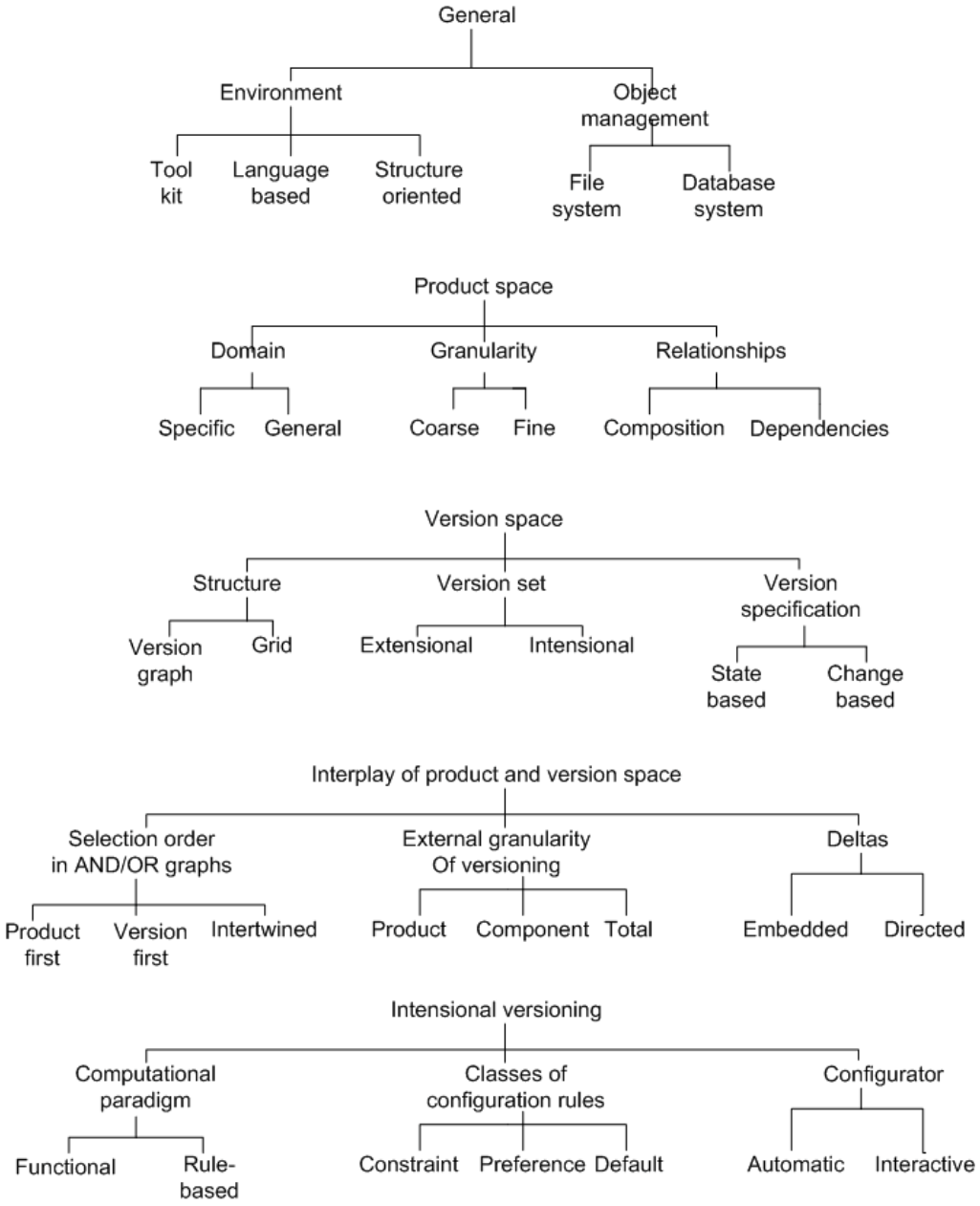


Figure 47: SCM Taxonomy [80, 81]

General

Environment
SCM systems can be a toolkit, language based or structure oriented. This distinction is not relevant when applying SCM systems to improve process variability management.

Object management
SCM systems can use a database management system or a file-based system to manage variants and revisions of a software entity. This is not relevant when applying SCM systems to improve process variability modeling.

NB1: "*A version intended to supersede its predecessor is called a revision. Revisions evolve along the time dimension and may be created for various reasons, such as fixing bugs, enhancing or extending functionality, adapting to changes in base libraries, and the like* [80]".

NB2: "*Versions intended to coexist are called variants. For example, variants of data structures may differ with respect to storage consumption, run-time efficiency, and access operations* [80]".

Product space
The product space takes only the structure of a software product into consideration; versioning is thus not done here.

Granularity
Process models are rather coarse-grained artifacts. Relevant finer grained elements of a process model are sub-processes or hierarchies of sub-processes. The modeling concepts used by a business process modeling language are not viable elements to describe a product space composed of process models: in the case of EPC, these are events, functions and connectors. In business process modeling, the product space can best be described in terms of process models and sub-process models. This taxonomical element is thus relevant when applying SCM systems to improve process variability modeling.

Domain
The distinction between domain-independent models and domain specific models is not relevant, in the case of business process models. As was said previously, the sole distinction that is made is between process models and sub-process models. Internal details and information of the process models are thus abstracted from. The distinction between domain specific and independent process models can thus not be made because this information is hidden within the process models. This taxonomical element is thus not relevant when applying SCM systems to improve process variability modeling.

Relationships
Furthermore a distinction can be made between composition and dependency relationships:

- Composition relationships
  "*Composition relationships are used to organize software objects with respect to their granularity [80].*"

- Dependency relationships
  "*Dependency relationships establish directed connections between objects that are orthogonal to composition relationships [80].*"

Composition relationships are relevant to describe process models because they state the relationships between process models and sub-process models. Dependency relationships can be used to state relationships between process models. This taxonomical element is thus relevant when applying SCM systems to improve process variability modeling.

Version Space
The focus of this research is on improving process variability modeling, however elements of versioning can still be applied to improve process variability modeling.

Structure
The version space can be represented using:

- Version graphs
  They have different shapes and a limited ability to represent variants. Only a small number of variants can be represented using branches, especially in the case of multidimensional variations.

- Grids
  A grid is an n-dimensional space whose dimensions are variant attributes.

Process variants within a domain can thus be represented using either version graphs in the case of a small number of process variants or grids. This taxonomical element is thus relevant when applying SCM systems to improve process variability modeling.

Version set
A versioned item is a set V of versions.

Using extensional versioning, V is constructed by an enumeration of its members: $V = \{v_1, \ldots, v_n\}$. By applying this concept on process variability modeling, V should be the set of process models of respective process variants occurring within a domain.

In intensional versioning, the version set is constructed using predicates: $V=\{vlc(v)\}$. "*The predicate c defines the constraints that must be satisfied by all members of V*". Furthermore, intensional versioning is best suited for the flexible and automatic construction of consistent versions in a large version space. By applying this concept on process variability modeling, V should also be the set of process models of respective process variants occurring within a domain.

This taxonomical element is thus relevant when applying SCM systems to improve process variability modeling.

Version specification
The most important distinction that can be made between SCM systems is the version model or specification they are using [80, 82]: "*A version model defines the objects to be versioned, version identification and organization, as well as operations for retrieving existing versions and constructing new versions"*.

Conradi and Westfechtel make a distinction between the following two version models:

- Version oriented models or state based versioning
  "*Version-oriented models describe configurations (i.e. product versions) in terms of explicit versions of components* [82]".

- Change oriented models or change based versioning
  "*Change-oriented models describe configurations in terms of changes relative to some base configuration* [82]".

These two modeling concepts can be applied to improve process variability modeling by improving configuration modeling. This taxonomical element is thus relevant when applying SCM systems to improve process variability modeling.

Interplay of product and version space
The interplay between product space and version space is considered future research. The focus of this research is on improving process variability modeling and not process variability modeling and process model evolution.

Selection order in AND/OR graphs
Applying AND/OR graphs is thus future research.

External granularity of versioning
Applying component versioning, total versioning and product versioning are considered future research.

Fine-grained deltas
Applying embedded and directed deltas is also considered future research.

Intensional versioning
Using intensional versioning, versions (revisions, variants) are assembled by combining finer grained software elements. However the configurator must ensure that consistent configurations or combinations have been assembled.

Computational paradigm
The construction of versions can be achieved using mainly two different computational paradigms:

- Using a functional framework, intensional versioning is modeled by applying a function or a query q to a set of arguments $a_1 \ldots a_n$: a version v is designed by evaluating the expression $q(a_1, \ldots, a_n)$. This approach assumes a deterministic version construction: the version selection is unique.

- Using a rule-based framework, intentional versioning is modeled by evaluating a query against a deductive database. This approach is non-deterministic because version selection may not be unique. In this case, the configurator may resolve ambiguous choices automatically or interactively.

The construction of process models of respective process variants within a domain can thus either be achieved using a functional or rule-based framework.

This taxonomical element is thus relevant when applying SCM systems to improve process variability modeling.

Classes of configuration rules
Strictness classes dictate the evaluation order of configuration rules:

- Constraint
  "*A constraint is a mandatory rule that must be satisfied. Any violation of a constraint indicates an inconsistency* [80]."

- Preference
  "*A preference is an optional rule that is applied only when it can be satisfied* [80]."

- Default
  "*Finally, a default is also an optional rule, but is weaker than a preference: a default rule is applied only when no unique selection could be performed otherwise* [80]."

Constraints are evaluated first, then preferences and finally default rules are evaluated.

Strictness classes of configuration rules can be used to guide the construction of process models of respective process variants within a domain. This taxonomical element is thus relevant when applying SCM systems to improve process variability modeling.

Configurator
Conradi and Westfechtel define elegantly what a configurator is: "*A configurator is a tool that constructs a version by evaluating a query against a versioned object base and rule base [80]*". The version object base consists of the product and version space, while the rule base consists of stored configuration rules. The assembled version must comply with product and version constraints. The configurator can achieve this compliance automatically or interactively.

A configurator can thus be used to construct process models of all respective variants within a domain. This taxonomical element is thus relevant when applying SCM systems to improve process variability modeling.

5.5.3 SCM system selection

Previously has been concluded that the following criteria can be used to select relevant SCM systems that can be applied to improve process variability modeling (Figure 48):

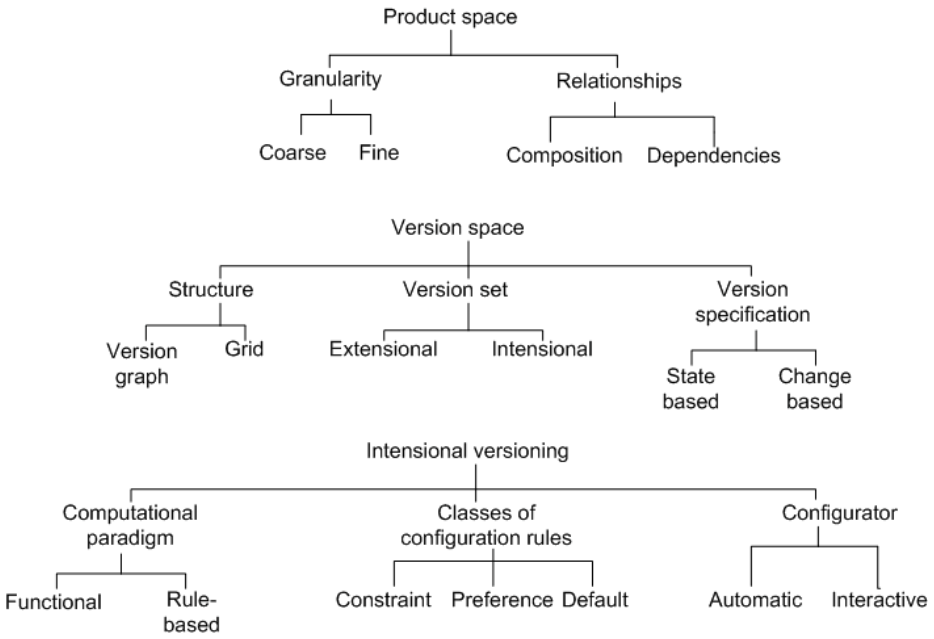


Figure 48: SCM taxonomical subset [80, 81]

Sadly time heavily constraints this research project, therefore at most two different and orthogonal SCM systems that implement as many of the taxonomical elements shall be chosen for the purpose of this research project. Conradi and Westfechtel classified and analyzed twenty four SCM systems [80]. Using the research results provided by Conradi and Westfechtel, the twenty four SCM systems shall be analyzed using the taxonomical subset that was just obtained (Figure 48, Table 1, Table 2, Table 3).

		Product space			
		Granularity*		Relationships *	
		Coarse	Fine	Composition	Dependencies
1	Conditional Compilation		+		
2	SCCS	+	+	+	
3	PIE	+	+	+	+
4	RCS	+	+	+	
5	Gandalf	+	+	+	+
6	DSEE	+	+	+	+
7	Pedit/MVPE		+		
8	CEDAR	+		+	+
9	ADELE I	+		+	+
10	DAMOKLES	+	+	+	+
11	PCTE	+		+	+
12	Shape	+	+	+	+
13	Aide de camp	+	+	+	+
14	COV	+	+	+	+
15	SIO	+		+	+
16	Inscape	+	+	+	+
17	POEM	+		+	+
18	CoMa	+	+	+	+
19	ClearCase	+	+	+	+
20	PCL	+		+	
21	Voodoo	+	+	+	
22	Adele II	+	+	+	+
23	Asgard	+		+	
24	ICE	+	+	+	+

Legend: ⊗ single valued feature, x feature value. * multi-valued feature, + feature value.

Table 1: Analyzing and classifying SCM systems using the product space taxonomical subset [80]

		Version space					
		Structure *		Version set *		Version specification *	
		Version graph	Grid	Extensional	Intensional	State-based	Change-based
1	Conditional Compilation		+		+	+	+
2	SCCS	+		+		+	
3	PIE		+	+	+		+
4	RCS	+		+	+	+	
5	Gandalf	+	+	+	+	+	
6	DSEE	+		+	+	+	
7	Pedit/MVPE		+		+	+	+
8	CEDAR		+	+		+	
9	ADELE I	+	+	+	+	+	
10	DAMOKLES	+		+		+	
11	PCTE	+		+		+	
12	Shape	+		+	+	+	
13	Aide de camp		+	+	+		+
14	COV		+	+	+	+	+
15	SIO		+	+	+	+	
16	Inscape		+	+		+	
17	POEM	+		+		+	
18	CoMa	+		+		+	
19	ClearCase	+		+	+	+	
20	PCL		+	+	+	+	
21	Voodoo		+	+		+	
22	Adele II	+	+	+	+	+	
23	Asgard	+	+	+	+	+	+
24	ICE		+	+	+	+	+

Legend: ⊗ single valued feature, x feature value. * multi-valued feature, + feature value.

Table 2: Analyzing and classifying SCM systems using the version space taxonomical subset [80]

		Intensional versioning						
		Computational paradigm ⊗		Classes of configuration rules *			Configurator *	
		Functional	Rule-based	Constraint	Preference	Default	Automatic	Interactive
1	Cond. Comp.	x					+	
2	SCCS							
3	PIE	x					+	
4	RCS		x				+	
5	Gandalf		x	+	+	+	+	
6	DSEE		x				+	
7	Pedit/MVPE	x					+	
8	CEDAR							
9	ADELE I		x	+	+	+	+	+
10	DAMOKLES							
11	PCTE							
12	Shape		x		+	+	+	
13	Aide de camp	x					+	
14	COV		x	+	+	+	+	+
15	SIO		x	+	+		+	
16	Inscape							
17	POEM							
18	CoMa							
19	ClearCase		x				+	
20	PCL	x					+	+
21	Voodoo							
22	Adele II		x	+	+	+	+	+
23	Asgard		x				+	
24	ICE		x	+	+		+	+

Legend: ⊗ single valued feature, x feature value. * multi-valued feature, + feature value.

Table 3: Analyzing and classifying SCM systems using the intensional versioning taxonomical subset [80]

In bold boxes, in *Table 1*, *Table 2* and Table 3, are shown the two chosen SCM systems that will be applied to improve process variability modeling: the Proteus Configuration Language (PCL) and Change Oriented Versioning (COV). These two systems are not perfectly orthogonal or different, and do not differ on the structure of the version space. However, they differ on the computational paradigm, the class of configuration rules and the version specification.

5.6 Conclusion

Variability management concepts provided by SPLE can be used and applied to improve process variability modeling. Two main variability management paradigms have been identified: FODA and SCM. FODA's notation is only one of the many available feature diagrams. There are furthermore Czarnecki's notation, Riebisch's notation, Bosch's notation and FeatureSREB's notation. Riebisch's feature diagrams were chosen to be applied in Chapter 6. Additionally SCM can be classified into two main disciplines: SCM as a management support discipline and SCM as development support discipline. A subset of the SCM taxonomy developed by Conradi and Westfechtel has been used to select two SCM systems that shall be further applied in Chapter 6 to improve process variability modeling: these are PCL and COV.

Chapter 6 Designing innovative solutions to business process variability modeling problems

6.1 Introduction

Some of the limitations of current businesss process modeling languages (BPMLs) have been uncovered in Chapter 4. EPC and BPMN were found unsuitable to model process variability while C-EPC lead to big configurable process models and were lacking domain modeling concepts. Here C-EPC are extended with domain modeling capacity by combining them with Riebisch feature diagrams. Moreover EPC are combined with change oriented versioning (COV) and afterwards the Proteus Configuration Language (PCL) to extend them with modeling concepts that can handle process variability with the domain space.

6.2 Combining Riebisch's feature diagrams with C-EPC

Feature diagrams are used to model the domain space of members of a family. They can thus be used to model the domain space of process variants. The main idea will be to apply feature diagrams as is described in *Figure 49*: feature diagrams shall be used to model the variability within the domain space that have an impact on the variability of the process model.

Figure 49: Domain modeling and process model configuration

6.2.1 Abstract meta model

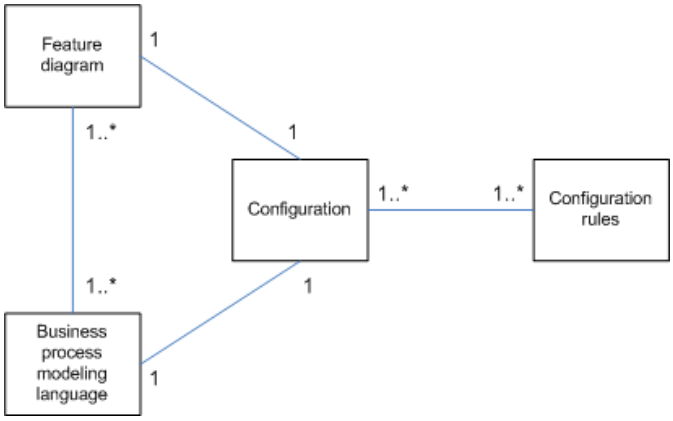


Figure 50: Abstract meta model of the combination of feature diagram with BPMLs

The meta model illustrated in Figure 50 describes at an abstract level how feature diagrams shall be combined with a BPML to enable the modeling of business process variability. A suitable BPML will have to be chosen to combine it with Riebisch's feature diagrams. Furthermore configuration rules need to be specified that guide the configuration of process models based on the configuration of feature diagrams.

6.2.2 Related work

Basically feature diagrams shall be used the same way questionnaires were used to configure EPC [32-34]. Other good examples of applications of feature diagrams to improve the configurability of process models are provided by:

- Schnieders and Puhlmann [83], where they combine feature diagrams with BPMN, while extending BPMN with encapsulation, extension points, parameterization, inheritance and design patterns.
- Czarnecki and Antkiewicz [84], where they combine feature diagrams with UML.

C-EPC are already configurable and there is therefore no need to extend them with encapsulation, extension points, parameterization, inheritance and design patterns. The approach chosen by Czarnecki and Antkiewicz shall thus be chosen and possibly applied to guide the integration of Riebisch's feature diagrams with C-EPC.

6.2.3 Selecting a business process modeling language

Combining a BPML with a feature diagram requires that the configuration of the feature diagram guide the configuration of the process model. The configurable elements of the BPML must thus be clear.

Of the three BPMLs (BPMN, EPC, C-EPC) that have been evaluated in Chapter 4, C-EPC are the only configurable BPML. To save time and effort, it is best to combine Riebisch's feature diagrams with C-EPC to improve their configurability and domain modeling. Combining feature diagrams with BPMN has already been described in the literature [83, 85]. Moreover combining feature diagrams with extended EPC, would require first to define and identify those elements of the BPML that can be configured (dynamically added, deleted, etc) in response to dynamic changes of feature model configurations. It is thus clear that Riebisch's feature diagrams can best be combined with C-EPC to improve their configurability and domain modeling.

6.2.4 Domain modeling using Riebisch's feature diagrams

In the literature, feature models or diagrams have been applied to design configurable UML class diagrams [84]. The goal in this research project shall be to connect feature diagrams to the configurable nodes, configurable attributes, configuration guidelines and configuration requirements of C-EPCs.

Feature diagrams can capture several aspects of the domain of a process:

- The domain variability that has an impact on the variability of the process (Figure 51).
- The variability of the domain and process captured into one feature model (Figure 52).
- The process variability captured into the feature model (Figure 53).

This shall be illustrated using the healthcare running example.

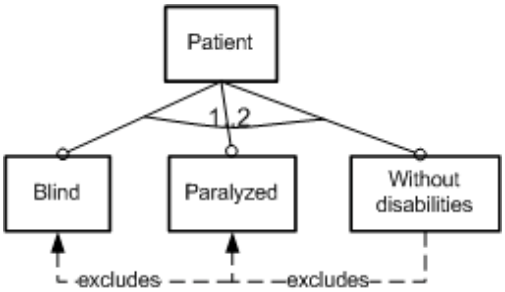


Figure 51: Modeling domain space variability using Riebisch's feature diagrams

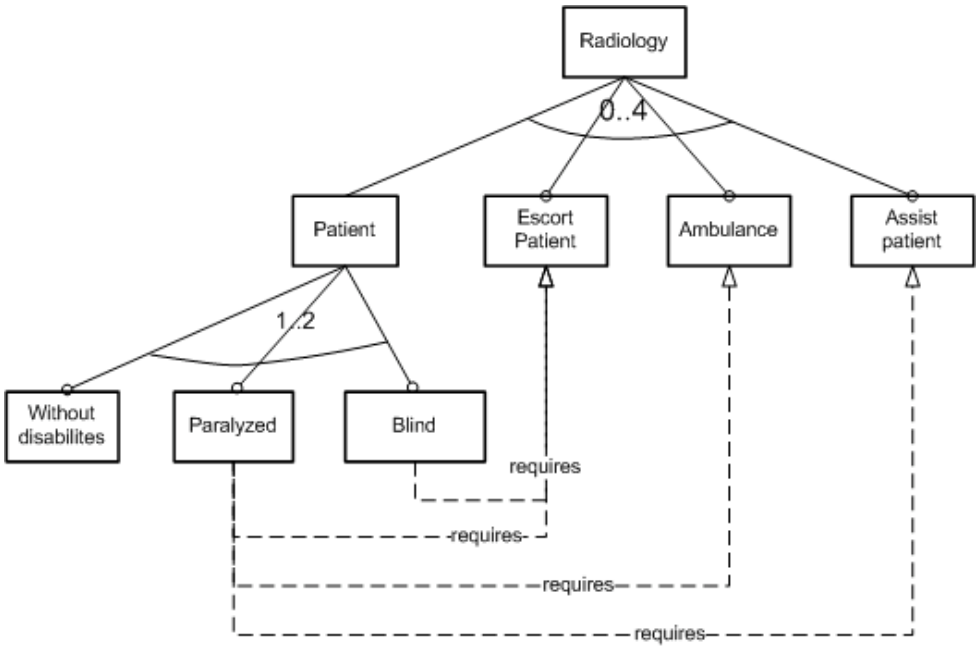


Figure 52: Modeling domain space and process variability using Riebisch's feature diagrams

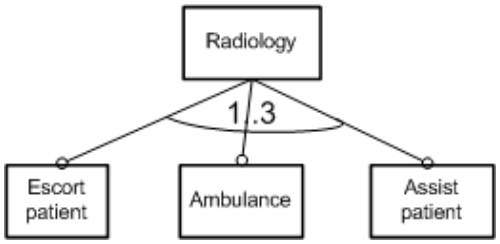


Figure 53: Modeling process variability using Riebisch's feature diagrams

The more elements are modeled or incorporated into feature diagrams, the more complicated becomes the integration task between the feature diagrams and the C-EPC process model. It is therefore advised to choose or formulate modeling guidelines that do not add unnecessary details or complications to the feature diagrams.

Therefore modeling only variable aspects of the domain of a process that cause process variability (Figure 51) is probably the best feature modeling choice. Capturing variable aspects of the process in the feature diagram should be avoided (Figure 52, Figure 53): this should be left hidden and captured in configuration rules. Capturing the variability of a process using a feature diagram does not state the

cause of the variability (Figure 53). It requires from the user to know when an ambulance or escort should be scheduled, while this should be the purpose of the business process management system (BPMS).

6.2.5 Feature-EPC

The integration of Riebisch's feature diagrams with C-EPC results in Feature-EPC (Figure 54). Riebisch's feature diagram configuration shall drive the configuration of the C-EPC process model. To enable this, configuration rules are needed to specify how chosen features lead to C-EPC process model adaptation.

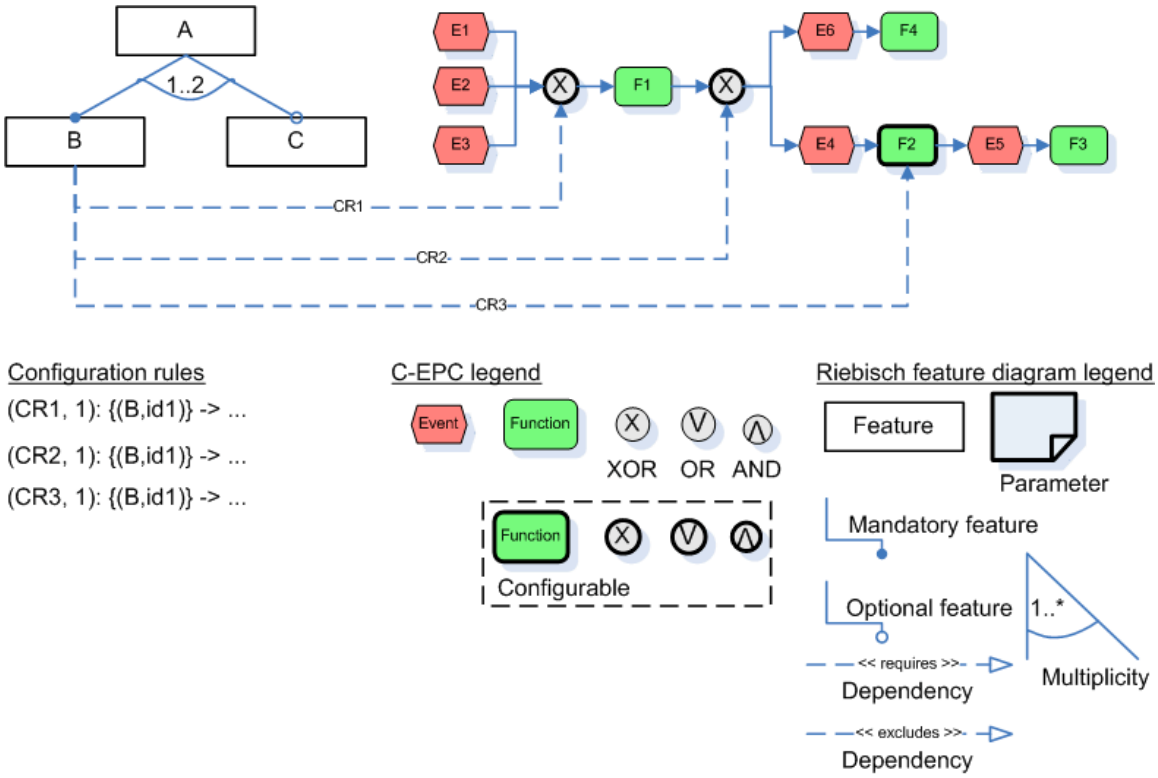


Figure 54: Illustration of Feature-EPC

6.2.6 Configuration rules

The impact of chosen features on the configuration of the C-EPC process models must be captured simply and effectively in configuration rules (Figure 54). Czarnecki and Antkiewicz advocate the use of Xpath expressions to specify the configuration rules in the form of constraints [84]. Rosemann and van der Aalst advocate the use of more graphical notations like dependency constraints [4]. The UML object constraint language 2.0 (OCL) can be used to specify configuration rules: a subset of OCL is used in Riebisch's feature diagrams to specify dependencies and constraints between pairs of features [76, 86]. Z and formal notation sets were also explored to specify the configuration rules. Finally a context free grammar in Backus-Naur Form has been used to specify the configuration rules.

Using modular configuration rules
Using a context free grammar in Backus-Naur form permits the explicit specification of the syntax, contextual constraints and semantics of the configuration rules [87] (section 10.2). The objective here is to specify modular, reusable and easy to implement configuration rules.

Strictly configurable nodes are the logical operators XOR and OR, as well as configurable functions. For every configurable node, configuration rules shall be specified, this should permit the automatic generation or the programming of configuration rules. The specification of complete configuration rules is then done by assigning features to these configuration rules.

To make the configuration rules modular, they have been defined to specify or reflect the configuration of C-EPC process models: every C-EPC process model shall come with its respective set of configuration rules. To have an exact and precise specification of the rules these have been specified using Backus-Naur Form (BNF).

NB: See section 10.2 Appendix 2, for a complete specification of the configuration rules in BNF [87].

Configurable functions
(ID, NumericPriority): {features} → (Name, ID, ON)
(ID, NumericPriority): {features} → (Name, ID, OFF)

Configurable XOR
(ID, NumericPriority): {features} → (xor, ID, XOR, {})
(ID, NumericPriority): {features} → (xor, ID, SEQ, EPCSequence)

Configurable OR
(ID, NumericPriority): {features} → (or, ID, XOR, {})
(ID, NumericPriority): {features} → (or, ID, AND, {})
(ID, NumericPriority): {features} → (or, ID, OR, {})
(ID, NumericPriority): {features} → (or, ID, SEQ, EPCSequence)

Operator precedence
Configuration rules reconfiguring a logical operator (OR, XOR) shall be applied before the configuration rules of configurable functions. If by chance, an OR logical operator is reconfigured into a sequence (or, or1, SEQ, ((“surf”,F1),(“Enjoy”,E1)) and the function “surf” happens to be configurable and turned OFF, the configuration rule would be inapplicable.

NB: see section 7.3.2 for a better-illustrated explanation.

Conflict resolution
Two options are available to specify the configuration rules:

1) To specify for every consistent combination of features, explicit configuration rules. This approach does not lead to inconsistent, conflicting or redundant rules. However it is quite time consuming to specify explicitly all configuration rules for all consistent combinations of features, especially in the case of a great number of features. Furthermore this approach generates long lists of configuration rules.

2) To specify only the necessary configuration rules. Afterwards based on the feature configuration, select all the applicable configuration rules and apply them. However this approach does lead to conflicting and redundant configuration rules. To resolve these problems several conflict resolution algorithms and strategies are available [88]. Thankfully this approach results in the specification of less configuration rules.

Options #2 shall be applied and chosen in this research project because it results in a more compact set of configuration rules. To resolve the situation of redundant and conflicting configuration rules, an algorithm using a conflict resolution strategy with numeric priorities shall be used:

1) Based on selected features, select applicable configuration rules.
    a. Power set selection
2) Detect conflicts:
    a. Redundant configuration rules
    b. Conflicting configuration rules
3) Resolve conflicts by selecting only redundant and conflicting configuration rules with the highest priority. If rules have the same priority, choose one randomly.
4) Apply the new subset of configuration rules.

For every set of selected features, based on its power set the corresponding configuration rules are selected. For example for the set of selected features {F1, F2, F3} has the following power set: {{}, {F1}, {F2}, {F3}, {F1, F2}, {F1, F3}, {F2, F3}, {F1, F2, F3}}. For every power set the corresponding configuration rules shall be selected.

Configuration rules are assigned numeric priorities between 1 and 1000; the higher this number is, the higher the priority of the corresponding configuration rule.

Furthermore redundant configuration rules can be detected because they have the same CEPCConfigRule or right hand expression after the '→' symbol. Conflicting configuration rules can be detected because they modify the same configurable nodes differently.

Tacit issues
When a configurable function is set to OFF its following nodes must be deleted from the C-EPC process model: generally this means the deletion of the following event(s).

Furthermore when a configurable connector such as an XOR or OR is configured as a sequence. Only the nodes of the selected sequence remain; all nodes of other possible sequences must be deleted from the C-EPC process model.

One of the limitations of the conflict resolution algorithm is that it cannot resolve the issue of conflicting configuration rules caused by human mistakes. This case arises when for example several configurable connectors are modeled consecutively one after another. When the first connector is configured as a sequence (SEQ), this may cause the deletion of one or more of the following configurable connectors. If by mistake one of these configurable connectors is reconfigured while it also scheduled for deletion a conflict arises. The question then arises whether this configuration rule should be deleted or not. In this research project, the configuration rules shall simply be deleted.

6.2.7 Concrete meta model

Finally the integration between Bosch feature diagrams and C-EPC lead to the following concrete meta model of Feature-EPC process models (Figure *55*). Most importantly a Feature-EPC is composed of one feature diagram, one C-EPC process model and one set of configuration rules.

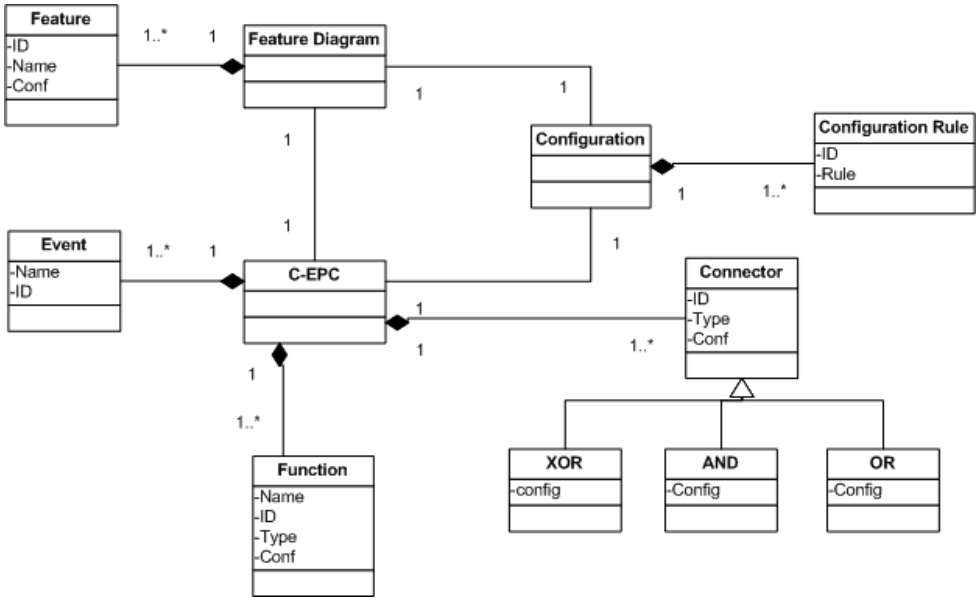


Figure 55: Concrete meta model of Feature-EPC

6.2.8 Business process variability modeling evaluation

As was described previously in Figure 54, Feature-EPC extend C-EPC with domain modeling concepts where features drive the configuration of the C-EPC process model. This leads to process models that are adaptable and thus well suited to model process variability. This is illustrated using the healthcare running example. Modeling the healthcare running example of section 3.4 using Feature-EPC leads to the process model described in Figure *56*. The healthcare running example has been extended here with an extra case: a blind and paralyzed patient. This new case is important to illustrate consistent combinations of features and consistent configuration rules.

However one of the main weaknesses of this approach is that process variants are modeled into one process model: this leads to long lists of configuration rules and big and incomprehensible process models. Extending Feature-EPC with a software tool can overcome some of the weaknesses of this approach: the tool only showing process models based on the current feature configuration and thus hiding the big process model beneath.

Furthermore the consistency of feature combinations can now explicitly be ensured using the constraints 'requires' and 'excludes'. This task would have been difficult and some times impossible using for example Bosch feature diagrams (Figure 44). Looking at Figure *56* here under, consistent combinations of features have been guaranteed using the constraint 'excludes'*:*

- The feature 'Without disability' cannot be combined with 'Blind' or 'Paralyzed', because indeed a patient cannot be for example without disability and blind at the same.
- The features 'Blind', 'Paralyzed' or both at the same time can be selected.

When discussing consistency issues, the problem of inconsistent configuration rules must not be forgotten. When strictly more than one feature is selected the risk of inconsistent configuration rules arises. This is the case of for example configuration rules CR1, CR9 and CR24 (see here under section configuration rules). In this research project was chosen to resolve the problem of inconsistent configuration rules using a conflict resolution algorithm with numeric priorities.

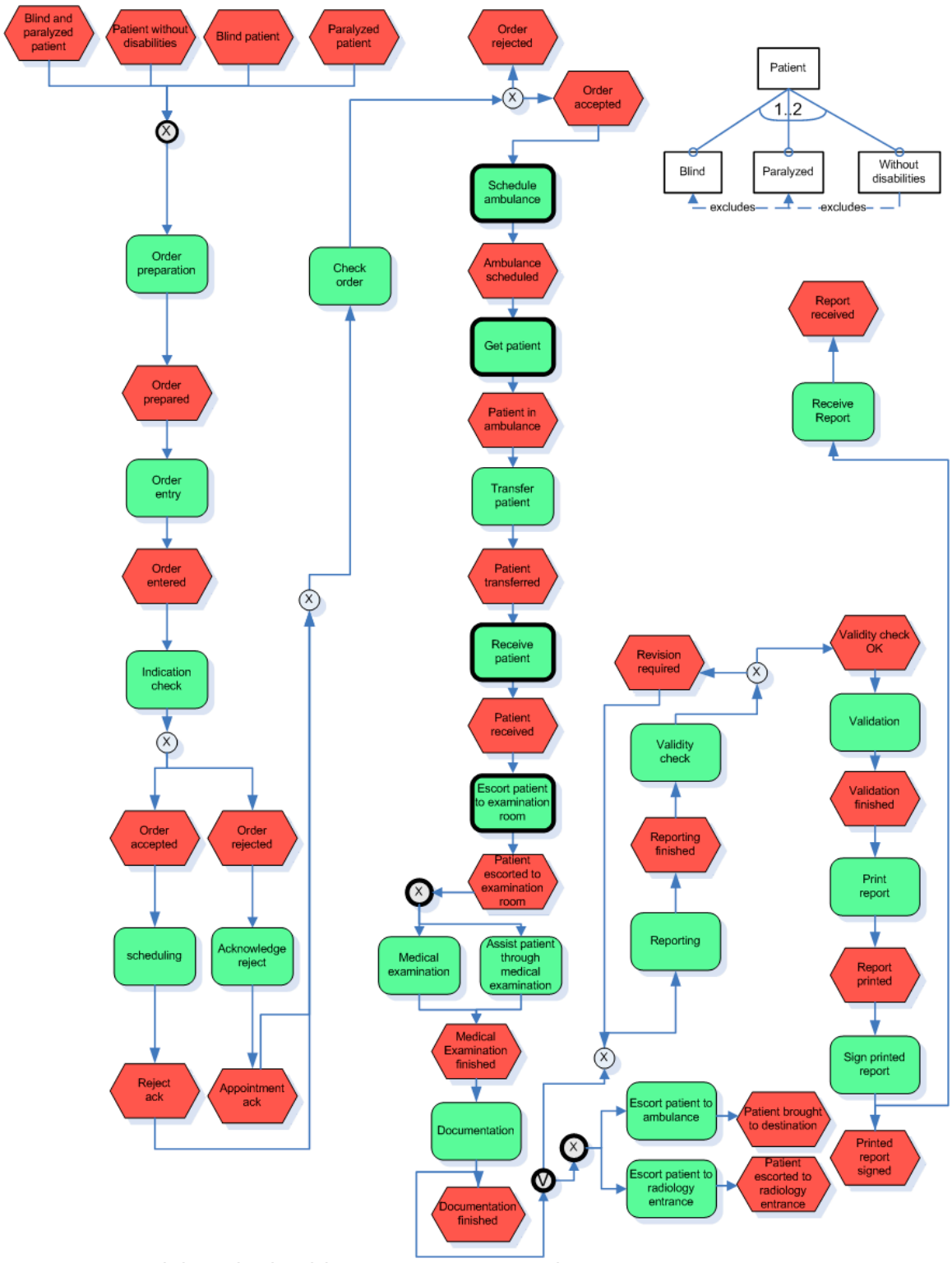


Figure 56: Modeling the healthcare running example using Feature-EPC

Configuration rules

To guide the configuration of the feature-EPC process model specified in Figure 56, configuration rules are necessary. The minimal amount necessary of configuration rules were specified.

(CR1, 2): {(Blind, Fe2)} → (xor, xor1, SEQ, ((Blind patient, E1), (Order preparation, Fu1)))
(CR2, 2): {(Blind, Fe2)} → (Schedule Ambulance, Fu2, OFF)
(CR3, 2): {(Blind, Fe2)} → (Get patient, Fu3, OFF)
(CR4, 2): {(Blind, Fe2)} → (Receive patient, Fu4, ON)
(CR5, 2): {(Blind, Fe2)} → (Escort patient to examination room, Fu5, ON)
(CR6, 2): {(Blind, Fe2)} → (xor, xor2, SEQ, ((Patient escorted to examination room, E2), (Medical Examination, Fu6)))
(CR7, 2): {(Blind, Fe2)} → (or, or1, AND, {})
(CR8, 2): {(Blind, Fe2)} → (xor, xor3, SEQ, ((or, or1), (Escort patient to radiology entrance, Fu7)))

(CR9, 3): {(Paralyzed, Fe3)} → (xor, xor1, SEQ, ((Paralyzed patient, E3), (Order preparation, Fu1)))
(CR10, 3): {(Paralyzed, Fe3)} → (Schedule Ambulance, Fu2, ON)
(CR11, 3): {(Paralyzed, Fe3)} → (Get patient, Fu3, ON)
(CR12, 3): {(Paralyzed, Fe3)} → (Receive patient, Fu4, ON)
(CR13, 3): {(Paralyzed, Fe3)} → (Escort patient to examination room, Fu5, ON)
(CR14, 3): {(Paralyzed, Fe3)} → (xor, xor2, SEQ, ((Patient escorted to examination room, E2), (Assist patient through medical examination, Fu8)))
(CR15, 3): {(Paralyzed, Fe3)} → (or, or1, AND, {})
(CR16, 3): {(Paralyzed, Fe3)} → (xor, xor3, SEQ, ((or, or1), (Escort patient to ambulance, Fu9)))

(CR17, 1): {(Without disability, Fe3)} → (xor, xor1, SEQ, ((Patient without disabilities, E4), (Order preparation, Fu1)))
(CR18, 1): {(Without disability, Fe3)} → (Schedule Ambulance, Fu2, OFF)
(CR19, 1): {(Without disability, Fe3)} → (Get patient, Fu3, OFF)
(CR20, 1): {(Without disability, Fe3)} → (Receive patient, Fu4, OFF)
(CR21, 1): {(Without disability, Fe3)} → (Escort patient to examination room, Fu5, OFF)
(CR22, 1): {(Without disability, Fe3)} → (xor, xor2, SEQ, ((Patient escorted to examination room, E2), (Medical examination, Fu6)))
(CR23, 1): {(Without disability, Fe3)} → (or, or1, SEQ, ((Documentation finished, E5), (xor, xor4)))

(CR24, 4): {(Blind, Fe2), (Paralyzed, Fe3)} → (xor, xor1, SEQ, ((Blind and paralyzed patient, E5), (Order preparation, Fu1)))

Applying the conflict resolution algorithm, the selection of the feature Blind results in the selection of the following configuration rules: CR1, CR2, CR3, CR4, CR5, CR6, CR7, CR8. No conflicting and redundant rules are detected. Thus all the selected rules are applied to generate the Feature-EPC process model described in Figure 57.

Applying the conflict resolution algorithm, the selection of the feature Paralyzed results in the selection of the following configuration rules: CR9, CR10, CR11, CR12, CR13, CR14, CR15, CR16. No conflicting and redundant rules are detected. Thus all the selected rules are applied to generate the Feature-EPC process model described in Figure 58.

Applying the conflict resolution algorithm, the selection of the feature 'Without disability' results in the selection of the following configuration rules: CR17, CR18, CR19, CR20, CR21, CR22, CR23. No conflicting and redundant rules are detected. Thus all the selected rules are applied to generate the Feature-EPC process model described in Figure 59.

Applying the conflict resolution algorithm, the selection of features Blind and Paralyzed results in the selection of the following configuration rules: CR1, CR2, CR3, CR4, CR5, CR6, CR7, CR8, CR9, CR10, CR11, CR12, CR13, CR14, CR15, CR16, CR24.
The following rules are redundant:

- CR4 and CR12
- CR5 and CR13
- CR7 and CR15

The following rules are conflicting:

- CR1, CR9 and CR24
- CR2 and CR10
- CR3 and CR11
- CR6 and CR14
- CR8 and CR16

The configuration rules with the highest priority are the following: CR9, CR10, CR11, CR12, CR13, CR14, CR15, CR16 and CR24. These rules applied to generate the Feature-EPC process model described in Figure 60.

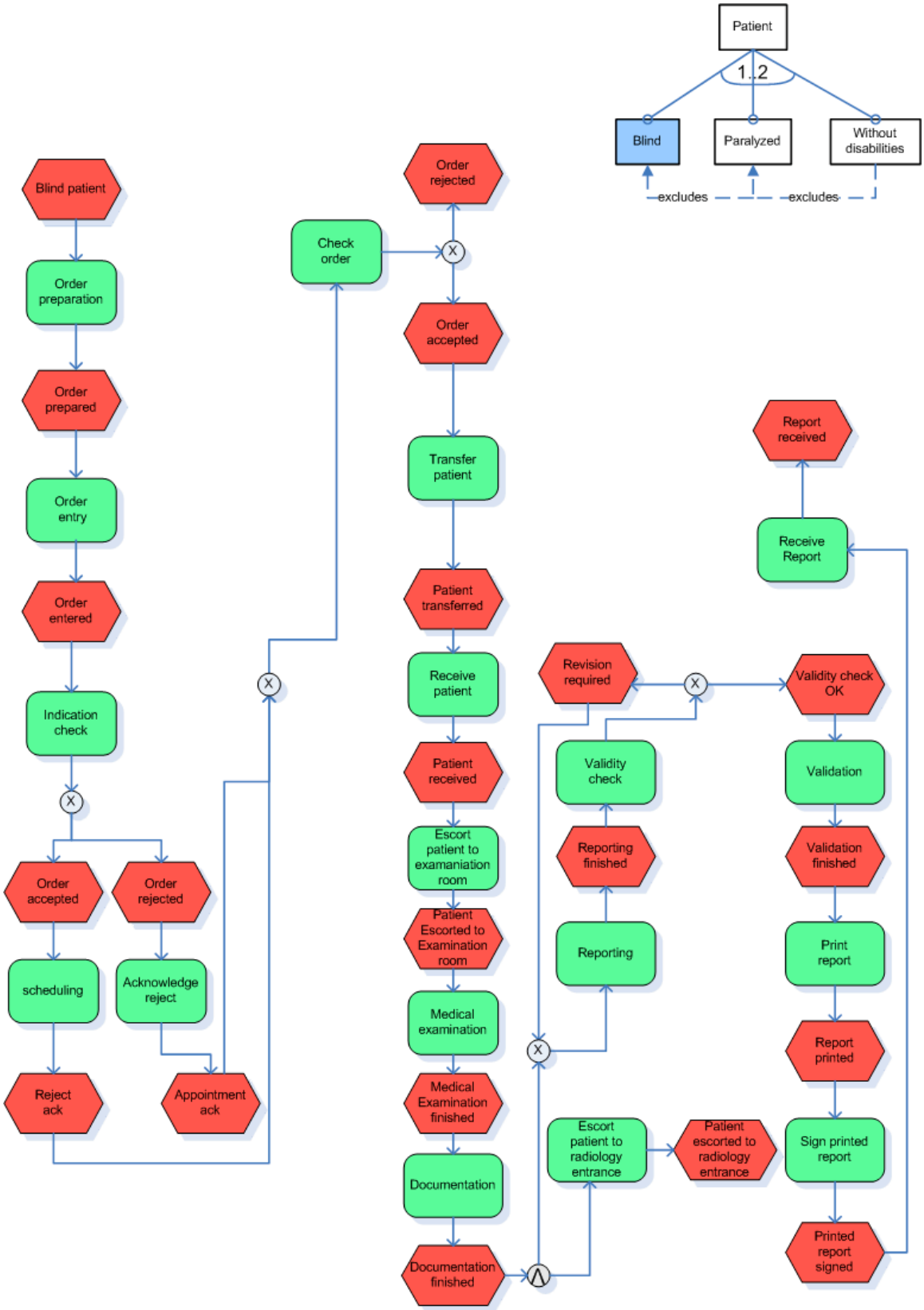


Figure 57: Feature-EPC configuration for a blind patient

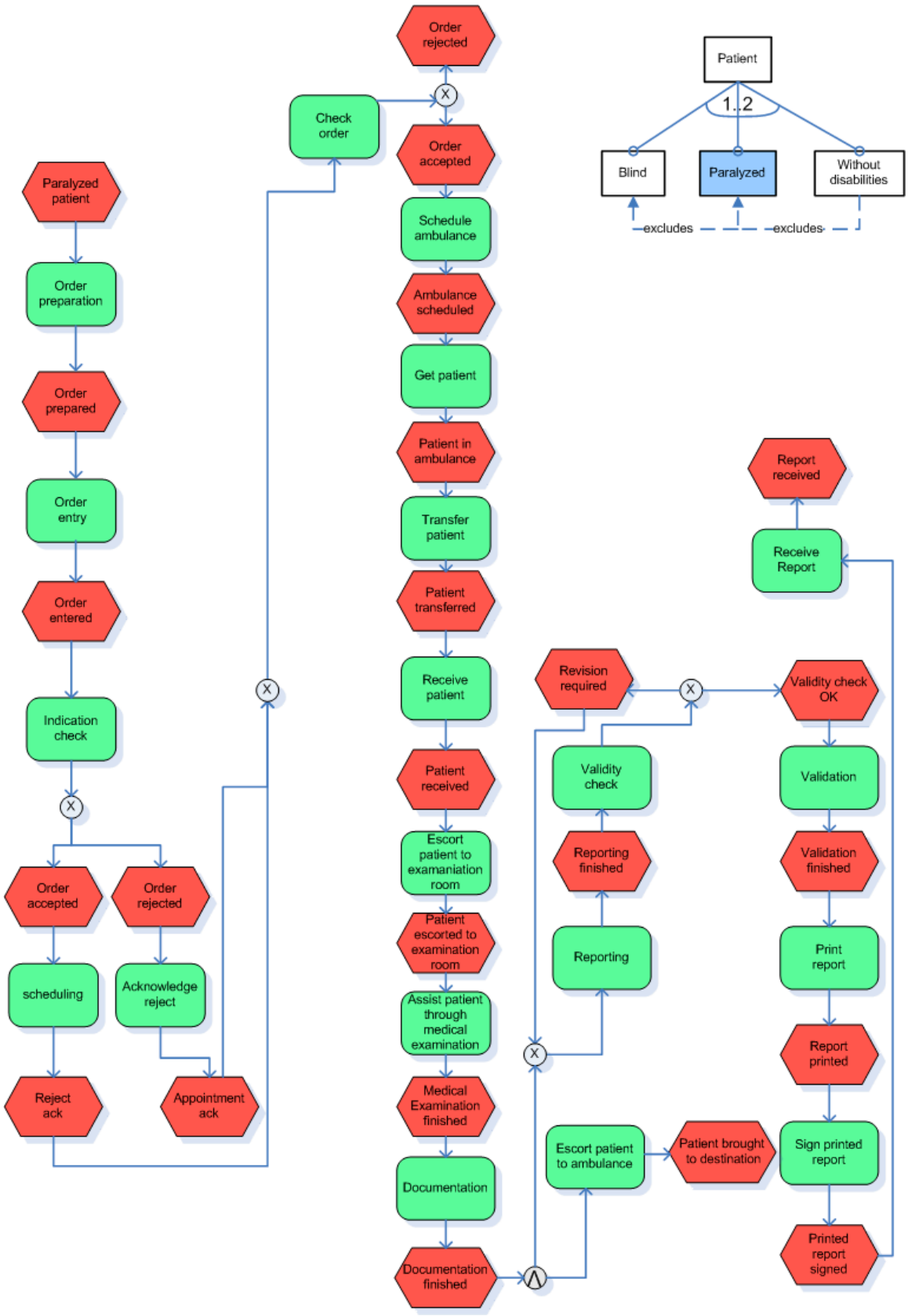


Figure 58: Feature-EPC configuration for a paralyzed patient

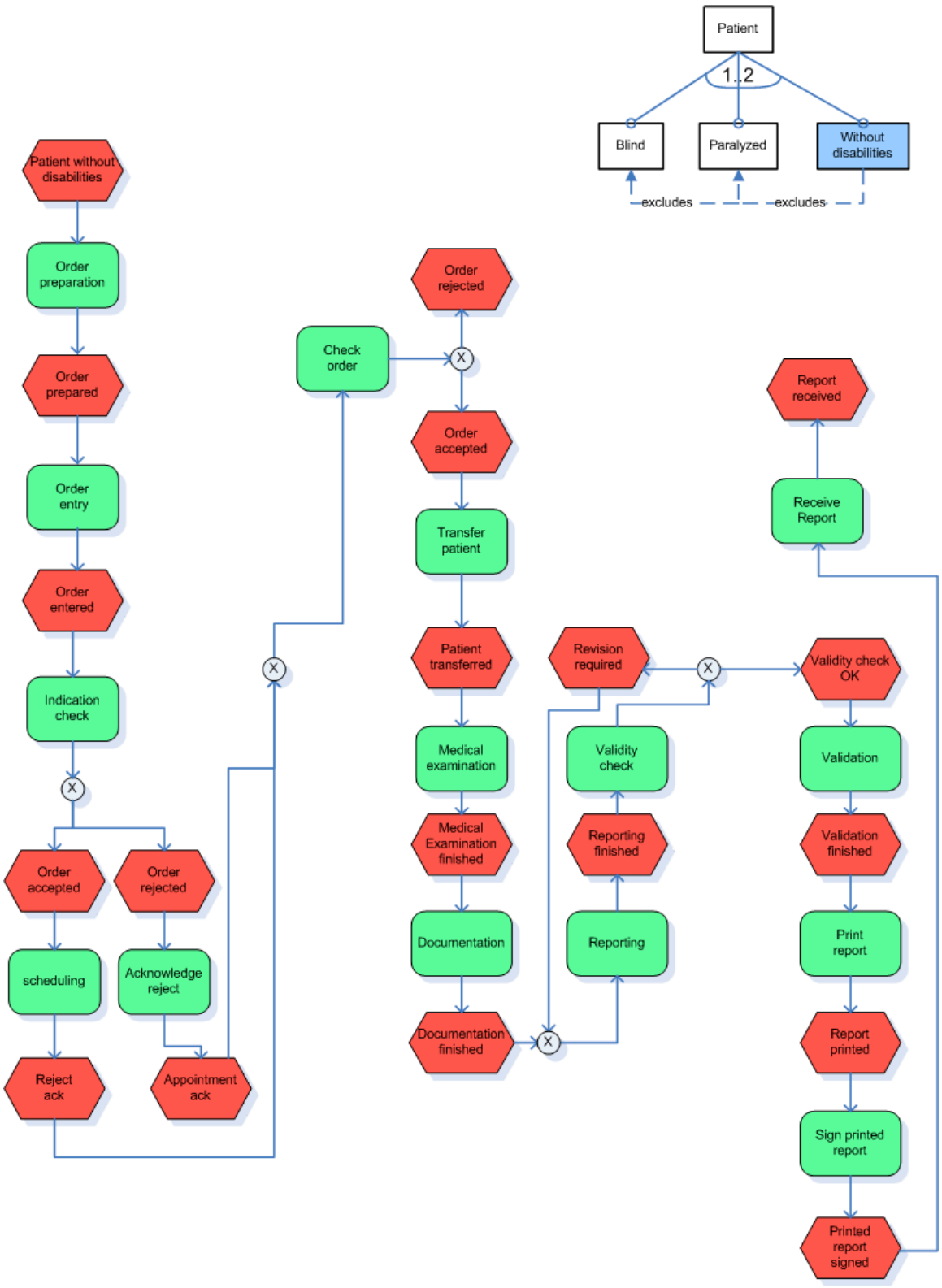


Figure 59: Feature-EPC configuration for a patient without disabilities

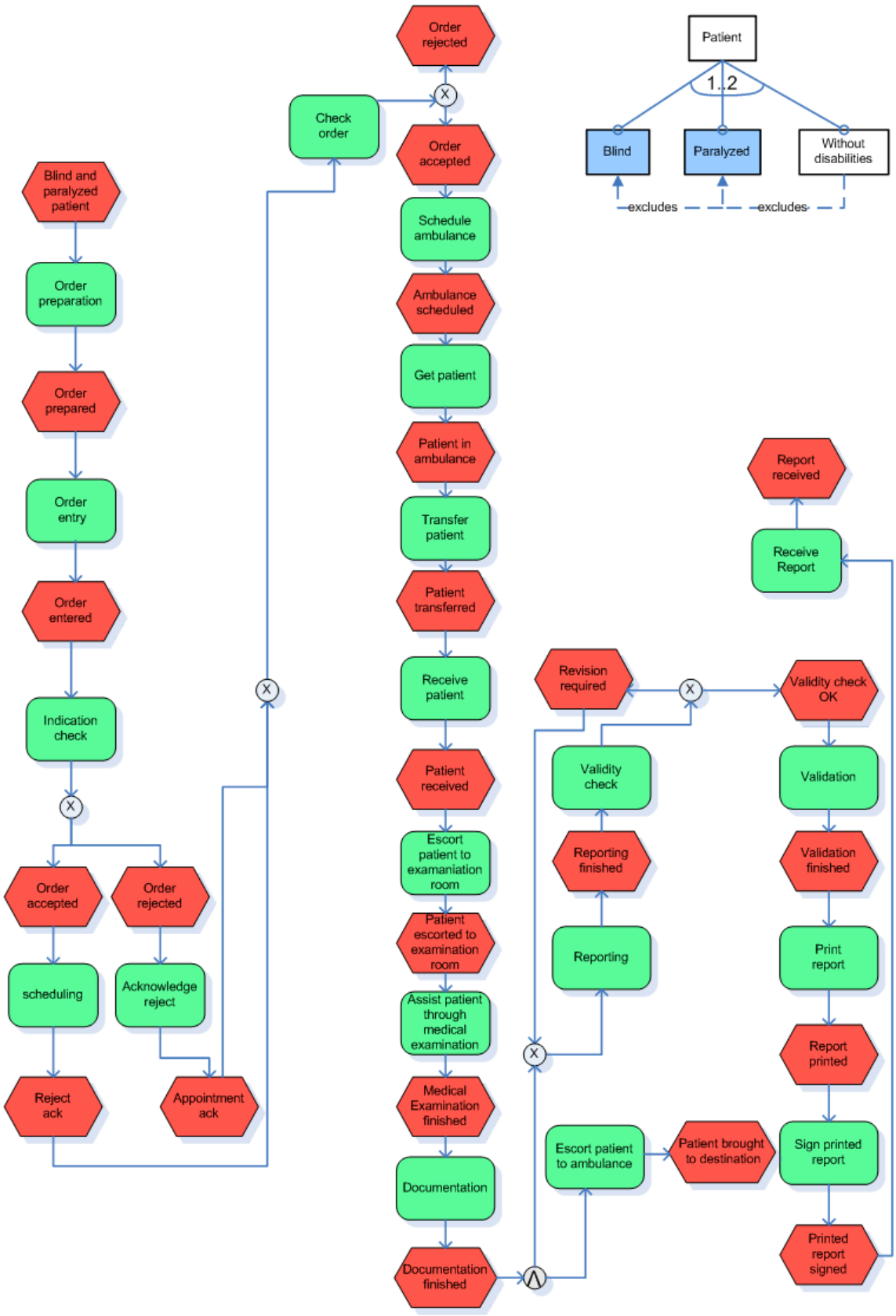


Figure 60: Feature-EPC configuration for a blind and paralyzed patient

EC1: Mark variable elements
Feature-EPC provides the means to mark variable elements of a process model: functions and logical connectors can be marked as configurable (Figure 54). Configurable nodes have bold lines.

Strengths
Marking configurable nodes using bold lines is simple and effective.

Weaknesses
Without configuration rules, it is unclear how and when these nodes should be configured.

EC2: Support of change patterns
Configurable functions support the delete process fragment (AP2) change pattern: a function marked as configurable can be deleted from the process model with its respective end events.

AND logical connectors marked as configurable are actually not configurable because they do not support any change patterns.

XOR logical connectors marked as configurable support the delete process fragment (AP2) change pattern: a configurable XOR can turn into a sequence by deleting process fragments or remain an XOR.

OR logical connectors marked as configurable support the delete process fragment (AP2) and replace process fragment (AP4) change patterns:
- A configurable OR can turn into a sequence by deleting process fragments.
- A configurable OR can be replaced by an OR, XOR or AND.

Strengths
C-EPC support only two of the change patterns:
- **AP2**: Delete process fragment
- **AP4**: Replace process fragment

C-EPC are hereby relatively simple to use and configure.

Weaknesses
Feature-EPC mostly supports the delete process fragment (AP2). This requires modeling the commonality and variability of process variants into one process model because process configuration can mainly be achieved by deleting parts of the process model. This leads to fairly big configurable process models, which is a weakness of this approach: big and complex process models are difficult to modify and maintain.

EC3: Configuration rules that adapt process model
Feature-EPC supports the specification of process configuration rules. These were specified using a context free grammar in Backus-Naur form. Only necessary configuration rules are specified with the goal to reuse as much as possible existing rules.

Strengths
These configuration rules are precise and unambiguous. The resulting list of configuration rules is furthermore fairly compact.

Weaknesses
The maintenance of the configuration rules is problematic, because adding, deleting or modifying them requires the update of numeric priorities. Furthermore the configuration rules capture strictly configuration information: they do not state explicitly how the CEPC process model should be modified when a configurable function is set to 'OFF', or when a configurable XOR or OR connector is configured as a sequence (SEQ).

EC4: Visualization of configuration rules that adapt process models
The configuration rules can be visualized using dashed arrows (see EC5).

Strengths
This approach is quite simple and effective.

Weaknesses
With many features and configurable nodes, the Feature-EPC process model gets clouded with dashed arrows pointing from features to configurable nodes. Furthermore the dashed arrows are unnecessary because a selected feature has an impact on all configurable nodes of a Feature-EPC process model.

EC5: Domain visualization and process model configuration
Feature-EPC provides Riebisch's feature notation to model the domain space of process variants. Dashed arrows point from the features to the configurable nodes of the feature-EPC to indicate their impact on the nodes: these are configuration rules.

NB: in Feature-EPC, features have an impact on all the nodes. The dashed arrows are thus not necessary: they can be used to model or display critical configuration rules or configurable elements.

Strengths
Visualizing and modeling the domain space using Riebisch's feature diagrams is simple and effective.

Weaknesses
The dashed arrows pointing from features to configurable nodes can over cloud the Feature-EPC process models. Furthermore as stated above the dashed arrows are often unnecessary.

EC6: Domain and process configuration rules
The configuration rules specify how the configuration of Riebisch's feature diagram leads to adaptations of the C-EPC process model. These configuration rules have been specified using a context free grammar in Backus-Naur form.

Strengths
See EC3.

Weaknesses
See EC3.

EC7: Selective display
Displaying process configurations is supported manually. The process engineer must follow the Feature-EPC process model and the configuration rules to build and visualize the process models of desired process variants.

Strengths
Manually configuring Feature-EPC process models leads to understanding the workings of the Feature-EPC process models; this could possibly lead to the discovery and correction of errors.

Weaknesses
Manually configuring Feature-EPC process models requires understanding the workings of the Feature-EPC process models; this could to inconsistent configurations because of human mistakes. Having to configure process models manually to display the results of desired process configurations is a weakness of this approach because it is time consuming. A solution is automating this task using a software tool.

EC8: Correctness
C-EPC can be represented using an XML based language such as the EPC Markup Language (EPML). Using the EPML representation of the C-EPC, syntactic correctness of the C-EPC can be verified using a tool [58, 69].

Strengths
The syntactic correctness of the process models can be verified.

Weaknesses
Verifying the correctness of Feature-EPC process models requires a tool.

EC9: Consistency
Consistent combinations of features can be ensured by using the constraints "*requires*" and "*excludes*". Furthermore consistent configuration rules are ensured using a conflict resolution algorithm with numeric priorities.

Strengths
The consistency problems are resolved.

Weaknesses
Ensuring consistency needs to be done manually; this is time-consuming. Furthermore the maintenance of the Feature-EPC process models has become problematic: modifying the feature diagram or the C-EPC process model will require the addition, deletion or modification of configuration rules but also the update of their numeric priorities. However some conflicts are resolved implicitly.

6.3 Modeling process variability using change oriented versioning

6.3.1 Change oriented versioning

Change oriented versioning (COV) was born within the EPOS project as a new way of doing versioning. Another major work of the EPOS project was the EPOS database. This database implements COV and fulfills the function of a storage repository for works within the EPOS project [89]. In this research project, only COV shall be applied to model process variability; whether a database management system or file-based system is used to implement COV is not relevant to this research. COV can be used to model both process variability within the domain space and over time. As was said previously the focus of this research project is on solving the problems inherent to process variability within the domain space.

NB: EPOS stands for "*Expert System for Programming and ("Og") System Development [89]*"

6.3.2 COV concepts

A distinction can be made between two types of version models or version specifications as was previously described in section 5.5.2 [80, 82]:

- Change oriented or change based version models
- Version oriented or state based version models

COV uses a change oriented version model: versions (variants or revisions) are described in terms of logical changes or options instead of concrete versions. Core concepts of COV are options, choices and ambitions. However additional and helpful concepts are available such as visibilities and high-level version descriptions [89].

Options
"*A logical change is represented by a boolean variable called an option* [89]":

- If the option has value *true*, then it must be included.
- If the option has value *false*, then it must be excluded.

The value of an option may also be left unspecified and thereby *unset*. The set of all selectable options is named the version space. Options are mainly identifiable by their name. As is specified by Conradi and Westfechtel, "*each option corresponds to a global change that can be either included or omitted when configuring a product version* [80]".

Choice
A choice or version choice is a set of options with their respective Boolean value. A choice captures the user's intent to view a specific version of the database or product. Options that are not explicitly specified within a choice can be implicitly bound to the value *unset* or false [89].

Ambition
"*A physical change to the database will be marked with an ambition* [90]". An ambition is a set of options with their respective value bindings. "*These options indicate for which versions of the database the physical change is to apply* [89]."

Visibility
"*This is a logical expression over options, and is attached to every single fragment in the database. Given the option/truth value binding of a choice, a visibility should evaluate to either TRUE or FALSE. This value indicates whether the fragment is to be included in the view, or not* [90]".

High-level version descriptions
High-level version descriptions are mechanisms that can result in more convenient and consistent version selection for users.

Validities
Validities can be used to [89]:
- Assign status values (tested, delivered, etc) to versions.
- Freeze versions to prevent further changes.
- Restrict combinations of options.

Constraints
Constraints limit or restrict the combination of chosen options. Good examples of constraints are mutual exclusion and implications [89].

A new and special symbol is introduced to express mutual exclusion [89]: $\otimes$. Furthermore the concept of an option group is introduced to specify that only one option within the group can evaluate to true. A formal description of such a constraint is defined here under:

Mutex: $O1 \otimes O2 \otimes O3 \otimes O4 \otimes O5$

An implication or "*dependency of one option on another means that the option cannot be bound explicitly to either TRUE or FALSE in an ambition or choice unless another option is bound to TRUE* [89]". The dependency (O1 depends on O2) is formalized the following way [89]:

Dep: $\forall C : (C \Rightarrow O1) \vee (C \Rightarrow \neg O1) \Rightarrow (C \Rightarrow O2)$

A new symbol is created to indicate and formalize this type of dependency $\hookrightarrow$:

Dep: $O1 \hookrightarrow O2$

Preferences
Preferences are positive and negative weights attributed to an option: they describe how much an option is desired by a user [89].

Aggregates
Aggregates permit the construction of "*higher level structures*", they are simply attached to the names of version descriptions [89].

Defaults
"*These are settings attached to particular projects or tasks. These are functions of the environment, rather than of the COV system itself* [89]."

6.3.3 Applying COV to model business process variability

Originally COV is applied at the database level. Here COV shall be applied at the process model level within the domain space of choice. Ambitions, choices and options are considered the core concepts of COV [90]. These three concepts must therefore be applied to model process variability. Visibilities shall thus not be applied because implementing these will unnecessarily complicate the integration task between COV and basic EPC. Validities and constraints will be applied to improve and guarantee the consistency of combinations of options.

Options
To model process variability, options must be modeled as elements or attributes of the domain, which have an impact on the variability of the business processes. Using the healthcare running example, the following options shall be specified:

- O1: Patient without disability
- O2: Blind patient
- O3: Paralyzed patient

As was said previously options are mainly identifiable by their name. However the situation could occur where two different options have the same name. The decision was therefore made to identify options using their names and additionally unique IDs.

Choice
The space of all version choices in the healthcare running example is thus summarized by the following set: {{}, {O1}, {O2}, {O3}, {O1, O2}, {O1, O3}, {O2, O3}, {O1, O2, O3}}.

However the combination of some options are inconsistent: {{O1, O2}, {O1, O3}, {O1, O2, O3}}. A patient cannot be blind and without disability at the same time. The same holds for a paralyzed patient and a patient without disability.

Consistent combinations of options are thus the following:{{O1}, {O2}, {O3}, {O2, O3}}. However there is no explicit process model that has been defined for the following combination of options {O2, O3}.

For the empty set of options {} no process model is being defined.

Ambition
To specify ambitions, a base process model must be used to apply changes on. This base process model can be a reference process model [37, 44, 91], the process model of one of the process variants within the considered domain, etc. However specifying a base process model that captures most of the commonality of the process model of the process variants shall lead to the specification of significantly less ambition rules: viewing the process model of the desired process variant shall thus be achieved by slightly modifying this base process model and thus applying few ambition rules. If by chance the chosen base process model is significantly different from the process model of the respective process variants: viewing the process model of the desired process variant shall thus be done by modifying greatly this base process model and thus applying a great number of ambition rules. Thus creating good base process models could be achieved by merging the process model of the respective process variants into one process model. In this research project the process model of a patient without disability as described in the healthcare running example shall be used as the base process model (Figure 17): this process model is relatively similar to the process model of the other two process variants and is thus a relatively good choice for a base process model. The choice was made to use basic EPC to model the business processes, as shall be explained in more detail later on in this subchapter.

Physical changes shall be applied on this base process model to construct the process model of the desired process variant. The following physical changes can be made to the base process model, these are actually process change patterns [54, 90]:

- Add a process fragment
- Delete a process fragment
- Update/Replace a process fragment

An ambition shall be specified here as a mapping between logical changes (options) and physical changes (process change patterns): Option → process change pattern. Later in this subchapter ambitions will be formalized. However for now the following ambition or configuration rules shall be used:

Ambition rule: replace
(ID, NumericPriority): {(OptionID1, …, OptionIDn)} → (REP, OldNode, NewNode)
OldNode is replaced by NewNode.

Ambition rule: Add
(ID, NumericPriority): {(OptionID1, …, OptionIDn)} → (ADD, {**(**(BeginNode, ID), (NewNode1, ID)**)**,…, **(**(NewNodeN, ID), (EndNode, ID)**)**})
Between BeginNode and EndNode are added new nodes: NewNode1,…,NewNodeN.

Ambition rule: Delete
(ID, NumericPriority): {(OptionID1, …, OptionIDn)} → (DEL, (BeginNode, ID), (EndNode, ID))
Everything between BeginNode and EndNode is deleted and replaced by a dynamic connector (an arrow).

Application of ambition rules to healthcare running example
No changes need to be made to the base process when options O1 is true; implicitly meaning that O2 and O3 are false. Only those options that are true are included into the ambition rules, implicitly meaning that all the other ones are false.

This is not the case when option O2 or O3 is true as can be seen here under.

(A1, 2): {(O2)} → (REP, (Patient without disability, E1), (Blind patient, E2))

(A2, 2): {(O2)} → (ADD, **{(**(Patient transferred, E3), (Receive patient, F1)**)**, **(**(Receive patient, F1), (Patient received, E4)**)**, **(**(Patient received, E4), (Escort patient to examination room, F2)**)**, **(**(Escort patient to examination room, F2), (Patient escorted to examination room, E4)**)**, **(**(Patient escorted to examination room, E4), (Medical examination, F7)**)}**)

(A3, 2): {(O2)} → (ADD, **{(**(Documentation finished, E5), (OR, C1)**)**, **(**(OR, C1), (XOR, C2)**)**, **(**(OR, C1), (Escort patient to radiology entrance, F3)**)**, **(**(Escort patient to radiology entrance, F3), (Patient escorted to radiology entrance, E6)**)}**)

(A4, 3): {(O3)} → (REP, (Patient without disability, E1), (Paralyzed patient, E7))

(A5, 3): {(O3)} → (ADD, **{(**(Order accepted, E7), (Schedule ambulance, F4)**)**, **(**(Schedule ambulance, F4), (Ambulance scheduled, E8)**)**, **(**(Ambulance scheduled, E8), (Get patient, F5)**)**, **(**(Get patient, F5), (Patient in ambulance, E9)**)**, **(**(Patient in ambulance, E9), (Transfer patient, F6)**)}**)

(A6, 3): {(O3)} → (ADD, **{(**(Patient transferred, E3), (Receive patient, F1)**)**, **(**(Receive patient, F1), (Patient received, E4)**)**, **(**(Patient received, E4), (Escort patient to examination room, F2)**)**, **(**(Escort patient to examination room, F2), (Patient escorted to examination room, E4)**)**, **(**(Patient escorted to examination room, E4), (Medical examination, F7)**)}**)

(A7, 3): {(O3)} → (REP, (Medical examination, F7), (Assist patient through medical examination, F8))

(A8, 3): {(O3)} → (ADD, **{(**(Documentation finished, E5), (OR, C1)**)**, **(**(OR, C1), (XOR, C2)**)**, **(**(OR, C1), (Escort patient to ambulance, F9)**)**, **(**(Escort patient to ambulance, F9), (Patient brought to destination, E10)**)}**)

As can be noticed, ambition rules A7 and A6 conflict with each other. If ambition rule A7 is applied before A6, A6 cannot be applied correctly as the function "Medical examination" has been replaced by the function "Assist patient through medical examination". To solve these problems operator precedence is introduced:

- The ADD operator has a higher priority than DEL and REP operators.
- The REP operator has a higher priority than DEL operators.
- The DEL operator has the lowest priority of all operators.

If both options O2 and O3 are true, all the ambition rules A1, A2, A3, A4, A5, A6, A7 and A8 must be applied. However these rules are redundant (A2, A6) and conflicting (A1 and A4, A3 and A8). Furthermore when both O2 and O3 are true, the following ambition rule must be added:

(A9, 4): {(O2), (O3)} → (REP, (Patient without disability, E1), (Blind and paralyzed patient, E2))

These issues can mainly be solved following these two approaches:

1) For all consistent combinations of options, specify explicitly the corresponding configuration rules. This approach does not lead to inconsistent configuration rules but is however time consuming and results in long lists of configuration rules.

2) Specify only the necessary amount of rules and resolve conflicting configuration rules using an appropriate conflict resolution strategy. This is the approach chosen in this research project, because it requires less time to specify the configuration rules and the resulting lists of configuration rules are shorter.

To resolve these issues several conflict resolution strategies are available [88]. However for the sake of simplicity and because of timing constraints numeric priorities shall be applied to resolve the problem of redundant and conflicting configuration rules. Every ambition or configuration rule is assigned a number between 1 and 1000; the higher the number assigned the higher the priority.

Applying the ambition rules shall then be done using the following algorithm:

1) Determine the set of applicable rules for the selected options.
2) Detect conflicts
    a. Redundant rules
    b. Conflicting rules
3) Resolve conflicts
    a. Redundant and conflicting rules with the highest priority are selected.
    b. If rules have the same priority, choose one randomly.
4) All the selected rules are applied.

For every set of selected options, based on its power set the corresponding configuration rules are selected. For example for the set of selected options {O1, O2, O3} has the following power set: {{}, {O1}, {O2}, {O3}, {O1, O2}, {O1, O3}, {O2, O3}, {O1, O2, O3}}. For every power set the corresponding configuration rules shall be selected.

Redundant ambition rules are detected because they have the same right hand expression, this is for example the case of ambition rules A2 and A6. Conflicting ambition rules are detected because they change the same element differently, this for example the case of ambition rules: A1, A4, A9.

Thus if both options O2 and O3 are true, using the algorithm above results in the following:

1) Ambition rules for O2 is true, O3 is true, and both O2 and O3 true are selected: A1, A2, A3, A4, A5, A6, A7, A8, A9.
2) Rules A1, A4 and A9 conflict. Rules A8 and A3 conflict. Rules A2 and A6 are redundant.
3) Rule A9 has a higher priority than rule A1 and A4. Rule A8 has a higher priority than rule A3. Rule A6 has a higher priority than rule A2.
4) The following rules are thus applied: A5, A6, A7, A8, A9.

NB: see section 10.3 Appendix 3, for a complete specification of the ambition rules using a context free grammar in Backus-Naur form [87].

Validity/Constraints

A validity is a disjunction of complete or incomplete version choices: choices or validity terms being conjunctions of option bindings themselves [89].

To eliminate inconsistent combinations, the following validity V1 is specified:

$V1 = (O1 \wedge \neg O2) \vee (O1 \wedge \neg O3) \vee (O1 \wedge \neg O2 \wedge \neg O3)$

To avoid inconsistency of ambition rules because explicit rules have not been specified for option combination $O2 \wedge O3$, the following validity V2 is specified:

$V2 = (O1 \wedge \neg O2) \vee (O1 \wedge \neg O3) \vee (O1 \wedge \neg O2 \wedge \neg O3) \vee (O2 \wedge \neg O3) \vee (\neg O2 \wedge O3)$

However using the mutual exclusion constraint, the validity V2 can be rewritten in a more compact manner as the validity V3:

$V3: O1 \otimes O2 \otimes O3$

BPML selection

By applying COV strictly, the base process model does not need to be configurable: C-EPC will thus not be combined with COV. However to simplify and ease the integration process between a BPML and COV, a very simple and clear BPML shall be chosen: basic EPC. Extended EPC and BPMN were not chosen because of the richness and complexity of their modeling concepts.

Modeling process variability using COV-EPC

The choice was made to annotate the base process model simply with black boxes either labeled "*Replace*", "*Add*", "*Delete*" to indicate the variable elements of the base process model (Figure 61). This modeling notation is fairly simple and does not significantly raise the complexity of the process models. When modeling additions this modeling notation can be imprecise: it is not clear if the new nodes must be added, after or in between of the selected nodes (Figure 61). To solve these problems, black triangles are added in the corners of the "*Add*" black rectangle to indicate whether new nodes are added before or after the designated node:

- In Figure 62, new nodes are added before the selected node.
- In Figure 63, new nodes are added after the selected node.

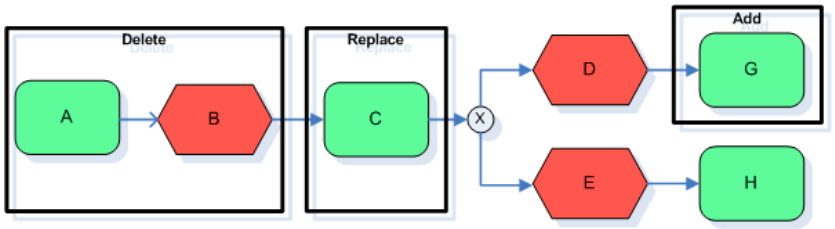


Figure 61: COV-EPC base process model annotated with variability markings

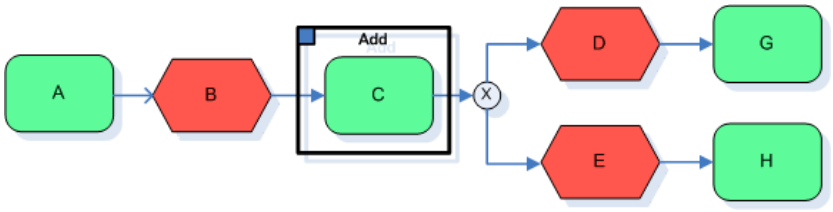


Figure 62: COV-EPC base process model annotated with improved Add rectangle (before)

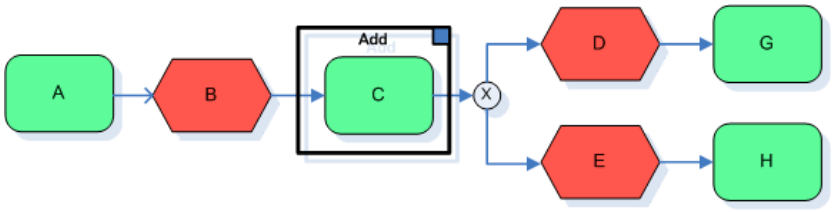


Figure 63: COV-EPC base process model annotated with improved Add rectangle (after)

An alternative choice would have been to include inside of the black rectangles the process fragments that shall be used as replacements or added to the process model (Figure 64). However this modeling notation is unusable in case of a large number of process fragments that can be either added to the process model or used as replacements. Furthermore the size of the process fragments is also an issue. Process fragments that are too big won't fit nicely into a black rectangle.

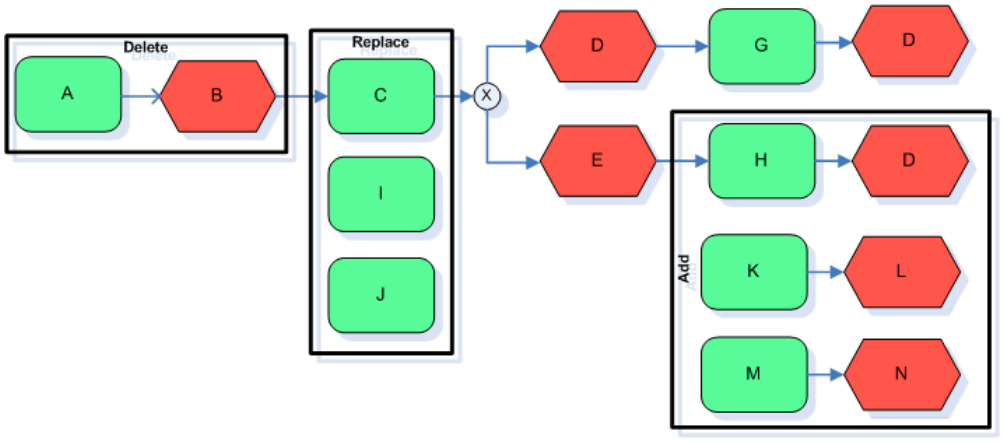


Figure 64: COV-EPC base process model annotated with variability markings and process fragments

The modeling notation described in Figure 62 and Figure 63 shall be used to model the base process model of the healthcare running example because of its compactness and precision.

Meta model
The COV-EPC meta model specified here under can be used to build consistent COV-EPC process models. A choice consists of a set of options. Ambitions form the link between options and configuration rules (process change patterns). The set of ambition rules (options, process changes patterns) is assigned to exactly one base process model.

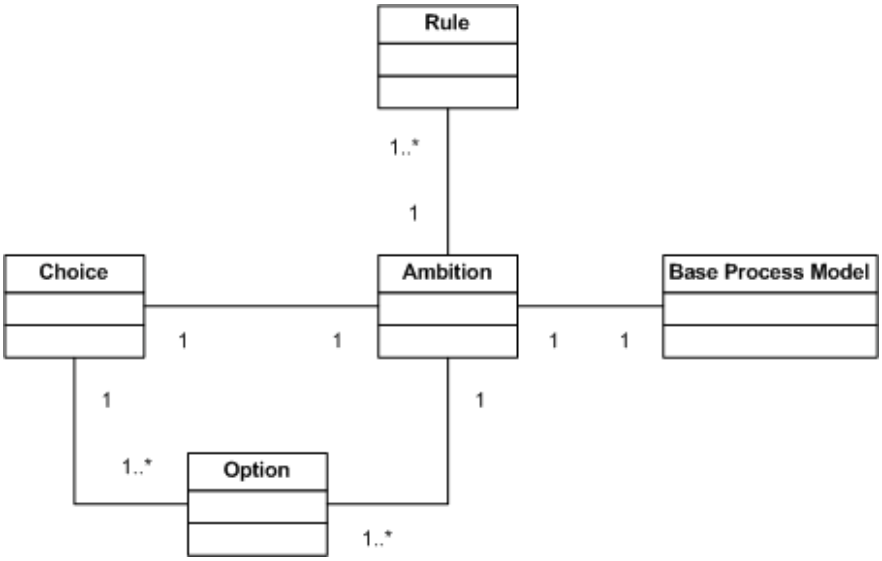


Figure 65: COV-EPC meta model

6.3.4 Business process variability modeling evaluation

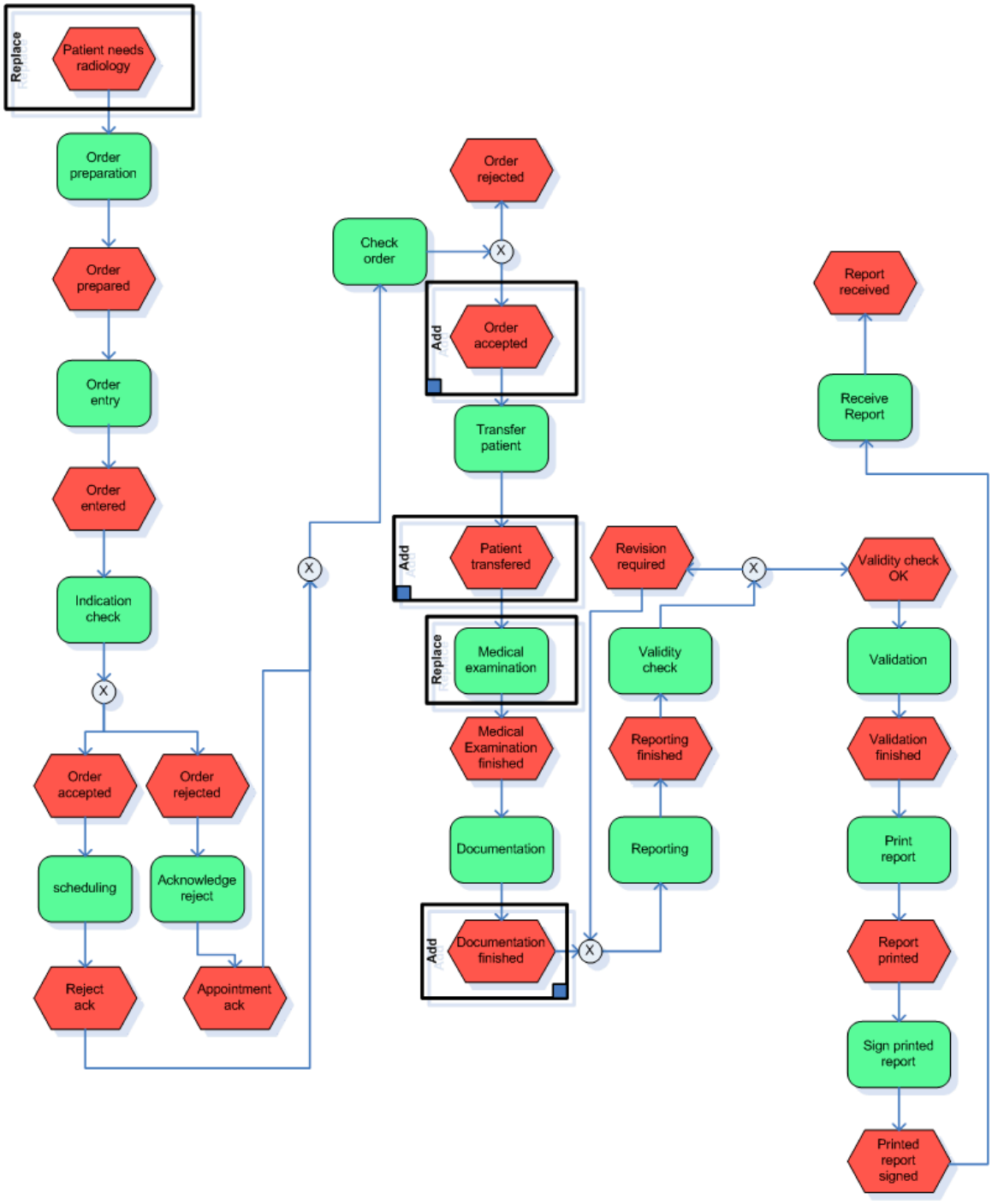


Figure 66: COV-EPC base process model with variability markings

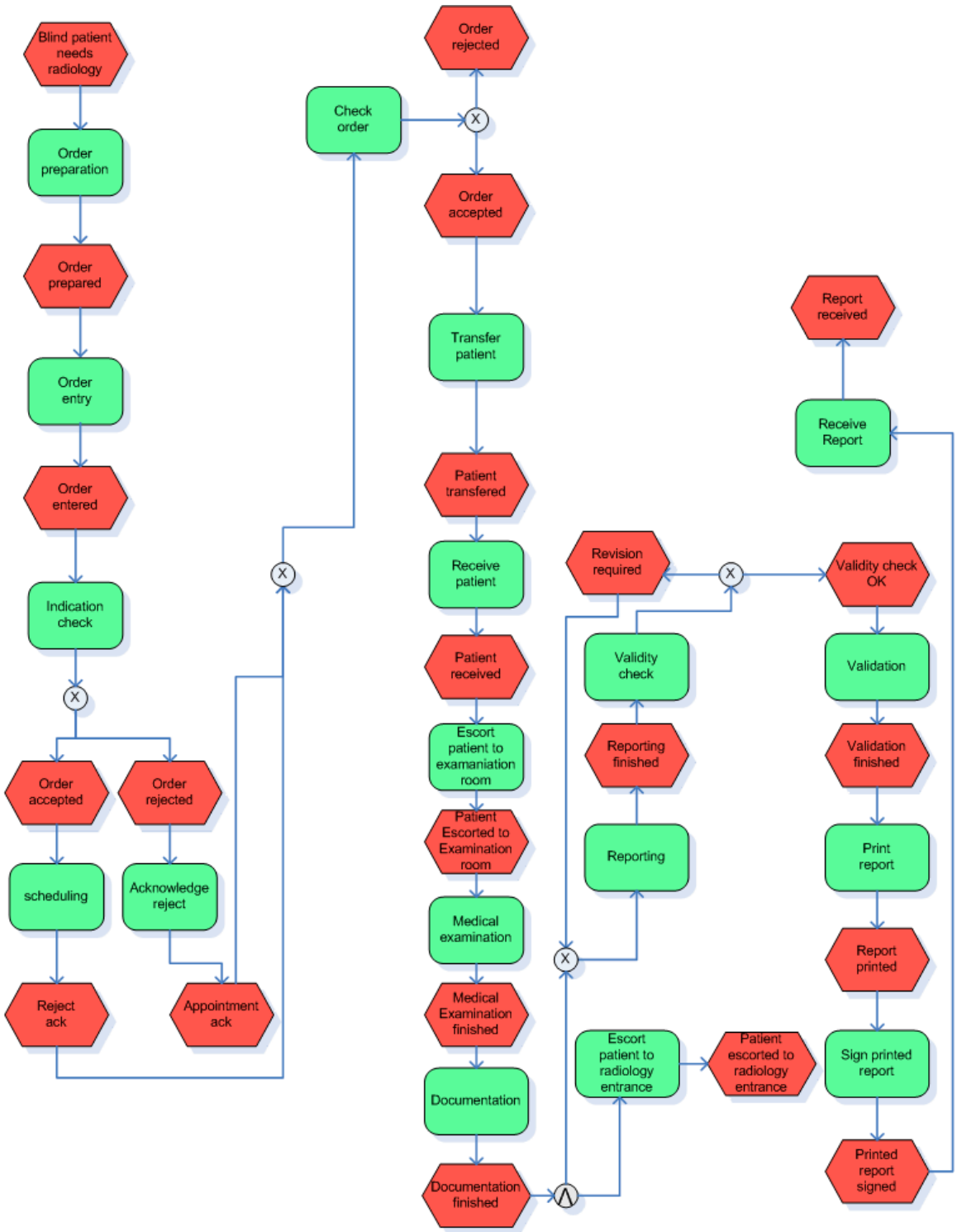


Figure 67: COV-EPC process model configuration with option "Blind patient"

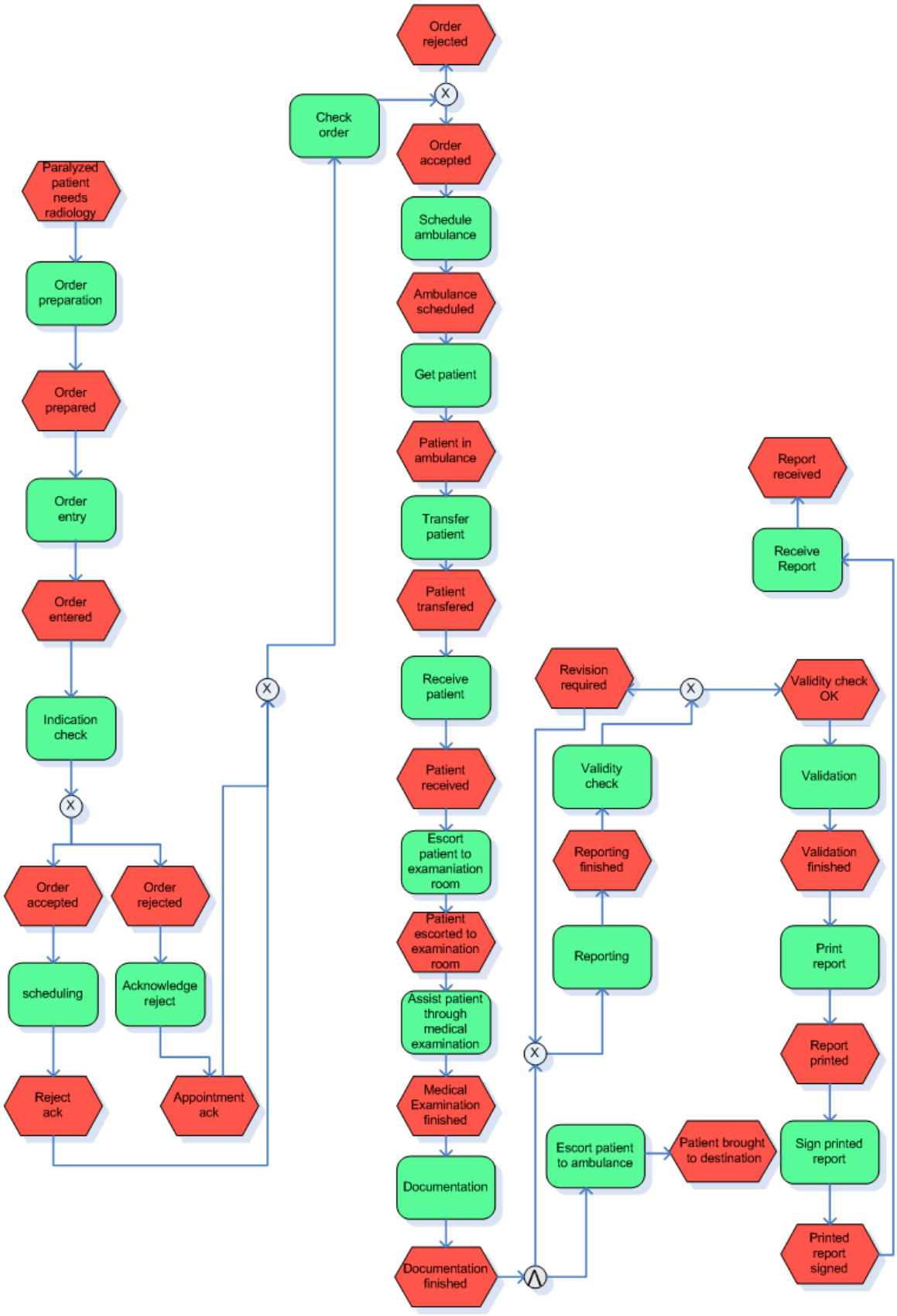


Figure 68: COV-EPC process model configuration with option "Paralyzed patient"

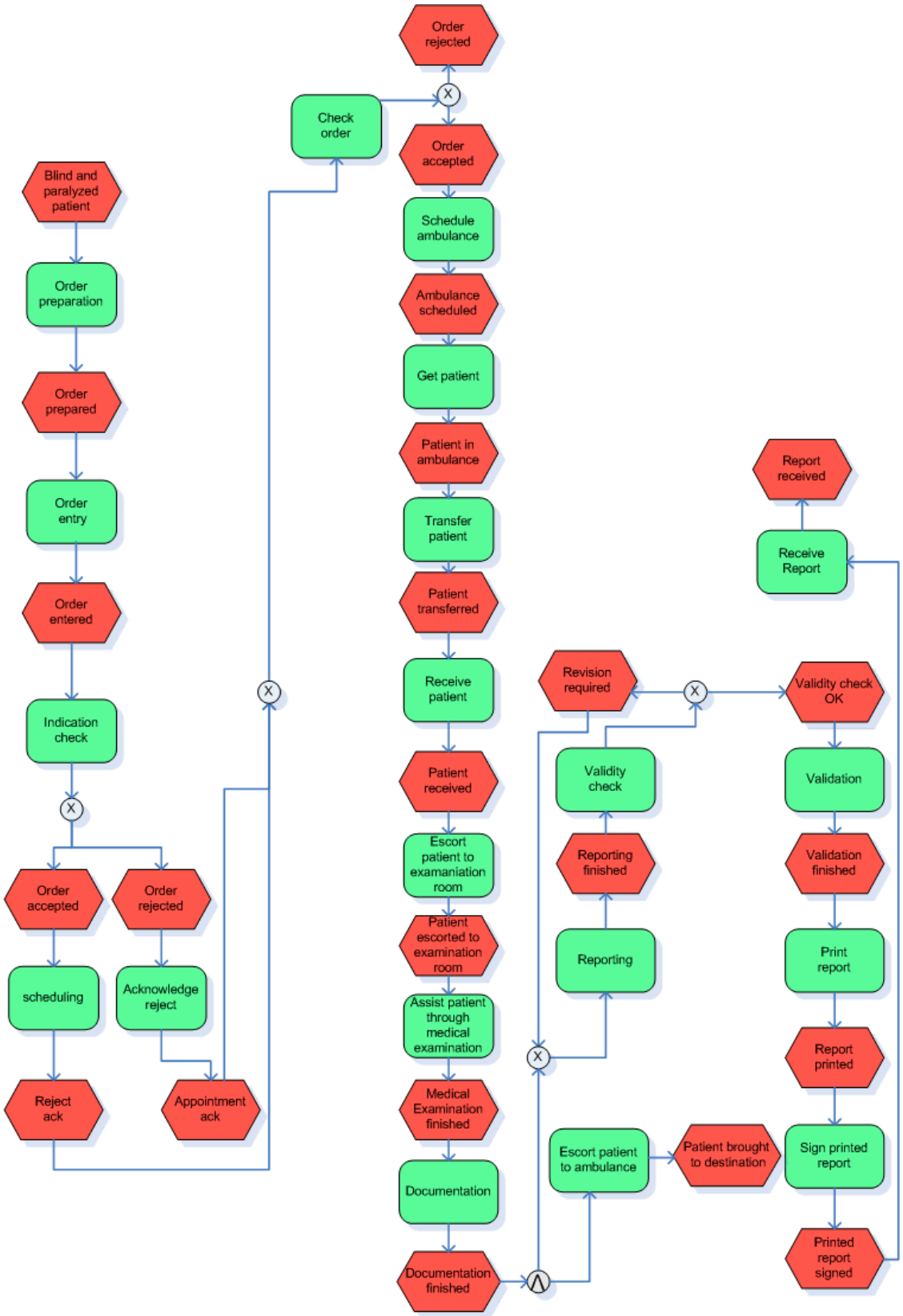


Figure 69: COV-EPC process model configuration with options "Blind patient" and "Paralyzed Patient"

The options, validity/constraints, choices and ambitions shall be defined briefly and precisely because they have already been described in detail in previously.

Options
(Patient without disability, O1)
(Blind patient, O2)
(Paralyzed patient, O3)

Validity/Constraint
Validity: (O1 ∧¬ O2) ∨ (O1 ∧¬ O3) ∨ (O1 ∧¬ O2 ∧¬ O3)

Choice
ValidChoices = {{O1}, {O2}, {O3}, {O2, O3}}.

Ambition
(A1, 2): {(O2)} → (REP, (Patient without disability, E1), (Blind patient, E2))

(A2, 2): {(O2)} → (ADD, **{(**(Patient transferred, E3), (Receive patient, F1)**)**, **(**(Receive patient, F1), (Patient received, E4)**)**, **(**(Patient received, E4), (Escort patient to examination room, F2)**)**, **(**(Escort patient to examination room, F2), (Patient escorted to examination room, E4)**)**, **(**(Patient escorted to examination room, E4), (Medical examination, F7)**)}**)

(A3, 2): {(O2)} → (ADD, **{(**(Documentation finished, E5), (OR, C1)**)**, **(**(OR, C1), (XOR, C2)**)**, **(**(OR, C1), (Escort patient to radiology entrance, F3)**)**, **(**(Escort patient to radiology entrance, F3), (Patient escorted to radiology entrance, E6)**)}**)

(A4, 3): {(O3)} → (REP, (Patient without disability, E1), (Paralyzed patient, E7))

(A5, 3): {(O3)} → (ADD, **{(**(Order accepted, E7), (Schedule ambulance, F4)**)**, **(**(Schedule ambulance, F4), (Ambulance scheduled, E8)**)**, **(**(Ambulance scheduled, E8), (Get patient, F5)**)**, **(**(Get patient, F5), (Patient in ambulance, E9)**)**, **(**(Patient in ambulance, E9), (Transfer patient, F6)**)}**)

(A6, 3): {(O3)} → (ADD, **{(**(Patient transferred, E3), (Receive patient, F1)**)**, **(**(Receive patient, F1), (Patient received, E4)**)**, **(**(Patient received, E4), (Escort patient to examination room, F2)**)**, **(**(Escort patient to examination room, F2), (Patient escorted to examination room, E4)**)**, **(**(Patient escorted to examination room, E4), (Medical examination, F7)**)}**)

(A7, 3): {(O3)} → (REP, (Medical examination, F7), (Assist patient through medical examination, F8))

(A8, 3): {(O3)} → (ADD, **{(**(Documentation finished, E5), (OR, C1)**)**, **(**(OR, C1), (XOR, C2)**)**, **(**(OR, C1), (Escort patient to ambulance, F9)**)**, **(**(Escort patient to ambulance, F9), (Patient brought to destination, E10)**)}**)

(A9, 4): {(O2, O3)} → (REP, (Patient without disability, E1), (Blind and paralyzed patient, E7))

When option O1 is selected, applying the conflict resolution algorithm results in the following steps:

1. Ambition rules for O1 are selected: there are none.
2. No conflicting or redundant rules are detected.
3. No conflict resolution needs to be done.
4. No rules are applied, the resulting process model is the process model described in Figure 66.

When option O2 is selected, applying the conflict resolution algorithm results in the following steps:

1. Ambition rules for O2 are selected: these are A1, A2 and A3.
2. No conflicting or redundant rules are detected.
3. No conflict resolution needs to be done.
4. Rules A1, A2 and A3 are applied, the resulting process model is the process model described in Figure 67.

When option O3 is selected, applying the conflict resolution algorithm results in the following steps:

1. Ambition rules for O3 are selected: these are A4, A5, A6, A7 and A8.
2. No conflicting or redundant rules are detected.
3. No conflict resolution needs to be done.
4. Rules A4, A5, A6, A7 and A8 are applied, the resulting process model is the process model described in Figure 68.

When option O2 and O3 is selected, applying the conflict resolution algorithm results in the following steps:

1. Ambition rules for O2, O3, and both O2 and O3 are selected: A1, A2, A3, A4, A5, A6, A7, A8, A9.
2. Rules A1, A4 and A9 conflict. Rules A8 and A3 conflict. Rules A2 and A6 are redundant.
3. Rule A9 has a higher priority than rule A1 and A4. Rule A8 has a higher priority than rule A3. Rule A6 has a higher priority than rule A2.
4. The following rules are thus applied: A5, A6, A7, A8, A9.

EC1: Mark variable elements
Black and labeled rectangles were introduced to mark variable aspects of the base process model. There are three kinds of rectangles:

- Rectangles labeled with "Add" to denote that process fragments will be added.
- Rectangles labeled with "Delete" to denote that process fragments will be deleted.
- Rectangles labeled with "Replace" to denote that process fragments will be replaced. However only one event, function or connector can be replaced at the same time.

Strengths
The strength of this modeling notation lies in its simplicity: by taking a quick look at the base process model, its variable aspects can be identified.

Weaknesses
However the weakness of this modeling approach is that it is not clear what fragments are added or being replaced with. It is also not clear when a process fragment is being added, deleted or replaced. These process fragments could be added inside of the black rectangles, however this approach is only feasible with small process fragments (Figure 64).

EC2: Support of change patterns
COV-EPC support three change patterns:

- AP1: Insert process fragment
- AP2: Delete process fragment
- AP4: Replace process fragment

Strengths
COV-EPC supports three of the most basic change patterns. Using these three changes patterns most of the necessary process adaptations can be implemented simply and effectively. Furthermore the support of more change patterns such AP2 and AP4 results in smaller configurable process models.

Weaknesses
N/A

EC3: Configuration rules that adapt process model
COV-EPC provides configuration rules to adapt the basic EPC process models. These are captured in the form of ambition rules, where options are linked to process change patterns: Add, Delete, Replace process fragments.

Strengths
The configuration rules have been specified using a context free grammar in Backus-Naur form: the syntax, the contextual constraints and semantics of the configuration rules are thus clear. Furthermore they are precise and unambiguous.

Weaknesses
Specifying the configuration rules leads to long lists of configuration rules: it is thus a time consuming task.

EC4: Visualization of configuration rules that adapt process models
The configuration rules can be visualized using three kinds of black rectangles (Figure 66):

- Add
- Delete
- Replace

Strengths
It is clear by looking at the black rectangles or boxes what type of configuration rule is being applied.

Weaknesses
The weakness of this modeling approach is that it is unclear which options result in additions, deletions or replacements.

EC5: Domain visualization and process model configuration
Attributes of the domain space that have a direct impact on the configuration of the basic EPC process models are captured by options. However these options are not modeled or represented in any way.

Strengths
The resulting base process models annotated with variability markings are simpler without a domain model.

Weaknesses
The domain space cannot be visualized and neither its impact on the configuration of the base process model.

EC6: Domain and process configuration rules
Options capture domain space attributes that have an impact on the configuration of the basic EPC process models. Depending on the Boolean value (true, false, unset) of an option, different change patterns are applied to the base process model.

Strengths
See EC3.

Weaknesses
See EC3.

EC7: Selective display
Selective display can be achieved manually or automatically if a tool is built to support COV-EPC.

Strengths
Manually reconfiguring the base process model to visualize a desired process variant leads to understanding the internal workings of COV-EPC.

Weaknesses
Manually reconfiguring the base process model to visualize a desired process variant requires understanding the internal workings of COV-EPC. This is approach is also quite time consuming.

EC8: Correctness
Ensuring the correctness of process models after deletion, addition or replacement of process fragments is not done in COV-EPC. Eleven and simple design rules can be followed to model correct control flow and avoid problematic behavior such as deadlocks when using basic EPC [3]. EPC can also be extended with formal concepts (Petri-Nets [61-64]) to ensure the syntactic or semantic correctness of the process models [65, 66].

Strengths
As was said in the evaluation of E-EPC (section 4.3.1), eleven design rules can be used to verify the syntactic correctness of basic EPC process models.

Weaknesses
Ensuring the syntactic correctness of the EPC process models needs to be done manually; this is a time-consuming activity. This weakness can be turned into a strength by implementing a tool that validates the correctness of the basic EPC process models by formalizing them using Petri-nets (section 4.3.1).

EC9: Consistency
Using validities and constraints, consistent combinations of options are ensured in COV-EPC. The consistency of configuration rules is guaranteed by using a conflict resolution algorithm with numeric priorities.

Strengths
COV-EPC ensures consistent combinations of options by specifying validities/constraints. Furthermore the consistency of configuration rules is also ensured.

Weaknesses
Assigning numeric priorities to configuration rules must be done manually, although conflict resolution can be done automatically. The maintenance of the configuration rules has also become cumbersome because numeric priorities have to be updated when rules are added, deleted or modified.

6.4 Modeling process variability using the Proteus Configuration Language

6.4.1 Proteus configuration language

The Proteus configuration language (PCL) is a configuration language inspired by module interconnection languages (MILs) [92]. MILs can be used to specify "*architectural, programming language independent, perspective on software design* [92]". Good examples of MILs are MIL75, INTERCOL, NuMIL, SySL, etc.

Sommerville and Dean say that PCL was built with the goal to build an ideal configuration language with the following requirements [92]:

- Integrated systems modeling
  "*The language must be able to model all of the entities and dependencies which make up a system* [92]*.*"

- Multiple structural views
  "*The language must allow different structural views of an entity and system to be constructed* [92]*.*"

- Variability expression
  "*The language must include facilities to represent different versions of a system and to show clearly how one version differs from another* [92]*.*"

- Object-oriented modeling
  "The language must be able to model object-oriented systems [92]."

- User tailorability
  "*The language must allow multi-dimensional, extensible entity classification and user-defined relations* [92]*.*"

Furthermore Sommerville and Dean state that PCL was furthermore designed with the goal to support the following types of system variability [92]:

- Structural variability
  "*The designs of different versions may have different architectures* [92]*.*"

- Implementation variability
  "*The implementation of system components may vary depending on non-functional requirements such as performance and differences in implementation platform* [92]*.*"

- Installation variability
  "*The run-time configuration of the system may vary depending on the execution platform where it is installed* [92]*.*"

PCL provides the following basic concepts or entity types to support the evolution of software systems, hardware systems, documents and their relationships [93]:

- Family entities
  "*Used to define the architecture of hardware, software or documentation components in a system. Family entities may incorporate variability and therefore a single family entity can represent a set of versions of a component [93].*"

- Version descriptor entities
  "*Used to define the specific attributes of a single version of a system [93].*"

- Tool entities
  "*Used to define tools which may be used to build a system whose modelled in PCL. Tool descriptions include a description of the tool inputs and outputs and the command syntax required to execute these tools [93].*"

- Classification definitions
  "*Used to define classification terms which may be associated with a family entity. All classifications are derived from basic classifications including hardware, software and document [93].*"

- Relation definitions
  "*Used to define relations which may exist between family entities, family entities and version descriptor entities or family entities and tool entities in a system description. The relationships derived from these relations are established within entity descriptions as described below [93].*"

- Attribute type definitions
  "Used to define attribute types as an enumerated set of identifiers *[93]*."

An entity description is sectioned and consists of a sequence of named slots (Table 4):

Entity type	**Sections**
family	classification, attributes, interface, parts, physical, relationships
version description	attributes, parts
tool	inputs, outputs, attributes, scripts
relation	domain, range
class	physical, tool
attribute type	enumeration

Table 4: PCL entity types and sections [94]

As suggested by Tryggeseth, Gulla and Conradi family entities form the core concept of PCL [94]: structures of logical components are specified by sets of family entities. Furthermore "*attributes are used to characterize a family and its potential variability* [95]". The following types of attributes are available:

- "*Information attributes state properties common to all members of the family* [95]".

- "*Variability control attributes indicate possible variability among the members of a family* [95]".

6.4.2 Applying PCL to model process variability

PCL describes different versions (variants, revisions) in terms of logical components [94]: a component can be a part of another component, or can have subcomponents, etc. These logical components are afterwards mapped onto physical components, which can be files of source code, documents, modules, etc.

Using PCL, process variability must be modeled using process model components. The whole PCL tool shall not be applied here but only the Proteus Configuration Language's ability to model process variability within the domain space shall be evaluated here.

BPML Selection
A business process modeling language needs to be chosen and combined with PCL. To simplify and smooth the integration between PCL and the BPML, the chosen BPML needs to be simple and extensible: basic EPC shall thus be chosen (Figure 1). BPMN, C-EPC and E-EPC being too complex to be integrated with PCL.

PCL-EPC
By combining basic EPC with PCL, EPC process models must be constructed out of process model components. Capturing the commonalities of process variants within a domain space into one base process model and the differences using distinct process model components is a viable strategy to reduce the number of components that have to be maintained. The base process model could furthermore be constructed by merging the process model of the respective process variants into one process model. The goal is to enable the construction of the process models of the respective process variants within a domain with a minimal amount of components.

This requires the extension of EPC with late selection of process fragments, late modeling of process fragments or late composition of process fragments [54]: concretely basic EPC will be extended with placeholders that can be replaced with an element of a set of pre-existing process fragments (*Figure 70*). An EPC base process model extended with placeholders shall capture the commonalities of the process variants within the domain space. Placeholders mark the differences between process variants within a domain space. Using PCL, these placeholders are deterministically replaced by the appropriate basic EPC process fragments. The integration of PCL with EPC is named PCL-EPC: its meta model is described in Figure 71. Every PCL-EPC is composed of one PCL specification, one base process model and several process model components.

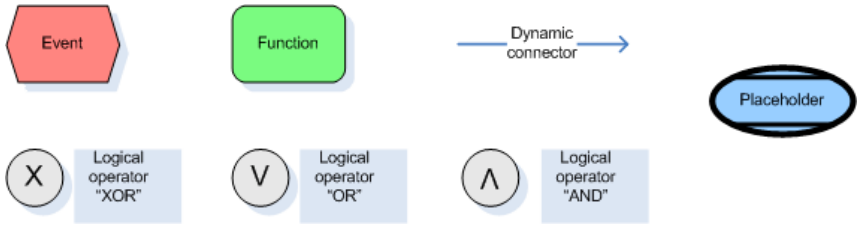


Figure 70: PCL-EPC legend

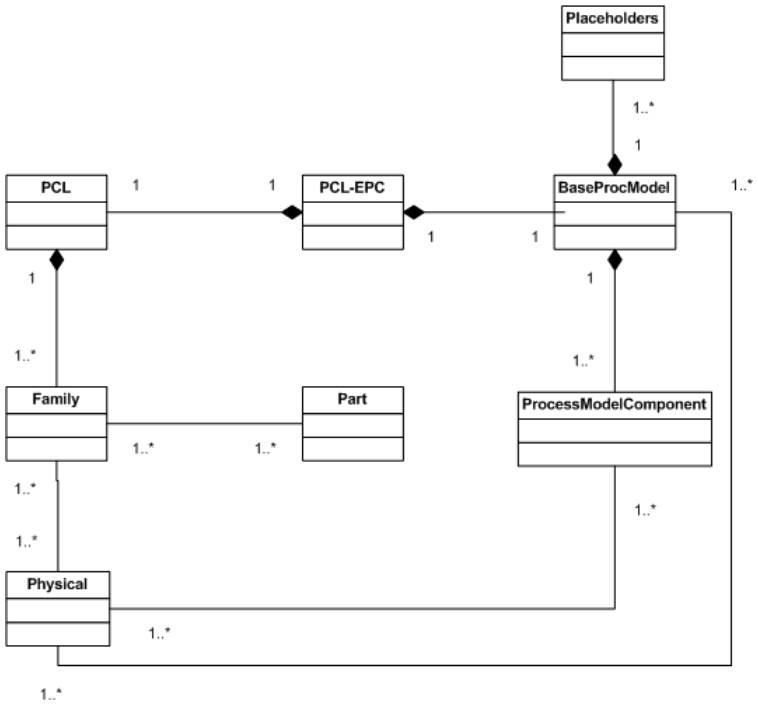


Figure 71: PCL-EPC meta model

Illustration of PCL-EPC using the healthcare running example
The healthcare running example shall thus be modeled using a combination of EPC and PCL: PCL-EPC. The base process model has been modeled in Figure 72, while the other process model components have been modeled in Figure 73, Figure 74, Figure 75, Figure 76, Figure 77, Figure 78, Figure 79, Figure 80 and Figure 81. Finally the PCL has been used to specify configuration rules.

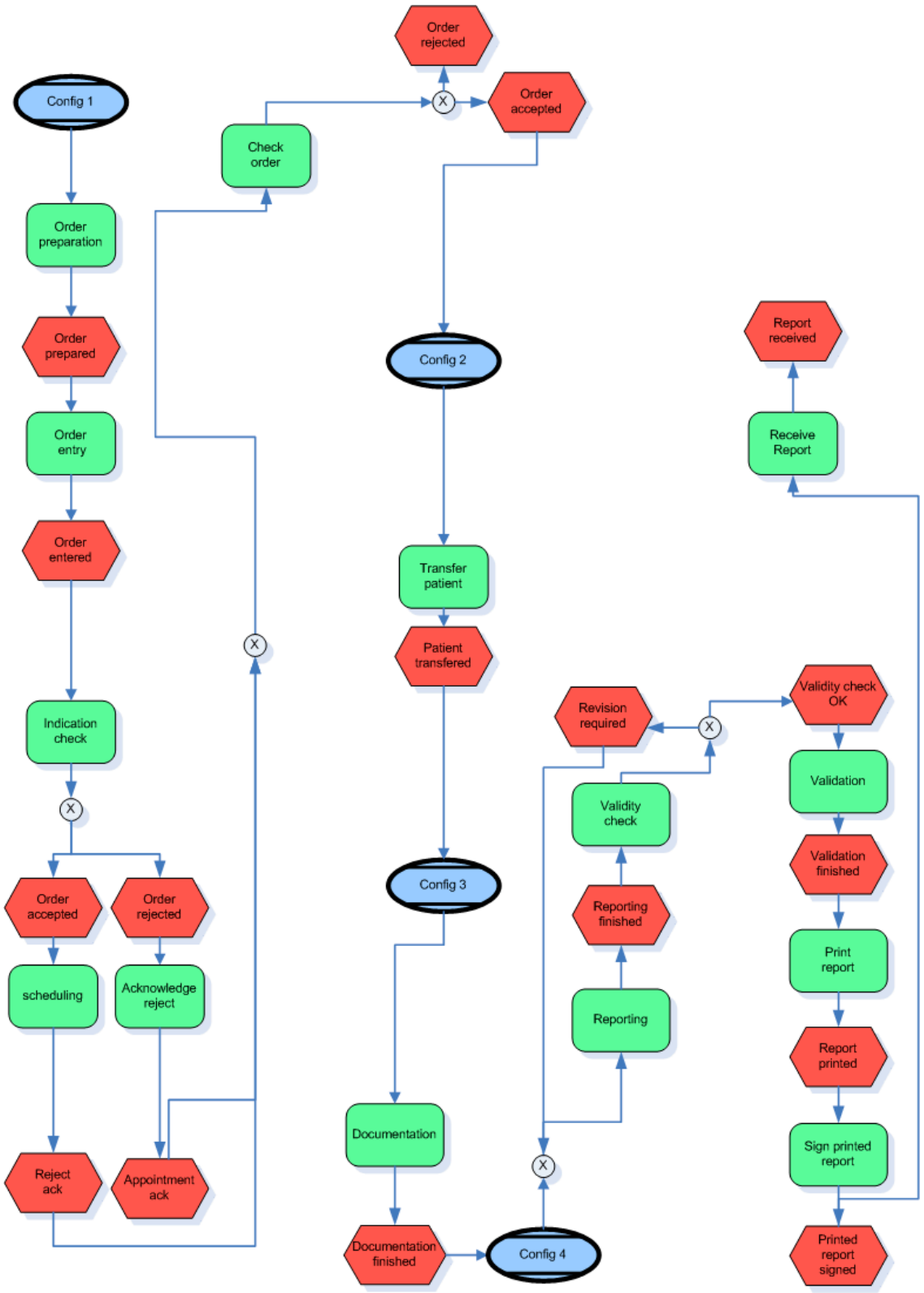


Figure 72: PCL-EPC base process model

```
family RadiologyBaseProcModel
        physical
                proc_model ⇒ "radiology_base_proc_model.pclepc"
        end
end
```

Figure 73: Blind_patient process model component

```
family Blind_patient
        physical
                proc_model ⇒ "blind_patient.pclepc"
        end
end
```

Figure 74: paralyzed_patient process model component

```
family Paralyzed_patient
        physical
                proc_model ⇒ "paralyzed_patient.pclepc"
        end
end
```

Figure 75: patient_without_disability process model component

```
family Patient_without_disability
        physical
                proc_model ⇒ "patient_without_disability.pclepc"
        end
end
```

Figure 76: schedule_ambulance process model component

```
family Schedule_ambulance
        physical
                proc_model ⇒ "schedule_ambulance.pclepc"
        end
end
```

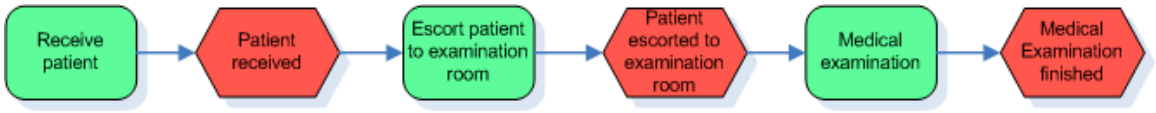


Figure 77: escort process model component

family Escort
    physical
        proc_model ⇒ "escort.pclepc"
    end
end

Figure 78: escort_assist process model component

family Escort_assist
    physical
        proc_model ⇒ "escort-assist.pclepc"
    end
end

Figure 79: medical_exam process model component

family Medical_exam
    physical
        proc_model ⇒ "medical_exam.pclepc"
    end
end

Figure 80: escort_entrance process model component

family Escort_entrance
    physical
        proc_model ⇒ "escort_entrance.pclepc"
    end
end

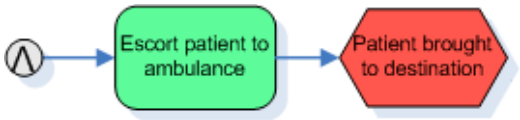


Figure 81: escort_ambulance process model component

```
family Escort_ambulance
        physical
                proc_model ⇒ "escort_ambulance.pclepc"
        end
end

family None
end

attribute_type Patient_type
        enumeration Blind, Paralyzed, Without_disability end
end

family Healthcare_Radiology_Processes
        attributes
                Name: string = "PCL-EPC healthcare running example";
                Patient: Patient_type;
        end

        parts
                BasePM ⇒ RadiologyBaseProcModel

                Config1 ⇒ if Patient = Blind then
                                (Blind_Patient)
                          elsif Patient = Paralyzed then
                                (Paralyzed_patient)
                          elsif Patient = Without_disability then
                                (Patient_without_disability)
                          endif

                Config2 ⇒ if Patient = Paralyzed then
                                (Schedule_ambulance) else (None)
                          endif

                Config3 ⇒ if Patient = Blind then
                                (Escort)
                          elsif Patient = Paralyzed then
                                (Escort_assist)
                          else (Medical_exam)
                          endif

                Config4 ⇒ if Patient = Blind then
                                (Escort_entrance)
                          elsif Patient = Paralyzed then
                                (Escort_ambulance)
                          else (None)
                          endif
        end
end
```

Process model instances of the PCL-EPC specification are the same as the basic EPC process models specified in the healthcare running example (Figure 17, Figure 18, Figure 19).

In the family Healthcare_Radiology_Processes, the slots config2 and config4 can be replaced by None. The family None is completely empty. Replacing the placeholders config2 and config4 by None in the base process model (Figure 72) would result in the deletion of the placeholder and replacing their preceding and outgoing arrows into one single arrow.

Using PCL, consistent combinations of process fragments are explicitly specified. A process model cannot be constructed for a patient that is without disability and blind at the same time. The current PCL specification does not allow the construction of a process model for a patient that is blind and paralyzed at the same. However would this combination be allowed, explicit PCL rules would have to be specified as well as new process fragments.

When PCL-EPC is used to model the healthcare running example some issues arise. Should the use of placeholders be restricted to the modeling of base process models only or all the process model components? Only the core concepts of PCL have been used to model process variability, object oriented inheritance could prove itself helpful when modeling process variability. However Sommerville and Dean assess the limitations of inheritance when modeling variability [93]:

"*The inheritance facility is, of course, an alternative construct for modelling variability. It is possible to define generic components at the base of an inheritance hierarchy and to extend these in different variations. However, variability may be controlled by multiple attribute values (e.g. if a and b and c..). This complex conditional variability is very difficult to express using inheritance [93].*"

6.4.3 Business process variability modeling evaluation

EC1: Mark variable elements
PCL-EPC is extended with placeholders that can be replaced by an element of a set of process fragments or process model components.

Strengths
The chosen modeling approach is really simple and quite easy to use. It results in small and configurable process models.

Weaknesses
Placeholders do not indicate or show the process model components they can be replaced with. Extending PCL-EPC with a tool would make it possible to view the process model components the placeholders can be replaced with by double clicking on them. A domain model with a great number of process variants can result in a base process model with a great number of placeholders.

EC2: Support of change patterns
PCL-EPC support the following change patterns:

- **AP4**: Replace process fragment
- **PP1**: Late selection of process fragments
- **PP2**: Late modeling of process fragments
- **PP3**: Late composition of process fragments

Strengths
PCL-EPC supports only a few of the change patterns, making PCL-EPC quite simple and easy to use.

Weaknesses
Supporting only these change patterns requires the maintenance of quite some process model components.

EC3: Configuration rules that adapt process model
The configuration rules that help construct the process model of the desired process variant within the domain space are specified using PCL.

Strengths
Family entities provide powerful concepts to handle process variability: variability and commonality of process variants is modeled using logical components, which are respectively mapped onto process model components and a base process model.

Weaknesses
The PCL syntax can sometimes be unclear. Furthermore specifying the configuration rules using PCL has resulted in a long list of Proteus configuration rules considering the small size of the healthcare running example.

EC4: Visualization of configuration rules that adapt process models
PCL-EPC does not support the visualization of PCL within the PCL-EPC process models.

Strengths
This results in process models that are quite simple and easy to understand.

Weaknesses
N/A

EC5: Domain visualization and process model configuration
PCL-EPC does not support the visualization of the domain and its impact on the configuration of the PCL-EPC process model.

Strengths
This results in simple and comprehensible process models.

Weaknesses
PCL-EPC should at least support the visualization of the configuration of the domain.

EC6: Domain and process configuration rules
The impact of the domain space on the configuration of the PCL-EPC process models can be specified using PCL.

Strengths
The variability control attributes provided by family entities to model the dimensions of the domain space that have an impact on the configuration of the PCL-EPC process models can be applied simply and effectively. Furthermore the values of slots can be specified using conditional if-statements.

Weaknesses
See EC3.

EC7: Selective display
The current implementation of PCL-EPC supports only manual selective display of the process model of the desired process variant.

Strengths
This leads to understanding the internal workings of PCL-EPC and possibly the detection and correction of errors.

Weaknesses
This requires understanding the internal workings of PCL-EPC. This could also result in inconsistent process models: process models generated out of an inconsistent combination of process model components because of human mistakes.

EC8: Correctness
PCL-EPC does not explicitly support the verification of the syntactic or semantic correctness of the EPC process models. Eleven and simple design rules can be followed to model correct control flow and avoid problematic behavior such as deadlocks [3]. EPC can also be extended with formal concepts (Petri-Nets [61-64]) to ensure the syntactic or semantic correctness of the process models [65, 66].

Strengths
It is extensible with formal concepts to verify and guarantee the syntactic correctness of the generated EPC process models.

Weaknesses
Only the syntactic correctness can be verified using the above mentioned eleven design rules; this is furthermore time consuming.

EC9: Consistency
By specifying strict Proteus configuration rules and applying them strictly, consistent combinations of process model components can be ensured.

Strengths
By automating the combination of process model components using the Proteus configuration language, consistent combinations can be guaranteed.

Weaknesses
Combining process model components by following the Proteus configuration rules is done manually, and there is therefore room for human mistakes.

6.5 Conclusion

Riebisch's feature diagrams have been combined with C-EPC successfully to design Feature-EPC. The feature diagram functions as a domain model of which its configuration leads to C-EPC process model adaptations. Feature-EPC improves C-EPC with domain modeling capability and clearly defined configuration rules. However the problem of big configurable C-EPC models has not been solved. Thankfully the problem of big configurable process models was solved by merging basic EPC with respectively COV and PCL.

COV-EPC models process variability in terms of changes made to a base process model. COV-EPC extend EPC with configuration rules, variability markings, the support of the insert, delete and replace change pattern. The main drawback of COV-EPC is most likely the extensive list of configuration rules that have to be specified and maintained.

PCL-EPC models process variability in terms of family entities composed mostly out of logical and physical components. PCL-EPC extends EPC with configuration rules, placeholders and the support of the late modeling, late composition or late selection change pattern. The main weakness of PCL-EPC is the maintenance of the PCL specification and the process model components.

Chapter 7 Software prototypes

7.1 Introduction

To illustrate the newly designed business process modeling languages (BPMLs), software prototypes were built. A full fledge software demonstration was designed for Feature-EPC, while only a small software demonstration was built for PCL-EPC because of timing constraints.

7.2 Feature-EPC software prototype

7.2.1 Description

This software prototype is a prototype of an environment that can assist the business process modeler in automatically configuring C-EPC process models based on the configuration of feature models.

The prototype of the software environment consists of the Eclipse IDE[3], two Eclipse plug-ins XFeature[4], EPC Tools[5] and two self-programmed Java classes addConfig and FeatureEPC. The eclipse IDE contains and integrates the whole software environment.

The prototype of the software environment works in several stages. Its workings shall be illustrated using the healthcare running example. However a simplified version of the healthcare running example is used in this software prototype.

7.2.2 Designing feature diagrams using XFeature

First a feature diagram of the domain in question is designed using the eclipse plug-in XFeature. In section 6.2.4 has already been explained that only those features having an impact on the business processes should be included into the feature model. The feature diagram of the simplified healthcare running example is illustrated in Figure 82.

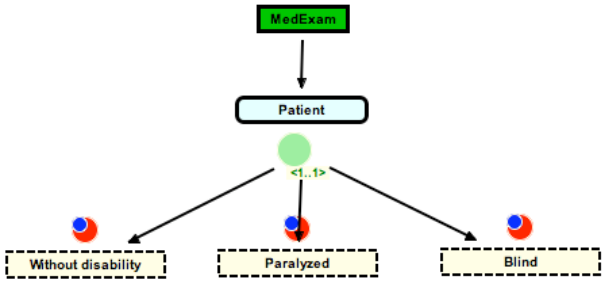


Figure 82: XFeature model of the healthcare running example

[3] http://www.eclipse.org/

[4] http://www.pnp-software.com/XFeature/

[5] http://wwwcs.uni-paderborn.de/cs/kindler/research/EPCTools/

The red and blue ball floating above the features "Without disability", "Paralyzed" and "Blind" are attributes. The program XFeature cannot determine explicitly when a feature is being selected. Therefore an Integer attribute was added:

- When this Integer has value "1", the feature is selected.
- When this Integer has value "0", the feature is not selected.

XFeature saves the XFeature diagram in XML format, easing the analysis and modification of the XFeature XML files.

7.2.3 Creating EPC process models using EPC Tools

Using EPC Tools, a non-configurable EPC process model can be created of a variable business process. A non-configurable EPC process model was thus created for the simplified healthcare running example.

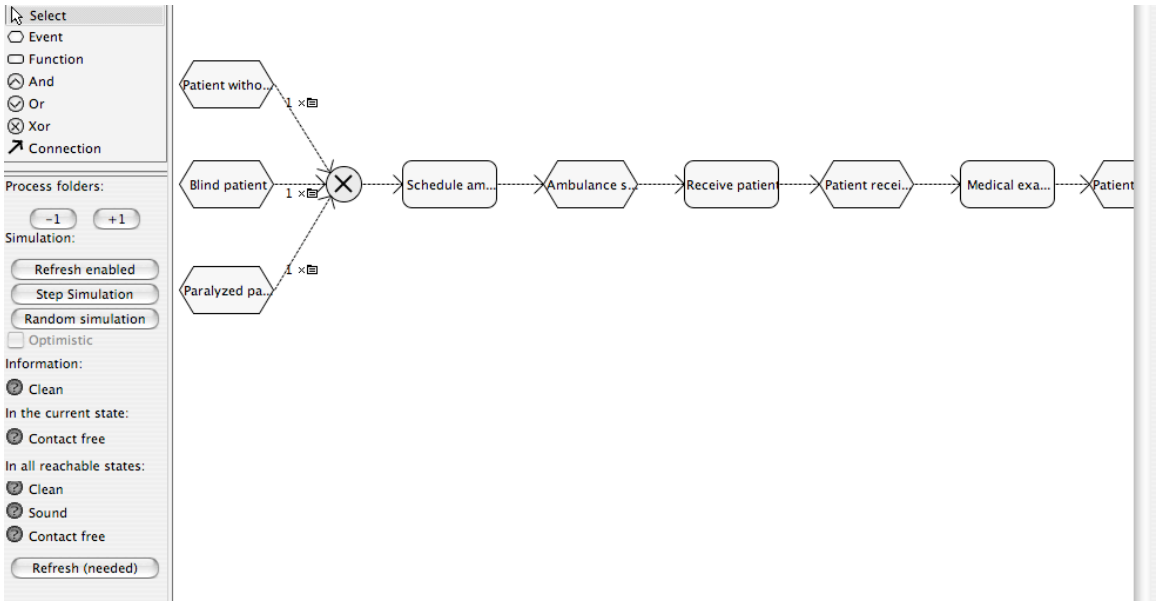


Figure 83: Simplified EPC process model of healthcare running example created using EPC Tools

7.2.4 Transforming EPC into C-EPC process models

Using the java class AddConfig the EPML representation of the EPC process models is annotated with extra information. The attribute "*config*" is added to the elements "*function*", "*xor*" and "*or*" to state their configurable nature:

- If the value of attribute config is set to "*YES*", the function, the XOR or OR logical operator is configurable.
- If the value of attribute config is set to "*NO*", the function, the XOR or OR logical operator is not configurable.

Setting the value of "*config*" is done using the simple user interface provided by the Java program Addconfig.

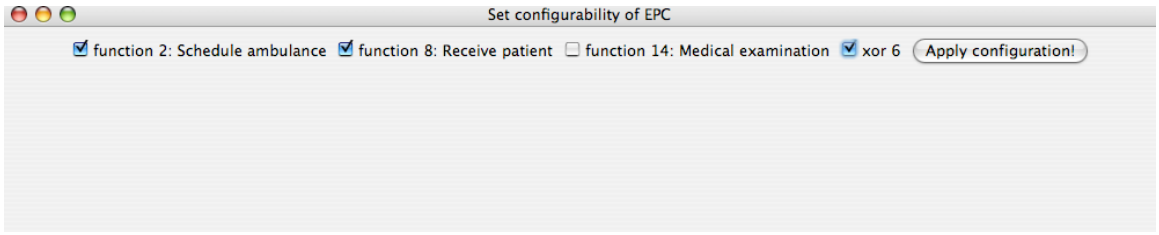


Figure 84: Using AddConfig to transform EPC into C-EPC

As can be seen in Figure 84, function #2, function #8 and xor #6 were selected to be configurable. This is achieved by pressing the button "Apply configuration!". Once the button is pressed an intermediate EPML file is generated, which has been annotated with the new attribute "config".

7.2.5 Generating and applying configuration rules

Using the java class FeatureEPC, configuration rules are automatically generated based on the C-EPC process model: for each configurable node are generated its respective configuration rules. The configuration rules are then linked to each feature, step by step and manually by the business process modeler through the user interface.

First the configuration rules for the feature "Paralyzed" are chosen and afterwards the button "Save configuration rules!" is pressed. For each feature the appropriate configuration rules must be chosen and saved.

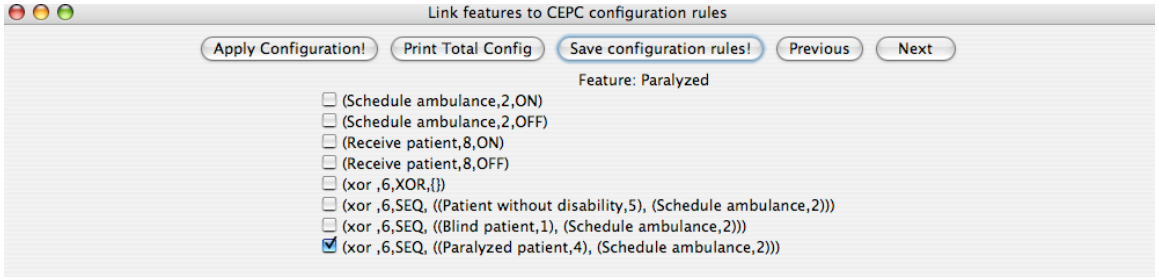


Figure 85: Configuration rules of feature Paralyzed

In order to specify the configuration rules for the feature "Blind", the button "next" is pressed. The configuration rules for the feature "Blind" are specified and saved the same way as the feature paralyzed.

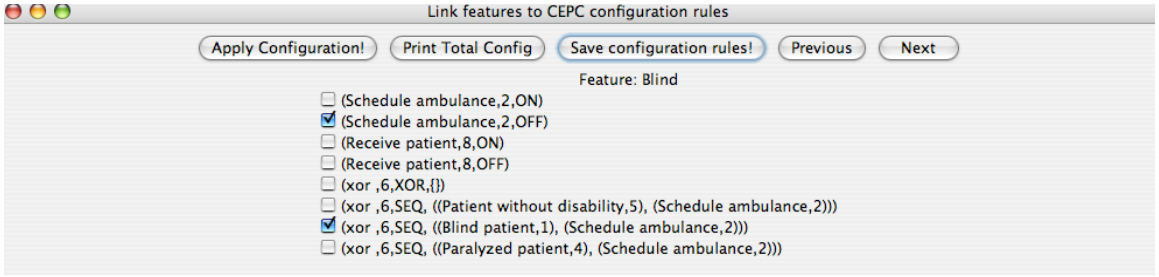


Figure 86: Configuration rules of feature Blind

Again by pressing the button "next", the configuration rules for the next feature can be specified; in this case the feature "Without disability".

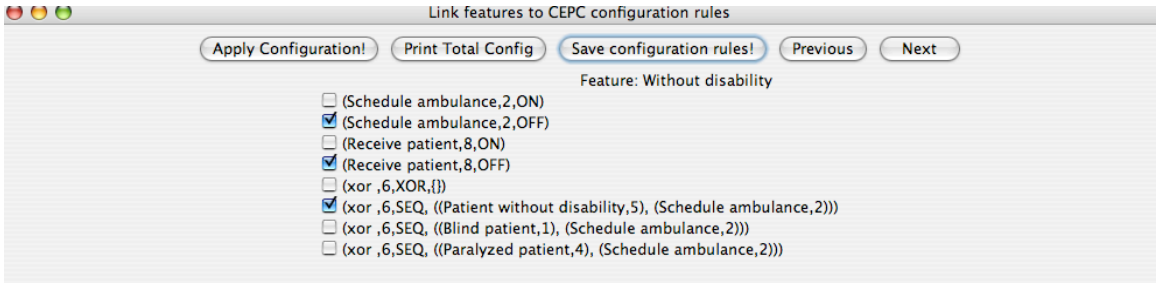


Figure 87: Configuration rules of feature Without disability

When for all features the right configuration rules have been specified, these can be listed and visualized by pressing the button "Print Total Config".

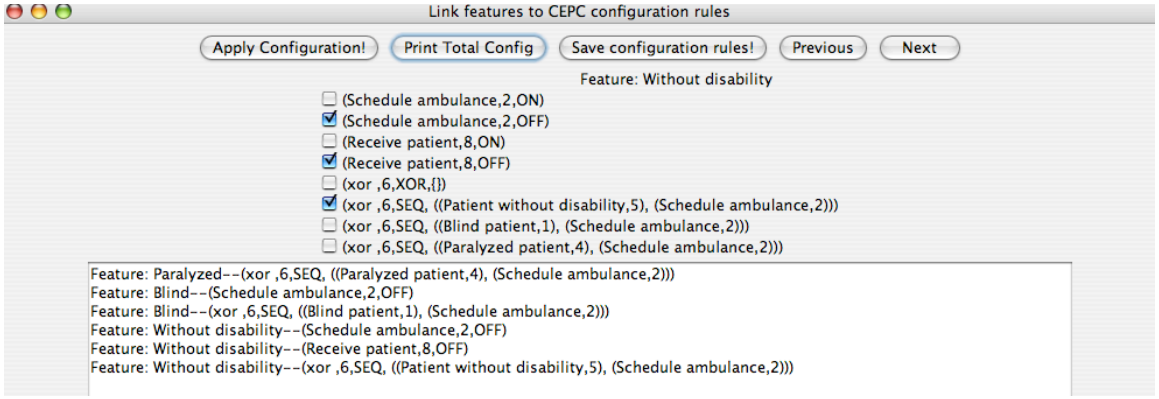


Figure 88: Print and visualize configuration rules

Once all the rules have been linked appropriately to features, by pressing the button "Apply Configuration!", based on the selected features of the XFeature diagram, configuration rules are selected and applied. This results in the automatic configuration of the C-EPC process model. When the feature blind is selected, the following EPC process model is generated automatically (Figure 89).

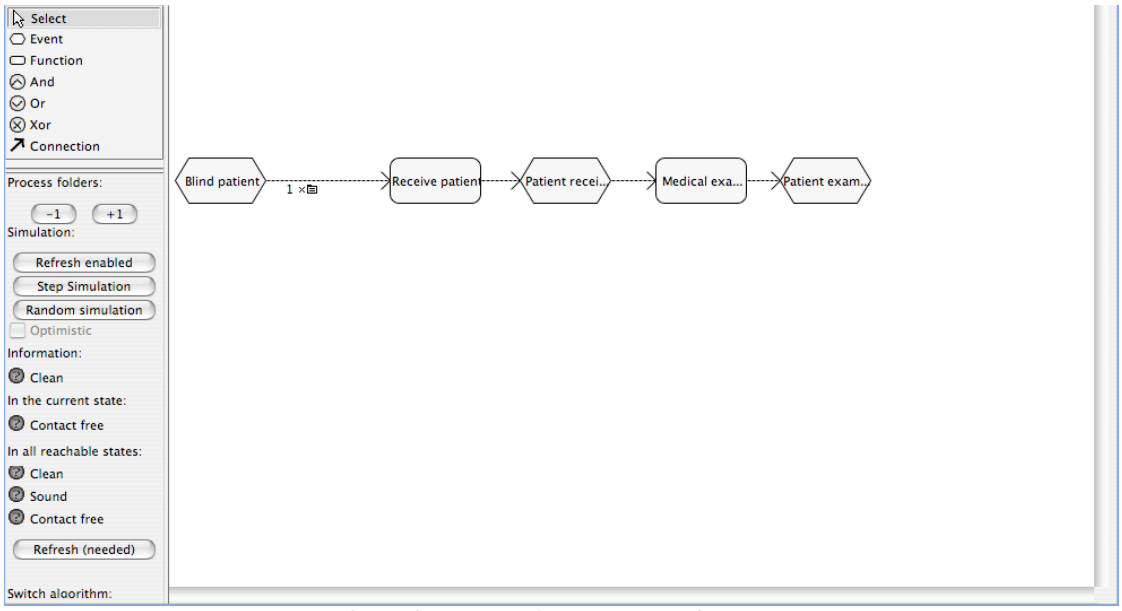


Figure 89: Generating and applying configuration rules using Feature-EPC

7.2.6 Limitations and improvements

The software environment that was built is a prototype and therefore has its limitations. The software prototype shows its limitations when the configuration rules are applied and when several features have been selected at once.

A configurable function can either be turned "*ON*" or "*OFF*". When a configurable function is turned "*OFF*", the EPC process model needs to be modified in the following way:

1. When the configurable function is followed by an event; the function, its following event and the arc that connects the two nodes must be deleted. However the location of the configurable function also plays an important role in the deletion task:

    a. When the configurable function marks the start of a process model then an additional arc must be deleted: the arc following the event that follows the configurable function.

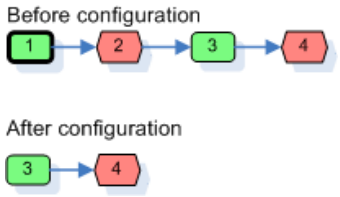


Figure 90: Deletion of configurable function (case first)

    b. When the configurable function marks the end of a process model then an additional arc must be deleted: the ingoing arc that points to the configurable function.

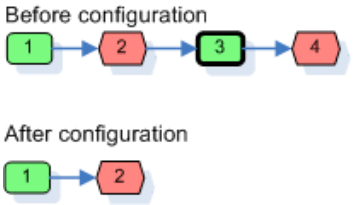


Figure 91: Deletion of configurable function (case last)

    c. When the configurable function is enclosed by several nodes, two additional arcs must be deleted: the arc following the event that follows the configurable function and the ingoing arc that points to the configurable function.

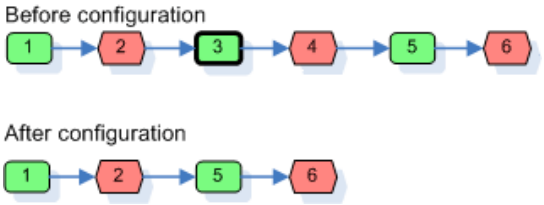


Figure 92: Deletion of configurable function (case enclosed)

2. When the configurable function is followed by a logical operator (XOR, OR, AND) things become more complicated.

    a. In the situation described by *Figure 93*, everything including the configurable function and what follows it must be deleted.

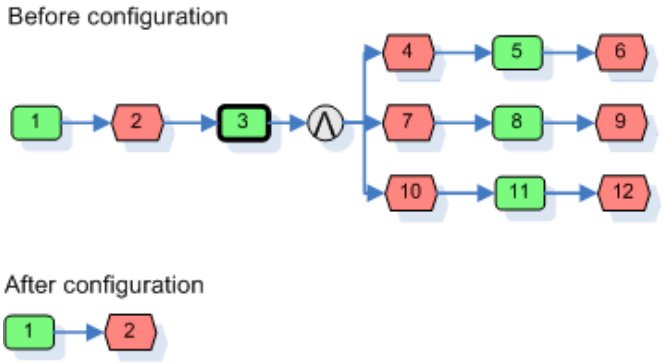


Figure 93: configurable function followed by logical operator (case simple)

    b. In the situation described by *Figure 94*, only the configurable function, and what is between the two logical operators, including the two logical operators must be deleted.

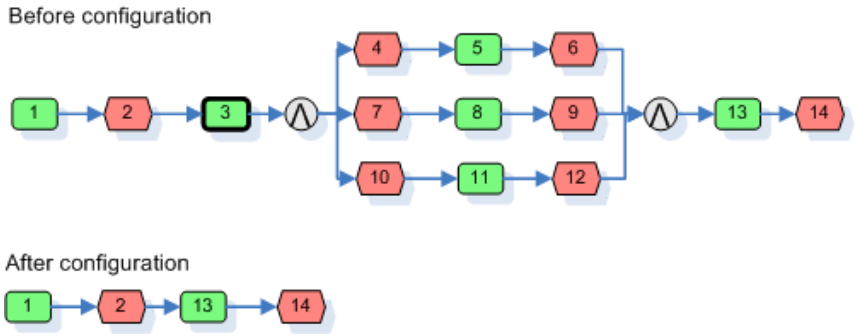


Figure 94: configurable function followed by logical operator (case complex)

A configurable logical operator (XOR, OR) can be reconfigured into a sequence; quite some complications can then arise.

3. In *Figure 95*, the assumption is made that sequences of nodes are interconnected by one single logical configurable operator. The application of the configuration rule (or,1,SEQ,((3,3),(13,13)) implies the following (*Figure 95*):
    a. The deletion of node 6 and everything that precedes node 6.
    b. The deletion of nodes 7 and everything that follows node 7.
    c. The deletion of nodes 10 and everything that follows node 10.
    d. The deletion of all arcs connected to the deleted nodes.
    e. The deletion of the logical operator itself.
    f. Drawing a new arc between 3 and 13.

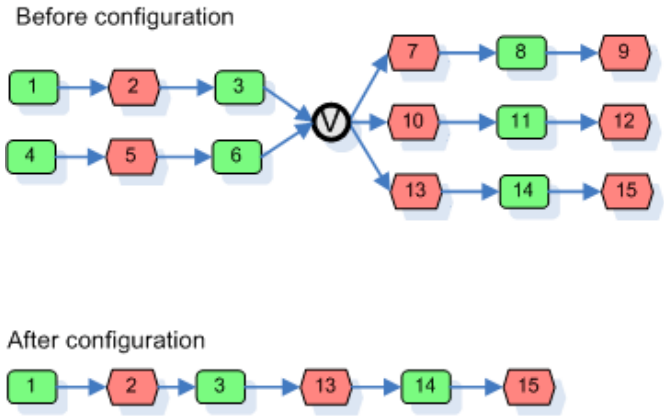


Figure 95: configurable logical operator (case simple)

4. In *Figure 96*, the situation becomes more complicated: sequences of nodes are interconnected by several logical operators, of which some are configurable. Analyzing the process model is thus needed to determine which sequences of nodes may be deleted.

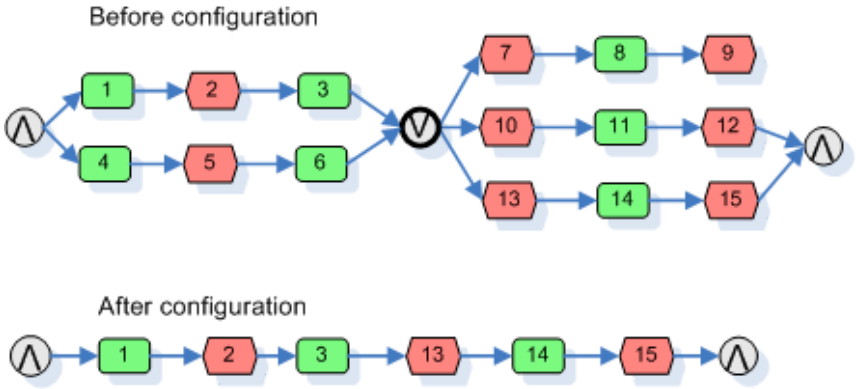


Figure 96: configurable logical operator (case complex)

5. Operator precedence was also introduced into the prototype by ordering the configuration rules and applying first the configuration rules involving logical operators (XOR, OR) and afterwards configurable functions. Observing and analyzing *Figure 97*, applying the following configuration rules shall lead to a conflict:

   CR1: (9,9,"OFF")
   CR2: (or,1,SEQ,((2,2),(9,9))

   Configuration rule CR2 must be applied before CR1 to avoid a conflict. If CR1 is applied before CR2, the function with name "9" shall be turned OFF with the following consequence: configuration rule CR2 is now inapplicable because function "9" has been deleted from the process model.

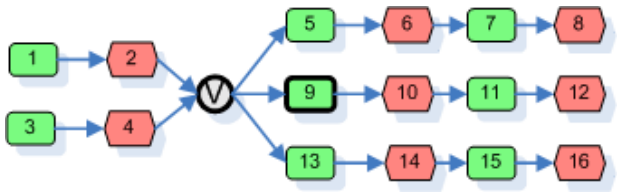


Figure 97: simple conflict resolution

6. The nesting of configuration rules was not introduced into the prototype. These are rules of the following kind:
   (xor, ID, SEQ, XORRule)
   (xor, ID, SEQ, ORRule)
   (or, ID, SEQ, EPCSequence)
   (or, ID, SEQ, XORRule)
   (or, ID, SEQ, ORRule)

7. However if several features are selected with common and different rules, the prototype shall function as long as there are no conflicts between the configuration rules: the prototype of the software environment does not implement any conflict resolution algorithm.

Only the features 1a, 1b, 1c, 3, 5 and partially 7 have been implemented in the prototype. The software prototype has yet to be improved with the implementation of Feature 2, 4, 6 and partially 7.

7.3 PCL-EPC software demo

7.3.1 Description and application scenario

This software prototype has been constructed using the Eclipse IDE, Java classes and the eclipse plug-in EPC Tools. A simplified version of the healthcare running example shall be used to illustrate PCL-EPC.

This software prototype is quite simple. As described in Figure 98, all the user needs to do is type in either the values "Blind", "Paralyzed" or "Without disability". This will automatically construct the resulting EPC process model by integrating the base process model with the correct process model components. The resulting process model is afterwards saved in the EPML file MedExamConfig. This file can be opened and viewed using the Eclipse plug-in EPC Tools (Figure 99).

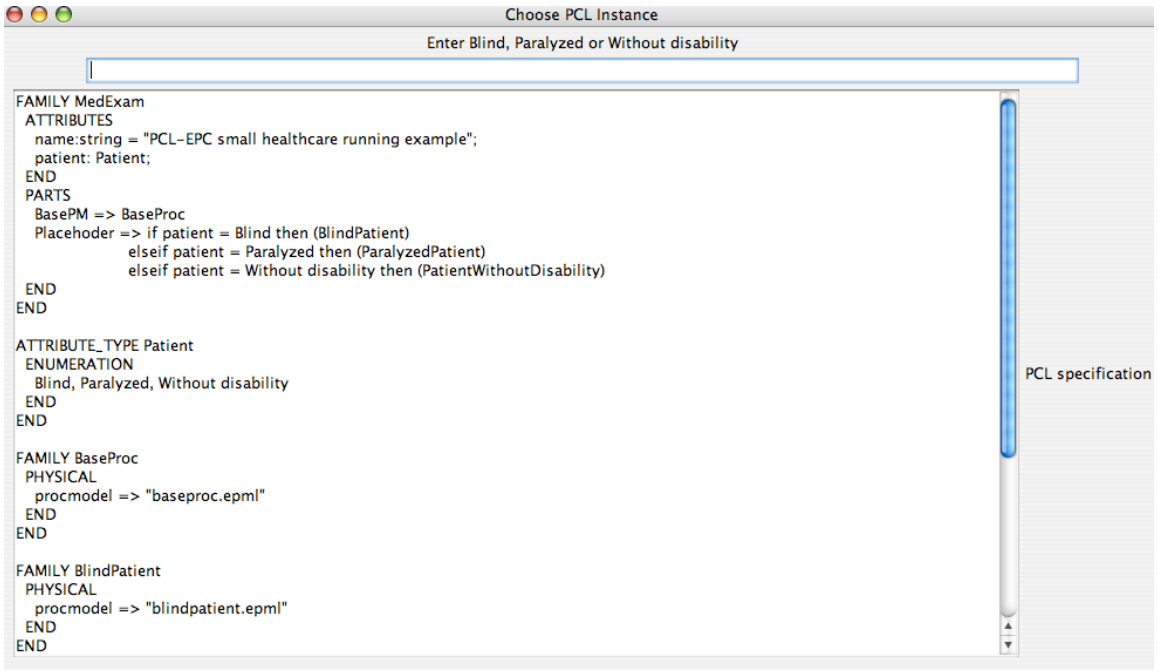


Figure 98: PCL-EPC demo

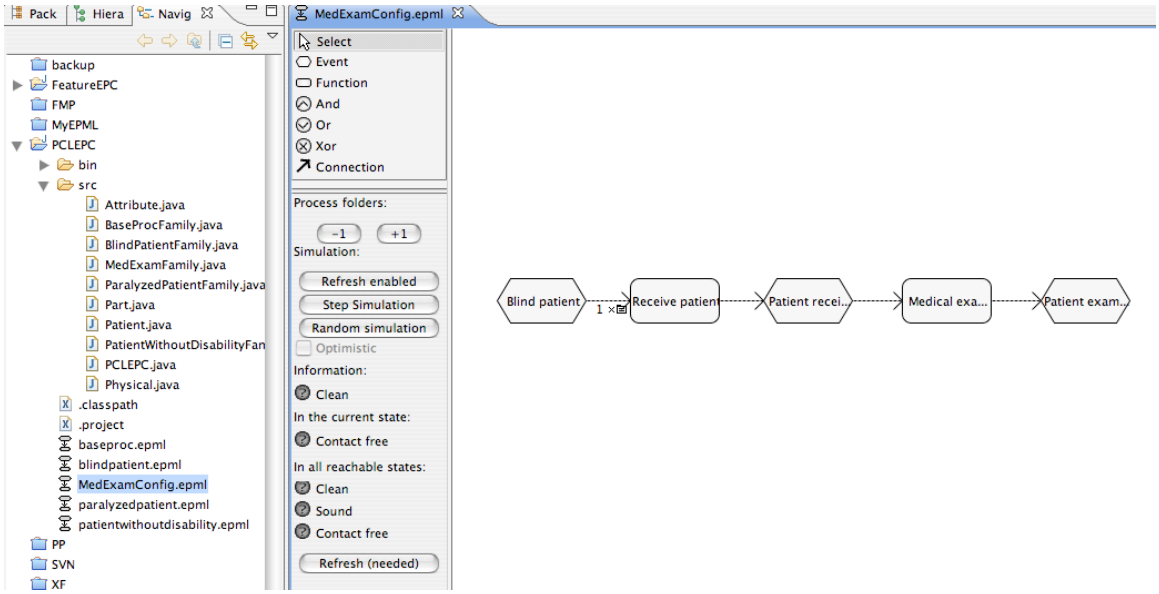


Figure 99: PCL-EPC MedExamConfig process model of blind patient (EPC-Tools)

7.3.2 Limitations and improvements

This prototype is not configurable. A change to the PCL specification or EPC process models requires the manual update of the software code, which is quite cumbersome.

This PCL-EPC software prototype can be improved by specifying a simple context free grammar for the PCL. Analyzing the grammar of the PCL specification and by passing the value of relevant attributes through a simple GUI would enable the automatic generation of the process model of the desired process variant.

7.4 COV-EPC software demo

No software demonstration was built for the newly designed business process modeling language COV-EPC.

7.5 Conclusion

With these software prototypes has been demonstrated that PCL-EPC and Feature-EPC are mature enough to be used when modeling business process variability within the domain space.

Chapter 8 Conclusion, evaluation and future work

Business process variability has been characterized by business process variability within the domain space and over time (Figure *10*). Business process variability within the domain space leads mainly to business process variability modeling problems, while business process variability over time leads to process model evolution problems (*Figure 22*). In this research project, the focus is on analyzing and solving process variability modeling problems. Crafting a set of evaluation criteria (section 4.2), current and newly created business process modeling languages are evaluated on the modeling concepts they provided to model process variability (Table 5).

EPC provide specific modeling concepts to model goals, resources, outputs, and organizational units whereas C-EPC, Feature-EPC, COV-EPC, PCL-EPC and BPMN don't. Using EPC, process variability can only be modeled using one or separate process models. In case of a large number of process variants within the domain space, the best solution is modeling all the process variants using distinct and separate process models. However this modeling choice comes with process model evolution problems (*Figure 22*).

BPMN provides swimlanes that can be used to model and visualize quite effectively resources, organizational units, etc. Using BPMN, process variability can only be modeled using one or separate process models. BPMN and EPC have the same weaknesses when it comes to modeling process variability. In case of a large number of process variants, the best solution is also modeling all the process variants using separate and distinct process models. As was said previously this approach introduces process model evolution problems.

C-EPC comes with configurable nodes and attributes. Functions and connectors are configurable. Furthermore, C-EPC provides configuration guidelines and requirements to direct the configuration of the process models whereas EPC and BPMN don't; these are specified using logical expressions of which the syntax is found unclear. Moreover C-EPC only support the delete and replace process fragments change patterns; this leads to big configurable C-EPC process models because their configuration is mainly done by deleting parts of the process model.

The field of software product line engineering (SPLE) has in common with business process variability that they suffer from the same type of variability: variability within the domain space and variability over time (Figure 41). Software product line engineering has come up with its own solutions to manage software variability: feature diagrams and software configuration management (SCM) (Figure *42*). The goal was to extend or combine current business process modeling languages, with feature diagrams and variability management concepts borrowed from software configuration management systems, to provide them with useful concepts to model process variability. This lead to the creation of three new business process modeling languages: Feature-EPC, COV-EPC and PCL-EPC.

Feature-EPC improves and extends C-EPC on several points. The logical expressions used to specify the configuration rules have been replaced by modular configuration rules specified using a context free grammar in Backus-Naur form: they are precise and unambiguous. Furthermore C-EPC have been extended with domain modeling capacity by integrating C-EPC with Riebisch's feature diagrams. The configuration of

the Riebisch feature diagram guides the configuration of the C-EPC process models using configuration rules. Using the constraints "*requires*" and "*excludes*" provided by Riebisch feature diagrams, consistent combinations of features can be ensured. Furthermore consistent combinations of configuration rules are guaranteed by applying a conflict resolution algorithm using numeric priorities. However Feature-EPC and C-EPC still have one weakness in common, the support of only the delete and replace process fragments change patterns; this has as a consequence big configurable process models that are hard to modify and maintain.

In COV-EPC, process variants within a domain space are specified in terms of changes made to a base process model. The core concepts of COV-EPC are options, choices and ambitions. Options are logical changes that capture aspects of the domain space that cause process variability. A choice is a set of selected options. Specifying validities or constraints upon these options ensures consistent combinations of options. Ambition rules map option combinations to changes made to the base process model using either insert, delete or replace process fragments; this leads to small configurable base process models. Furthermore the ambition rules have been specified using a context free grammar in Backus-Naur form: they are thus precise and unambiguous. Thankfully applying a conflict resolution algorithm using numeric priorities also ensures consistent combinations of ambition rules here. However the main weakness of this approach is the large number of ambition rules necessary to specify the process models of the respective process variants within the domain space and thus their maintenance.

In PCL-EPC, process variants are mainly specified in terms of families of logical components and process model components. EPC have been extended with placeholders that can be replaced by an element of a set of process model components. An EPC process model extended with placeholders fulfills the function of a base process model that captures the commonality of process variants within the domain space. Thus the change pattern late selection, late modeling or late composition of process fragments has been implemented here; this leads to small and simple configurable base process models. The variability of the process variants within the domain space is captured in separate process model components. The placeholders of the EPC base process model can selectively be replaced by one of these process model components to construct the process model of the desired process variant. This process is guided by rules specified using the Proteus configuration Language of which the syntax can sometimes be unclear to the untrained eye. A main drawback of this approach is the maintenance of the large amount of Proteus configuration rules and process model components.

	Current business process modeling languages			New business process modeling languages		
Evaluation Criteria	**EPC**	**BPMN**	**C-EPC**	**Feature-EPC**	**COV-EPC**	**PCL-EPC**
EC1	n/a	n/a	+/-	+/-	+/-	+/-
EC2	n/a	n/a	+/-	+/-	+	+/-
EC3	n/a	n/a	-	+/-	+/-	+/-
EC4	n/a	n/a	+/-	+/-	+	+
EC5	-	+/-	+/-	+/-	+/-	+/-
EC6	n/a	n/a	-	+	+/-	+/-
EC7	-	-	+/-	+/-	+/-	+/-
EC8	+/-	+/-	+/-	+/-	+/-	+/-
EC9	n/a	n/a	-	+	+	+

Legend: very good(++), good(+), Just good enough(+/-), bad(-), very bad(--), (n/a) not available

Table 5: Total process variability modeling evaluation summary

EPC and BPMN do not provide any explicit means to model process variability. The advice is therefore not to use them when trying to model business process variability, or use them exceptionally in the case of very few process variants. When modeling business process variability, the advice is to choose Feature-EPC, PCL-EPC or COV-EPC over C-EPC because they come with domain modeling capability, concise configuration rules and conflict resolution algorithms. Finally when having to model business process variability, the advice would be to choose Feature-EPC over PCL-EPC and COV-EPC because of their visual character and that despite their main drawback of larger configurable process models. Using Feature-EPC, the domain of the business process is visualized using a feature diagram, and upon selection of features the business process is reconfigured to comply with the configuration of the domain. PCL-EPC and COV-EPC are also capable of doing this but do not support the visualization of the domain. Moreover interesting future research would be to combine feature diagrams with EPC base process models extended with placeholders supporting mainly the change pattern late selection, late modeling or late composition of process fragments: this would lead to the creation of an innovative business process modeling language with domain visualization and compact configurable process models.

8.1 Recommendations and future research

Future research includes mainly the following three points:

- The complete formalization of the newly created business process modeling languages.
- The extension of the newly created business process modeling languages with complete software environments.
- Solving the process model evolution problems (Figure 22) of the newly created business process modeling languages.

Future work includes the complete formalization of the newly created business process modeling languages, Feature-EPC, COV-EPC and PCL-EPC, the same way previously created business process modeling languages have been formalized [4, 33, 35].

The three newly created business process modeling languages should be extended with complete software environments. This software environment should permit:

- The creation, deletion and modification of process models.
- The automatic conflict resolution of redundant and conflicting configuration rules.
- The verification of the syntactic correctness of the process models.
- The automatic construction of base process models by merging the process model of respective process variants into one process model.

Finally and most importantly is resolving the process model evolution problems of Feature-EPC, COV-EPC and PCL-EPC. New research questions arise: how can the newly created business process modeling languages be used to model both business process variability within the domain space and over time? How can the configuration rules be maintained simply and effectively? SCM was originally applied to model and handle software variability within the domain space and over time. It is highly probable that variability modeling concepts borrowed from SCM can be applied to model and handle process variability within the domain space and over time. In this research project, SCM variability modeling concepts have been combined with current business process modeling language to only model business process variability within the domain space. Future research thus involves the application of SCM variability modeling concepts to model and handle both business process variability within the domain space and over time. Using for example COV-EPC, options would not only be used to model attributes of the domain space but also incremental changes, improvements, etc. The same line of thought can be applied when using PCL-EPC, process models of process variants and process revisions can be constructed using PCL specifications and process fragments.

Chapter 9 References

1. H.A. Reijers, *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. Lecture Notes in Computer Science. Vol. 2617. 2003: Springer.
2. CIO Definitions. *business process management*. 16-01-2006 [cited 13-08-2007]; Available from: http://searchcio.techtarget.com/sDefinition/0,290660,sid19_gci1088464,00.html.
3. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede, *Process-Aware Information Systems Bridging People and Software Through Process Technology*. 2005: Wiley Interscience.
4. M. Rosemann and W.M.P. van der Aalst, *A configurable reference modeling language.* Information systems, 2007. **32**(1): p. 1-23.
5. A. Dreiling, M. Rosemann, W.M.P. van der Aalst, W. Sadiq, and S. Khan. *Model-Driven Process Configuration of Enterprise Systems*. in *Wirtschaftsinformatik: eEconomy, eGovernment, eSociety, 7. Internationale Tagung Wirtschaftsinformatik 2005, Bamberg, 23.2.2005 - 25.2.2005*: Physica-Verlag.
6. S.A. White, *Introduction to BPMN*. 2006, IBM Software Group.
7. OMG, *Business Process Modeling Notation (BPMN) Specification - Final Adopted Specification dtc/06-02-01*. 2006.
8. Object Management Group/Business Process Management Initiative. *Business Process Modeling Notation (BPMN) Information*. 09-07-2007 [cited 28-08-2007]; Available from: http://www.bpmn.org/.
9. S.F. King and O.A. Johnson, *VBP: An approach to modelling process variety and best practice.* Information and Software Technology, 2006. **48**(11): p. 1104-1114.
10. M. Haag. *Random Processes: Mean and Variance*. 05-04-2005 [cited 31-05-2007]; Available from: http://cnx.org/content/m10656/latest/.
11. S.-C. Chou and J.-Y.J. Chen, *Process evolution support in concurrent software process language environment.* Information and Software Technology, 1999. **41**(8): p. 507-524.
12. B. Burmeister, H.-P. Steiert, T. Bauer, and H. Baumgärtel. *Agile Processes Through Goal- and Context-Oriented Business Process Modeling*. in *Business Process Management Workshops*. 2006: Springer.
13. M. Aoyama. *Agile Software Process model*. in *COMPSAC*. 1997: IEEE Computer Society.
14. B. Weber, M. Reichert, W. Wild, and S. Rinderle. *Balancing Flexibility and Security in Adaptive Process Management Systems*. in *OTM Conferences (1): CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part I*. 2005: Springer.
15. M. Soto and J. Münch. *Process Model Difference Analysis for Supporting Process Evolution*. in *EuroSPI*. 2006: Springer.
16. W.M.P. van der Aalst. *Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information?* in *CoopIS*. 1999: IEEE Computer Society.
17. W.M.P. van der Aalst, *Flexible workflow management systems: an approach based on generic process models.* Lecture Notes in Computer Science, 1999. **1677**: p. 186-195.

18. W.M.P. van der Aalst and T. Basten, *Inheritance of workflows: an approach to tackling problems related to change.* Theoretical Computer Science, 2002. **270**(1-2): p. 125-203.
19. M. Zhang and M.M. Tseng, *A Product and Process Modeling Based Approach to Study Cost Implications of Product Variety in Mass Customization.* Engineering Management, IEEE Transactions on, 2007. **54**(1): p. 130-144.
20. J. Jiao and M.M. Tseng, *Customizability analysis in design for mass customization.* Computer-Aided Design 2004. **36**(8): p. 745-757.
21. C.W. Krueger. *Variation Management for Software Production Lines*. in *SPLC*. 2002: Springer.
22. A. Ocampo and J. Münch. *Process Evolution Supported by Rationale: An Empirical Investigation of Process Changes*. in *SPW/ProSim*. 2006: Springer.
23. A.-W. Scheer, *ARIS - Business process modeling*. Second, Completely Revised and Enlarged Edition. 1999: Springer-Verlag Berlin - Heidelberg.
24. R. Lenz and M. Reichert, *IT support for healthcare processes - premises, challenges, perspectives.* Data Knowl. Eng., 2007. **62**(1): p. 39-58.
25. K. Sadegh-Zadeh, *Fundamentals of clinical methodology - 4. Diagnosis.* Artificial Intelligence in Medecine, 2000. **20**(3): p. 227-241.
26. A. Sutcliffe, *The domain theory: patterns for knowledge reuse and software reuse*. 2002, Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc., Publishers.
27. R. Duray, P.T. Ward, G.W. Milligan, and W.L. Berry, *Approaches to mass customization: configurations and empirical validation.* Journal of Operations Management, 2000. **18**(6): p. 605-625.
28. G. Da Silveira, D. Borenstein, and F.S. Fogliatto, *Mass customization: Literature review and research directions*, in *International Journal of Production Economics*. 2001, Elsevier. p. 1-13.
29. S. Rinderle, M. Reichert, and P. Dadam, *Correctness criteria for dynamic changes in workflow systems - a survey.* Data Knowl. Eng., 2004. **50**(1): p. 9-34.
30. L.T. Ly, S. Rinderle, and P. Dadam, *Semantic Correctness in Adaptive Process Management Systems.* Lecture Notes in Computer Science, 2006. **4102**: p. 193-208.
31. J. Jiao, L. Zhang, and K. Prasanna, *Process Variety Modeling for Process Configuration in Mass Customization: An Approach Based on Object-Oriented Petri Nets with Changeable Structures.* International Journal of Flexible Manufacturing Systems, 2004. **16**(4): p. 335-361.
32. M. La Rosa, F. Gottschalk, M. Dumas, and W.M.P.v.d. Aalst. *Domain-driven Reference Process Model Configuration*. in *Proceedings of the BPM 2007 Workshops (to appear)*. 2007.
33. M. La Rosa, J. Lux, S. Seidel, M. Dumas, and A.H.M.t. Hofstede, *Questionnaire-driven Configuration of Reference Process Models.* Lecture Notes in Computer Science, 2007. **4495**: p. 424-438.
34. M. La Rosa, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede, *Variability Modeling for Questionnaire-based System Configuration*, in *QUT ePrints*. 2007, Queensland University of Technology.
35. F. Gottschalk, W.M.P.v.d. Aalst, M.H. Jansen-Vullers, and M.L. Rosa, *Configurable Workflow Models*. 2007, Eindhoven University of Technology: The Netherlands.
36. W.M.P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M.H. Jansen-Vullers. *Configurable Process Models as a Basis for Reference Modeling*. in *Business Process Management Workshops*. 2005.

37. O. Thomas. *Understanding the Term Reference Model in Information Systems Research: History, Literature Analysis and Explanation*. in *Business Process Management Workshops*. 2005.
38. Wikipedia. *Database Management System*. 11 July 2007 [cited 12 July 2007]; Available from: http://en.wikipedia.org/wiki/Database_management_system.
39. M.L. Jaccheri and R. Conradi, *Techniques for process model evolution in EPOS.* IEEE Trans. Software Eng., 1993. **19**(12): p. 1145-1156.
40. B. Weber, S. Rinderle, W. Wild, and M. Reichert. *CCBR-Driven Business Process Evolution*. in *ICCBR*. 2005: Springer.
41. S.-C. Chou and J.J.-Y. Chen, *Process program change control in a process environment.* Softw., Pract. Exper., 2000. **30**(3): p. 175-197.
42. P. Zipkin, *The limits of mass customization.* MIT Sloan Management Review, 2001. **42**(3): p. p81-p88.
43. J. Jiao, Q. Ma, and M.M. Tseng, *Towards high value-added products and services: mass customization and beyond.* Technovation 2003. **23**(10): p. 809-821.
44. A. Wasser, M. Lincoln, and R. Karni, *ERP Reference Process Models: From Generic to Specific*, in *Business Process Management Workshops*. 2006, Springer. p. 45-54.
45. Royal Children's Hospital Melbourne Australia. *MRI GA Clinical Path: MR 960*. September 2005 [cited 14 June 2007]; Available from: http://www.rch.org.au/emplibrary/rch_clinpath/MRIpath.pdf
46. B. Cahill, D. Carrington, B. Song, and P. Strooper, *An Industry-Based Evaluation of Process Modeling Techniques*, in *Software Process Improvement*. 2006. p. 111-122.
47. B.-J. Hommes and V. van Reijswoud. *Assessing the Quality of Business Process Modelling Techniques*. in *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 1*. 2000: IEEE Computer Society.
48. R.F. Paige, J.S. Ostroff, and P.J. Brooke, *Principles for modeling language design.* Information & Software Technology, 2000. **42**(10): p. 665-675.
49. E. Söderström, B. Andersson, P. Johannesson, E. Perjons, and B. Wangler, *Towards a Framework for Comparing Process Modelling Languages*, in *Advanced Information Systems Engineering: 14th International Conference, CAiSE 2002 Toronto, Canada, May 27-31, 2002. Proceedings*. 2002. p. 600.
50. Workflow Patterns Initiative. *Workflow Patterns*. 2007 [cited 17 July 2007]; Available from: http://www.workflowpatterns.com/index.php.
51. S.A. White, *Process Modeling Notations and Workflow Patterns.* BPTrends, 2004(March).
52. N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and P. Wohed. *On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling*. in *APCCM*. 2006: Australian Computer Society.
53. M. Dumas and A.H.M. ter Hofstede, *UML Activity Diagrams as a Workflow Specification Language.* Lecture Notes in Computer Science, 2001. **2185**: p. 76-90.
54. B. Weber, S. Rinderle, and M. Reichert, *Change Patterns and Change Support Features in Process-Aware Information Systems.* Lecture Notes in Computer Science, 2007. **4495**: p. 574-588.
55. W.M.P. van der Aalst, *The Application of Petri Nets to Workflow Management.* Journal of Circuits, Systems, and Computers, 1998. **8**(1): p. 21-66.
56. K. Salimifard and M. Wright, *Petri net-based modeling of workflow systems: An overview.* European Journal of Operational Research, 2001. **134**(3): p. 664-676.

57. J. Mendling, J. Recker, M. Rosemann, and W.M.P. van der Aalst. *Generating Correct EPCs from Configured C-EPCs*. in *SAC* 2006: ACM.
58. J. Recker, M. Rosemann, W.M.P.v.d. Aalst, and J. Mendling. *On the Syntax of Reference Model Configuration - Transforming the C-EPC into Lawful EPC models*. in *Business Process Management Workshops*. 2005.
59. J. Recker, J. Mendling, W.M.P. van der Aalst, and M. Rosemann. *Model-Driven Enterprise Systems Configuration*. in *CAiSE* 2006: Springer.
60. Workflow Patterns Initiative. *Workflow Patterns*. 2007 [cited 20-08-2007]; Available from: http://www.workflowpatterns.com/patterns/control/index.php.
61. J. Dehnert and P. Rittgen. *Relaxed Soundness of Business Processes*. in *CAiSE* 2001: Springer.
62. K. van Hee, O. Oanea, and N. Sidorova, *Colored Petri Nets to Verify Extended Event-Driven Process Chains.* Lecture Notes in Computer Science, 2005. **3760**: p. 183-201.
63. P. Langner, C. Schneider, and J. Wehler. *Petri Net Based Certification of Event-Driven Process Chains*. in *ICATPN*. 1998: Springer.
64. B.F. van Dongen, W.M.P. van der Aalst, and H.M.W. Verbeerk. *Verification of EPCs: Using Reduction Rules and Petri Nets*. in *CAiSE* 2005: Springer.
65. N. Cuntz and E. Kindler. *On the Semantics of EPCs: Efficient Calculation and Simulation* in *Business Process Management*. 2005: Springer.
66. E. Kindler. *On the Semantics of EPCs: A Framework for Resolving the Vicious Circle*. in *Business Process Management: Second International Conference, BPM 2004, Potsdam, Germany, June 17-18, 2004. Proceedings*. 2004: Springer.
67. R.M. Dijkman, M. Dumas, and C. Ouyang, *Formal Semantics and Analysis of BPMN Process Models using Petri Nets*, in *QUT ePrints*. 2007, Queensland University of Technology.
68. I. Raedts, M. Petkovic, Y.S. Usenko, J.M. van der Werf, J.F. Groote, and L. Somers. *Transformation of BPMN models for Behaviour Analysis*. in *5th International Workshop on Modeling, Simulation, Verification and Validation of Enterprise Information Systems*. 2007. Funchal, Madeira - Portugal.
69. J. Mendling, J. Recker, M. Rosemann, and W.M.P. van der Aalst. *Towards the Interchange of Configurable EPCs: An XML-based Approach for Reference Model Configuration*. in *Enterprise Modelling and Information Systems Architectures*. 2005. Klagenfurt, Austria: GI.
70. G. Succi, W. Pedrycz, J. Yip, and I. Kaytazov. *Intelligent Design of Product Lines in Holmes*. in *Canadian Conference on Electrical and Computer Engineering* 2001.
71. Software Engineering Institute Carnegie Mellon. *Software Product Lines*. 2007 [cited 26-06-07]; Available from: http://www.sei.cmu.edu/productlines/.
72. P. Noordhuizen, *Analyzing Aspects in Production: Plans for Software Product Lines*, in *Electrical Engineering, Mathematics and Computer Science*. 2006, University of Twente: Enschede. p. 176.
73. A. Schnieders and M. Weske, *Activity Diagram Based Process Family Architecture for Enterprise Application Families*, in *Enterprise Interoperability: New Challenges and Approaches*. 2007, Springer-Verlag: London. p. 67-76.
74. M. Sinnema and D. Sybren, *Classifying variability modeling techniques.* Information & Software Technology, 2007. **49**(7): p. 717-739.
75. M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. *COVAMOF: A Framework for Modeling Variability in Software Product Families*. in *SPLC*. 2004: Springer.

76. J.-C. Trigaux and P. Heymans, *Modeling variability requirements in Software Product Lines: a comparative survey*, in *Technical report*. 2003, Computer Science Institute, University of Namur.
77. K. Czarnecki, *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*. 1998, Technische Universtität Ilmenau: Ilmenau.
78. J. Estublier. *Software Configuration Management: A Roadmap*. in *ICSE - Future of Software Engineering* 2000. Limerick Ireland.
79. U. Asklund, L. Bendix, and T. Ekman. *Software Configuration Management Practices for eXtreme Programming Teams*. in *11th Nordic Workshop on Programming and Software Development Tools and Techniques (NWPER)* 2004. Turku, Finland.
80. R. Conradi and B. Westfechtel, *Version models for Software Configuration Management.* ACM Computing Surveys, 1998. **30**(2): p. 232-282.
81. R. Conradi and B. Westfechtel, *Towards a Uniform Version Model for Software Configuration Management.* ACM Comput. Surv., 1997. **30**(2): p. 232-282.
82. R. Conradi and B. Westfechtel. *Configuring Versioned Software Products*. in *SCM*. 1996: Springer.
83. A. Schnieders and F. Puhlmann. *Variability Mechanisms in E-Business Process Families*. in *BIS*. 2006: GI.
84. K. Czarnecki and M. Antkiewicz. *Mapping Features to Models: A Template Approach Based on Superimposed Variants*. in *GPCE*. 2005: Springer.
85. F. Puhlmann, A. Schnieders, J. Weiland, and M. Weske, *Variability Mechanisms for Proces Models*, in *PESOA-Report No. 17/2005*. 2005, DaimlerChrysler Research and Technology, Hasso-Plattner-Institut.
86. M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. *Extending Feature Diagrams with UML Multiplicities*. in *6th Conference on Integrated Design & Process Technology* 2002. Pasadena, California, USA.
87. D.F. Brown and D.A. Watt, *Programming Language Processors in Java - Compilers and Interpreters*. 2000: Prentice Hall.
88. E.N. Hanson and J. Widom, *An Overview of Production Rules in Database Systems.* The Knowledge Engineering Review, 1993. **8**(2): p. 121-143.
89. B.P. Munch, *Versioning in a Software Engineering Database -The Change Oriented Way*. 1993: Trondheim, Norway.
90. B.P. Munch, J.-O. larsen, B. Gulla, R. Conradi, and E.-A. Karlsson. *Uniform Versioning: The Change-Oriented Model*. in *Proceedings of the 4th International Workshop on Software Configuration Management (Preprint)*. 1993. Baltimore, Maryland.
91. J.M. Küster, J. Koehler, and K. Ryndina. *Improving Business Process Models with Reference Models in Business-Driven Development*. in *Business Process Management Workshops*. 2006: Springer.
92. I. Sommerville and G. Dean, *PCL: A configuration language for modelling evolving system architectures*, in *SE/8/1994*. 1994, Software Engineering Research Group, Computing Department, Lancaster University.
93. I. Sommerville and G. Dean, *PCL: a language for modelling evolving system architectures.* Software Engineering Journal, 1996. **11**(2): p. 111-121.
94. E. Tryggeseth, B. Gulla, and R. Conradi, *Modelling systems with variability using the PROTEUS configuration language*, in *Selected papers from the ICSE SCM-4 and SCM-5 Workshops, on Software Configuration Management*. 1995, Springer-Verlag
95. B. Gulla and J. Gorman, *Experiences with the use of a configuration language*, in *Software Configuration Management*. 1996. p. 198-219.

96. The Stationary Office. *Glossary of terms*. 2000 [cited 31-05-2007]; Available from: http://www.tso.co.uk/demo/itil2/cd/content/ss/ss_apdx_a_02.htm
97. D. Müller, M. Reichert, and J. Herbst. *Flexibility of Data-Driven Process Structures*. in *Business Process Management Workshops*. 2006: Springer.

Chapter 10 Appendices

10.1 Appendix 1: Glossary of terms

Business process

A collection of activities that takes one or more kinds of input and creates an output that is of value to the customer (Hammer and Champy, 1993) [1].

Business process management

Business process management (BPM) is the design and control of business processes (Leymann and Altenhuber, 1994) [1].

Business process modeling

Business process modeling is the activity of representing or mapping business processes using a business process modeling language with the goal to describe, analyze or reengineer business processes.

Business process modeling language

A business process modeling language is a notation that can be used to map or represent business processes.

Business process redesign

Fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical measures of performance, such as cost, quality, service, and speed (Hammer and champy, 1993) [1].

Business process variability

Business process variability occurs within the domain space and over time. It leads to variable business processes.

Change management

Process of controlling Changes to the infrastructure or any aspect of services, in a controlled manner, enabling approved Changes with minimum disruption [96].

Feature diagram

A feature diagram is a featural description of the individual instances of a concept. A feature diagram constitutes a tree composed of nodes and directed edges. The tree's root represents the concept which is refined using mandatory, optional, alternative (X-OR-features) and OR-features (Trigaux, Heymans) [76].

Process

Read definition of "*Business process*".

Process variant

A process variant is a business process created to comply with the configuration of its domain.

Process revision

A process revision is a business process created by an evolutionary change of another business process.

Reference process model

Every reference model is a model which can be consulted for the development of other models (Hars) [37].

Software configuration management

Software configuration management (SCM) is the portion of software project management concerned with identifying, organizing and controlling changes to the components of a software project [97].

Software product lines

A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [71].

Software product line engineering

Read definition of "*Software product lines*".

10.2 Appendix 2: Context free grammar of Feature-EPC configuration rules

Using a context free grammar in Backus-Naur form permits the explicit specification of the syntax, contextual constraints and semantics of the configuration rules [87]. The objective here is to specify modular, reusable and easy to implement configuration rules.

Strictly configurable nodes are the connectors XOR and OR, and configurable functions. For every configurable node, configuration rules shall be specified, this should permit the automatic generation or the programming of configuration rules. The specification of complete configuration rules shall then be done by assigning features to these configuration rules.

To make the configuration rules modular, they have been defined to specify or reflect the configuration of CEPC process models: every CEPC process model shall come with its respective set of configuration rules. To have an exact and precise specification of the rules these have been specified using Backus-Naur Form (BNF).

Syntax

Terminal Symbols
: , ; → { } ( ) xor or XOR OR SEQ AND

Non terminal Symbols

FEPCConfiguration	ConfigurationRule
CEPCConfigRule	ConfFunctionRule
XORRule	ORRule
Feature	Features
FeatureSet	EPCSequence
Event	Function
Connector	Config
Status	Name
ID	Letter
Digit	

Start Symbol
The start symbol is FEPCConfiguration.

Production rules
FEPCConfiguration ::= ConfigurationRule FEPCConfiguration | ConfigurationRule
ConfigurationRule ::= (ID, NumericPriority): FeatureSet → CEPCConfigRule
CEPCConfigRule ::= ConfFunctionRule | XORRule | ORRule

Feature ::= (Name, ID)
Features ::= Feature, FeatureList | Feature
FeatureSet ::= {Features}

ConfFunctionRule ::= (Name, ID, Status)

XORRule ::= (xor, ID, XOR, {})
| (xor, ID, SEQ, EPCSequence)
| (xor, ID, SEQ, XORRule)
| (xor, ID, SEQ, ORRule)

ORRule ::= (or, ID, OR, {})
| (or, ID, AND, {})
| (or, ID, XOR, {})
| (or, ID, SEQ, EPCSequence)
| (or, ID, SEQ, XORRule)
| (or, ID, SEQ, ORRule)

EPCSequence::= (Event, Function)
| (Function, Event)
| (Connector, Event)
| (Connector, Function)
| (Event, Connector)
| (Function, Connector)
| (Connector, Connector)

Event ::= (Name, ID)
Function ::= (Name, ID)
Connector ::= (xor, ID) | (or, ID) | (and, ID)

Status ::= ON | OFF

NumericPriority ::= Digit NumericPriority | Digit
Name ::= Letter | Letter Name | Name Digit
ID ::= Letter | Letter ID | ID Digit

Letter ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Contextual constraints
There are two types of scope:

- The scope of a single configuration rule: local scope
- The scope of all configuration rules: global scope.

Every ID must be unique and has a global scope.

Every function, event and connector must be an existing node of the configurable EPC process model.

Every feature must be an existing feature of the Riebisch feature diagram.

A NumericPriority is an integer number between 1 and 1000.

Semantics

There are three types of configuration rules:

- Configuration rules that configure functions.
- Configuration rules that configure XOR connectors.
- Configuration rules that configure OR connectors.

(CR2, 2): {(Blind, Fe2)} → (Schedule Ambulance, Fu2, OFF)
The configuration rule here above configures a function and has the following meaning (Figure 100):

- It has the ID 'CR2'.
- It has the numeric priority '2'.
- The feature set is composed of a single feature with name 'Blind' and ID 'Fe2'
- It configures the function with the name 'Schedule ambulance; and with ID 'Fu2' to 'OFF'.

A configurable function can also be configured to 'ON'.

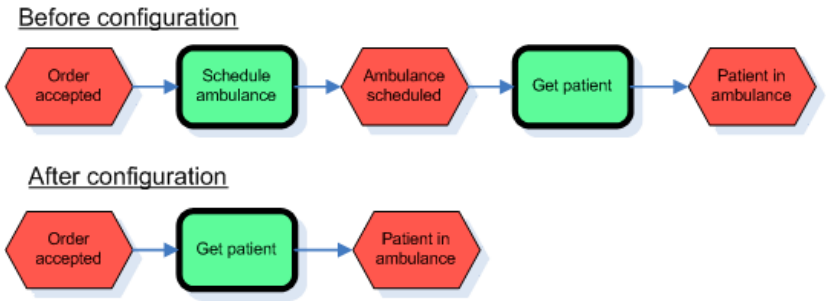


Figure 100: illustration of a configurable function

(CR6, 2): {(Blind, Fe2)} → (xor, xor2, SEQ, ((Patient escorted to examination room, E2), (Medical Examination, Fu6)))
The configuration rule here above configures an XOR connector and has the following meaning (Figure 101):

- It has the ID 'CR6'.
- It has the numeric priority '2'.
- The feature set is composed of a single feature with name 'Blind' and ID 'Fe2'.
- It configures connector 'xor' with ID 'xor2' as a sequence. This causes the deletion of the xor connector and its replacement with the sequence event 'Patient escorted to examination room' and function 'Medical examination'.

An XOR connector can also be configured as an XOR connector.

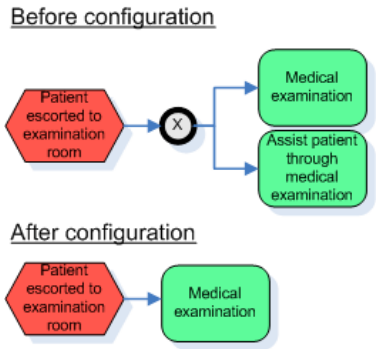


Figure 101: illustration of a configurable XOR connector

(CR7, 2): {(Blind, Fe2)} → (or, or1, AND, {})
The configuration rule here above configures an OR connector and has the following meaning (Figure 102):

- It has the ID 'CR2'.
- It has the numeric priority '2'.
- The feature set is composed of a single feature with name 'Blind' and ID 'Fe2'
- It configures connector 'or' with ID 'or1' as an 'AND' connector.

An OR connector can also be configured as an XOR, OR and sequence (SEQ).

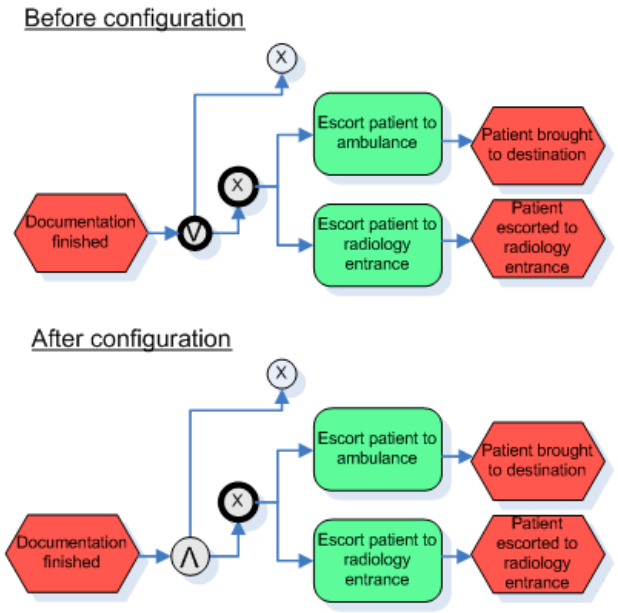


Figure 102: illustration of a configurable OR connector

10.3 ***Appendix 3: Context free grammar of COV-EPC ambition rules***

To specify the configuration rules precisely, a context free grammar in Backus-Naur form shall be used [87]. The syntax, the contextual constraints, the semantics and a motivation of current ambition rules have been specified here.

Syntax

Terminal Symbols
( ) , : { } → xor and or REP DEL ADD

Non terminal Symbols

Ambition	AmbitionRule
AddRule	DelRule
RepRule	Choice
Options	Option
EPCFragment	EPCSequence
Node	Event
Function	Connector
NumericPiority	Name
ID	Letter
Digit	Rule

Start Symbol
Ambition is the start symbol.

Production rules
Ambition ::= AmbitionRule Ambition | AmbitionRule

AmbitionRule ::= (ID, NumericPriority): Choice → Rule

Rule ::= AddRule | DelRule | RepRule
AddRule ::= (ADD, {EPCFragment})
DelRule ::= (DEL, Node, Node)
RepRule ::= (REP, Node, Node)

Choice ::= {Options}
Options ::= OptionID, Options | OptionID
Option ::= (Name, OptionID)
OptionID ::= ID

EPCFragment ::= EPCSequence, EPCFragment | EPCSequence
EPCSequence::= (Event, Function)
| (Function, Event)
| (Connector, Event)
| (Connector, Function)
| (Event, Connector)
| (Function, Connector)
| (Connector, Connector)

Node ::= Event | Function | Connector

Event ::= (Name, ID)
Function ::= (Name, ID)
Connector ::= (xor, ID) | (or, ID) | (and, ID)

NumericPriority ::= Digit NumericPriority | Digit
Name ::= Letter | Letter Name | Name Digit
ID ::= Letter | Letter ID | ID Digit

Letter ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Contextual constraints
There are two types of scope:
- the scope of a single AmbitionRule: local scope.
- the scope of an Ambition: global scope.

IDs have global scope and are unique.

Further important contextual constraints need to be defined on the AddRule:
- The EPCFragment must consist of at least two EPCSequence.
- The first node of the first EPCSequence needs to be an existing node of the EPC base process model.
- The last node of the last EPCSequence needs to be an existing node of the EPC base process model.

The following contextual constraints need to be defined on the DelRule:
- The two nodes 'Node' need to be existing nodes of the EPC base process model.

The following contextual constraints need to be defined on the RepRule:
- The first node 'Node' needs to be an existing node of the EPC base process model.

A choice must consist of a set of existing options (optionIDs).

A NumericPriority is an integer number between 1 and 1000.

Semantics

Most importantly the following rule has the following meaning:
AmbitionRule ::= (ID, NumericPriority): Choice → Rule
For the following 'Choice' or set of selected options apply the corresponding 'Rule'.

There are three types of rules:
- AddRule ::= (ADD, {EPCFragment})
- DelRule ::= (DEL, Node, Node)
- RepRule ::= (REP, Node, Node)

AddRule has the following meaning: add the new nodes between the first node of the first EPCSequence and the last node of the last EPCSequence to the EPC base process model (see syntax).

DelRule has the following meaning: delete all the nodes between the two nodes ‘Node’ from the EPC base process model.

RepRule has the following meaning: replace the first node ‘Node’ with the second node ‘Node in the EPC base process model.

Motivation
The syntax, contextual constraints and semantics of the configuration rules were specified to minimize the amount of configuration rules that need to be specified.

It is indeed thus better to have the following AddRule::= (ADD, {EPCFragment}) where several nodes can be added at the same time than an AddRule ::= (ADD, Node, Node) where only a single node can be added at the same time. The second AddRule would require the specification of more configuration rules when trying to add several elements to the base process model. The same reasoning holds for the DelRule.

However the same reasoning does not hold for the RepRule. The new RepRule would then look like the following rule:

RepRule ::= (REP, Node, Node) | (REP, {EPCFragment}, {EPCFragment})

The usefulness of the AddRule is then questionable as the subrule (REP, {EPCFragment}, {EPCFragment}) can be used to add new rules. Furthermore unnecessary complexity is introduced by two different sub-RepRules.