Goal-oriented Pattern Family Framework for Business Process Modeling

Saeed Ahmadi Behnam

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Ph.D. in Computer Science

Under the auspices of the Ottawa-Carleton Institute for Computer Science

uOttawa

University of Ottawa
Ottawa, Ontario, Canada
October 2012

This work is dedicated to
my mother and to the memory of my father;
both instilled in me a passion for science and the
continuous curiosity to understand, and they will
be eternal sources of inspiration for me.

ABSTRACT

While several approaches exist for modeling goals and business processes in organizations, the relationships between these two views are often not well defined. This inhibits the effective reuse of available knowledge in models. This thesis aims to address this issue through the introduction of a Goal-oriented Pattern Family (GoPF) framework that helps constructing business process models from organization goals while expanding these goals, establishing traceability relationships between the goal and process views, and improving reusability. Methods for extracting domain knowledge as patterns, which are composed of goal model building blocks, process model building blocks, and their relationships, and for maintaining the patterns over time are also presented. The GoPF framework provides the infrastructure for defining pattern families, i.e., collections of related patterns for particular domains. The foundation of GoPF is formalized as a profile of the User Requirements Notation, a standard modeling language that supports goals, scenarios, and links between them. A method for the use of GoPF is defined and then illustrated through a case study that targets the improvement of patient safety in healthcare organizations. The framework and the extraction/maintenance methods are also validated against another case study involving aviation security in a regulatory environment.

The GoPF framework is expected to have a positive impact on the scientific community through the formalization, evolution, and reuse of patterns in domain-specific business domains. From an industrial viewpoint, this framework will also help intermediary organizations (such as consulting firms) who are required to repeatedly create and document goal and process models for other organizations in their business domain.

ACKNOWLEDGEMENT

First and foremost, I wish to thank my supervisor, Dr. Daniel Amyot, for his support, patience, and insightful comments all these years, making this research experience enjoyable, motivating and productive. I will always be inspired by his dedication to scientific research and to the well-being of his students. I also extend thanks to my committee, Dr. Liam Peyton, Dr. Michael Weiss, Dr. Morad Benyoucef, and Dr. Luiz Marcio Cysneiros, for accepting to review and comment on this thesis.

My case study related to aviation security went smoothly thanks to insightful and valuable inputs from Nick Cartwright, Edna Braun, and Mario Saucier. I am also grateful to Dr. Alan Forster and his team for their help and suggestions in the patient safety case study.

To my research colleagues, thank you all for your support and feedback. I am particularly indebted towards Dr. Gunter Mussbacher for his thoughtful ideas and suggestions during the course of my research on pattern families. I would like to thank the system support and administration staff of EECS, and particularly Jacques Sincennes, at the University of Ottawa.

Then, I would like to express my thanks to my mom, Mahin, and to my siblings, Fariba, Hamid, and Farzaneh for their encouragement and love which gave me enough strength to focus on my research during these years. I am also grateful for the encouragement and support by my wonderful friends, Julie, Daniel, Danielle, Helen, Maryam, Ali, Sarah, and Houman. The time I spent with my friends and thought provoking dialogs and discussions with Daniel, Reza, Farzaneh, Julie, and Houman have always been a source of joy and comfort and helped me through these years of research.

Finally, this research was made possible through the financial support of the Ontario Research Network for Electronic Commerce, and of the Natural Sciences and Engineering Research Council of Canada (Discovery and Collaborative Health Research Projects grants).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronym	Definition
BP	Business Process
BPBB	Business Process Building Block
BPDM	Business Process Definition Metamodel
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Management
BPMN	Business Process Modeling Notation
EHR	Electronic Health Record
FDM	Family Development Method
FMM	Family Metamodel
GBPM	Goal-driven Business Process Modeling
GDM	Goal-driven Method
GoPF	Goal-oriented Pattern Family
GMBB	Goal Model Building Block
GRL	Goal-oriented Requirement Language
IT	Information Technology
ITU	International Telecommunication Union
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
NFR	Non-Functional Requirement
OCEM	Organization-driven Customization and Extraction Method
OCL	Object Constraint Language
ODP	Organizational Development Process
OMG	Object Management Group
PDCA	Plan Do Check Act
PF	Pattern Family
RE	Requirements Engineering
SOA	Service-Oriented Architecture
TQM	Total Quality Management
UCM	Use Case Map
UML	Unified Modeling Language

URN	User Requirements Notation

Chapter 1. INTRODUCTION

This thesis provides a framework for reusing knowledge captured in the form of patterns at the level of goal models and business process models. This chapter summarizes the problem and motivation for this research, concisely defines the research hypothesis, highlights the research methodology, provides a summary of the solution, lists the main contributions of this research, and outlines the content of this thesis.

1.1 Problem Statement

In today's competitive and global economy, companies and other types of organizations are faced with many challenges such as (i) the need for fast information transfer (ii) the need for quick decision making (iii) the need to adapt to changes (iv) increased competition, and (v) the need for higher quality services and products [1]. In the past two decades, there have been efforts to harness software applications in order to address such challenges [2][3][4]. Organizations attempt to take advantage of software solutions to solve their problems and achieve their organizational objectives.

In a software development process, goals drive the definition of requirements. The value of software application solutions to an organization is based on how well business goals are satisfied through their use. When developing valuable software applications, organizations often have two major issues. First, they often have difficulties in properly identifying and documenting their goals, their business processes, and the links between these two views [5][6][7]. Second, there are additional challenges in transforming business processes to executable software applications that realize them. Figure 1 illustrates the gaps to be filled when going from business goals to business processes and then to software applications. This figure also shows the conventional roles typically associated with the artifacts discussed so far: business analyst for business goals, business process analyst for business processes, and developer for software applications.

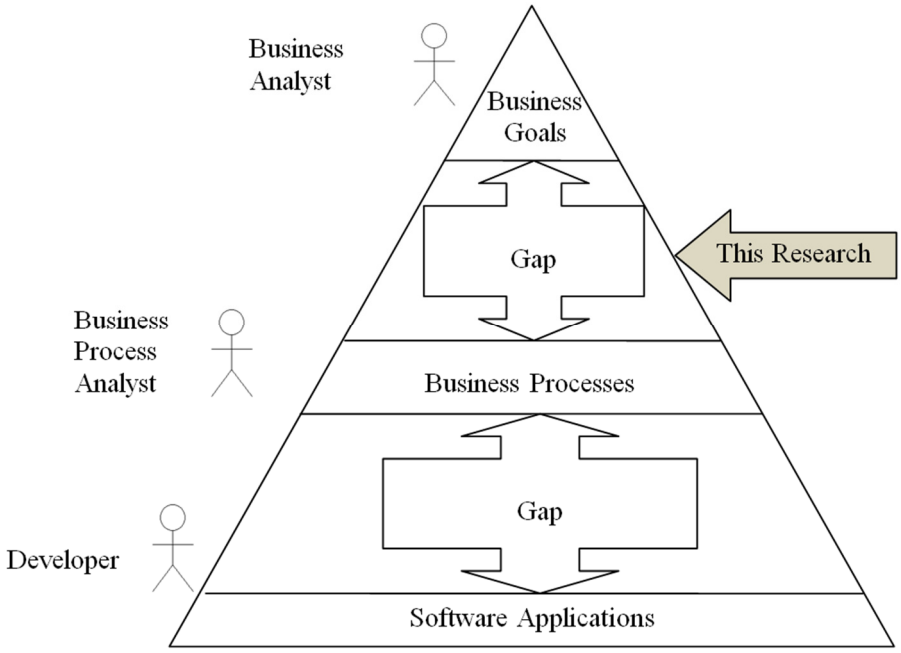

Figure 1 Gaps between different levels of the software development process

The gap between business processes and software applications has received much attention over the past decade, and dedicated technologies such as the Web Service Business Process Execution Language (WSBPEL) [8] have emerged. In addition, the Object Management Group's Model-Driven Architecture (MDA) [9], a well-known incarnation of the Model-Driven Engineering (MDE) transformation approach [10], can help formalize and facilitate the generation of software from business process models. However, the gap between business goals and business processes has received far less attention. Yet, not addressing this gap leads to problems in effectively and appropriately identifying stakeholders' goals and devising business processes that satisfy them. Symptoms of these problems typically identified by senior executives [11], such as:

- Software investments are unrelated to business strategies;
- Payoff from software related investments is inadequate;
- There is too much "technology for technology's sake";
- Relations between users and software specialists are poor; and
- System designers do not consider users' preferences and work habits.

Consequently, many software development projects yield disappointing results or are simply canceled because software applications and business processes are not aligned properly with business goals [12].

Currently, there are few approaches that capture and analyze goal models within the software development process [13]. When goals are indeed captured, even small problems can lead to complex and large goal models [14]. Furthermore, modeling business goals and processes separately is not sufficient to bridge the gap between these two views; alignment of goals and processes as well as traceability must also be taken into account. Defining such models remains challenging, especially when done from scratch. Reusing domain knowledge captured in the form of *patterns* can often help address this issue. For instance, design patterns have been quite successful in the construction of software applications [15]. However, patterns that span business goals and processes are far less common, and reusing existing knowledge with goals and business process remains an open problem [16].

1.2 Motivation

A paradigm shift towards model-driven development is happening in the field of software engineering in order to deal with the challenges mentioned in the previous section. This is changing the way software applications are developed [17]. As Greenfield and Short observe in [18], new artifacts beyond those offered by object-orientation are required:

"The software industry remains reliant on the craftsmanship of skilled individuals engaged in labour intensive manual tasks. However, growing pressure to reduce cost and time to market and to improve software quality may catalyse a transition to more automated methods. We look at how the software industry may be industrialized, and we describe technologies that might be used to support this vision. We suggest that the current software development paradigm, based on object orientation, may have reached the point of exhaustion…"

Suggested approaches to address the mentioned challenges are mostly process-oriented but do not fully tackle the importance of goal models [19]. This is particularly significant because successful software applications must address the problems of stakeholders [20]. Stakeholders' concerns must be captured, analyzed, and reasoned about, and

creating goal models is an appropriate means to this end. Goal models can drive the creation of business processes that address the concerns of stakeholders.

Still, creating high-quality goal models and business process models that represent requirements is challenging and requires much effort. Doing this is particularly difficult in ground-up approaches where models are created from scratch. Ground-up approaches for software development seldom take advantage of reusing the captured domain knowledge, especially at the goal and process levels. Creating solutions from scratch for recurring problems makes them more vulnerable to: unsuccessful solutions that may run against working routine [21], difficulties in maintenance [6], and failure of the new solution because of conflicts that are hard to locate in early stages of requirement engineering [22][23]. Furthermore, in such approaches, quality and resource consumption (related to time and cost) have a reverse correlation. This is also known as the time, cost, and quality triangle [24]. In other words, developing high-quality applications is a resource-consuming process and lowering the costs often decreases the quality of the final product. For instance, in the field of healthcare software applications, there are many failure stories that result from the above difficulties [25][26][27][28].

These important concerns can be addressed by reusing domain knowledge in the form of goal models and business process models. It is becoming increasingly difficult to ignore the benefits of knowledge reusability in this context, as often emphasized in the literature [6][7][29][30][31]. However, two difficulties challenge the reuse of knowledge for organization in a specific domain. First, it is usually not possible for an organization to reuse entirely the goal models and business process models that represent the requirements and know-how of another organization. The reason is that despite the similarity of objectives in a domain, there are differences in the organizational objectives and their priorities, which lead to different hierarchies of requirements. Therefore, it is more plausible for organizations to reuse pieces of the business goal and process models. However, it is difficult to reuse only pieces of these models because they are usually captured as complex hierarchical models. Second, reusing business goal and process models is successful when an organization can evaluate the potential effects of alternative solutions considering its particular context. However, when capturing the knowledge about the requirements is done holistically, it is difficult to capture the alternative effects of solu-

tion along with models. This makes it difficult for organizations to find and reuse those pieces of model that are best suitable to the context of a particular organization and have the most positive effect on how to solve the problems.

Considering the continuity of the spectrum of artifacts from requirements to software applications, it is important to recognize that business process models are conceptually linked to goal models. These links indicate which business processes realize particular business goals. Capturing the links between business process models and goal models facilitates the finding of known solutions in the context of conditions and requirements of an organization. Hence, such links help bridge the gap between requirements of a particular organization and corresponding (existing) solutions.

Patterns are reusability mechanisms for capturing and reusing domain knowledge. They can also capture goal models and business process models along with links that define the realization relationships between them. The knowledge about domains captured in such a way can be systematically reused by a framework for creating solutions for a particular organization. A means for capturing the knowledge enables solving complex problems by reusing patterns that capture building blocks of problems and solutions. These can be used for identifying the requirement and providing mechanisms that address these requirements. However, the domain knowledge about business goals and processes is volatile and changes over time at a more rapid pace than for design patterns. Therefore, pattern families in a business modeling context can be useful only if they can adapt to the changes that happen in the domain and reflect the respective solutions for emerging problems. Introducing evolution mechanisms that systematically help evolving pattern families is hence a necessity. Inspired from the concepts of evolution[1] and adaptive software maintenance, these mechanisms should enable a gradual and iterative development where a pattern family changes into a different and better form that more accurately represent the knowledge about the current problems and solutions within a domain.

---

[1] *Evolution* in this thesis refers mainly to adaptive maintenance activities in conventional software evolution, which are (manual) modifications of a software product (or pattern here) performed after delivery to keep it usable in a changed or changing environment. Evolution here is not related to genetic algorithms or other automatic evolutionary algorithms from the artificial intelligence community.

1.3 Research Hypothesis

Our main research hypothesis is defined as follows:

> *We can reuse and maintain, in a rigorous way, the knowledge about business goals, business processes and the links between them, captured as patterns to create suitable business processes in the context of a different organization.*

The main objective of this research is to develop a goal-oriented pattern-based framework that facilitates knowledge reusability based on business goals and processes for a given domain. Furthermore, this framework serves as key enabler for creating organization-specific goal models and business processes that realize them. Finally, by providing mechanisms for pattern evolution, the knowledge captured in the framework will adapt to changes in the domain.

1.4 Solution: the GoPF Framework

The solution developed in this thesis, named *Goal-oriented Pattern Family (GoPF) framework*, is a framework that aims to facilitate the discovery and documentation of recurring solutions to recurring problems in the form of patterns. It also provides mechanisms to enable reusing and maintaining the knowledge at the level goal model and business process models. The term *family* is used here to reflect the parent-child refinement relationships that exist among the patterns that the family contains. GoPF is composed of a Family Metamodel (FMM) and a Goal-driven Method (GDM), as shown in Figure 2. FMM is a metamodel that lays down a structure for Pattern Families (PF). A PF captures the knowledge about a particular domain with patterns formalized with goals, business processes, and links between them. It specifies typical refinements of goals in terms of processes for a particular domain (e.g., healthcare). A PF is the key enabler for reusing knowledge. The method in the framework, GDM, is composed of two major components: (i) a Family Development Method (FDM), and (ii) the Organization-driven Customization and Extraction Method (OCEM).

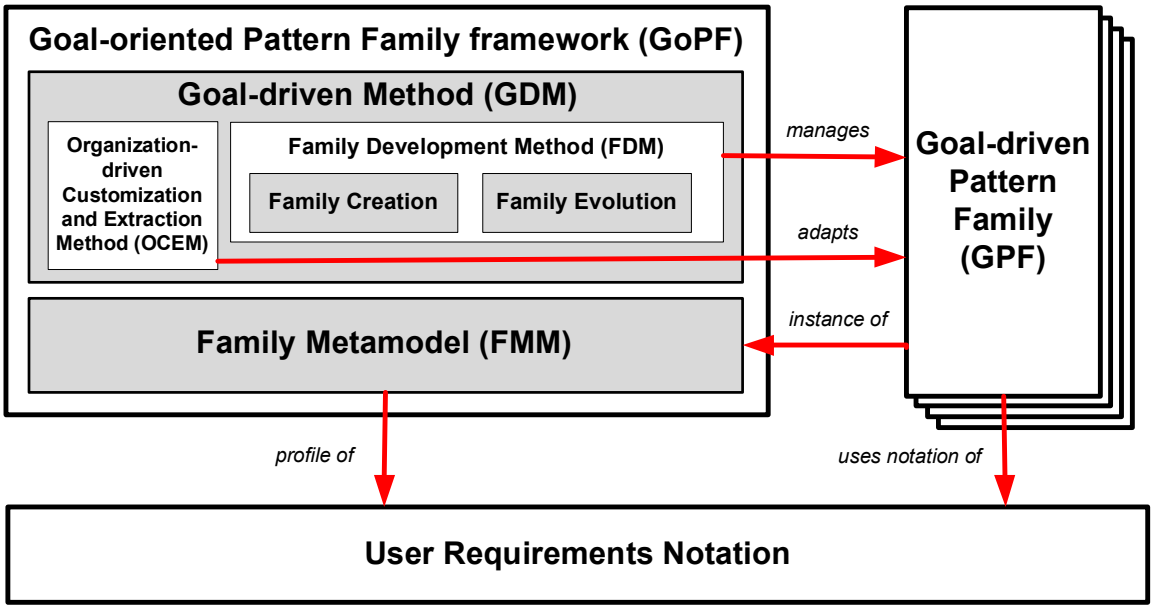

Figure 2 Architecture of GoPF framework

FDM provides algorithms for creating a PF and evolving it over time:

- Family Creation is a method that provides the steps for capturing existing solutions to recurring problems in the form of patterns. The output of this method is a PF; and
- Family Evolution is composed of algorithms for evolving a PF to adapt to the changes in the domain in order to improve its accuracy and overall quality. Family Evolution algorithms accommodate eliminating obsolete patterns, extending a PF by adding new patterns, modifying current patterns, and combining a PF with another PF.

Another component of GDM is the Organization-driven Customization and Extraction Method (OCEM), which includes algorithms that guide the selection of instances of solutions for particular organizations within the domain. OCEM uses a PF as an input and assesses the impact of alternative solutions for achieving the high-level goals of a given organization in a step-by-step, top-down approach. Another input of OCEM is a partial business goal model where only some of the high-level goals of an organization need to be identified. The main output of the OCEM is a more complete goal model combined with business processes that are aligned with the identified goals, as well as additional traceability links between the two views.

The main beneficiaries of this framework are requirements engineers and business analysts whose objective is to model the business goals and business processes for a particular organization in a specific domain. The GoPF framework enables them to reuse available knowledge of the domain at the level of goals and related business processes. However, many other stakeholders of such organizations also indirectly benefit from this framework by being exposed to potential problems and solutions in similar organizations. The GoPF framework complies with the spirit of MDE while including goal models in the chain of transformations of artifacts. The framework builds on the User Requirements Notation (URN) [32][33], an international standard that already combines goal and scenario modeling into a single language.

1.5 Research Methodology

In order to contribute to the research community, as characterized by March and Smith [34], this research attempts to provide a *construct*, i.e., the Goal-oriented Pattern Family framework, to address a *problem*, namely how to reuse knowledge about requirements and solutions in specific domains. The *design-oriented methodology*, an approach toward research in information systems suggested by Hevner *et al.* [35], is the underlying research methodology for this thesis. Following this approach, first the gap between business goals and business processes as well as challenges for reusing the knowledge in domains have been studied and characterized through a literature review and further elaborated and illustrated through representative scenarios. Furthermore, a framework is proposed to address the aforementioned challenges. This framework is composed of a metamodel that formalizes patterns and of creation, usage, and evolution methods, which together serve as the constructs contributing to research as highlighted by Hevner *et al.* and March and Smith. Finally, this framework is evaluated against representative case studies. A Patient Safety case study is used as an ongoing example throughout the thesis, and then a second case study related to Aviation Security is discussed. The evaluation is intended to highlight the usefulness and applicability of the framework in regard to addressing the motivating problems, but it is not intended to be an empirical evaluation [34][35].

1.6 Thesis Contributions

The contributions of this research are provided here in order of importance.

- A framework, called GoPF, for capturing the goal and process knowledge in the enterprise domain in the form of patterns:
  - This framework includes a metamodel (FMM) that provides the structure for Pattern Families (PF);
  - This framework is formalized as a URN profile that enables using the URN standard and tools for capturing the patterns.
- A Goal-driven Method (GDM) that guides and partially automates the creation and evolution of a PF for a particular domain:
  - This GDM includes a Family Creation algorithm;
  - This GDM also includes several Family Evolution algorithms; and
  - A library of Java classes that implement these algorithms in order to assess the feasibility of evolving pattern families through evolution mechanisms.
- An Organization-driven Customization and Extension Method (OCEM):
  - OCEM includes algorithms for extracting and adapting models for a specific organization from patterns in a PF.

In order to evaluate the hypothesis of this research, two case studies have been carried out. Considering that the generated PFs in these case studies capture reusable real-world knowledge in their respective domains, they can be considered as minor contributions of this thesis.

- Validation case studies:
  - A pattern family for patient safety in healthcare is created and evolved. This family is then used for extracting and adapting models for a department in a healthcare organization;
  - A pattern family for aviation security screening is created, with an emphasis on the use of indicators to enable measurement of compliance.

Several publications based on the above ideas are already available. The first one relates to the healthcare case study whereas the second one uses patterns to capture knowledge in a specific healthcare domain. The third and fourth papers represent GoPF's approach for maintaining the captured knowledge in pattern families. The next two papers highlight the requirements of outcome-based regulatory compliance, which is then used as the basis for creating patterns in the aviation security domain. Using this domain as a case study, the last paper illustrates the use of indicators in GoPF-based patterns, highlights the process of creating patterns, and reports on the use of the framework for aviation security.

1. S.A. Behnam, D. Amyot, A.J. Forster, L. Peyton, and A. Shamsaei, "Goal-Driven Development of a Patient Surveillance Application for Improving Patient Safety" *E-Technologies: Innovation in an Open World*, LNPIB 26, Springer, 2009, pp. 65-76.
2. S.A. Behnam, D. Amyot, and G. Mussbacher, "Towards a Pattern-Based Framework for Goal-Driven Business Process Modeling", *8th Int. Conf. on Software Engineering Research, Management and Applications (SERA2010)*, Montréal, Canada, 2010. IEEE CS, pp. 137-145.
3. S.A. Behnam, D. Amyot, "Evolution of Goal-driven Pattern Families for Business Process Modeling", *5th International MCETECH Conference on eTechnologies (MCETECH 2011)*, LNBIP 78, Springer, 2011, pp. 17-31.
4. S.A. Behnam, D. Amyot, "Evolution of Goal-driven Pattern Families for Business Process Modeling", *Int. J. Electronic Business*, Inderscience Publishers, *(to appear, accepted Feb. 2012).*
5. R. Tawhid, M. Alhaj, G. Mussbacher, E. Braun, N. Cartwright, A. Shamsaei, D. Amyot, S.A. Behnam, and G. Richards, "Towards Outcome-Based Regulatory Compliance in Aviation Security", *20th IEEE Int. Requirements Engineering Conference (RE'12)*, Chicago, USA, September 2012. IEEE CS, pp. 267-272.

6. E. Braun, N. Cartwright, A. Shamsaei, S.A. Behnam, G. Richards, G. Mussbacher, M. Alhaj, and R. Tawhid, "Drafting and Modeling of Regulations: Is It Being Done Backwards?", *4th Int. Workshop on Requirements Engineering and Law (RELAW)*, Chicago, USA, September 2012. IEEE CS, pp. 1-6.
7. S.A. Behnam, D. Amyot, G. Mussbacher, E. Braun, N. Cartwright, and M. Saucier, "Using the Goal-Oriented Pattern Family Framework for Modelling Outcome-Based Regulations", *Second International Workshop on Requirements Patterns (RePa)*, Chicago, USA, September 2012. IEEE CS, pp. 35-40.

1.7 Thesis Outline

The thesis is organized as follows. Chapter 2 presents background information on URN, MDE, patterns, approaches that attempt to reuse domain knowledge, and other approaches that attempt to bridge the gap between business goals and processes. Then, Chapter 3 gives an overview of the GoPF framework metamodel and of how it is formalized with the help of URN, with restrictions in OCL. Chapter 4 presents the process of capturing recurrences, creating individual patterns, and creating pattern families. Chapter 5 describes four evolution mechanisms, provides related algorithms, and describes their application through case studies. Chapter 6 presents the mechanism for using the pattern families through customizing and extracting methods for particular organizations. This chapter also provides an algorithm and illustrates its application in the domain of patient safety. Chapter 7 evaluates the usefulness, applicability, and limitations of the suggested framework through case studies and comparisons with related work. Chapter 8 follows with conclusions and future work.

Chapter 2. RELATED WORK

This chapter provides an overview of background concepts and notations as well as of the research that proposes existing solutions for bridging the gap between business goals and business processes. First, related notations and standards are recalled in section 2.1. The User Requirement Notation (URN) and its two complementary sub-notations, the Goal-oriented Requirement Language (GRL) and Use Case Maps (UCM), which are used in this research, are briefly reviewed. Furthermore, the Business Process Modeling Notation (BPMN), an alternative to UCM, and the Business Process Metamodel Definition (BPDM), which is a related standard, are also discussed. Section 2.2 then reviews some core concepts of model-driven engineering. Section 2.3 gives a brief description of patterns and their formalization, which are some of the fundamental elements of this research. Finally, section 2.4 reviews existing concepts and approaches related to the bridging of the goal-process gap.

2.1 Related Standards and Notations

This section first presents the User Requirement Notation (URN) standard, which provides the underlying foundations and notations for goal and business process modeling used in this thesis. Next, OMG's Business Process Modeling Notation and Business Process Definition Metamodel are briefly introduced as standardized alternative foundations for business process and goal modeling, respectively.

2.1.1 User Requirements Notation

The development of notations for capturing and analyzing requirements is a major achievement of the last ten years [13]. The User Requirements Notation (URN), a standard of the International Telecommunication Union (ITU-T Z.151) [32][36][37][38], is intended for the elicitation, analysis, specification, and validation of requirements. URN contains two complementary graphical modeling languages for goals (GRL) and scenarios (UCM). URN allows software and requirements engineers to discover and specify requirements for a proposed system or an evolving system, and analyze such requirements for correctness and completeness. URN can also be used as a medium for commu-

nication with stakeholders about their requirements. The seamless presentation of goals and behavior is done with GRL and UCM diagrams respectively. Although the main application domains for URN include reactive systems and telecommunications systems, this language has also been applied successfully to the modeling and analysis of business goals and processes in many application domains [39][40].

Goal-oriented Requirement Language (GRL)

The Goal-oriented Requirement Language is a graphical language that enables the modeling of stakeholders, business goals (including functional and non-functional requirements), alternatives, and rationales. Modeling stakeholders' requirements with GRL makes it possible to define and understand the problem that ought to be solved [41]. Business analysts, requirement engineers, and software architects can achieve these objectives by using various types of intentional elements and relationships, as well as their stakeholders called *actors* (⬭). Intentional elements include *goals* (⬭) *softgoals* (⬭) for qualities and non-functional requirements, *resources* (▭) for conditions, *tasks* (⬭) for activities and alternative solutions, and *indicators* (⬭) for measures. Intentional elements can also be linked by AND/OR/XOR *decompositions*, by *dependencies*, and by *contributions*. Various qualitative positive and negative contribution types exist (see legend in Figure 3) as well as quantitative contribution levels on a [-100, 100] scale.

On the analysis side, GRL *evaluation strategies* enable modelers to assign initial satisfaction values to some of the intentional elements (usually alternatives at the bottom of a goal graph) and propagate this information to the other elements through the decomposition, dependency, and contribution links [42]. In addition, *importance* values are usually defined for high-level goals of stakeholders in a quantitative range of [0, 100] or with qualitative labels such as High, Medium, Low, or None. This ultimately helps assess the impact of alternative solutions on high-level goals of the involved stakeholders. Such models are also useful for evaluating trade-offs and documenting decision rationales. Figure 4 (left) is a GRL diagram that represents part of a goal model for increasing patient safety (and various contributing factors from other softgoals) in the case study. Note that despite of these features, the GRL notation and other goal modeling languages such as i* lack good modularizion constructs [43].

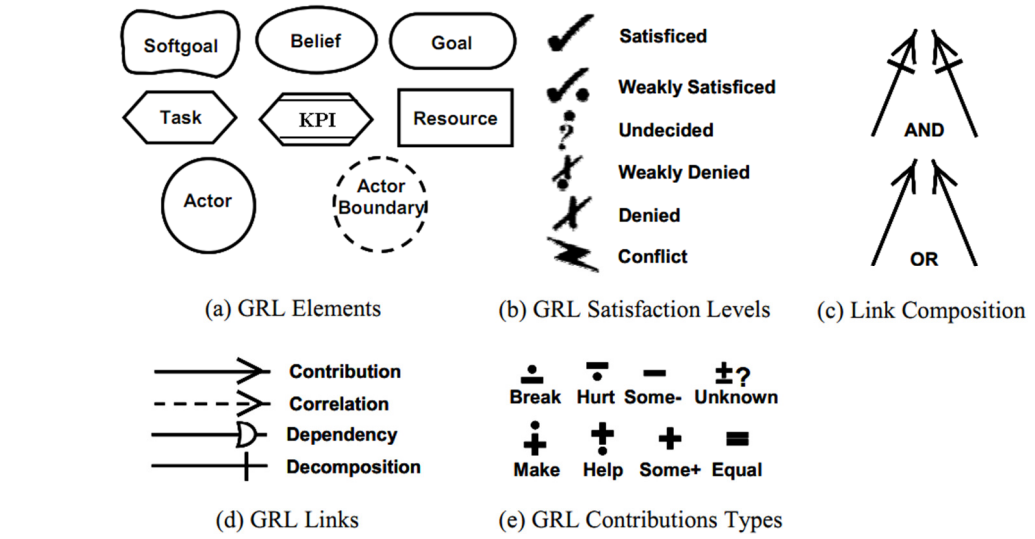

Figure 3 GRL notational elements

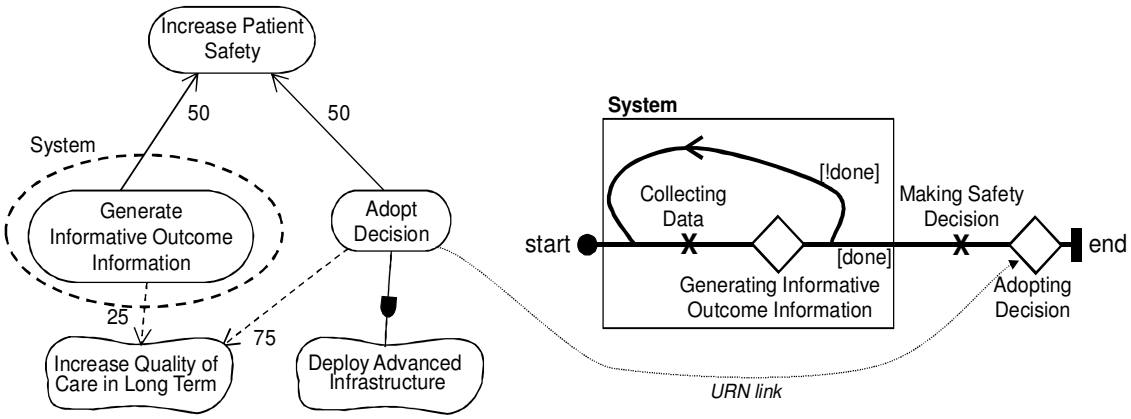

Figure 4 URN overview with GRL (left) and UCM (right)

Use Case Map

The Use Case Map (UCM) notation is a visual process modeling language for specifying causal scenarios and optionally binding their *responsibilities* (✕) to an underlying structure of *components* (☐). Responsibilities represent activities performed in a process whereas components represent actors, systems, and system parts. UCMs support most of the concepts used in common workflow modeling notations [44] including start points (●), end points (▌) as well as alternative and concurrent flows. Stubs (◇) are containers for sub-maps and may be used to organize a complex model in a hierarchical structure. Figure 5 shows common UCM elements. As an example, Figure 4 (right) illustrates a

UCM diagram that depicts the process that leads to Increase Patient Safety by providing the sequencing between relevant responsibilities and stubs (i.e., where the details are specified in a different UCM diagram, not shown here). In this figure, a URN link is used to trace the Adopt Decision goal in the GRL view to the corresponding stub in the UCM view.

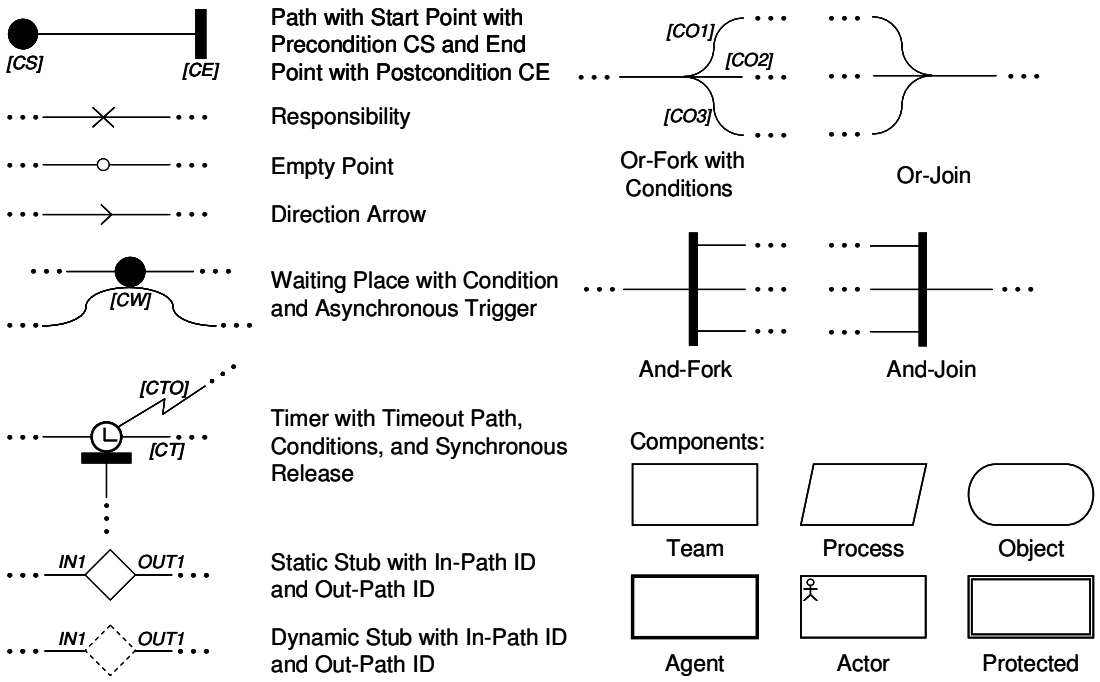

Figure 5 UCM notational elements

URN Links and Metadata

URN allows typed links called *URN links* to be established between modeling elements (e.g., between goal and scenario model elements). URN also supports the annotation of any model element with *metadata*, which are name-value pairs.

URN Profile

URN profiles are used for extending and tailoring the URN notation for a given domain. The URN standard includes several mechanisms that allow defining domain specific profiles. URN links and metadata, together with the possibility of adding constraints in UML's Object Constraint Language (OCL) [45], enable URN to be profiled to a particular application domain [46].

Tool Support

jUCMNav is an open source URN tool for the creation, analysis, and management of URN models [47]. It allows for the qualitative, quantitative, or hybrid evaluation of GRL models according to strategies, together with the abstract execution of UCM scenarios. jUCMNav is an Eclipse plug-in that also supports extensions to URN for modeling key performance indicators in the context of business process analysis, monitoring and performance management [48]. It also supports the verification of user-defined OCL constraints to enforce compliance to URN profiles [46].

2.1.2 Business Process Modeling Notation

Historically, there has been a gap between business process models that are mostly business-oriented and software applications that implement them [6][7]. The Business Process Modeling Notation (BPMN) helps alleviate this gap by providing a standard notation that can be mapped to execution models [49][50]. It was originally created by the Business Process Management Institute to answer the need for graphical business processing languages. This organization later became part of the Object Management Group (OMG), who released the first version of the BPMN specification in 2004. BPMN provides a comprehensive, integrated notation for business process modeling [51]. It is a graph-oriented and informal notation in which nodes can be connected almost arbitrarily [52]. BPMN is targeted towards analysts and its models will look familiar to most business analysts. This notation has attained a significant popularity after its introduction to the business process modeling community and it is now supported by dozens of modeling tools.

Figure 6 represents a typical BPMN model. Such models can also be mapped to BPEL4WS, which is the de facto standard for business process execution modeling. In other words, BPMN can be used to bridge the gap between business process models and executable models (Figure 1). However, unlike the UCM notation, BPMN lacks a complementary notation for modeling business goals and relationships to these goals.

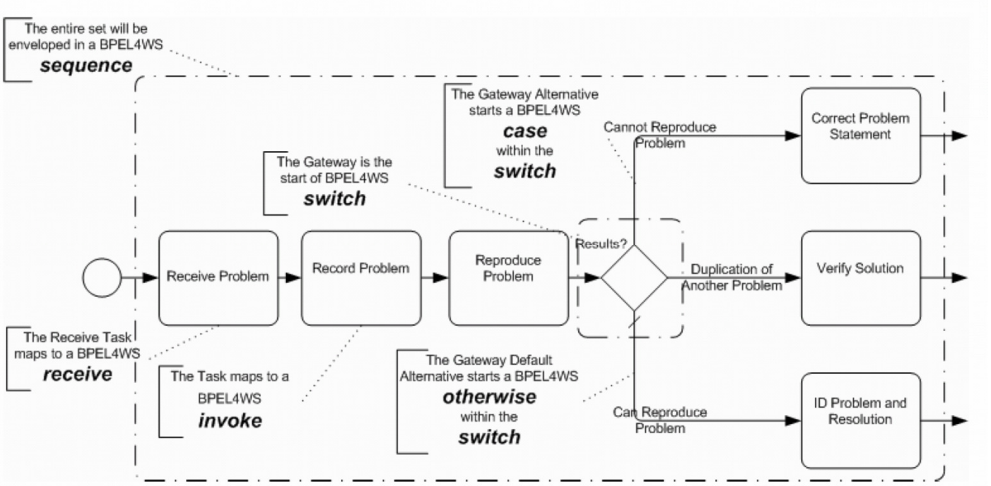

Figure 6 A BPMN model with annotation that shows mapping to BPEL4WS [49]

2.1.3 Business Process Definition Metamodel

As mentioned in [53], business processes have been at the core of business and technology improvement under the guise of many terms, notations, and methodologies. Business Process Engineering or Re-Engineering, Business Process Management (BPM), Business Process Execution, Process Improvement, Business Process Modeling and Workflow, and Service Oriented Architectures (SOA) are among dozens of such approaches and notations. Such methodologies and approaches have provided substantial benefits to organizations. However, many of these approaches are islands of particular technologies, methodologies or notations and do not work well with the others. Therefore, it is very difficult to develop solutions that integrate different types of models in order to address complex problems.

In order to address these difficulties, the Business Process Definition Metamodel (BPDM) was proposed as an infrastructure for specifying the business processes of an organization, independently of notations and methodologies [53][54]. BPDM is a well-defined, consistent, technology-independent, and precise metamodel, which was finalized by OMG in 2008. It provides a language syntax and semantics for business process modeling, but it omits the details of concrete syntax and semantics of such languages. In other words, BPDM documents the necessary concepts for business process modeling, and notations can be built on this standard. Therefore, BPDM-based notations and languages can be used together, while avoiding tight coupling with one particular business process

modeling infrastructure. BPDM also supports the separation of concerns in which the intended outcome of processes will be separate from how the process achieves those outcomes. Support for agility, easy integration and linkage of models, and loose coupling are some of the benefits promised by BPDM when modeling.

Currently, there is no known goal modeling language based on BPDM. On the other hand, URN is a standard that has a well-defined structure with complementary goal modeling and business process modeling notations and links between them. The solution in this research requires the combined use of goals and business processes in models. Consequently, using the URN standard is a better choice for formalizing this solution. However, despite the fact that the proposed solution (i.e., the GoPF framework) is based on URN, the underlying concepts of the patterns in the solution are not dependent on URN at the conceptual level. Thus, it may be possible to reconfigure the solution to use BPDM-based notations in the future, if there is a compelling argument to do so.

2.2 Model-Driven Engineering

This section discusses Model-Driven Engineering and its well-known incarnation, namely OMG's Model-Driven Architecture.

2.2.1 Model-Driven Engineering

As Atkinson and Kuhne observed [55], over the past five decades, software developers and researchers have been rising the level of abstraction in development artifacts. This has allowed them to specify *what* computers must do rather than *how* to perform it, and hence, shielded them from increasing complexity of the problems. The Model-Driven Engineering (MDE) approach is the continuation of this trend. MDE helps bridge the gap between different levels of abstraction and integrate different bodies of knowledge [56]. MDE is based on the premise that "everything is a model" and that a software development process can be considered as a set of transformations between models from different views and at different levels of abstraction. MDE claims to bring three major benefits to the process of software development. First, it simplifies and partially automates the process of developing software applications that satisfy their requirements. Second, it shields developers from the complexities of the environment in the process of software development [10]. Third, it flattens the learning curve and facilitates contribution of ex-

perts to the software development process by enabling them to create models at different levels of abstractions and with familiar notations in their domains of expertise. In their work, Atkinson and Kuhne predict that this move towards more abstraction holds the potential to drastically reduce the complexity of the problems that are considered hard by today's standards [55].

Figure 7 compares the underlying ideas of MDE and those of the object-oriented paradigm. Bézivin [17] argues that the basic principle in the object-oriented paradigm, "everything is an object", was most helpful in driving technologies of the 80s in the direction of simplicity, generality, and increased integration power. Similarly in MDE, the basic principle, "everything is model", offers many interesting properties in terms of simplicity and power of integration.

Although MDE does not limit the different bodies of knowledge that can be integrated for software development, many proposed MDE-based approaches, including the dominant Model-Driven Architecture (MDA) and its variations, focus on bridging the gap between the processes and applications discussed in Figure 1. Such approaches emphasize transformations from "what" is needed to "how" it can be done. In other words, the abstraction suggested in these approaches is mostly in the *solution domain* and not necessarily in the *problem domain* dominated by goals.

This thesis is concerned with abstractions in the problem domain. The knowledge about the problem is captured as part of the patterns in the GoPF framework. This can be seen as an extension of conventional MDE where goal models are included in the chain of transformations of models.

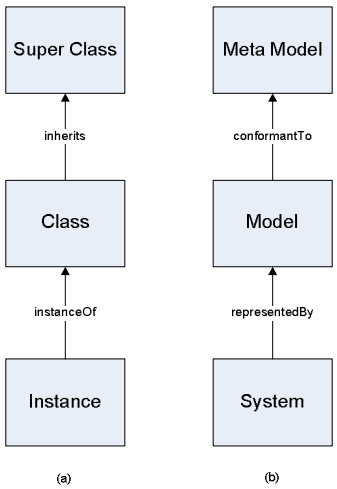

Figure 7 Basic notions in object-orientation (a) and MDE (b) [17]

2.2.2 Model-Driven Architecture

The Model-Driven Architecture (MDA) standard from Object Management Group (OMG) is a specific incarnation of the MDE approach [56]. Being a well-known standard, MDA is sometimes mistaken for the general concepts that MDE stands for [57]. A detailed description of OMG's MDA is provided in [58], including the following problems that MDA aims to solve:

"The MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. To this end, the MDA defines an architecture for models that provides a set of guidelines for structuring specifications expressed as models.

The MDA approach and the standards that support it allow the same model specifying system functionality to be realized on multiple platforms through auxiliary mapping standards, or through point mappings to specific platforms, and allow different applications to be integrated by explicitly relating their models, enabling integration and interoperability and supporting system evolution as platform technologies come and go."

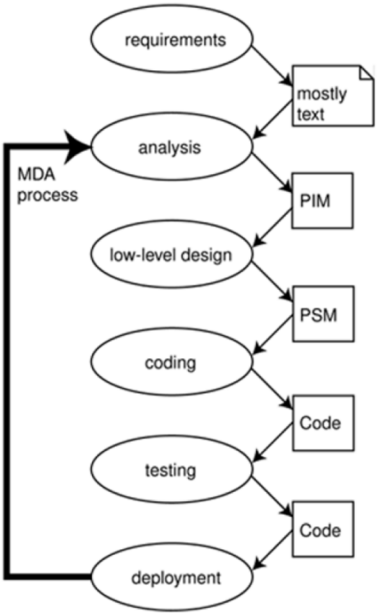

Figure 8 MDA software development life cycle [59]

The MDA approach includes ways of modeling functionality and implementation platforms as well as transformations from functional models to implementations. The life cycle of MDA in Figure 8 shows how it is mainly bridging the gap between an analysis model and the final software application. However, MDA does not include ways of specifying stakeholder goals and other intentional requirements.

2.3 Patterns

This section reviews the benefits of capturing knowledge in the form of patterns as a means of reusing knowledge. Some of the relevant approaches that take advantage of patterns are introduced and the importance of formalizing patterns is highlighted. Next, the evolution of patterns is discussed as one needs to maintain the knowledge captured as patterns when changes happen in the environment. Finally, frameworks are introduced as solutions that provide infrastructure and foundations for capturing patterns, organizing the knowledge, and reusing it.

2.3.1 Overview of Patterns

Patterns are three-part rules that express a relation between a *problem*, a *solution*, and a certain *context* [60]. They have been proposed to capture and categorize knowledge of recurring problems and give advice on possible solutions to those problems [15][60].

Patterns capture existing knowledge and important practices that have occurred repeatedly. They are known to be a means of increasing reusability. A pattern can be thought of as a reusable model that describes a need and solves a problem that may occur at different levels of abstraction and in different domains. Reusing knowledge helps improving software quality while minimizing the financial and temporal costs of creating software artifacts [61]. Another benefit of patterns is that they encapsulate recurring problems and solutions into modules. This is valuable because compared to holistic models, patterns are less subjected to changes. Patterns have been used in the area of software engineering with design patterns [12], in conceptual modeling with analysis patterns [62], in information system architectures with architecture patterns [63], in e-business with e-business patterns [64], and more recently in advanced distributed architectures with cloud computing patterns [65].

Patterns also provide a description of the forces at play. As Gabriel observed [66], patterns allow the forces in the systems to resolve themselves. Forces are design trade-offs that are affected by the problem and solution. In other words, forces typically discuss the reasons for using the suggested solution for the given problem in the given context. Describing forces and making the trade-off among them explicit is perhaps the most significant contribution of patterns. In some cases, as mentioned by Weiss in [67] and [68], forces cannot be resolved adequately by a single pattern. In such cases, a pattern includes references to other patterns, which help resolve forces that were unresolved by the current pattern. Patterns connected together in this way are often referred to as a *pattern language*.

Although the concept of patterns is usually practiced with well-known object-oriented design patterns, recently there have been efforts for using requirements pattern as enabler for reusing requirements knowledge [61]. For instance, Wen *et al.* [69] highlight the difficulties of reusing the knowledge about recurring security problems captured in the form of natural language. They suggest an approach for capturing requirements patterns containing the knowledge about the security problems in medical information system based on the *i** framework and on problem frames. In this approach, a large security problem in the domain is decomposed into sub-problems accompanied by an evaluation of threats for each sub-problem as well as potential protection measures. Wen *et al.*

did not include a systematic solution for capturing patterns but they underscore such solution as a desired extension of their work. In another example, Konard and Cheng [70] suggest an approach for using requirements patterns in the domain of embedded systems. They use these patterns for capturing and reusing requirements specifications. In this approach, the Unified Modeling Language (UML) is used for formalizing the structural and behavioral aspect of patterns. Amongst other benefits, Konard and Cheng report on two advantages of using requirements patterns in their case studies: i) using an even small set of requirements patterns helped novices in eliciting requirements of a fairly complete embedded systems, and ii) using these patterns facilitated the understanding and maintenance of the system specification by enabling the construction of more uniform system specifications. Although Konard and Cheng's approach suggests formal models for structural and behavioral aspects of requirements patterns, it does not provide a formal foundation, such as goal models, for capturing the intentions in the specification of patterns.

2.3.2 Pattern Formalization

Reusable knowledge in patterns enables efficient transfer of skills and experience of domain experts. However many pattern descriptions tend to focus on the solution to a problem and not so much on the problem and forces that are involved [71]. In addition, traditional pattern descriptions, including the Alexandrian [60], Gang-of-Four (GoF), and Coplien forms, are mostly expressed textually. The GoF form includes specific sections for intent, motivation, structure, participants, and collaborations [15]. In this form, the main emphasis is on the solution while the discussion about the forces is spread over different sections of the pattern. Motivated by drawbacks of the GoF form, Coplien provided a more rigid pattern structure by explicitly representing forces and consequences of patterns [72].

When patterns are represented in such forms, it becomes challenging to recognize under which conditions a pattern must be selected, how to compare the patterns that address the same problems, and how to integrate the consequences of applying multiple patterns to a model or system. This important issue motivated the development of better formalizations of patterns. For example, Taibi *et al.* describe why patterns must be formalized and propose that structural and behavioral aspects of design patterns be combined in one formal specification [73].

Formalizing the solution aspects of patterns has received more attention than the problem and force aspects, although there are noticeable exceptions. For instance, Araújo and Weiss have suggested an explicit representation of forces that are involved in patterns [74]. These forces and trade-offs are captured and then analyzed using the Non-Functional Requirements (NFR) framework [75]. In a related work by Ong and Weiss [76], the forces that are affected by a pattern are derived through close reading of the textual pattern description. This enables the discovery of pattern contributions to the overall system concerns, which were previously implicitly represented in textual descriptions. Their main finding is that the contributions of a pattern are a lot less apparent in textual descriptions.

Gross and Yu [77], together with Chung *et al.* [78], represent ways of reasoning about patterns using NFRs. Similarly, Mussbacher *et al.* [71] formalize patterns with URN, including problems and forces with GRL, and solutions with UCM. However, their work is more concerned with connections between patterns at the language level, as well as with establishing models of the forces and the trade-offs that exist in a particular domain.

Andrade and Logrippo [79] have used Use Case Map scenarios to describe patterns of behavior in wireless systems. Andrade, in her thesis [80], represents the need for requirements and analysis patterns and shows that these patterns can be used for reusing the knowledge about problems and solutions in mobile communication domain. Using Use Case Map scenarios for capturing the commonalities of solutions, she also provides methods for capturing and reusing these patterns. In similar work by Billard [81], Use Case Map scenarios are used to capture agent interaction patterns. Mussbacher and Amyot [82] proposed Use Case Map modeling patterns for describing and composing telephony features.

2.3.3 Evolution of Patterns

Changes in the stakeholder's concerns and domain's circumstances are unavoidable. This consequently leads to changes in the chain of artifacts in software development, from requirements models to software applications. This ripple effect in turn leads to resources being consumed for software artifact evolution and, unfortunately, artifact evolution is

also subject to errors. One benefit of creating artifact with the aid of patterns is that effects of changes in the models would be limited to specific modules. However, once patterns are established, it becomes somewhat difficult to maintain them and update their embedded knowledge. As Henninger and Corrêa observed [83], this should not, in general, be a major problem because patterns are ideally "timeless". Nevertheless, the rapid pace of change in technology has highlighted the need for evolution of various kinds of patterns. Requirements of stakeholders are changing even faster than foundation technology of design patterns. Hence, this evolution challenge is more pronounced for requirement patterns of specific domains. In the long term, a collection of patterns targeting requirements (including goals and processes) remains useful if the collection can be evolved to accurately address the current recurring problems and solutions of the domain.

Zhao *et al.* suggest an approach for the evolution of pattern-based designs [84] and of design patterns [85]. They propose a graph transformation method at the pattern level for evolving and validating patterns and pattern-based designs. Likewise, Dong *et al.* [86][87] propose a transformation based on two levels (primitive and pattern) to formulate the evolution process of design patterns. However, the approaches of Zhao *et al.* and of Dong *et al.* are limited in the context of this thesis because i) they are focused on design patterns and are mostly fine-tuned toward changing UML class diagrams, and ii) evolution is limited to variations of the initial pattern, i.e., the evolved pattern must be reducible to the initial graph. For instance, an *abstract factory* pattern can evolve to a new variation, which must still be based on the principles of abstract factory patterns. In this thesis, evolution of the patterns is considered at a more abstract level that captures the knowledge about the goals and requirements of stakeholders. Furthermore, patterns can be changed beyond the principles of their initial versions.

Aoyama [88] also highlights the importance of adapting patterns to rapid changes in requirements. His research first proposes a more formal representation of patterns, called *pattern type diagram*, which is then used as a basis for an evolutionary mechanism. Evolution of patterns in Aoyama's research is mainly focused on design patterns. Furthermore, evolution mechanisms in Aoyama's work (represented in *pattern evolutionary diagrams*) are mostly concerned with capturing changes that happen for a particular de-

sign pattern and for capturing the new emerging variations (e.g., evolution in the *Factory* pattern family.)

Kobayashi and Saeki studied the evolution of patterns from a different viewpoint [89]. They consider software development as pattern instantiation (i.e., applying a pattern to an actual problem) and pattern evolution. In their work, the evolution of patterns is about creating new artifacts from artifacts found in earlier stages, which finally results in the creation of a software application as the ultimate artifact in the chain of artifacts. Therefore, the "evolution" is not used for adapting to changes at a pattern level, but is used more like stepwise refinement. This is different from this thesis' objective for pattern evolution where the knowledge in the pattern is meant to be kept up to date.

2.3.4 Pattern Framework

Frameworks are a reuse technique for providing recurring solutions to sets of problems in a particular domain. Gabriel defines a framework as a system that can be customized, specialized, or extended to provide more specific, more appropriate, or slightly different capabilities [66]. In another popular definition from Johnson and Foote [90], a framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact. These definitions tend to define programming frameworks. However, frameworks can potentially be used at more abstract levels for reusing the knowledge and for bridging the gap between business goals and business processes. The importance of such systematic approaches for classifying and reusing requirements patterns is highlight by Naish and Zhao in [61].

Iida *et al.* [91] propose a process pattern framework composed of process templates. The framework assists software engineers in constructing their custom software processes by selecting and plugging process templates. This approach hence provides project-level reusability of process templates. Prior to that work, Iida had also described an early attempt at capturing process elements with patterns, for the software development domain [92]. Through transformations applied to a primitive process, customization to a particular organization was possible, which is similar in spirit to the OCEM method in the GoPF framework. However, these approaches consider only roles, products and activities (process definitions at the level of UCM), and not goals of the patterns or of the

organization. The selection of patterns to apply during transformations is hence done in an ad hoc way. In addition, patterns are fine-tuned for software development processes and there is no mechanism in place to maintain the framework.

In his thesis, Tran reviewed many pattern-based process modeling approaches that suffer from the same weaknesses [93]. His approach however is interesting in that it formalizes the process patterns with a metamodel and provides algorithms to apply them successively to processes for their evolution. Tran's technique hence shares some common objectives with those in this thesis, except that this thesis emphasizes the evolution of patterns, and also considers goals in addition to processes.

2.4 From Business Goals to Business Processes

This section discusses different approaches that aim to reuse domain knowledge in the form of business goal and process models. Some of the approaches also explore the advantages of capturing and using the connections that exist between the business goal and process models.

2.4.1 Enterprise Knowledge Patterns

Stirna *et al.* in [94][95][96] propose Enterprise Knowledge Patterns, a systematic approach for documenting, analyzing, and capturing patterns and for managing knowledge. Enterprise Knowledge Patterns contains patterns described with interrelated models such as goal, business process, and concepts models. These models are used to describe reusable solutions for enterprise problems. However, this approach does not provide a formal semantics for links between goal models and business processes. This prevents an automated transformation of goal models to business process models.

2.4.2 Goal-Oriented Legal Compliance of Business Processes

Ghanavati *et al.*, in a recent literature survey [97], have reviewed 88 papers (selected from 800 related research articles) and have shown that goal-oriented languages have often been used to model regulations and compliance. These approaches provide notations and tools for modeling the objectives of organizations. These goal models are then used for evaluating the compliance of business processes in those organizations. Shamsaei *et al.*, in another literature survey on the usage of goal-oriented languages for

managing compliance in organizations [98], have reviewed 32 papers (out of 198 related research articles) and have concluded that in spite of availability of individual pieces, current approaches are not effectively combining goals and indicators of business process performance in the organizations.

These surveys show the benefits of using goal-oriented language in connection with business process models for compliance management. However, the surveyed papers put less emphasis on providing approaches that benefit from such connection. This prevents the effective capturing of the domain knowledge and its reuse.

2.4.3 Linking Business Goals to Process Models

Markovic and Kowalkiewicz [99] provide a business process ontology, a goal modeling notation, and a modeling pattern for linking goals and business processes. The ontology captures the knowledge of conceptual models. The links then enable the integration of the intentional perspective into the business process ontology. This ontology is used to perform automated analysis on goal specifications, i.e., by identifying inconsistencies, redundancies, and conflicts. The ontology has been used as a basis for querying the knowledge embedded in the conceptual model.

Rimassa *et al.* [100][101] present an approach to business process management that leverages agent technology. This approach includes a goal-oriented business process modeling notation (GO-BPMN), which is a visual language for specifying business processes. GO-BPMN essentially enriches BPMN with explicit goal modeling. However, the relationships between goals and business process are limited. For example, business processes are not linked to goal models at different levels of abstraction.

2.4.4 Aspect-Oriented Business Process Improvement

The framework proposed by Pourshahid *et al.* [102] is also a pattern-based and URN-based approach for improving business processes when monitored key process indicators change over time. This framework describes redesign patterns as transformation rules that are applied to the existing organizational models with the help of an aspect-oriented extension of URN. Redesign patterns are more generic than the patterns described by the GoPF framework, which encode more domain-specific knowledge. An aspect-oriented

approach, however, could potentially be applied to GoPF to describe the family of business processes for a pattern, at the cost of additional complexity.

2.4.5 Use of Ontologies to Increase Reusability

An ontology is defined as an explicit specification of conceptualization [103]. Ontologies can be used to capture and share knowledge. This can be useful to consolidate one important challenge of software development, which is capturing and sharing knowledge. This challenge arises when different stakeholders and software development actors have disagreements over terminology, concepts, and how concepts are modeled [104]. Consequently, by representing and sharing common knowledge, ontologies decrease misunderstandings among stakeholders over terminologies and concepts.

In recent years, ontologies have been used to provide a unified view on the business process space of organizations [99][105]. For instance, Kaiya and Saeki use ontologies to propose methods for detecting incompleteness and inconsistency issues in created models [106][107]. Such approaches are useful for providing a unified view on requirements or business processes. Yet, they do not integrate these two views by capturing the reusable knowledge about both business goals and business processes. In addition, ontologies in these approaches are providing an infrastructure that can be modeled as metamodels that are specialized for modeling the infrastructures.

2.4.6 Reusability and Domain Engineering

Domain engineering aims at reusing common knowledge in the domain for addressing the problems within that domain [108][109]. An essential activity in domain engineering is domain modeling, which supports building models that contain the domain knowledge. These models can be reused in the software development process, leading in decreased costs of software development. These models have three main roles [108][110]: i) a unified source of reference when ambiguities arise in the analysis of problems, ii) a repository of knowledge that facilitates reusing, teaching and communicating the common knowledge in the domain, and iii) a specification for the implementer of reusable components.

Wang *et al.* have suggested a domain modeling framework with five layers of ontologies for networked software applications [20][111][112]. This framework uses ontol-

ogies to encapsulate knowledge in different views such as goal, process, and service views. This approach enables substitution of web resources based on different users' preferences. However, the main emphasis of their work is on reusing the functionalities that are captured in the form of services for taking advantage of the service-oriented computation paradigm shift. Consequently, this framework helps bridging the gap between process descriptions and the web services that realize these processes.

Čiukšys and Čaplinskas have suggested an ontology-based method for reusing business processes in a domain [16]. In their approach, a business process ontology, an application domain ontology, and a process ontology are used to enable reusing business processes in different application domains. Similarly, the suggested approach for Semantic Business Processes Management by Filipowska *et al.* [113] leverages ontologies for capturing the knowledge about business processes in a domain. However, these suggested domain engineering approaches mainly emphasize capturing the common knowledge about business processes. On the other hand, capturing the knowledge about requirements of stakeholders and linking them with processes that realize them has received far less attention.

2.4.7 Customization Approaches

Traditional process-oriented software development generally pays little attention to high-level goals of stakeholders. Such approaches put little emphasis on eliciting and analyzing stakeholder's business goals on one hand and linking them to the processes that realize them on the other hand. Motivated by the above drawbacks, Lapouchnian *et al.* [13] propose an approach in which goal models capture the needs of stakeholders. They enrich the goal models with annotations so models contain the necessary details about how the goals can be achieved. Reasoning and selecting processes enables finding customization alternatives that best accommodates stakeholders' goals.

In a related paper, Yu *et al.* [114] propose a two-step approach that utilizes goal models for reasoning and selecting configuration alternatives that lead to better satisfaction of stakeholders' goals. In the first step, software applications are reverse-engineered and a goal model is created. The leaf intentions are associated with the configurable items, which lead to different behaviors of the software application. In the second step,

selecting the alternative that satisfies the best the root-level goals of stakeholders determines the appropriate configuration. Similarly, Liaskos *et al.* [115] propose an approach for configuring common personal software applications. In this approach, a goal model captures the needs of stakeholders. Next, the goal model is used for reasoning about the best configuration alternative that satisfies the goals of stakeholders. While Yu's approach and Liaskos' configure the software to realize the goals of stakeholders, they have two limitations. First, they mostly focus on goal models and do not include explicit reasoning about business processes. Second, each application must be accompanied by a goal model that represents the alternatives so it can be used when configuration is necessary.

Hui *et al.* [19] propose a framework for customization of software applications based on goals, skills, and preferences of stakeholders. In this approach, a goal model represents stakeholders' needs and its leaf intentions are mapped to class diagrams that denote the possible alternatives at the design level. Consequently, choosing an alternative determines the static diagrams that represent the software architecture, which best accommodates the needs of stakeholders.

2.4.8 Product Line Software Engineering

Software Product Lines [116] increase reusability by guiding organizations toward using core assets rather than developing software applications from scratch. Clements and Northrop [117] provide the details of product line software engineering. They highlight core asset development and using those assets for product development as its two major activities. By means of these activities, product line software engineering exploits the commonalities amongst products while it manages their variability.

Feature modeling is one of the most popular techniques used for building reusable core assets for a given domain [118]. This technique analyzes the domain, captures the externally distinctive characteristics of its products, and organizes them in the form of feature models. Features are externally visible characteristics that differentiate amongst products. This differentiation is a key enabler for selecting the appropriate core assets in product engineering. Lee *et al.* [118] highlight this as the main difference between feature modeling and other means of reusability (e.g., objects or aspects) where conceptual ab-

stractions are identifiable by internal viewpoints. Similarly, this is the difference between the reusability perspective used in this thesis and the one used in product line software engineering. Note that product line software engineering is an active area of research and that further opportunities for integrating it with the GoPF framework defined in this thesis will only be highlighted in future work (section 8.2.7).

2.5 Summary

This chapter provided an overview of User Requirements Notation (URN), which supports integrated goal and scenario modeling and analysis. Next, an overview of patterns and of how they improve reusability was provided. Sections 2.3 and 2.4 also highlighted existing research efforts that aimed to reuse requirements knowledge. Many attempts for bridging the gap between goal modeling and business process modeling and their foundations have been summarized in this chapter. Furthermore, this chapter discussed several weaknesses of the surveyed approaches (including traceability between goals and processes, pattern formalization, documentation and evolution, and customization in a given context) that the GoPF framework intends to address. A comparison between the GoPF framework and many of the approaches discussed in this chapter will be provided in section 7.3.

The next chapter defines the core concepts of the GoPF framework with a metamodel formalized as a profile of the User Requirements Notation.

Chapter 3. FRAMEWORK METAMODEL (FMM)

A pattern in the GoPF framework contains a *goal model building block*, *business strategies*, *business process building blocks*, and *realization links* between business goals and business processes that loosely couple goals in the goal model building block with model elements in the business process building blocks. GoPF-based patterns are organized in a *pattern family*, in which they are connected through refinement links (the term family reflects the parent-child nature of these refinement links). This chapter describes these key elements of the framework metamodel (FMM). In addition, it provides a formalization of this metamodel as a URN profile in order to benefit from its integrated goal/scenario concepts and existing tool support. In this profile, OCL constraints are used to enforce well-formedness properties of pattern families.

The patient safety domain is used throughout this chapter for providing examples that highlight the architecture and usage of the GoPF framework for intermediary organizations. Examples are also given later on to show how pattern families are maintained (Chapter 5) and used (Chapter 6).

3.1 Foundational Elements of FMM

3.1.1 Goal Model Building Block

Goal model building blocks capture recurring meaningful excerpts of goal models, i.e., several goals and their relationships that can stand by themselves, thus identifying common problems faced by organizations, potential solutions, as well as the forces that have to be considered when solving these problems. Furthermore, goal model building blocks enable reasoning about solutions that best address the identified problem by selecting one strategy from a set of alternative solutions (captured in business process building blocks). Goal model building blocks are GRL-based goal models that emphasize *what* stakeholder requirements are and *what* must happen in order for the requirements to be realized. Each goal model building block contains a self-contained meaningful and reusable piece of goal model that represents a problem and elements of its solutions at a similar level of abstraction. This goal model may also include side effect intentions, which are affected

differently by alternative solutions. In addition, goal models may include the resources that represent conditions and their effects on the intentions of goal model. Finally, each goal model building block includes contributions connecting intentional elements of a model. A contribution is a weighted link that indicates the potential realization effect of one element on another element. Figure 9 illustrates a goal model building block that represents the highest-level problem and elements of solutions faced by some healthcare institutes in the patient safety domain.

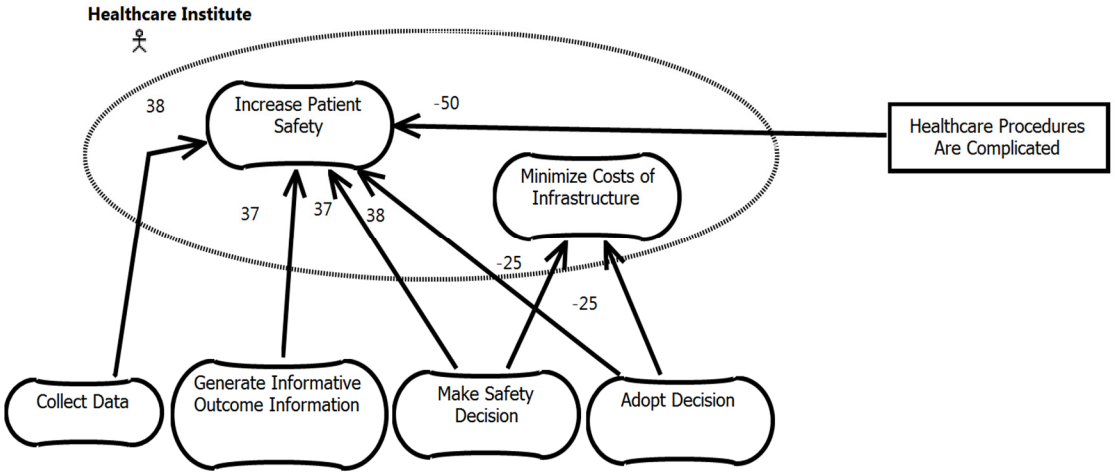

Figure 9 Goal model building block for Increase Patient Safety

Goal model building blocks are created by analyzing existing goal models and consulting domain experts, which leads to locating and capturing recurrent best practices. Because these models represent different levels of abstraction, the elements of solutions in one model can be potentially considered as problems at less abstract levels and so on. In other words, *what* must happen for one stakeholder's main goal to be realized, can in turn be the main goal of another stakeholder. In such cases, the stakeholders are at different levels within the organization and the latter stakeholder's achievement aids the former stakeholder's goals to be fulfilled. Goal model building blocks may need to be changed and improved over time as more knowledge is gained or as the circumstances of domain are changed, and GoPF encourages this practice through evolution mechanisms described in Chapter 5.

3.1.2 Business Process Building Block

A variety of definitions of business processes have been suggested [119][120][121][122]. In this thesis, a business process is identified as a structured set of activities that are designed to fulfill a goal for a particular stakeholder. *Business process building blocks* are recurrent abstract business process models that represent a solution, i.e., the process of achieving goals, while leaving the concrete implementation of the process elements to later deployment steps. Business process building blocks are Use Case Maps and are created by analyzing the solutions identified in existing business processes and by consulting domain experts. They capture the excerpts recurrently used as solutions for achieving recurrent goals. A business process building block specifies *how* a solution is carried out by laying down its steps and providing the sequencing of such steps. Figure 10 illustrates two business process building blocks defined at the same level of abstraction as the Increase Patient Safety goal model building block. They represent two alternative solutions for increasing patient safety. One alternative solution employs ad hoc improvements based on collecting and analyzing data. This alternative is more suitable to small healthcare institutes and institutes with limited resources. The other model represents a solution that systematically improves the healthcare institute but that also needs more resources from the underlying infrastructure.

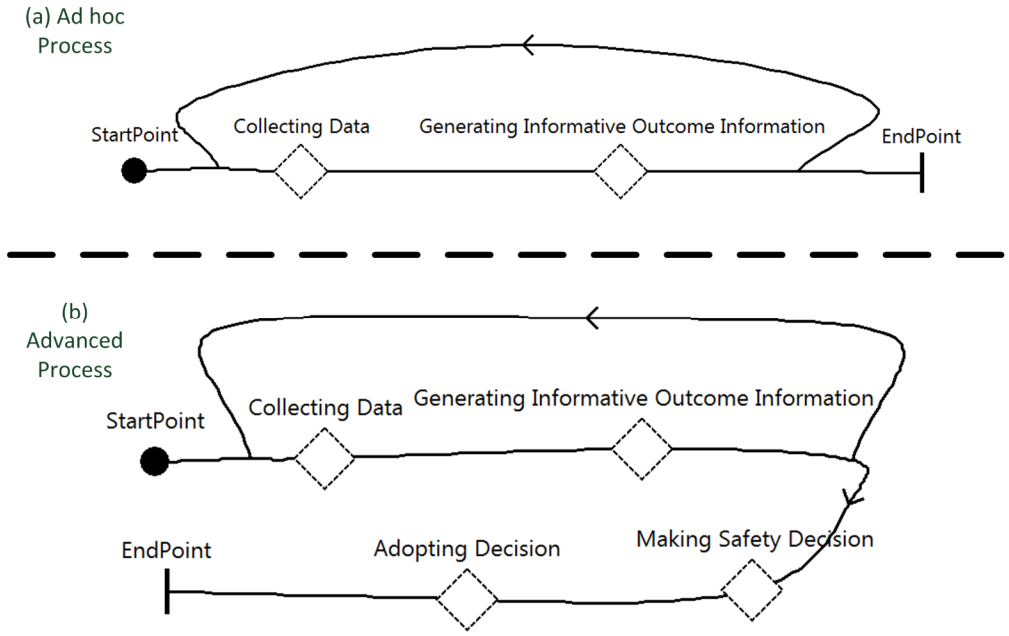

Figure 10 Two alternative business process building blocks for Increasing Patient Safety

3.1.3 Collection of Business Process Building Block

Individual elements of the business process building blocks may overlap, be shared, or be slightly altered. Hence, a set of business process building blocks that specify alternative solutions to a problem that is common to other members of the set may be viewed as a *collection of business process building blocks* similar to families of software products. In a collection of business process building blocks, each member represents one possible way of fulfilling a goal shared by all members of the collection. One can think of the members of the collection as design alternatives for fulfilling the common goal of the collection captured in the corresponding goal model building block. These collections are used for shaping the possible recurring solutions to the recurrent problems. For instance, Figure 11 shows a collection of business process building blocks that address a common goal in the patient safety context, i.e., Make Decision, in which each member represents an alternative solution. Although each member addresses the common goal, there are variations among the members of the collection, which may lead to differences in the degree of achieving the common goal and the conditions associated with a member.

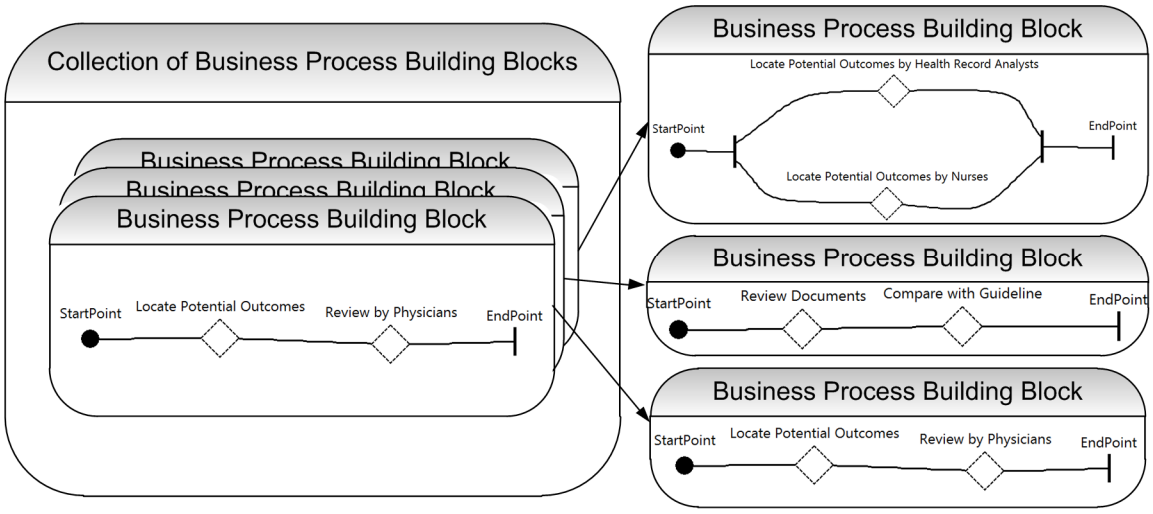

Figure 11 A collection of alternative business process building blocks for Make Decision

3.1.4 Business Strategy

Real problems may lead to complex goal models with many goals at different levels of abstraction. In many situations, goals may be satisfied in different ways (collection of business process building blocks) and to different degrees. Each of these ways (members of business process building block collection) provides an alternative solution that shapes

the business differently. This definition of business strategy complies with its usual definition in game theory as “a plan of action” [123], which is consistent with the definition of the word “strategy” in the Oxford Dictionary. Each *business strategy* is thus composed of a business process building block along with its anticipated effects (the *evaluation strategy*) on the intentions of the relevant goal model. Business process building blocks define the behavior and structure of the solutions in more detail. Evaluation strategies represent the anticipated effects of each business process building block that provide the rationale for choosing the most appropriate solution in a given context. Business strategies that contain pairs of <business process building block, evaluation strategy> provide a twofold advantage. First, a business strategy captures the knowledge of a domain in terms of reusable alternative solutions, their effects on the problems, and consequent satisfaction of stakeholders. Second, this structure benefits the stakeholder by providing the flexibility of selecting the solutions that suit best their specific requirement and circumstances. Therefore, this architecture enables reusing business process blocks for incrementally building complex process models for organizations, which are in line with their requirements.

3.1.5 Pattern

In the GoPF framework, *patterns* are containers that capture the knowledge about domain in three ways. First, patterns capture recurring problems and elements of solutions as well as conditions and secondary intentions (and side effects) in the form of goal models (goal model building blocks). Second, patterns include a group of alternative processes and their effects on the related goals, which are simply pairs of business process building block and evaluation strategies (i.e., business strategies). Finally, patterns include realization links that connect related processes from business process building blocks and goals from goal model building blocks. Figure 12 represents the architecture of a pattern in the GoPF approach.

Patterns and their constructs are formalized using URN. Such a formalized foundation brings three benefits. First, clear specification of requirements and solutions enable business analysts to better discover reusable requirements, to choose from a set of available solutions, and consequently to create hierarchical models for both requirements and corresponding solutions. Patterns based on such formalization are building blocks

that make it possible to reuse the domain knowledge by which customized goal and business process models can be built for specific organizations. The second benefit of formalizing patterns is avoiding unclear requirements and corresponding solutions. This facilitates maintaining the patterns when changes in the domain highlight the need for changing the patterns. Such an approach to pattern specification attempts to address the difficulties that arise by implicit and potential ambiguity in the knowledge captured in textual specifications of patterns, as highlighted in the literature review (section 2.3.2). Finally, a formalized foundation enables automatic mechanisms for creating, maintaining and using patterns. Such mechanisms will be defined in Chapter 5 and Chapter 6.

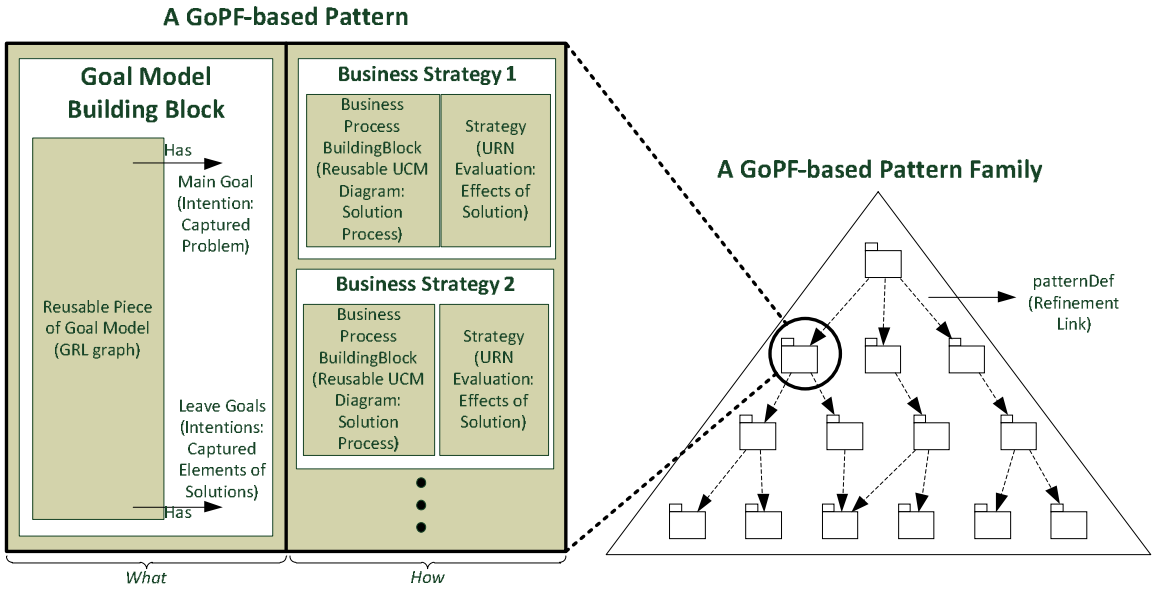

Figure 12 Architecture of a typical pattern in GoPF

3.1.6 Pattern Family (PF)

For a specific domain, different patterns may exist that describe problems and solutions at different levels of abstraction. Each of these patterns can be linked to those related patterns that potentially refine it. The GoPF framework defines the concept of *Pattern Family* (PF) as a container that includes all the patterns for a particular domain. Patterns in a PF are connected to each other through refinement links. A pattern family captures and organizes the knowledge of a domain in the form of patterns and their interconnections. Each pattern in the family is a self-contained, meaningful, and reusable building block of requirements and solutions in that domain. Figure 12 shows how a pattern family is composed of interconnected patterns.

Patterns refine each other when solutions at one level are considered as problems to be solved at other levels. Refinement links are in charge of capturing this type of information. These links between patterns represent how patterns can potentially refine other patterns. Capturing refinement links between the known patterns of a domain enables organizing the knowledge about that domain; this is an important advantage of pattern families.

Cysneiros *et al.* in [124] emphasize the importance of enabling the chain of reasoning from abstract to concrete models in requirements engineering. The characteristics of GoPF pattern families enable such reasoning at different levels of abstractions.

3.1.7 Roles

Different roles need to be played by people who want to use the GoPF framework.

Stakeholders

Freeman defines stakeholders as "any group or individual who can affect or is affected by the achievement of the organization's objectives" [125]. Freeman's definition is adopted here with emphasis on the fact that stakeholders at different levels in an organizational hierarchy have different goals in the organization's hierarchy of goals. *Satisfaction* measures to what level the expectations of a stakeholder are met when a corresponding goal is achieved.

Problems of stakeholders are *what* requirements engineers try to elicit, analyze, and address. The more important the stakeholders are in an organization, the more their satisfaction influences the success of a solution. Requirements of similar groups of stakeholders within a particular domain are also similar. In order to benefit from this similarity, GoPF captures recurring problems and recurring solutions in the form of patterns. Moreover, both the UCM and the GRL notations used in GoPF support modeling actors, which represent the stakeholders. This makes it possible to evaluate the effects of solutions on problems of stakeholders and their satisfaction.

The GoPF framework helps the PF users discover the requirements of stakeholders and create hierarchical solutions that are customized for them. Although most stakeholders are not primary users of GoPF, they are the key elements in creating and evolving

pattern families (FDM; Chapter 4). They will also reap the benefits of applications developed by PF users using the GoPF framework for creating hierarchal models of requirements and solutions (OCEM; Chapter 5).

PF Analysts

A *PF analyst* is a domain-specialized modeler in charge of creating or evolving a Pattern Family (PF). Based on interviews with stakeholders and domain experts, and on the analysis of available models, the PF analyst discovers requirements, locates recurrences in the models, builds patterns, creates a family of patterns, and maintains the pattern family by evolving it over time. In other words, the PF analyst analyzes the problems and solutions in a particular domain and captures this knowledge in the form of reusable patterns. A PF analyst usually works for an intermediary organization (e.g., a consulting firm), which provides services to organizations in a domain. Pattern families are the deliverable artifacts produced and maintained by PF analysts. *PF users,* who are the other common actors in intermediary organizations, then use these artifacts. Figure 13 shows the main interaction among these different roles.

PF Users

A *PF user* is a business analyst who uses a PF for exploiting the domain knowledge it contains. The PF user interacts with stakeholders of an organization within the relevant domain to elicitate the requirements and to extract and customize specific models addressing these requirements for that organization by using the patterns in the PF.

PF users usually work for intermediary organizations, which are the primary beneficiaries of the GoPF approach. For example, they may work for healthcare consulting firms in charge of developing applications for improving patient safety in hospitals. Such intermediary organizations are in a position to maintain their own pattern families through experience gained solving similar issues in different target organizations. They can also develop specific solutions for specific target organizations based on these pattern families. Reusing domain knowledge hence facilitates their development efforts. Stakeholders in a domain (e.g., hospitals and patients) will reap the benefits of applications developed by intermediary organizations but they are not considered as direct beneficiar-

ies of the GoPF framework. For instance, the GoPF framework is not responsible for of directly increasing patient safety; it only helps capturing and reusing problems and solutions that support stakeholders in achieving their objectives in the domain of patient safety.

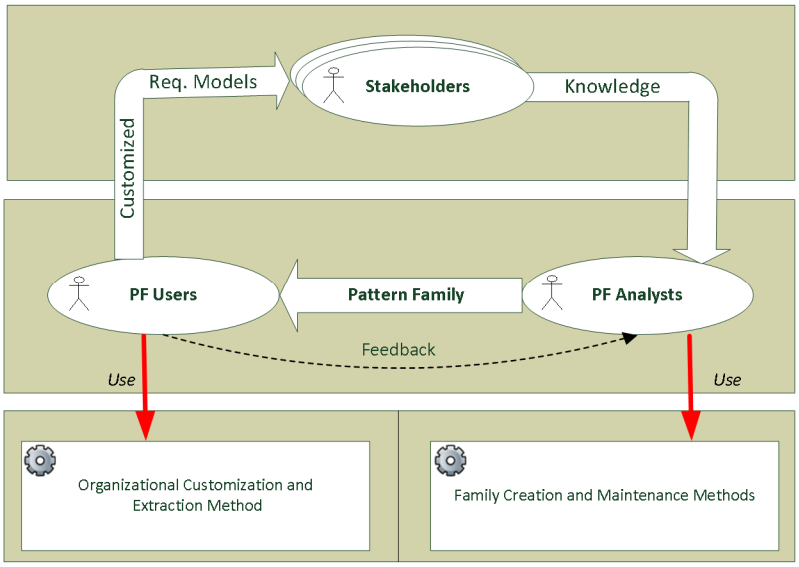

Figure 13 Interaction among different GoPF roles

3.2 Family Metamodel (FMM)

3.2.1 Formalizing the Family Metamodel (FMM)

The Family Metamodel formalizes the concepts of Pattern Family as a profile of the User Requirements Notation, as shown in Figure 14. The names between guillemets refer to corresponding metaclasses from the URN standard metamodel [32]. In URN, a *concern* is a model element that groups other model elements, including other concerns. URN *metadata* are used to associate stereotypes to model elements in a URN model that are part of this framework, as specified in Figure 14 (e.g., a URN concern may be stereotyped as a «pattern»).

A *pattern family* contains *patterns*, each of which includes one *goal model building block* (that formulates a problem and elements of its solutions) and at least one *business strategy* (that captures the arrangement of a solution along with its effect on the problem). Each goal model building block is essentially a GRL graph, and hence includes *intentions* (e.g., goals, tasks, and softgoals), *indicators*, and *element links* (i.e., contributions, decompositions, and dependencies) between them. In GRL, the intentions can be

contained in *actors* (not shown here because this concept is reused as is), which essentially represent stakeholders. The goal intentions contributing to the main goal of such a building block can themselves be refined by other patterns, through the new *patternDef* relationship, formalized as a URN link. Business strategies contain two main parts: an *evaluation strategy* (i.e., a regular GRL evaluation strategy, used for the evaluation of a goal model) and a corresponding *business process building block* (i.e., a UCM map describing the process that specifies among other things the ordering of the goals selected by the strategy). In addition, goals and tasks in the goal model building block can be realized by *process elements* (e.g., stubs and responsibilities) in the business process building block. *Realization links* connect goals that represent problems and elements of solutions on one hand and business process building blocks that realize the goals on the other hand. Such *realization links* are supported with URN links and enable traceability between goals and business processes. Further realization links between goals and business process building blocks are derived from existing associations (from Intention to BusinessProcessBuildingBlock via patternDef). Such links capture the connection between goals and those patterns that further refine them. The decomposition of patterns into goal model building blocks and corresponding strategies enables organizing the knowledge about the problem and its solutions in a reusable manner. Depending on the complexity of the system, a series of decompositions can recur to form a hierarchy of problems and solutions in a particular PF.

In this metamodel, goal model building blocks on one hand and business process building blocks and evaluation strategies on the other hand are two sides of the same coin (i.e., of a pattern). On one side, goal model building blocks represent the requirements and address the problems that are important to stakeholders. On the other side, business process building blocks and evaluation strategies represent recurring ways of doing business along with how they fulfill those particular goals. In other words, goals need business processes to be realized, and business processes are justified by goals.

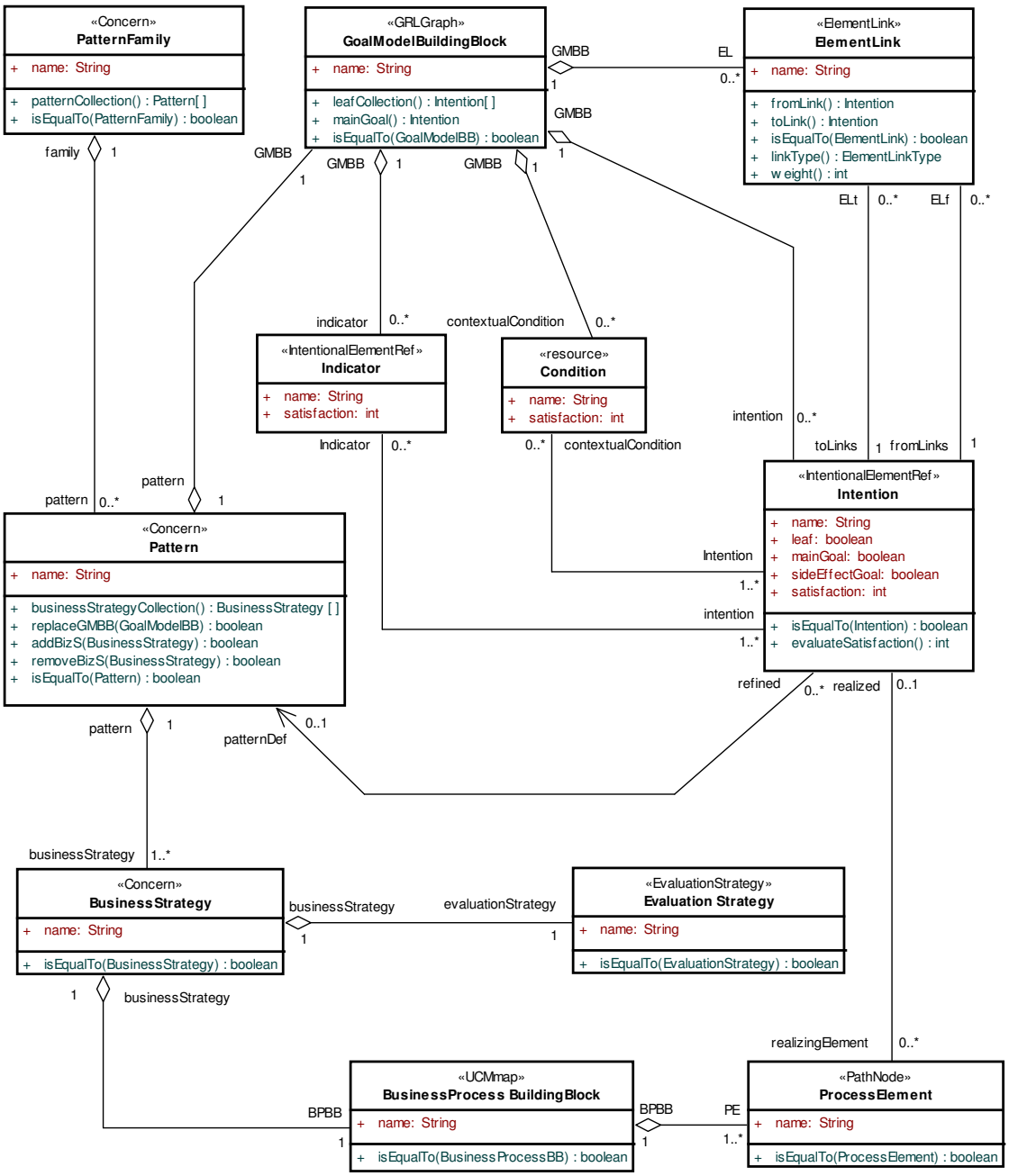

Figure 14 Framework metamodel

The benefits of casting the framework metamodel as a URN profile is that resulting models are expressed in a standard and familiar language, together with existing analysis algorithms (e.g., GRL propagation [42]) that are reused as is. In addition, jUCMNav can provide tool support for editing such models (with well-formedness checking through OCL constraints) and for analyzing them.

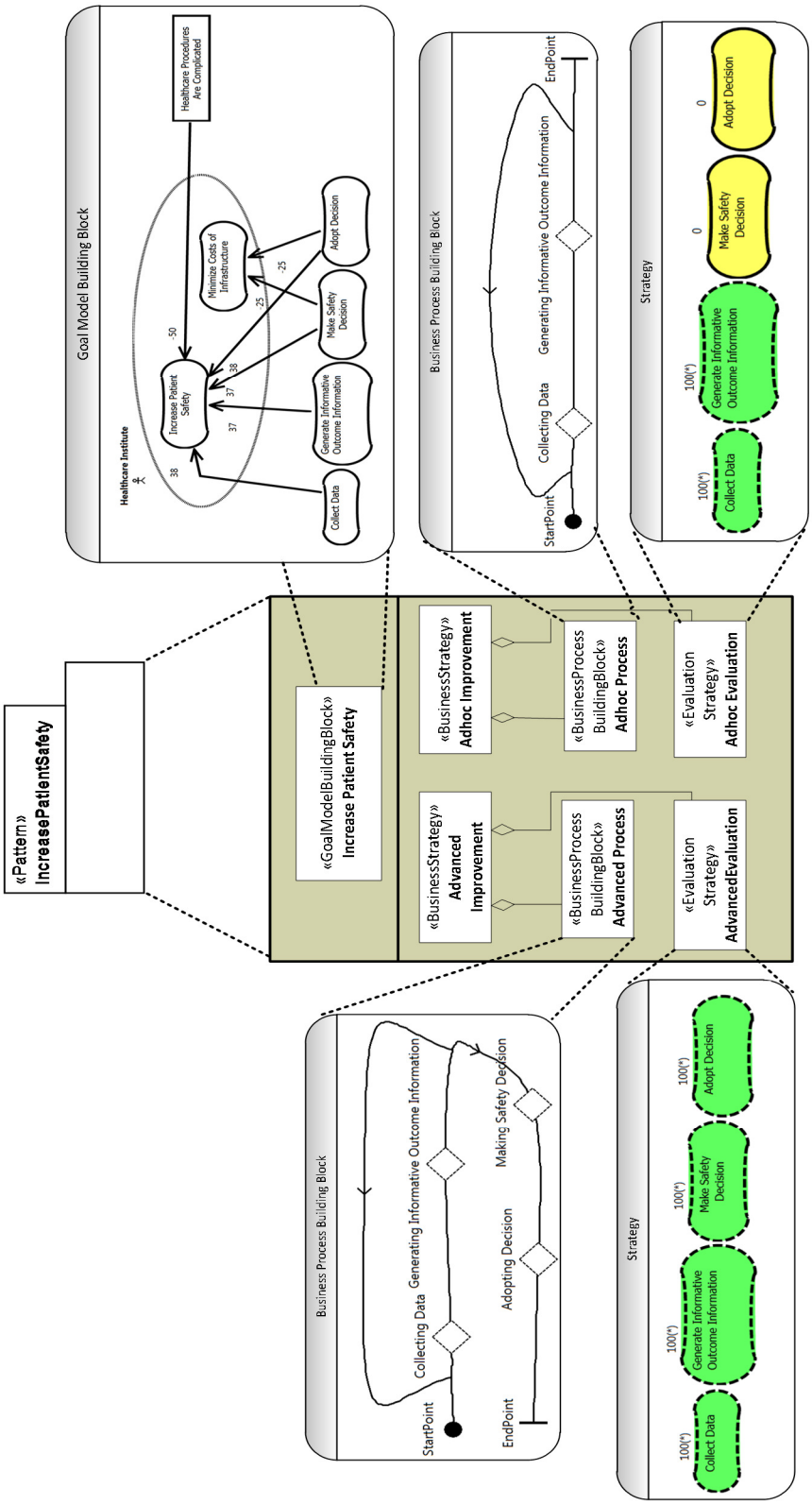

Figure 15 High-level representation of the Increase Patient Safety pattern

3.2.2 Example of FMM-based Pattern Family

FMM is used here to give a formal representation of a sample pattern family in the patient safety domain. Figure 15 illustrates the Increase Patient Safety pattern, which represents a top-level pattern (more abstract compared to other patterns of the family) concerned with the improvement of patient safety in a healthcare organization.

This pattern has a goal model building block capturing a goal model that formulates *how* to achieve *Increase Patient Safety* (i.e., the main goal of this pattern). It also contains two alternative business strategies, each composed of a business process building block (UCM) and of a GRL business strategy, which capture different solutions to address the problem as well as their effects.

Figure 16 represents this pattern in the form of an FMM-based object model. This object model was created with the UML Specification Environment (USE) [126][127], which is a tool for creating UML class diagrams and object models as well as for implementing and checking OCL constraints.

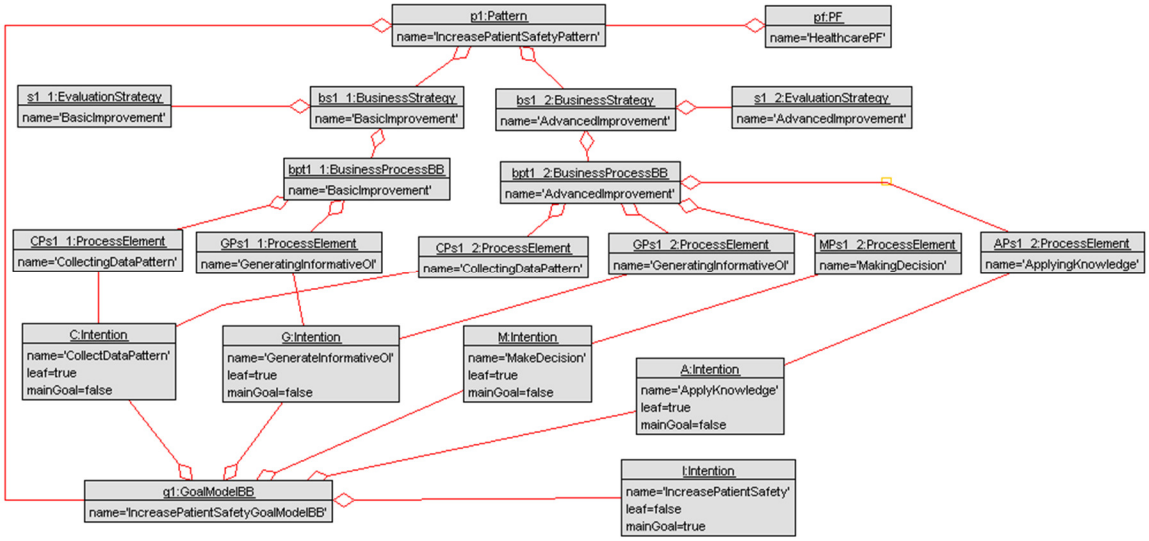

Figure 16 Increase Patient Safety pattern in the form of FMM-based object model

FMM is implemented with USE and is available at [128]. The FMM-based pattern family for patient safety has been built as an instance of this metamodel in the USE environment, and is also available online [129]. This sample family is a container for knowledge that includes the captured patterns. Figure 17 shows part of this pattern family in the form

of interconnected packages in which each package represents a pattern in the domain. As seen in the figure, patterns in this family are connected through patternDef links that capture the refinement relationships among the patterns. For instance, Collect Data is an intention that contributes to the satisfaction of the main goal of Increase Patient Safety pattern. In other words, this is an element of solution for satisfaction of the main stakeholder for the Increase Patient Safety pattern. Collect Data can in turn be considered as the main goal of another pattern that captures the refining excerpt of the goal model along with corresponding business processes and strategies. Consequently, Figure 17 shows how patternDef associates the refining Data Collection pattern to the Increase Patient Safety pattern. Similarly, other elements of solutions in the Increase Patient Safety pattern can be considered as recurring problems and captured as the main goals of new patterns. Figure 18 represents part of the formal, FMM-based object model of this family (for simplicity, all objects of ElementLink are hidden).

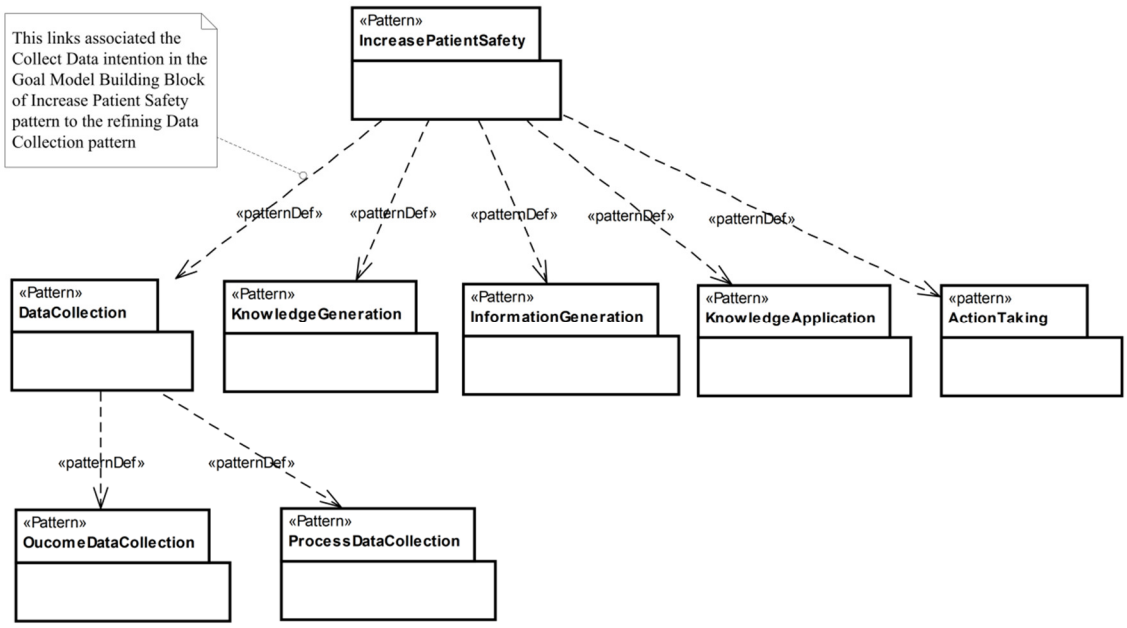

Figure 17 An excerpt of a pattern family for the patient safety domain

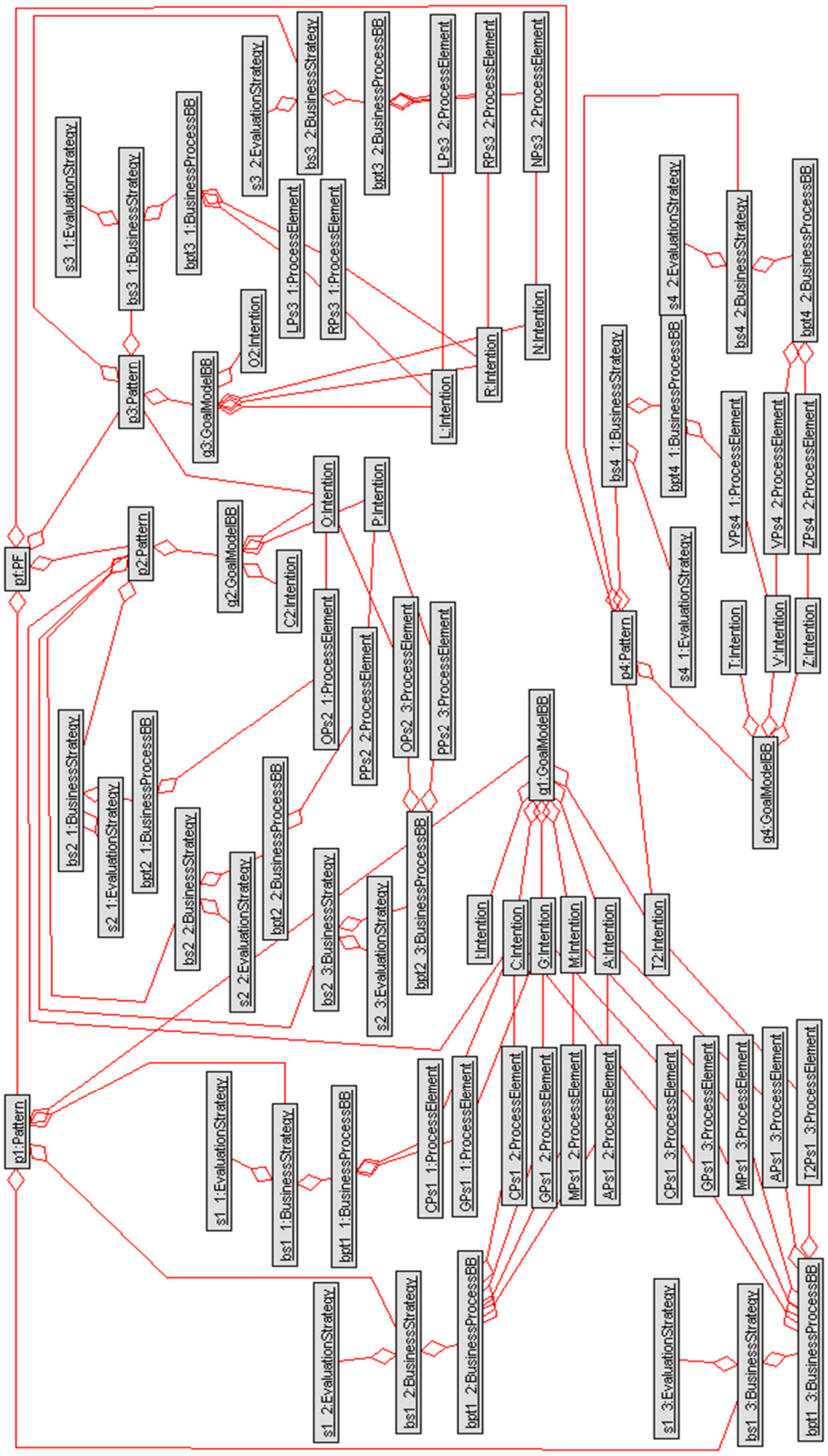

Figure 18 FMM-based object model of the sample patient safety pattern family

3.3 Well-formedness of FMM-based Models

3.3.1 Enforcing Well-formedness with OCL Constraints

The family metamodel (FMM) establishes the foundation for creating pattern families. However, this metamodel by itself is not sufficient to enforce well-formedness of the pattern family models. In order to address this issue, constraints on different elements of FMM were added for enforcing the well-formedness of FMM-based PFs. These constraints are specified formally with the Object Constraint Language (OCL) [45], which is a standard constraint language part of UML and maintained by the Object Management Group (OMG). OCL provides modelers with means of adding constraints to models, including invariants, preconditions, and post-conditions [130]. OCL 2.0 also enables modelers to query their models. These characteristics facilitate the specification of models in a formal yet comprehensive manner, which makes OCL a suitable option for enforcing restrictions in the GoPF framework.

Table 1 represents the constraints required for ensuring well-formedness of pattern families. These constraints are implemented in USE. In order to validate that the proposed constraints support well-formedness of the family metamodel, the following tasks were carried out:

- A UML class diagram was created that represents FMM in the USE environment [128];
- FMM-based object models representing sample pattern families for the patient safety examples were implemented in USE [129]. These object models offer a good coverage of the concepts in the metamodel;
- OCL constraints were implemented in USE; and
- The sample pattern families were validated against the OCL constraints.

The implementation details of these OCL constraints along with associated operations are provided in Appendix A. These OCL constraints are able to enforce the well-formedness of FMM-based models. Figure 62 on page 160 illustrates the result of validating a sample pattern family against the constraints.

Table 1 OCL constraints implemented to ensure the integrity of FMM-based models

Context	OCL Constraints	
PF	inv UniquePFName	C1
	inv UniquePatternNameInPF	C2
	inv UniquePatternMainGoalInPF	C3
	inv UniqueEvalStrategyNameInPF	C4
Pattern	inv UniqueBusinessStrategyNameInPattern	C5
	inv OnlyLeavesRefine	C6
	inv CorrectLeafMainGoalRefinement	C7
	inv NoOrphanPattern	C8
	inv NoCircularDefiningPatternExist	C9
	post businessStrategyCollectionisDone	P1
	post addGMBBisDone	P2
	post addBizSisDone	P4
	post removeBizSisDone	P5
GoalModelBuildingBlock (GoalModelBB)	inv UniqueIntenionNameInGMBB	C10
	inv UniqueElementLinkNameInGMBB	C11
	inv JustOneMainGoal	C12
	post leafCollectionisDone	P6
	post mainGoalisDone	P7
Intention	inv LeavesBeingRefined	C13
	inv ElementsAreIncludedInRelatedBSTs	C14
	inv EitherMainGoalOrLeaf	C15
	inv LeavesAreConnected	C16
	inv MainGoalsAreConnected	C17
	inv MiddleGoalsAreConnected	C18
	inv AllPossibleRefiningLeafConnected	C19
	inv NoMiddleIntentionPossibleRefinement	C20
	inv NoDanglingMiddleIntention	C21
	inv LeavesHaveOneMainGoal	C22
	inv MainGoalsHaveAtLeastOneLeaf	C23
BusinessProcess BuildingBlock (BusinessProcessBB)	inv UniqueProcessElementNameInBPBB	C24
ElementLink	inv DifferentSourceDestination	C25

3.3.2 Examples of OCL Constraints

This section discusses two of the constraints for presenting how OCL constraints maintain the integrity of FMM-based pattern families.

Preventing Circular Pattern Definitions

The capability that patterns in a pattern family can refine or be refined by other patterns in the family is one of the strengths of the GoPF framework. The framework metamodel provides the foundation for creating such refinement links. This is done by setting patternDef of leaf goals of a pattern, which can be potentially refined, to the refining patterns. The refining pattern can in turn be further refined by other patterns. For example, Figure 17 illustrates that the Data Collection pattern refines the Increase Patient Safety

pattern while also being refined by the Outcome Data Collection and Process Data Collection patterns.

Circular refinement happens when two patterns refine each other, directly or indirectly. Direct refinement happens when two patterns mutually refine each other without any other pattern in between. For example, in Figure 17, if the Increase Patient Safety pattern were to refine the Data Collection pattern, this would form a direct circular refinement. Indirect circular refinement happens when more than two patterns form a circular chain of refinements. In Figure 17, if the Increase Patient Safety pattern were to refine the Outcome Data Collection pattern, an indirect circular refinement would be formed.

Considering that the GoPF framework uses refinement for discovering and extracting details of solutions to recurring problems, circular refinements represent incorrect structures in the pattern family and prevent the discovering, extracting, and maintaining processes. The framework metamodel alone does not prevent circular refinements. It hence needs to be supplemented by an OCL invariant to ensure pattern families do not include such refinements:

```
context Pattern
inv NoCircularDefiningPatternExist:
      self.DefiningPatternSet()->excludes(self)
```

This invariant ensures that a pattern is not included in the set of its directly or indirectly refining patterns. In order to retrieve the set of refining patterns, and because OCL does not support transitive closure as a first-class construct of the language, there is a need to define two OCL operations, represented in Table 2, for the metaclass Pattern.

The first OCL operation, DownPatternSet, is recursive. It takes a set of patterns as input, finds all the patterns refining any member of the set, adds them the set, and recursively calls the operation with the new resulting set. This will continue until no new refining pattern is found. This happens when the resulting pattern set is equal to the set received as parameter. The final set represents all the patterns that directly or indirectly refine the initial set of patterns. This set is returned as the result of this OCL operation.

DefiningPatternSet is the second OCL operation. When DefiningPatternSet operates on a particular pattern, it initializes a set of patterns, calls the DownPatternSet operation, and returns the final refinement set. In order to create the initial set for the pattern it operates on, DefiningPatternSet assesses patternDef links of leaf intentions of the pattern's goal model building block and creates an appropriate set of refining patterns. If this set has at least one member, DownPatternSet will be invoked with this set as its initial parameter. The result of this recursive operation represents those patterns that are directly or indirectly refining the pattern that invokes DefiningPatternSet.

NoCircularDefiningPatternExist is an OCL invariant, and must hence be true all the time. In order to enforce the constraint on every pattern in the family, this invariant uses DefiningPatternSet for finding each pattern's refinement set. If this set excludes the pattern, NoCircularDefiningPatternExist returns *true*, indicating that no circular refinement link has been found for that pattern. Ensuring that this invariant is true for all the patterns in the PF prevents the family from including any circular refinements.

Table 2 OCL operations for retrieving a set of refining patterns

```
DownPatternSet(s:Set(Pattern)):Set(Pattern) =
        if s->includesAll(s.GMBB.intention.patternDef->asSet())
        then s
        else DownPatternSet(s->union(s.GMBB.intention.patternDef->
                                                        asSet()))
        endif
```

```
DefiningPatternSet():Set(Pattern) =
        if self.GMBB.intention.patternDef->asSet()->size() > 0
        then
           DownPatternSet(self.GMBB.intention.patternDef->asSet())
        else null
        endif
```

Figure 19 (a) shows the result evaluation of the NoCircularDefiningPatternExist OCL constraint against the sample patient safety pattern family presented in Figure 18. This constraint is evaluated to true because the pattern family is well formed. On the contrary, the pattern family presented in Figure 20 includes a circular refinement relationship between patterns. In this PF, Increase Patient Safety (p1) is refined by Data Collection (p2) through a patternDef link from Collect Data (C) to p2. Next, Data Collection (p2) is refined by Outcome Data Collection (p3) through a patternDef link from Collect Outcome Data (O) to p3. Finally, there is an erroneous refinement relationship from Outcome Data

Collection (p2) to Increase Patient Safety (p1) because of the patternDef link from Analyze Observations (N) to p1. Consequently, as can be seen in Figure 19 (b), NoCircularDefiningPatternExist is correctly evaluated to false, hence expressing the existence of a circular refinement relationship amongst the patterns. Finally, as illustrated in Figure 21 (a), an investigation of the NoCircularDefiningPatternExist violation in USE makes it possible to locate the circular refinement chain and possibly take steps to solve the indicted problems.

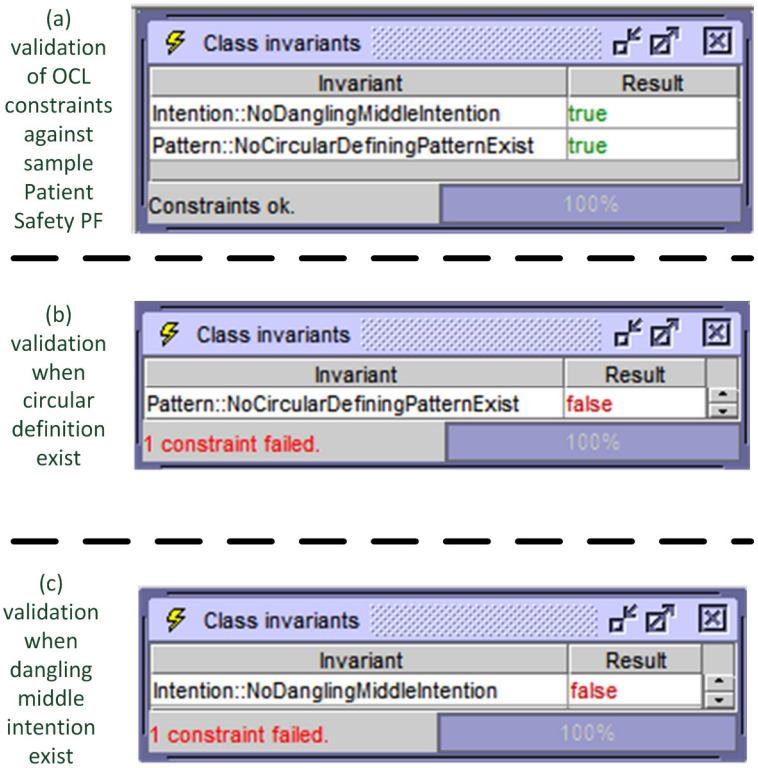

Figure 19 Validation of OCL constrains for preventing circular defining pattern and for dangling intentions

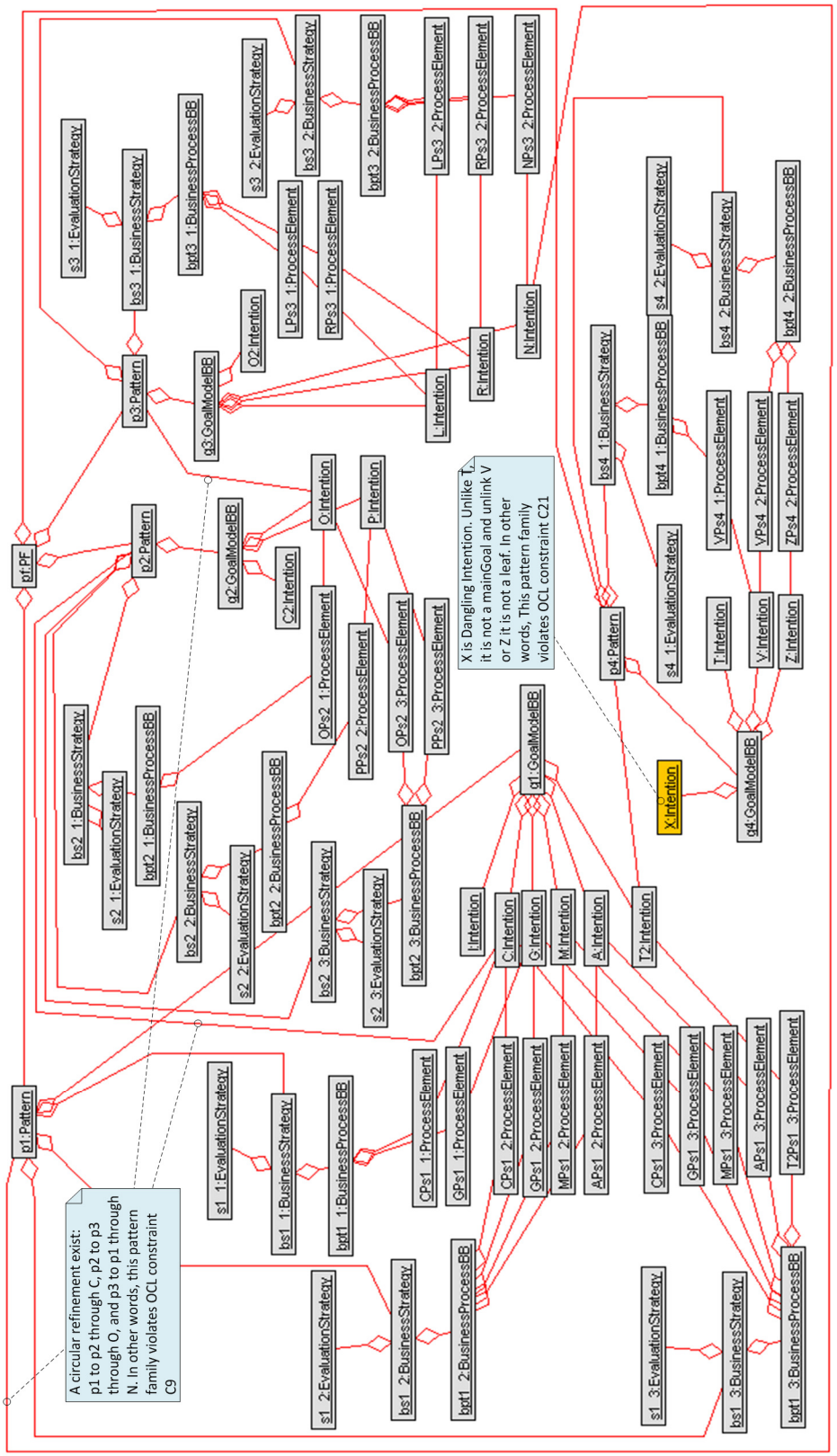

Figure 20 A pattern family with circular refinement patterns and dangling intentions

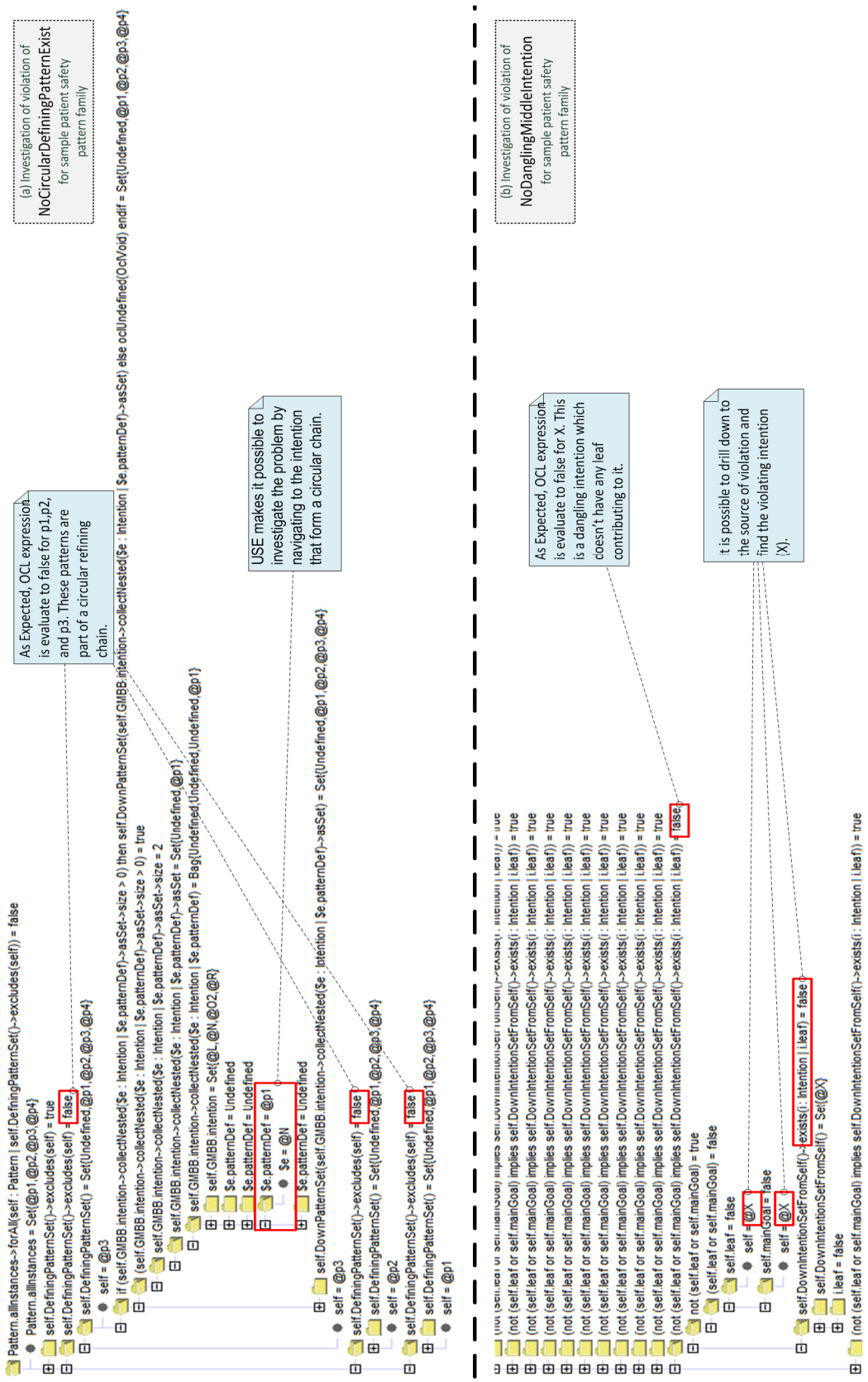

Figure 21 Investigation of the source of OCL violations in the examples

Preventing Dangling Middle Intentions

An intention in goal model building blocks could be a leaf, a main goal, or neither. In the latter case, the intention is located somewhere in the middle of the hierarchical goal model. In the GoPF framework, all intentions are directly or indirectly connected to leaf intentions. Therefore, goal models are well-formed only when intentions have at least one leaf from the same goal model building block that directly or indirectly contributes to their satisfaction. The FMM lays down the foundation for creating goal models within the goal model building blocks but it does not enforce the above restrictions on intentions. Ensuring the well-formedness of goal model building blocks requires the presence of the following OCL invariants, which must evaluate to true for all the intentions in the goal model building blocks.

```
context Intention
inv NoDanglingMiddleIntention:
      not (self.leaf or self.mainGoal)
      implies
      ( DownIntentionSetFromSelf() -> exists(i|i.leaf) )
```

In order to apply this invariant, two OCL operations on intentions are defined in Table 3.

Table 3 OCL operations for retrieving a set of intentions

```
DownIntentionSet(s:Set(Intention)):Set(Intention) =
       if s ->includesAll(s.ELt.fromLink()->asSet())
       then s
       else DownIntentionSet(s->union(s.ELt.fromLink()->asSet()))
       endif

DownIntentionSetFromSelf():Set(Intention) = DownIntentionSet (Set{self})
```

DownIntentionSet is a recursive OCL operation in charge of finding and returning a final set of intentions, which are directly or indirectly contributing to the satisfaction of an initial set of intentions passed as a parameter. To achieve this, DownIntentionSet takes a set of intentions, finds the other intentions that are contributing to their satisfaction, adds them to the set it received as parameter, and re-invokes the operation with the new set. DownIntentionSet continues to execute until all the intentions on the downside of the original set in the goal model (those that are contributing to the set) are discovered and included. Finally, the resulting set is returned.

DownIntentionSetFromSelf operates on an intention and returns a set of intentions from the same goal model building block that directly or indirectly contribute to the satisfaction of this intention. For this purpose, DownIntentionSetFromSelf creates the initial set with the intention that it operates on as the only member. It then passes the initial set to DownIntentionSet and returns the resulting set received from DownIntentionSet.

NoDanglingMiddleIntention is the OCL invariant enforcing the well-formedness of pattern families by preventing goal model building blocks from containing dangling intentions. For this purpose, this invariant must evaluate to true for every intention of all goal model building blocks. When encountered by an intention that is not a leaf or a main goal, the DownIntentionSetFromSelf operation invoked by this invariant finds the set of intentions that are contributing to that intention. The invariant returns true only if the intention is directly or indirectly connected to at least one leaf goal. In other words, this invariant indicates that no dangling intention exists in the goal model building blocks.

Figure 19 (a) shows the result evaluation of the NoDanglingMiddleIntention OCL constraint against the sample patient safety pattern family presented in Figure 18. This constraint is evaluated to true because the pattern family is well formed. On the contrary, the pattern family presented in Figure 20 includes a dangling intention. In this PF, Dangling Intention (X) is included in the goal model building block of Take Action (p4). It is a dangling intention because it is not a mainGoal or a leaf and yet none of the leaves contributes to it. Consequently, as can be seen in Figure 19 (c), NoDanglingMiddleIntention is correctly evaluated to false, hence expressing the existence of a dangling intention in the pattern family. As illustrated in Figure 21 (b), it is possible to investigate violations to NoDanglingMiddleIntention for finding the exact dangling intentions and take relevant actions for removing them.

3.4 Summary

This chapter described foundational elements of the GoPF framework, together with a metamodel that formalizes the concepts related to families of patterns (PFs). URN already supports some of the basic concepts of GoPF. To further benefit from the standard language, supporting concepts, and existing tools, URN is profiled to formalize FMM. In particular, the concepts for which URN did not have a direct equivalent have been

mapped to URN concepts, but with stereotypes captures as metadata. URN links are used to capture new associations in the metamodel. Finally, OCL constraints are introduced to enforce well-formedness of FMM-based pattern families as well as the stricter association multiplicities found in the framework metamodel.

The next section is concerned with a method for creating pattern families, which are instances of the metamodel just presented.

Chapter 4. BUILDING PATTERNS AND PATTERN FAMILIES

This chapter represents how FMM-based patterns and families are created for a particular domain. This is done by *building patterns* and going through *family creation*. Figure 22 gives an overview of the process used to create, evolve, and apply a PF. Note that many UCMs in this chapter actually describe how to use the GoPF framework rather than artifacts (patterns) produced by GoPF.

In order to build patterns, recurring problems and solutions must be located. Locating recurrences takes place by analyzing goal and business process models for organizations in a particular domain as well as by interviewing stakeholders and domain experts. Using these means helps to reveal the recurring problems and solutions, which together are used for building patterns. These patterns are then organized in a new PF or used to evolve an existing one. Then the new or evolved PF can be used for customizing and extracting models for a particular organization. This chapter focuses on locating recurrences and on building patterns and families.

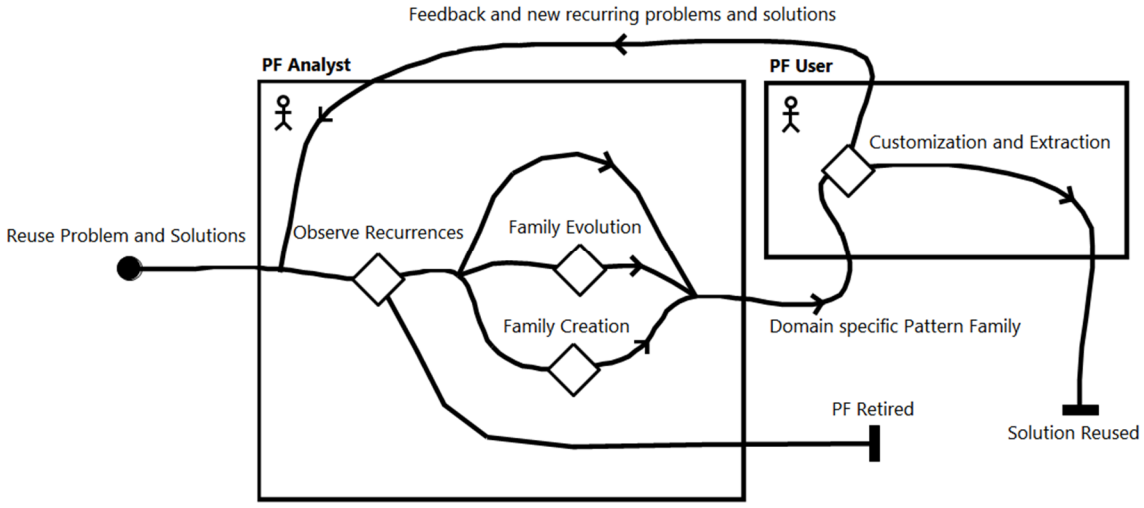

Figure 22 Overview of a process for creating, evolving and applying PFs

4.1 Building Patterns

In order to create a pattern family, the PF analyst must first form patterns by locating recurrent problems and recurrent solutions (best practices) for each pattern. These problems and solutions are those that can be reused to model the business goals and business

processes in a hierarchical fashion (this approach can be taken recursively to refine the solutions).

4.1.1 Locating Recurring Problems

The PF analyst has three means for finding the recurring problems, as described in Figure 23. First, she considers a problem as recurring when it has been observed repeatedly in the requirements models over time and at different organizations (or units). A problem is also recurring if similar types of domain stakeholders in different organization are highlighting it as an issue to be addressed. Finally, domain experts with a good understanding of the problems in the domain can reveal recurring problems.

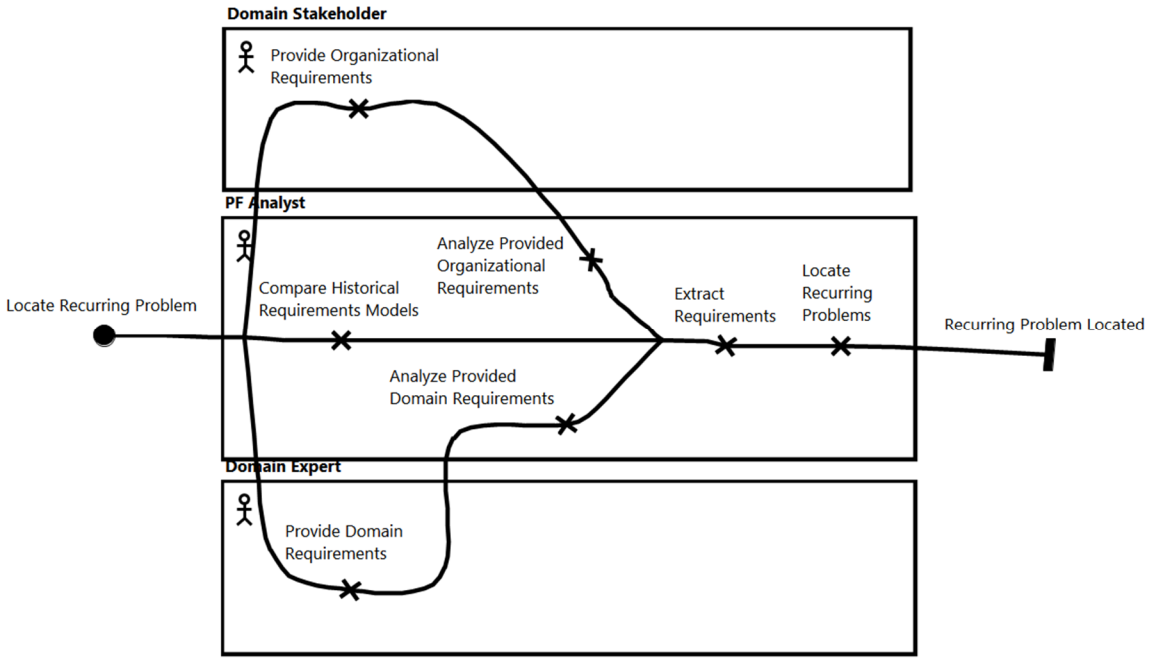

Figure 23 Locate a recurring problem in a particular domain

When a recurring problem is located, the PF analyst captures pieces of the goal model containing the corresponding knowledge in the form of a goal model building block. Such block includes a *main goal* representing a recurring problem as well as recurring intentions that may contribute to its satisfaction (see Figure 24, where side effects and conditions have been hidden for simplicity). Figure 25 represents an extracted goal model building block that includes Minimize Costs of Infrastructure as its side-effect intention as well as Healthcare Procedures Are Complicated as its condition. Each goal model building block represents one stakeholder's problem together with elements of solution.

It is important to note that goal model building blocks are at different levels of abstraction. This allows for contributing intentions in one block be considered as the main goals in other blocks. For instance, as shown in Figure 24, Collect Data is part of the solution for Increase Patient Safety whereas it becomes the main goal of the Data Collection goal model building block, where contributing intentions to Collect Data are captured. In such a hierarchy, higher-level goals are more abstract and intentional. They are consequently satisfied by those goals that are less abstract and more operational. The lowest-level goals typically represent concrete choices available to an organization. Using these goal models for extending an organizational goal model helps shaping and designing the required business processes along the way.

The UCM shown in Figure 26 gives an overview of the steps a PF analyst must carry out for building pattern and families. This process includes the *locating recurring problem* stub that invokes the process explained in this section (Figure 23) and uses the located recurring problem for creating a goal model building block. As can be seen in this UCM, recurring problem locations and the consequent creation of goal model building blocks are iterative activities that include interactions with stakeholders and experts in the domain.

4.1.2 Locating Recurring Solutions

The PF analyst locates recurring solutions for the problems found in the previous step. These solutions are the best practices (e.g., workflows, procedures, protocols, etc.) used to address a recurring issue or requirements in the domain. Similar to recurring problems, there are three important means for PF analysts to locate such recurrences. First, the PF analyst considers a solution as recurring when previous observations show its common use for addressing a problem. A solution is also considered as recurring when similar types of stakeholders in different organizations consider it as a means of addressing a common problem in their domain. Finally, domain experts who know about common problems can highlight the recurring solutions that are practiced by organizations in that domain. As illustrated in Figure 27, interviewing domain stakeholders and domain experts and analyzing the existing process models enable PF analyst to use the aforementioned sources of knowledge for locating recurring solutions.

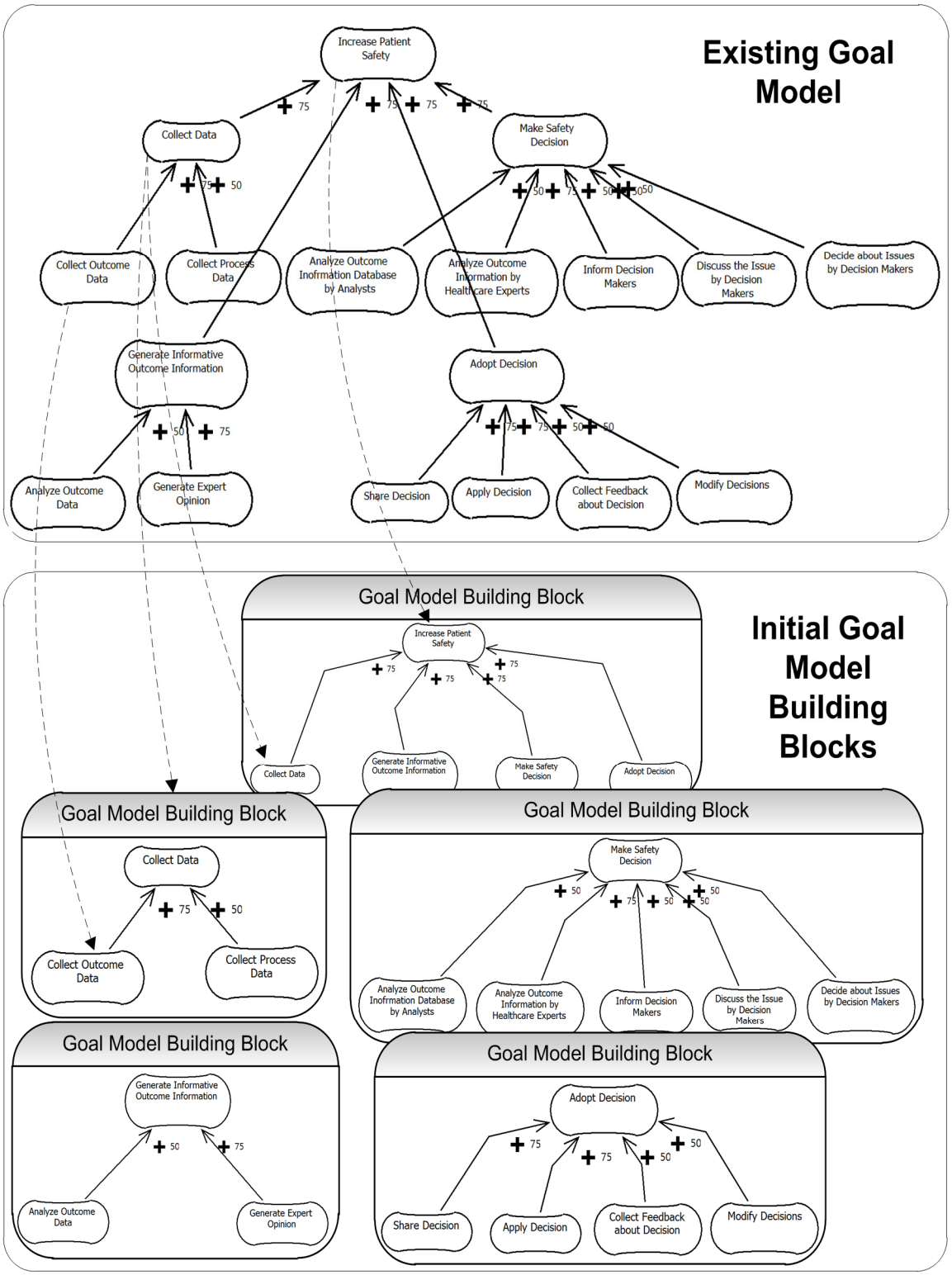

Figure 24 Extracting goal model building blocks

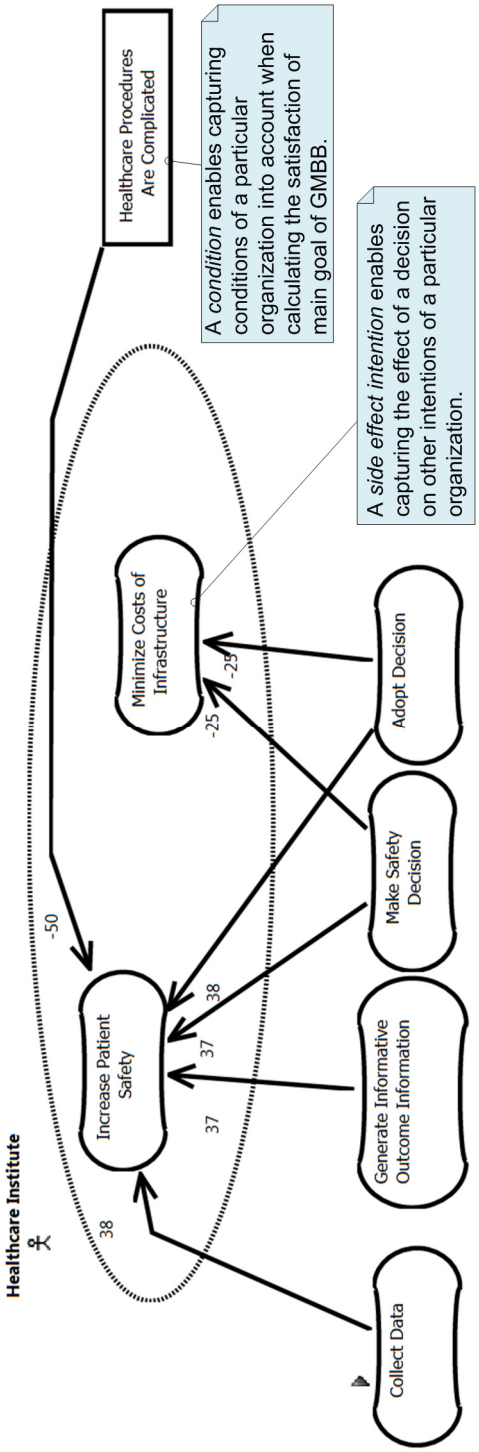

Figure 25 Example of an extracted goal model building block

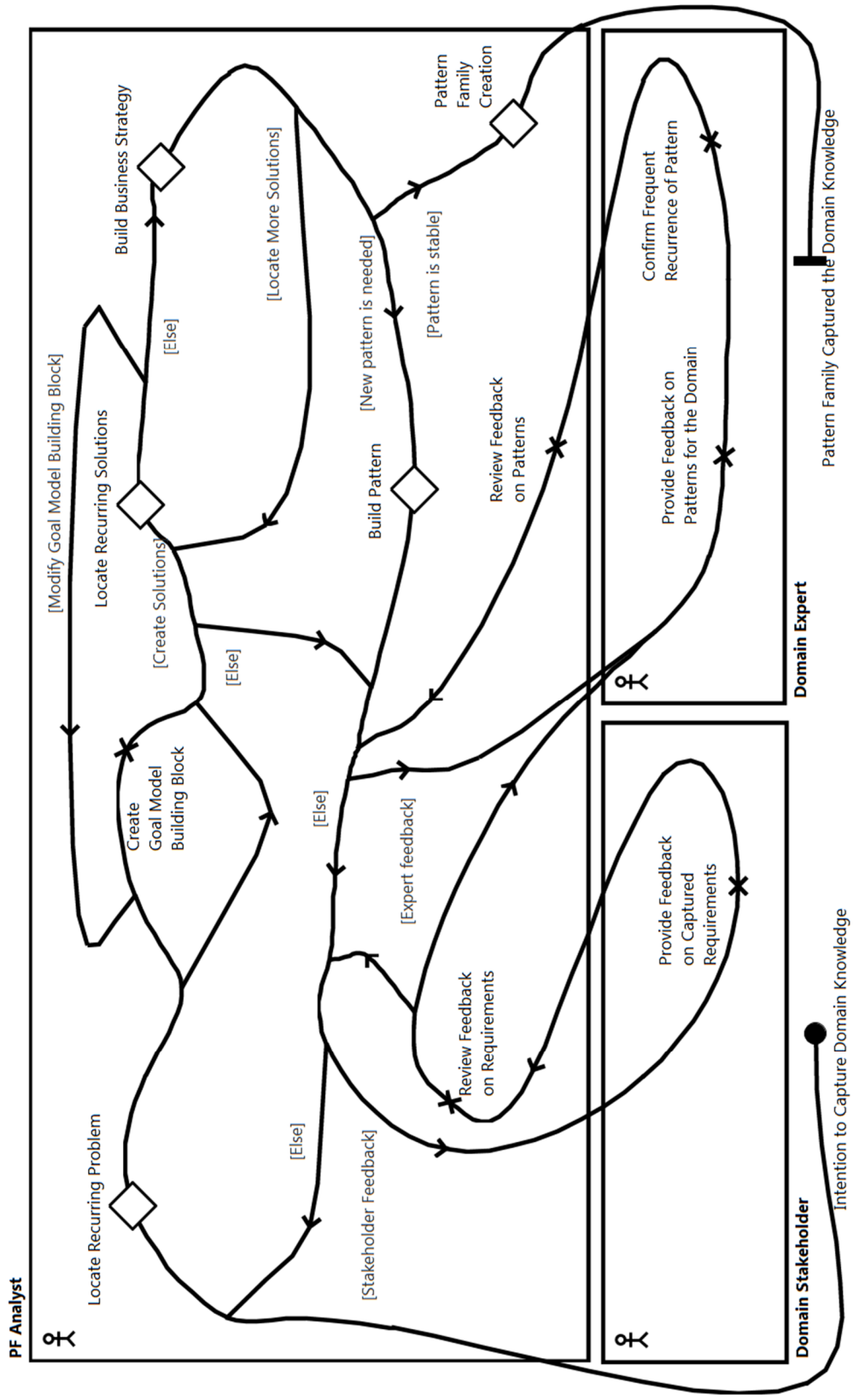

Figure 26 Process of building patterns and pattern families

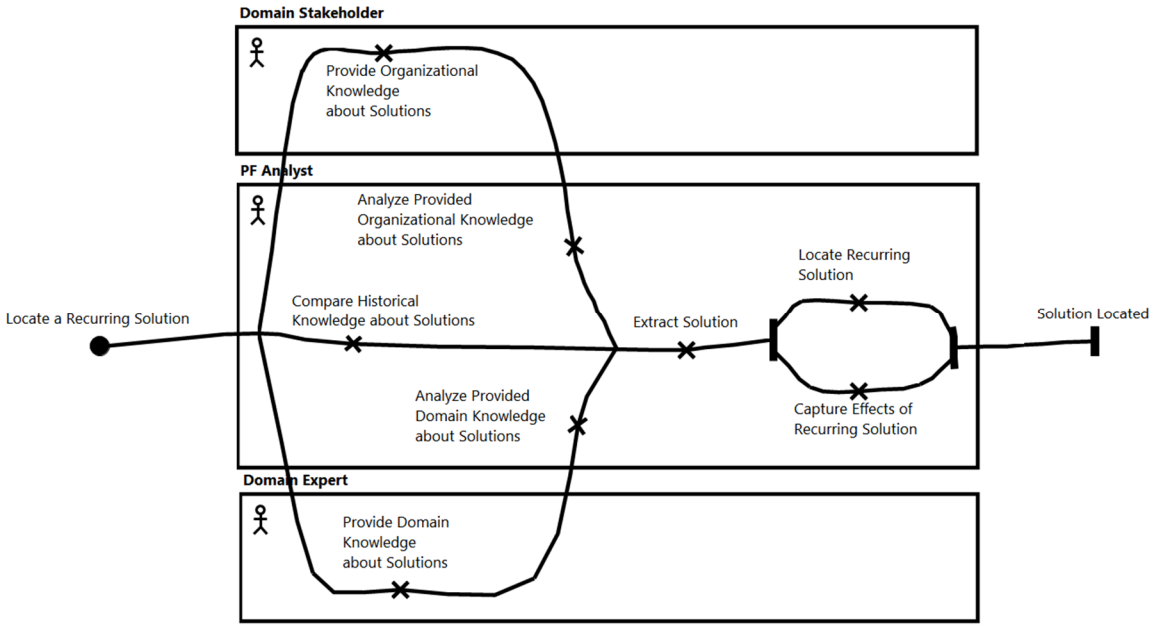

Figure 27 Locate a recurring solution in a particular domain

This approach helps the PF analyst to locate well-known recurring solutions for the same recurring problem. Each of these solutions is captured as a business process composed of elements, e.g., sub-processes, which are contributing to the main goal of the goal model building block representing the recurring problem. In other words, each solution represents a process implying an arrangement of contributing elements for addressing a common problem. Ideally, these elements have a counterpart, i.e., a contributing intention, in the captured goal model building block representing to the main goal. However, in practice this can only be achieved through a series of iterations as illustrated in Figure 26. While finding these elements and their arrangements, the PF analyst also captures the effects of each solution on the counterpart intentions of the respective goal model building block.

The recurring solutions and their effects are captured in the form of pairs of business process building block and evaluation strategy. Each business process building block is a UCM map that represents *how* a goal is satisfied and provides the ordering of elements of solutions, i.e., stubs and responsibilities. The bottom half of Figure 29 shows collections of business process building blocks, in which each member provides one possible way of fulfilling a goal shared by all members. Although all of these business process building blocks address a common goal, there are variations among the members of

the collection that may lead to differences in the quality of achieving the common goal and in the conditions associated with a member. The PF analyst uses URN-based evaluation strategies for capturing these differences. Consequently, she pairs a business process building block and the relevant evaluation strategy to form business strategies. Figure 28 illustrates the sub-process for building business strategies, which is invoked by the main UCM for building patterns and families presented Figure 26.

Figure 30 (a) shows the goal model building block for the Increase Patient Safety pattern and two related business strategies. For instance, the first business strategy, depicted in Figure 30 (b), captures a recurring solution commonly used by small healthcare organizations or units. Its business process building block shows the arrangement of sub-processes, i.e., Collecting Data and Generating Informative Outcome Information. The corresponding evaluation strategy shows that Collect Data and Generate Informative Outcome Information are the contributing elements of the goal model building block affected by this solution. The second business strategy in Figure 30 (c) represents a different solution and its corresponding effects. This solution is comprehensive compared the first solution because not only it collects data and generates information but it also uses them systematically to improve the procedures in a healthcare institute.

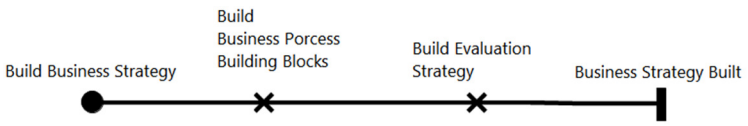

Figure 28 Build business strategy

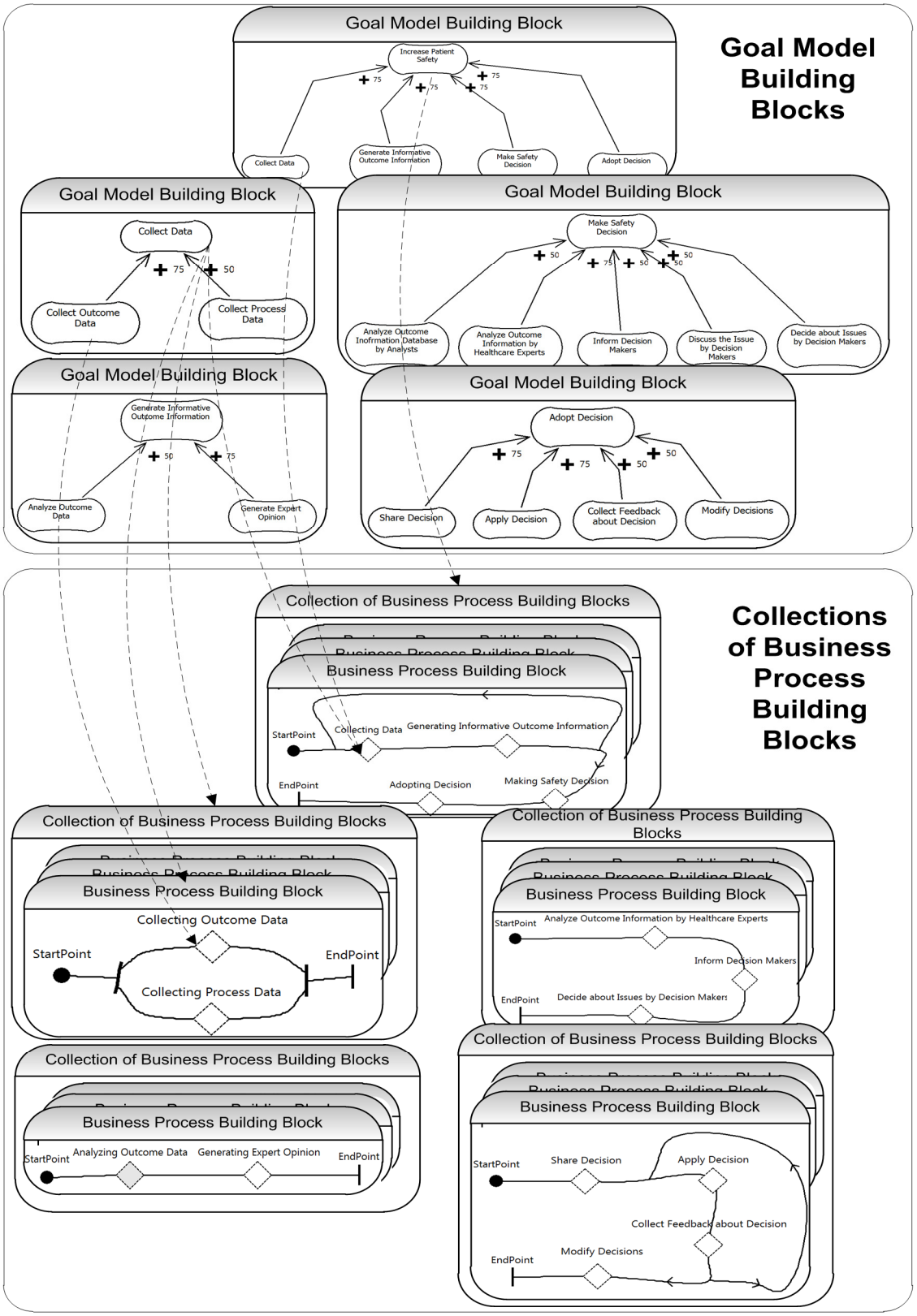

Figure 29 Collecting business process building blocks that address the problems in goal model building blocks

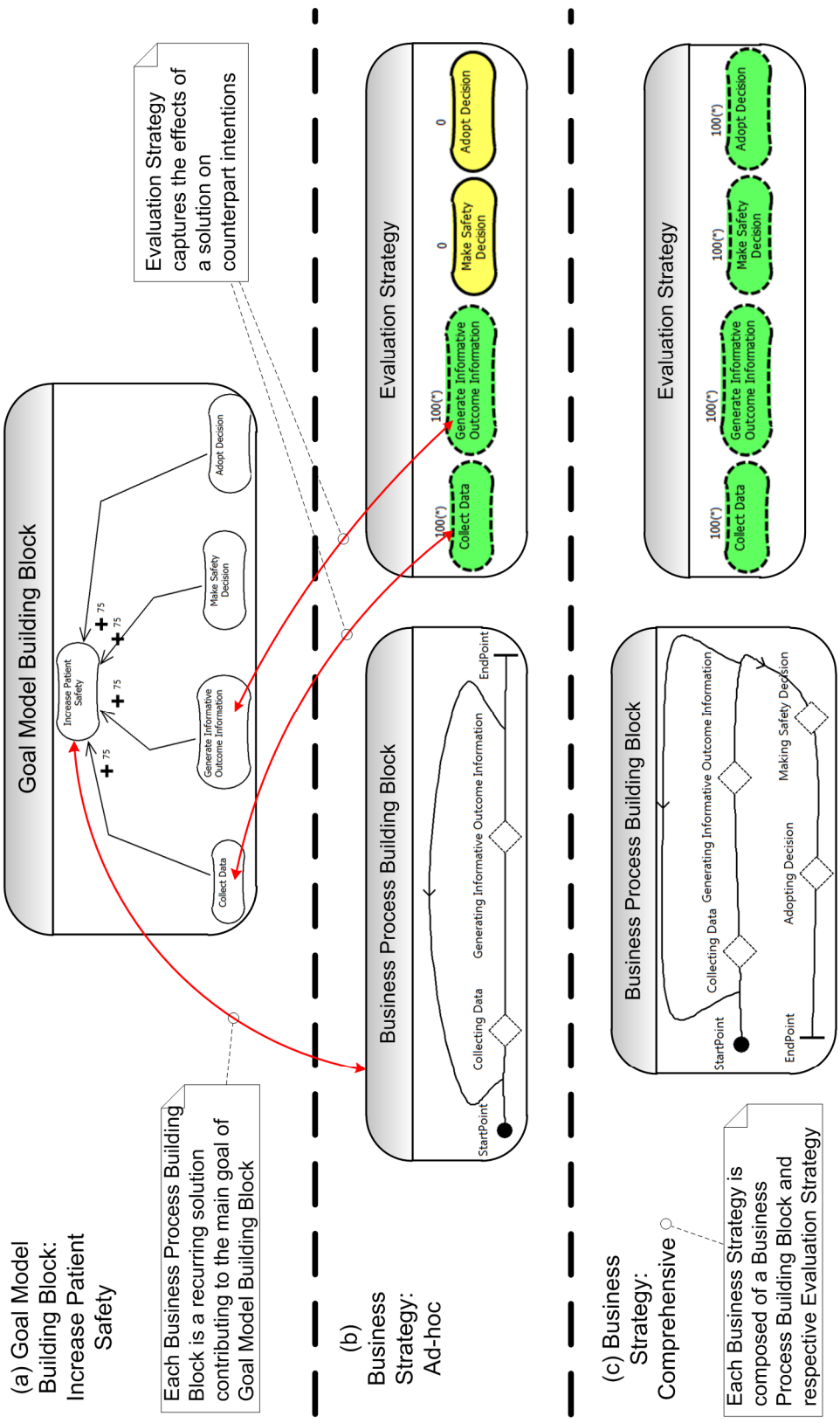

Figure 30 Goal model building block and business strategies for Increase Patient Safety Pattern

4.1.3 Forming Patterns

After locating recurring problems and solutions, the PF analyst combines a goal model building block together with the corresponding business strategies and forms a pattern. Figure 31 describes the approach for forming an FMM-based pattern. Figure 30 illustrates composing elements of the Increase Patient Safety pattern.

As shown in Figure 26, building patterns is carried out iteratively by invoking the sub-processes found in Figure 31. The optional paths in Figure 31 enable the gradual development of a pattern, when new knowledge discovered through interactions with stakeholders or experts can be combined with an existing pattern. For instance, the Increase Patient Safety pattern may be first built by combining one goal model building block (Figure 30 (a)), and one business strategy for ad-hoc improvement (Figure 30 (b)). In a subsequent iteration, another business strategy (Figure 30 (c)) is added when interactions with domain experts highlight the comprehensive approach for increasing patient safety.

In addition, realization links are established between the goals in the goal model building blocks and the elements in the business process building blocks. The patterns are then used as input for the family creation method.

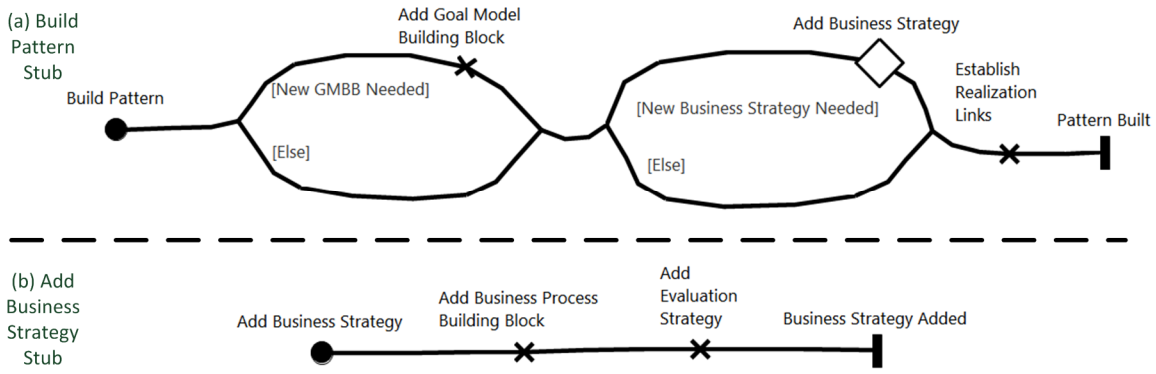

Figure 31 Forming a pattern using the components built when locating recurring problems and solutions

4.2 Family Creation

Family creation is a method for building pattern families. Family creation is a special case of family evolution of an initially empty PF by using the extension algorithm (explained in the next chapter). As illustrated in Figure 26, after patterns are built using the

located recurring problems and solutions, they are used as input for building a pattern family. If no pattern family exists for the domain, a new pattern family is built by evolving it from an empty one (family creation). On the other hand, if a PF for the domain already exists, instead of creating a new pattern family, the captured patterns are used for evolving the existing family.

Figure 32 shows this process and highlights its iterative nature. During the process of building family by evolutionary mechanisms, refinement links are being established to capture the refining relationship between a new pattern and existing patterns in a family. Section 5.2.3 on page 79 shows how the extension algorithm is used to add a pattern to an empty PF.

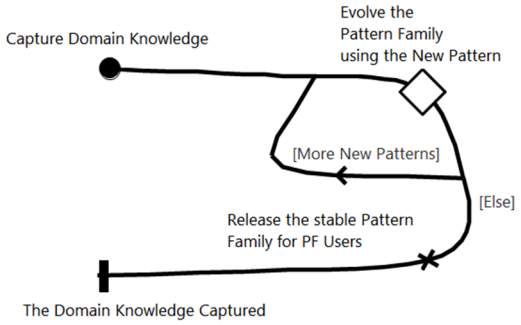

Figure 32 Add a newly built pattern to a pattern family

4.3 Case Study

This section illustrates how a PF analyst locates recurrences to form patterns.

4.3.1 Locating Recurrences: Example 1

The PF analyst analyzes the goal models and business process models created in the field of patient safety. The top of Figure 24 shows part of the goal model that was created by observing different departments (Cardiac Surgery Intensive Care, Intensive Care, and General Internal Medicine) of a real teaching hospital in Ontario between September 2008 and September 2009. Increase Patient Safety is an abstract, recurring requirement in different hospital departments and in other healthcare institutions. The bottom half the figure represents the recurrent pieces of goal models in the form goal model building blocks. Figure 33 shows the Increase Patient Safety goal model building blocks in more details, including the contributions of Collect Data, Generate Informative Outcome In-

formation, Make Decision, and Apply Knowledge to the realization of Increase Patient Safety and all side-effects (on quality and cost) as well as dependencies (on advanced infrastructure).

Two strategies have been located in the business process models that correspond to this goal model building block (see Figure 34). The first strategy (A) includes only the sub-goals Collect Data and Generate Informative Outcome Information and is described in more detail by the top business process building block in Figure 34. The business process building block shows that collecting data occurs before generating information. The second strategy (B) includes also the two other sub-goals, namely Make Decision and Apply Knowledge, and adds these activities to its business process building block (bottom of Figure 34).

Then Collect Data, Generate Informative Outcome Information, Make Decision, and Apply Knowledge are respectively linked (with URN links) to the Collecting Data, Generating Informative Outcome Information, Making Safety Decision, and Adopting Decision stubs in both business process building blocks in Figure 34. All GRL and UCM models linked together constitute the Increase Patient Safety pattern. Figure 40 illustrates this pattern. The created pattern is then used as the input for creating a new PF by extending an initially empty PF (an approach explained in detail in the next chapter).

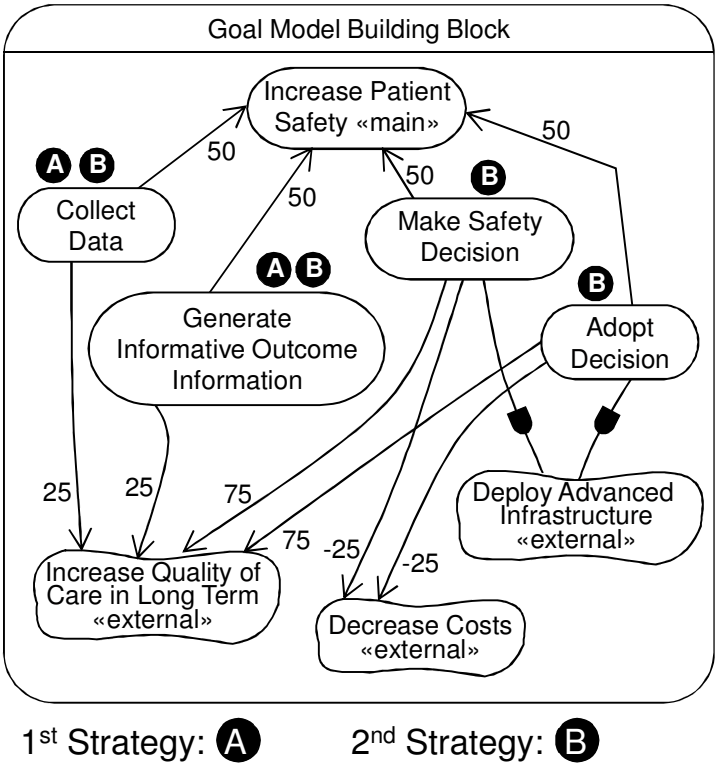

Figure 33 Goal model building block for Increase Patient Safety

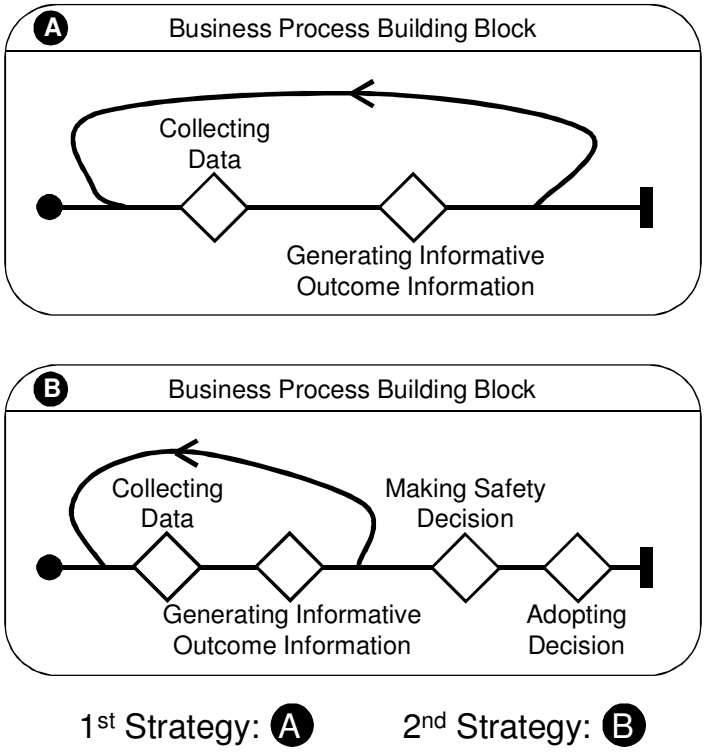

Figure 34 Business process building blocks for Increase Patient Safety

4.3.2 Locating Recurrences: Example 2

The satisfaction of the four sub-goals of goal model building blocks in the Increase Patient Safety pattern (Figure 33) can in turn be described in another layer of more detailed patterns. This section briefly illustrates the creation of a pattern that refines Collect Data.

Figure 35 presents such a goal model building block, located by the PF analyst. The Collect Data goal requires two sub-goals of its own (Collect Outcome Data and Collect Process Data). Figure 36 represents the three corresponding strategies (C, D, and E, i.e., one, or the other, or both in parallel). Then, Collect Outcome Data and Collect Process Data are linked to the Collecting Outcome Data and Collecting Process Data stubs in the business process building blocks (i.e., UCMs in Figure 36). Finally, this goal model building block together with the business process building blocks form the Collect Data pattern.

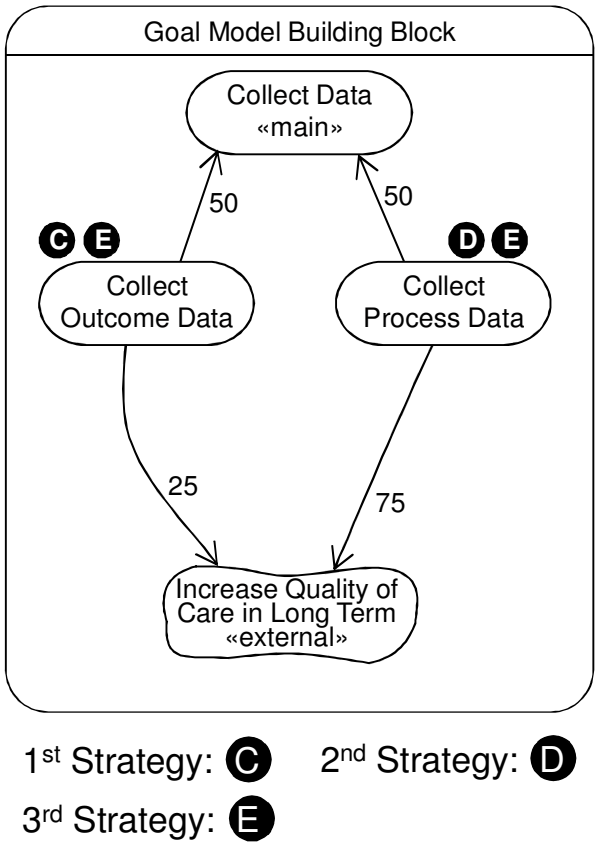

Figure 35 Goal model building block for Collect Data

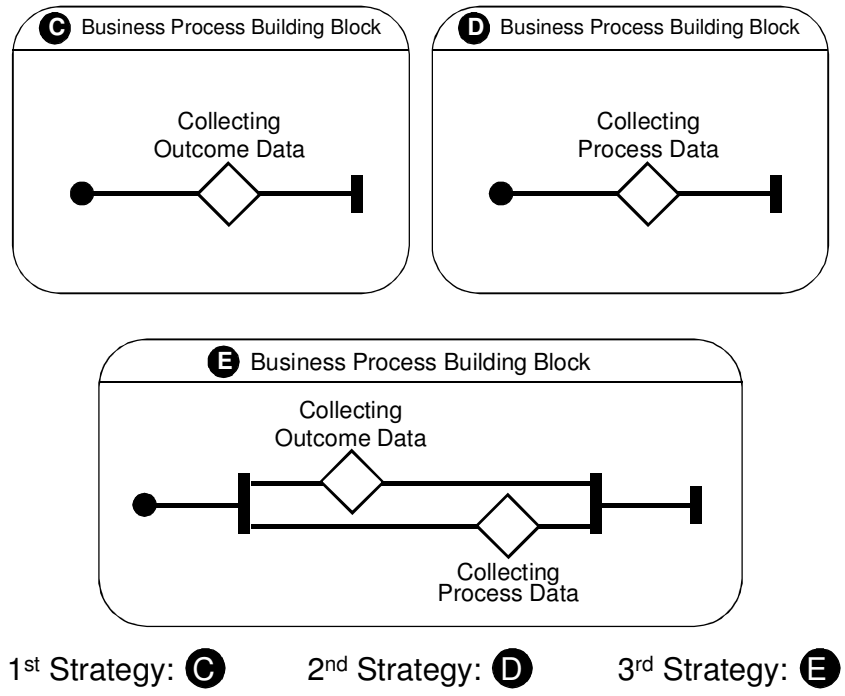

Figure 36 Business process building blocks for Collect Data

In addition, the PF analyst locates the refinement link between the Collect Data in the Increase Patient Safety pattern and the Collect Data pattern. This will be used when preparing the inputs for the extension algorithm so the refinement relationship will be established in the PF.

4.4 Summary

This chapter explained how the PF analyst builds patterns by locating recurring problems and solutions within a domain. A pattern that captures this knowledge contains a goal model building block, business strategies (i.e., pairs of business process building block and evaluation strategy), and realization links between business goals and business processes that loosely couple goals in the goal model building block with model elements in the business process building blocks.

This chapter also illustrated how the PF analyst generates patterns by locating the recurrent pieces of goal models and business processes models in patient safety and consequently uses these patterns as inputs of evolution mechanisms for creating a PF. Such mechanisms, which can also be used to create a first PF from a collection of patterns, are discussed in the next chapter.

Chapter 5. PATTERN FAMILY EVOLUTION

The Family Evolution method is comprised of a collection of solutions for maintaining the quality and accuracy of patterns in a pattern family. This is an important part of the Family Development Method (FDM) that enables ongoing development and improvement of a pattern family over their life span, especially as pattern families and patterns need to adapt to changes in the business domain.

5.1 Motivation and Overview

A Pattern Family (PF) captures the knowledge about recurring solutions that answer recurring problems in a specific context within a particular domain. When a PF is created for a domain, it can be used by another organization in that domain for finding and reusing solutions to a known problem that the particular organization is facing. This is done through customizing and extending solutions by "applying" a PF (with the OCEM method), as will be discussed in Chapter 6.

However, stakeholders' requirements in any domain are dynamic and constantly changing, which in turn leads to emerging new problems that must be addressed. Constant updates to requirements are caused by changes within the domain and outside of the domain. For instance, when a government introduces new legislation to improve patient safety in the healthcare domain (e.g., through stricter reporting of C.difficile cases and of other infections [131]), healthcare organizations must change their process to comply. Consequently, this compliance can affect the current way of dealing with a particular problem in the domain. Furthermore, when new successful practices gain recognition in the healthcare domain, stakeholders of other healthcare organizations will ask for these new practices to be integrated to their own. Thus in the long term, a PF can remain useful only if it can be evolved to comply with ongoing changes.

In the GoPF framework, the PF analyst is a domain-specialized modeler who is interested in creating and evolving a PF for a particular domain. The PF analyst observes the goal models and business processes of the organizations in the domain, and inter-

views with domain stakeholders as well as with domain experts in order to discover recurring problems and solutions. Figure 23 on page 59 and Figure 27 on page 64 represent the location of recurring problems and solutions, respectively. Figure 26 on page 63 illustrate how located problems and solutions are used for building patterns and families. When new observed patterns are related to a particular PF, the Family Evolution method will help maintaining this PF. On the other hand, when the PF analyst observes many new and unrelated patterns, then creating a new PF may be considered. Creating PFs is a special case of evolution in which Family Evolution mechanisms are used repeatedly on an initially empty PF.

Extension, *modification*, *elimination*, and *combination* are four types of evolution mechanisms (Figure 37) that help keep the patterns in a PF up-to-date [132][133]. They maintain the usefulness of a PF by increasing the quality and accuracy of its patterns and their interrelationships. They can address current problems and solutions that stakeholders within the domain are facing. These mechanisms respectively change a PF by (i) adding a new pattern, (ii) modifying a current pattern, (iii) eliminating an obsolete pattern, and (iv) combining two PFs that represent problems of the same domain. Each of these mechanisms keeps the integrity of the changing PF in addition to evolving individual patterns. Although the extension and elimination mechanisms would be functionally sufficient by themselves to maintain PFs, the modification and combination mechanisms provide additional and much needed usability, especially for micro-evolutions (through modifications addressing fine-grained changes) and macro-evolutions (through combinations addressing large-scale changes).

Gradual changes of patterns in a PF in response to changes in a domain resembles the concept of evolution in biology. However, changes in the GoPF framework are enforced by analysts based on their understanding of the domain, which differs from the source of changes in biology where changes are mutations providing survival advantages. Evolution in this thesis actually refers to adaptive maintenance activities in conventional software evolution, which are (manual) modifications of a software product (or pattern) performed after delivery to keep it usable in a changed or changing environment. Evolution here is therefore not related to biology or to automatic evolutionary algorithms from the artificial intelligence community.

A Java program was created to implement these algorithms and test these four evolution mechanisms. This program, discussed further in Chapter 7, demonstrates the feasibility of the algorithms, but it does not include advanced features such as rollback in case of errors. In addition, the performance of the program was not considered, as usually modification to pattern families (even complex ones) can be handled in less than a second.

The following four sections of this chapter provide detailed algorithms for each evolution mechanism.

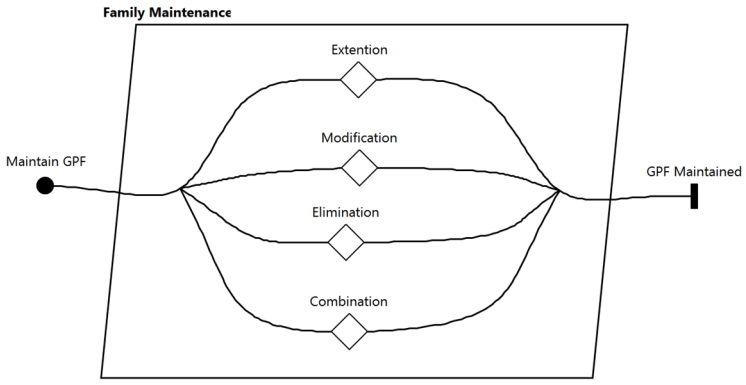

Figure 37 Evolution UCM

5.2 Extension Mechanism

When observing organizations and their processes reveals that a relevant pattern is not included in a PF, the Extension Mechanism helps the PF analyst add the new pattern to the PF and integrate it with the existing patterns within the family. The Extension Mechanism is composed of three major steps (Figure 38). First, it modifies those patterns that are affected by the new pattern, then it adds the new pattern to the PF, and, finally, it connects the new pattern to related patterns. In the latter step, all the patterns that are refined by the new pattern are first connected to the extending pattern and then the new pattern is also connected to those patterns that refine it.

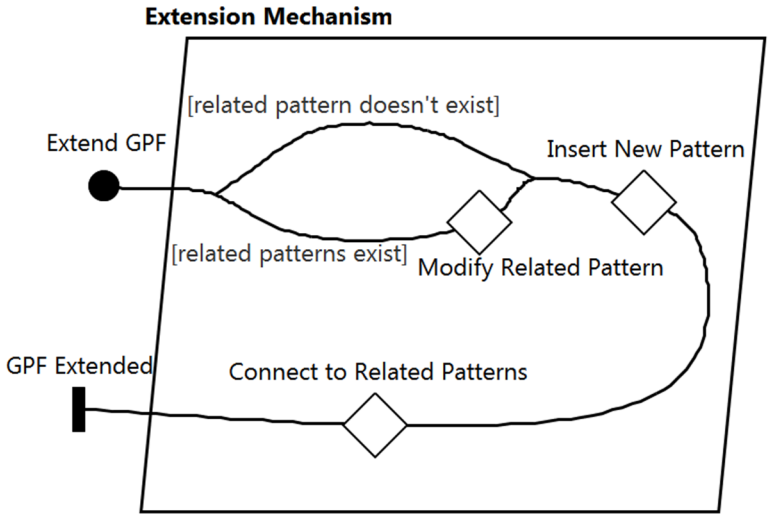

Figure 38 Main steps of the Extension Mechanism

In the following, the extension algorithm is presented, described, and finally illustrated through two examples.

5.2.1 Extension Algorithm

Algorithm 1 provides inputs, outputs, and steps of the extension algorithm. Algorithm keywords and constants are in **boldface** whereas types (from the metamodel) are *italicized*. Comments are shown between /* and */.

Inputs of the extension algorithm
I1. pf :*PF* /* initial pattern family */
I2. xp:*Pattern* /* the new extension pattern */
I3. modifications: **ordered set of** (rp:*Pattern*, link:*ElementLink*, action ∈ {*#Add, #Delete*},
bst:*BusinessStrategy*, oldbst:*BusinessStrategy*)
where link.toLinks.isEqualTo(rp.mainGoal()) ∧
(action == *#Add* ⇒ link.fromLinks.isEqualTo(xp.GMBB.mainGoal()))

Output of the extension algorithm
O1. **modified** pf:*PF* /* the original pattern family extended with xp */

Precondition and Post-conditions of the extension algorithm
Pre 1: modifications->**forAll**(m| rp.GMBB.intention->**exists**(i|i.isEqualTo(m.link.toLink)))
Post 1: modifications->**forAll**(m|m.action == *#Add* **implies** rp.GMBB.elementLink->**includes**(m.link) **and**
rp.GMBB.intention->**exists**(i|i.isEqualTo(m.link.fromLink)))
Post 2: modifications->**forAll**(m|m.action == *#Delete* **implies** rp.GMBB.elementLink->**excludes**(m.link))
Post 3: modifications ->**forAll**(rp|rp.GMBB.leafCollection()->**exists**(i|i.isEqualTo(xp.GMBB.mainGoal())) **or**
xp.GMBB.leafCollection() -> **exists**
(i|i.isEqualTo(rp.GMBB.mainGoal())))

Steps of the extension algorithm
S1. mg:*Intention* = xp.GMBB.mainGoal() /* mg is the main Intention of xp */
S2. **if** (**not** modifications.isEmpty()) **then**
S3. modification (pf, modifications) /* related patterns are modified by using the modification mechanism*/
S4. **endif**
S5. pf.insert (xp) /* xp is added to pf */

```
S6.  relatedIntentions: set of (ri:Intention ) where ∃ op ∈ pf.patternCollection() ∧
                                    not op.isEqualTo(xp) ∧ ri ∈ op.GMBB.leafCollection() ∧ ri.isEqualTo(mg)
     /* relatedIntentions represents those intentions in other patterns which are refined by xp */
S7.  leafIntentions: set of (Intention) = xp.GMBB.leafCollection()
S8.  foreach (i:Intention in relatedIntentions)
     S8.1.i.patternDef = xp
S9.  foreach (i:Intention in leafIntentions)
     S9.1.   foreach (p:Pattern  in pf.patternCollection()  where not p.isEqualTo(xp))
          9.1.1. if (i.isEqualTo(p.GMBB.mainGoal())) then
          9.1.2.   i.patternDef = p
          9.1.3. endif
```

Algorithm 1. Extension of PF

5.2.2 Applying the Extension Algorithm

Once observed recurrences highlight the need for adding a new pattern, the PF analyst prepares the inputs of the extension algorithm before using it.

This algorithm takes three inputs: pf is an initial pattern family, xp is a pattern used to extend the initial PF, and the modifications set (Table 4) highlights the effects that extending pf with xp has on other related patterns of the family. The PF analyst prepares the second and third inputs based on recurrences. As different types of modifications may be necessary, the second, forth, and fifth elements of the modifications set may be **null**. The precondition (Pre 1) ensures that the toLinks side of the link must always point to the main goal of rp. If the action is *#Add*, after the execution of the algorithm, the relevant pattern must include the link as well as fromLinks intention (post-condition 1). On the other hand, if the action is *#Delete*, after the execution of the algorithm, the relevant pattern must exclude the link (post-condition 2). Post-condition 3 limits the usage of the modifications only to patterns that are refined by or that refine the extension pattern (xp). These precondition and post-conditions prevent using the extension algorithm for merely changing an unrelated pattern in pf. Isolated modifications of patterns must use the modification algorithm, which is described in section 5.3.

Table 4 Elements of the modifications ordered set in the extension algorithm

Element	Description
rp	A related pattern that is affected and must be modified
link	A link between two intentions that highlights the part of the goal model building block that must be modified
action	An indicator of what must be done to the link
bst	A new business strategy that represents a new solution
oldbst	An old business strategy that must be eliminated from rp

Steps S1 to S9 carry out the three major activities illustrated in Figure 38. Step S1 initializes mg with the main goal of xp's goal model building block. Steps S2 to S4 take the modifications set and invokes the modification algorithm (Algorithm 2) to modify related patterns in pf. This captures the possible effects of adding xp on other patterns in PF. In step S5, the new pattern, xp, is added to pf. Although this is conceptually a simple insertion of a pattern, it is in fact an in-depth copy in which every element of xp is copied into pf. In step S6, a set of leaf intentions (without any incoming links) of other patterns in pf that are equal to the main goal of xp is assigned to relatedIntentions. This is the set of intentions that are refined by xp. In S7, the set of leaf intentions of xp's goal model building block is assigned to leafIntentions. Elements of leafIntentions may be refined by other patterns in pf. In step S8, the related intentions are linked to xp by assigning xp to patternDef of intentions in relatedIntentions. Finally, in step S9, intentions of xp's goal model building block that can be refined with other patterns in pf are linked to the appropriate pattern by setting their patternDef accordingly.

5.2.3 Example 1: Extension of an Empty PF

Extending an empty PF is a special case of extension that initializes a new PF. This section illustrates how to use the extension algorithm for creating a new PF in the patient safety domain. When observing the patterns underscores the need for a new pattern family, the PF analyst creates the inputs for the extension algorithm: pf is an empty PF (I1), xp is the new pattern recognized that must be added to pf (I2), and modifications is an empty set because no pattern in pf needs to be modified (I3).

The top half of Figure 39 represents the goal model building block and business processes templates of the Increase Patient Safety pattern. The goal model building block (in GRL form) depicts a recurring problem (Increase Patient Safety) and elements of solutions that influence its satisfaction (Collect Data, Generate Informative Outcome Information, Make Safety Decision, and Adopt Decision). Business process building blocks (in UCM form) capture the process of achieving Increase Patient Safety. In the bottom half of the Figure 39, the business process building blocks of two strategies in the pattern are provided. *Strategy A* represents a solution in which Collecting Data and Generating Informative Outcome Information increase the patient safety by ad hoc improvement of process brought to light by the collected data and the processing of such data.

Strategy B, on the other hand, uses Make Safety Decision and Adopt Decision in addition to the other two elements. Consequently, it improves the quality of care by systematically changing the underlying procedures in the hospital.

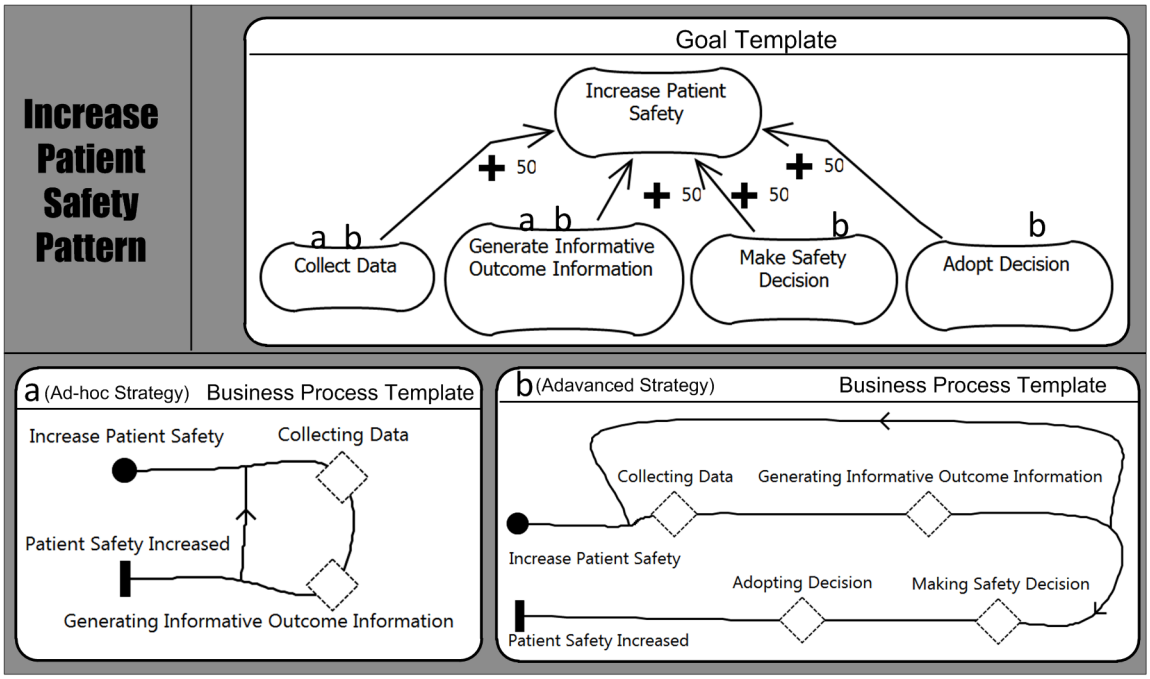

Figure 39 Top: goal model building block of xp - Bottom: business process building blocks of xp

Figure 40 shows the Increase Patient Safety pattern as a UML object diagram instantiating the Family Metamodel (FMM). In order to increase readability of the diagram, objects of the ElementLink class in the metamodel are not shown. Furthermore, to make the diagram more compact, I, C, G, M, and A are used respectively to identify the Increase Patient Safety, Collect Data, Generate Informative Outcome Information, Make Decision, and Adopt Decision intentions. Although the name attribute of these instances denotes the complete name of the objects, this attribute is hidden on large diagrams in the rest of this thesis.

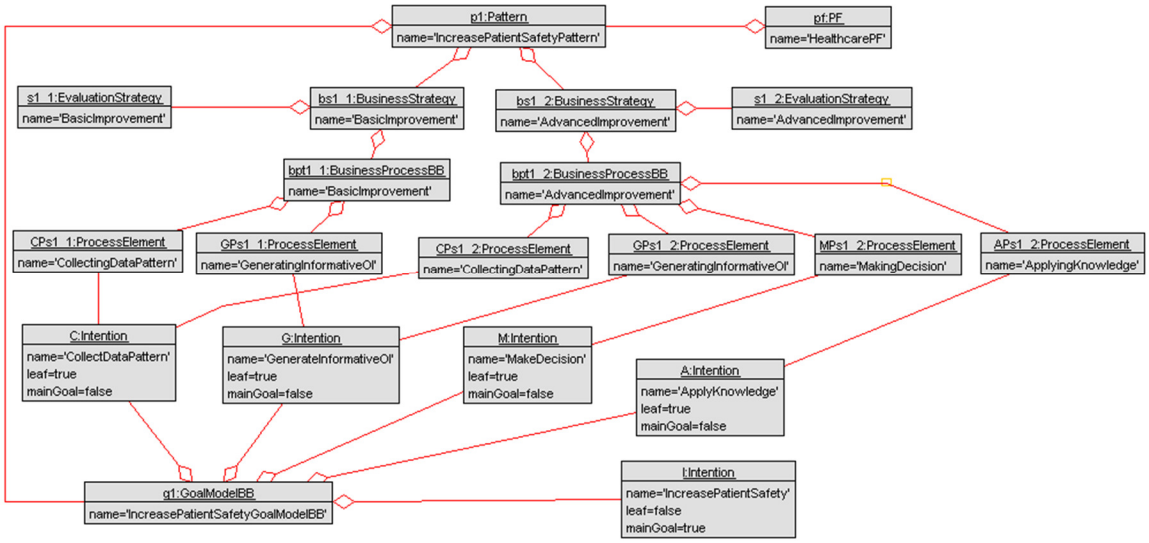

Figure 40 Object model of Increase Patient Safety pattern

Application of the Extension Algorithm to an empty PF

Step S1 initializes mg with I (Increase Patient Safety), which is the main goal of xp's goal model building block. The next three steps (S2, S3, and S4) are in charge of modification but because the modifications set is empty, these steps do not invoke the modification algorithm. Step S5 adds xp to pf through an in-depth copy. After this step, xp becomes a part of pf but it is still an isolated pattern as the need for links between xp and related patterns in pf is not yet explored. In step S6, relatedIntentions is set to **null** because no other pattern with a leaf intention equal to mg exists in pf. In other words, the new pattern is not refining any intention of other patterns. Step S7 assigns the leaves of xp ({C,G,M,A}) to leafIntentions. Steps S8 and S8.1 do not apply because relatedIntentions is empty. Steps S9 and S9.1 are applied for each element of leafIntentions. However, because no pattern in pf refines elements of leafIntentions, no further action is taken in steps 9.1.1 , 9.1.2 , or 9.1.3. Figure 41 shows the output of this algorithm (O1) as an object diagram. This figure represents the extended PF where Increase Patient Safety pattern (xp) was added to the initially empty PF (pf).

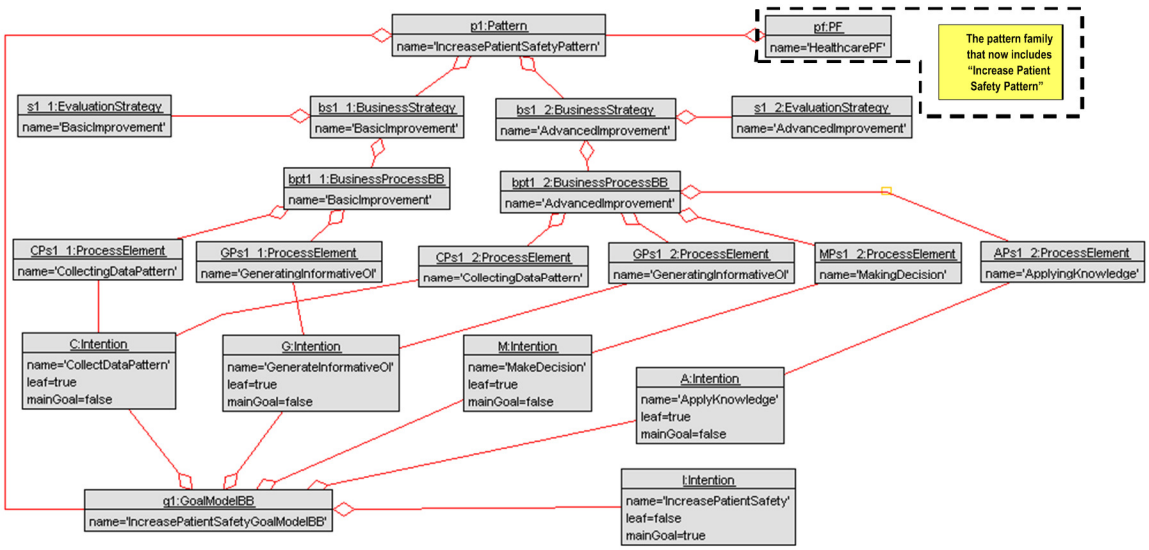

Figure 41 Object model of an empty PF extended to include Increase Patient Safety pattern

5.2.4 Example 2: Extension of Non-Empty PF

In this section, the extension algorithm is used for evolving a PF that is already established and that contains some patterns.

In the patient safety case study, a pattern family comprised of 32 interrelated patterns is created [39]. These patterns are mostly focused on the problem of improving patient safety by highlighting processes that can be improved. The patterns in this family do not use the data collected along the way for taking immediate actions to prevent an adverse event for a particular patient. Over time, it was observed that some hospitals use such collected data not only for *a posteriori* analysis but also for preventing the potential adverse events that may happen. Consequently, the PF analyst creates a new pattern (xp) that captures the problem of taking action to prevent predictable adverse events and its alternative solutions. The recurring excerpt of the observed goal model that formulates the main observed goal, Take Action, forms the goal model building block of xp (left side of Figure 42). In this goal model building block, the side effects of alternative business strategies are hidden in order to simplify the example. The goal model building block is composed of the main high-level goal, i.e., Take Action, and of the elements of solution, i.e., Prioritize Outcome and Prevent Outcome. The alternative solutions are captured in the form of business strategies that are composed of business process building blocks and

strategies. The two business process building blocks of the new pattern are shown in Figure 42 (right side).

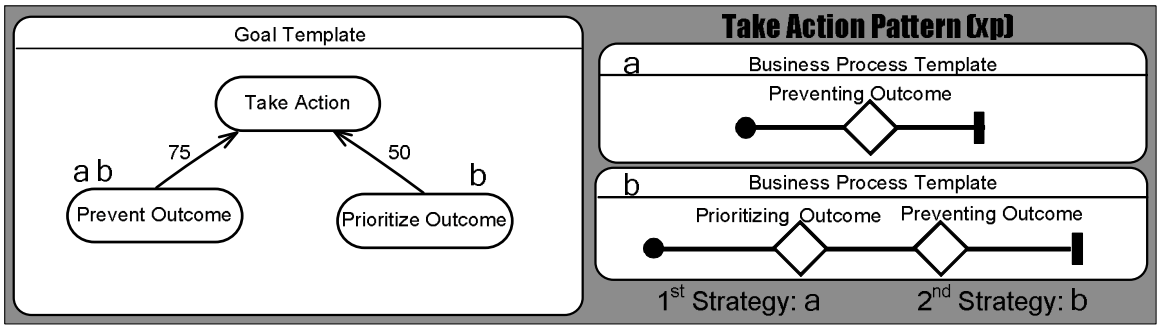

Figure 42 Left: goal model building block of xp - Right: business process building blocks of xp

The extension algorithm (Algorithm 1) changes the patient safety PF to include the Take Action pattern. The analyst prepares and provides the following inputs: pf as the initial PF that must be extended (I1), xp as the new pattern (I2), and modifications as a set that represents the needed modifications on other patterns in pf (I3).

The left part of Figure 46 (with grey background) illustrates the initial pf as an FMM-based UML object diagram, with only 3 out of the 32 patterns, for brevity. In this example, the Increases Patient Safety pattern (p1) is the only pattern affected by the extension because Take Action positively contributes to the Increases Patient Safety goal. Therefore, modifications is set to {(p1, link_I_T, *#Add*, bs1_3, **null**)}, indicating the new goal that must be added to p1's business goal model building block. It also indicates the new business strategy that represents an alternative solution that must be added to p1. Figure 44 shows the business processes template of the bs1_3 business strategy as a UCM diagram. As shown in this figure, the collected data and generated information about the current outcome and potential outcome may be used to prevent outcomes by taking immediate actions. Finally, because none of the existing business strategies is being eliminated, the last element of modifications set is **null**.

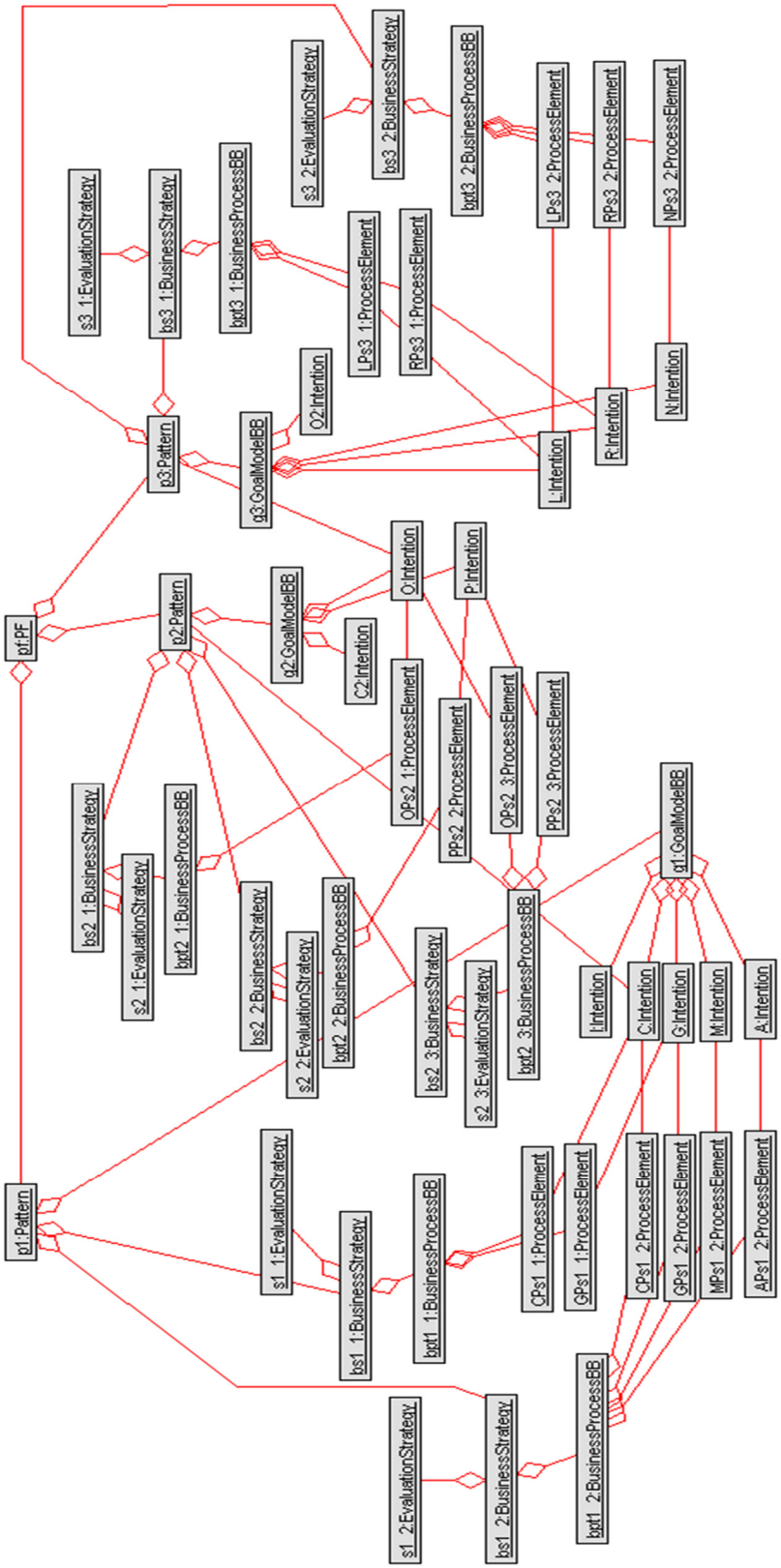

Figure 43 Object model of a non-empty PF used as initial PF

Figure 45 illustrates xp as an FMM-based UML object diagram (objects of the Element-Link class are not shown in favor of readability). In this diagram T, V, and Z respectively identify intention objects for Take Action, Prevent Outcome, and Prioritize Outcome.

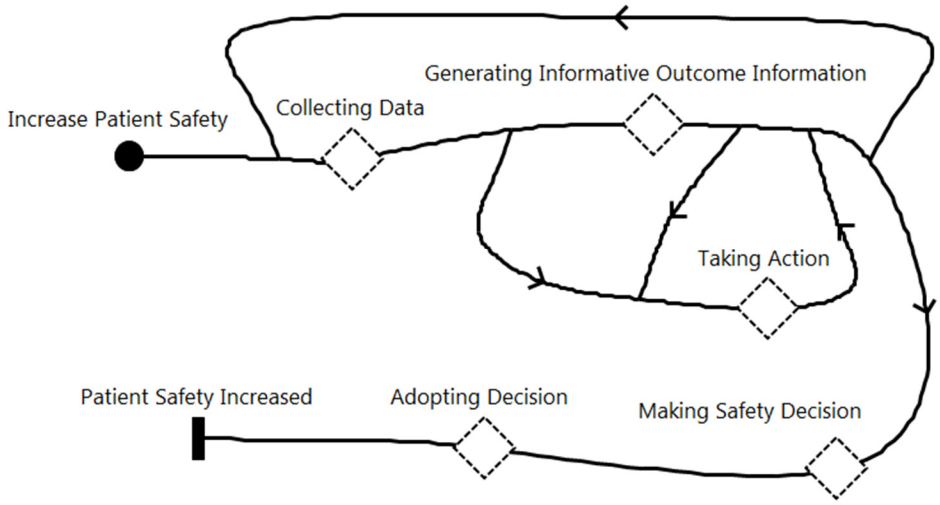

Figure 44 bpt1_3, which is the business process building block of bs1_3

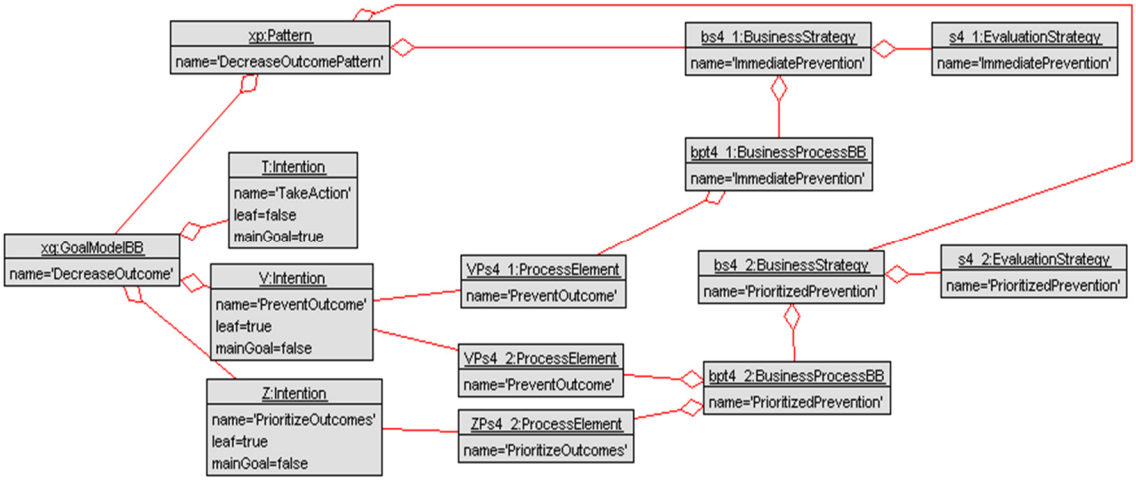

Figure 45 Object model of xp pattern

Application of the Extension Algorithm to a non-empty PF

Step S1 initializes mg with T (Take action), which is the main goal of xp's goal model building block. Steps S2, S3, and S4 invoke the modification method with pf and the modifications set as its inputs. The modification algorithm (section 5.3) applies these changes to the related patterns, i.e., p1, as it is the only related pattern in pf. Figure 46 shows the modified pf, where the area encapsulated within the dashed box (bottom right) represents the modifications. The details of T2 (Taking Action) at p1 level are dismissed so when the users of pf choose a solution at p1's level that includes T2, the solution will be refined by using the new pattern. Step S5 adds xp to pf. This is an in-depth copy of all elements of xp into pf. After this step, xp becomes a part of pf but it is still an isolated pattern as the links between xp and related patterns in pf are not yet established. Step S6 finds those intentions that are (i) leaves of other patterns in pf and (ii) equal to the main goal of xp's goal model building block. In this example, T2 (Take Action) is a leaf intention in p1 and is equal to mg. Therefore, relatedIntentions is set to {T2}. Step S7 assigns the leaves of xp's goal model building block ({V, Z}) to leafIntentions. Steps S8 and S8.1 set the patternDef of T2, which is the only member of relatedIntentions, to xp. This captures the fact that xp is refining the T2 intention in the goal model building block of p1. Steps S9 and S9.1 are applied to each element of leafIntentions. However because no pattern in pf refines either V or Z, no action takes place in steps 9.1.1 , 9.1.2 , or 9.1.3. Figure 46 shows the output of this algorithm (O1). It represents the extended PF after the new pattern xp (top-right) was added and integrated with the patterns in the initial PF (pf).

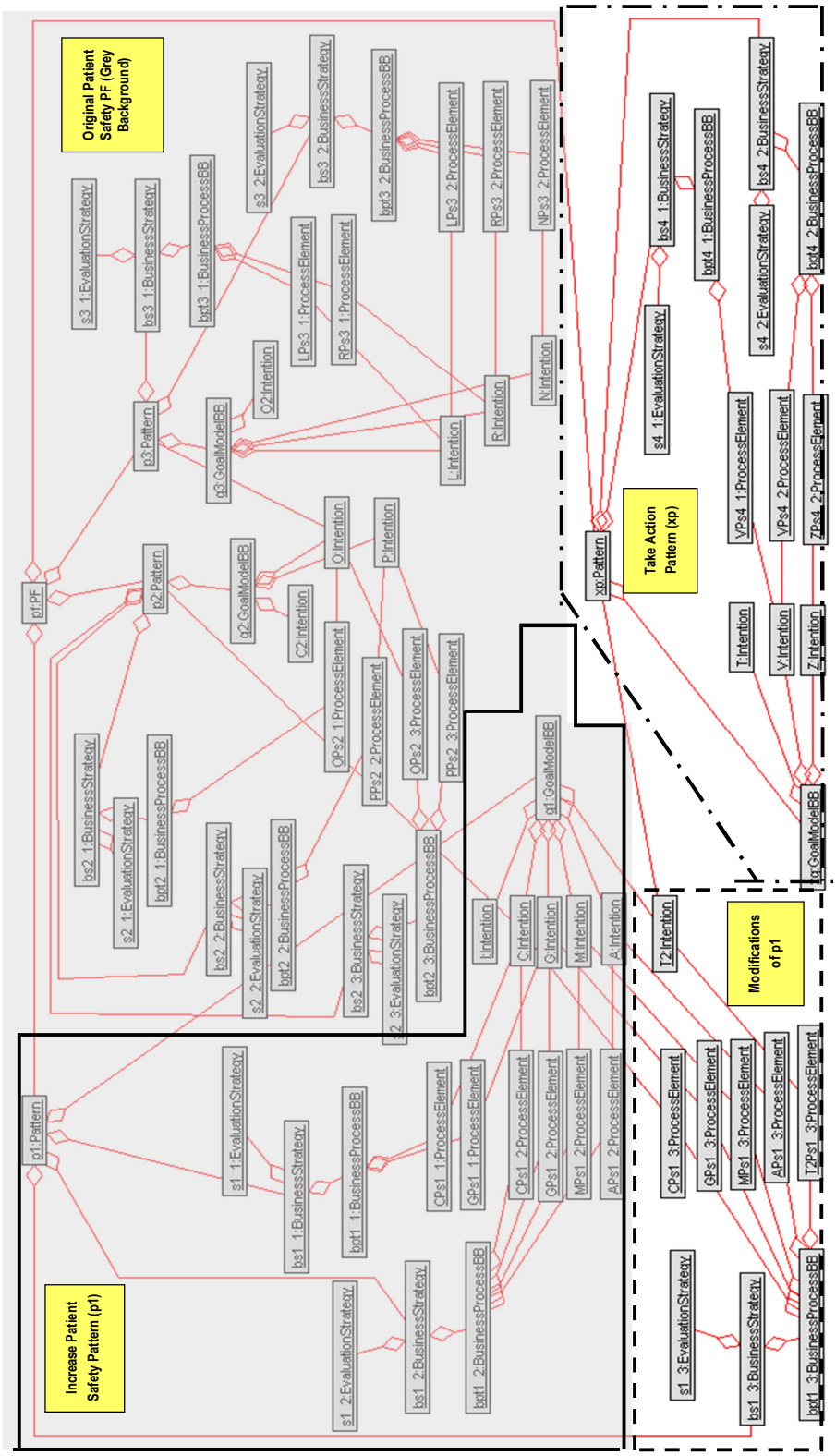

Figure 46 Object model of the extended PF pf that includes the xp pattern

5.3 Modification

This section describes the algorithm for evolving a PF by *modifying* its patterns within a PF. Such evolution is needed when:

I. changes in a domain indicate that the goal model building block or business strategy of a pattern must be updated,

II. the PF is being extended with a new pattern and introducing the new pattern affects a particular pattern (i.e., the way it is used within Algorithm 1), and

III. another pattern in the PF is eliminated and a particular pattern is affected.

The algorithm modifies the goal model building block to update the problem, elements of solutions, and the contributions between the elements. Similarly, the modification of business strategies enables the modification of solutions and their effects.

A modification is composed of three main optional and possibly cyclic steps (see Figure 47): first, it modifies the goal model building block of a pattern, then it removes the old business strategy (if already present), and, finally, a new business strategy is added (if available). The following subsections provide the details of the modification algorithm, the general application of this algorithm, and its illustration based on a specific example from the patient safety case study.

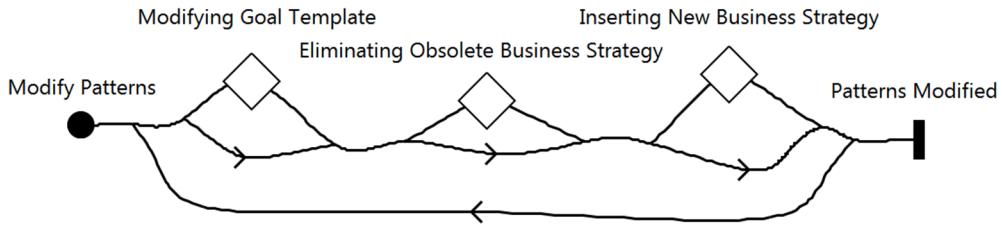

Figure 47 Main steps of the modification mechanism

5.3.1 Modification Algorithm

The following algorithm provides steps for modifying a PF.

Inputs of the modification algorithm

I1. pf:*PF* /* initial pattern family */
I2. modifications: **ordered set of** (p: *Pattern*, link:*ElementLink*, action ∈ {*#Add,#Delete*},
bst:*BusinessStrategy* , oldbst:*BusinessStrategy*)
where link.toLinks.isEqualTo(p.GMBB.mainGoal())

Output of the modification algorithm

O1. **modified** pf:*PF* /* the modified pattern family */

Preconditions and post-conditions of the modification algorithm

Pre 1: pf.pattern->size()>0
Pre 2: modifications->**forAll**(m| p.GMBB.intention->**exists**(i|i.isEqualTo(m.link.toLink)))
Post 1: modifications->**forAll**(m|m.action == *#Add* **implies** p.GMBB.elementLink->includes(m.link) **and**
p.GMBB.intention-
>**exists**(i|i.isEqualTo(m.link.fromLink)))
Post 2: modifications->**forAll**(m|m.action == *#Delete* **implies** p.GMBB.elementLink->**excludes**(m.link))

Steps of the modification algorithm

S1. **for each** m **in** modifications:
S2. mp:Pattern = a pattern **in** pf.patternCollection() **where** pattern.isEqualTo (m.p)
/* m.p is the first element of the m */
S3. **if** (m.action == *#Delete*) **then**
S4. mp.GMBB.delete(m.link)
S5. **elseif** (m.action == *#Add*) **then**
S6. mp.GMBB.add(m.link)
S7. **endif** /* if no action is provided then the GoalModelBuildingBlock is unchanged */
S8. **if** (m.oldbst ≠ **null**) **then**
S9. pbst:BusinessStrategy = a businessStrategy **in** mp.businessStrategyCollection()
where businessStrategy.isEqualTo(m.oldbst)
S10. mp.delete(pbst)
S11. **endif**
S12. **If** (m.bst ≠ **null**) **then**
S13. mp.add(m.bst)
S14. **endif**

Algorithm 2. Modification of PF

5.3.2 Applying the Modification Algorithm

Once observed changes in the domain highlight the need for modifying a PF, the PF analyst prepares the inputs and uses the modification algorithm standalone or through other evolution mechanisms to update the patterns.

The modification algorithm takes two inputs: pf is a PF in which some patterns must be modified (I1), and a modifications ordered set (see the structure of its elements in Table 5) that represents the required modifications (I2).

Table 5 Elements of the modifications ordered set in the modification algorithm

Element	Description
p	A pattern that is affected and must be modified
link	A link between two intentions that highlights the part of the goal model building block that must be modified
action	An indication of what must be done on the link
bst	A new business strategy that represents a new solution
oldbst	An old business strategy that must be eliminated from p

As different types of modifications may be necessary, the second, forth, and fifth elements of modifications set (Table 5) may be **null**. The first precondition prevents using the modification algorithm on empty pattern families. The second precondition makes sure the toLinks side of the link always points to an intention in the goal model building block of p. The first post-condition ensures that when the action is *#Add*, the algorithm adds the link as well as the link.toLinks intention to the relevant pattern's goal model building block. The second post-condition ensures that when the action is *#Delete*, the algorithm deletes the link from the relevant pattern's goal model building block. These preconditions and post-conditions prevent using the modification algorithm for modifying unrelated patterns. The output of the algorithm represents the modified pf (O1).

Steps S2 to S14 must be taken for each element of the modifications set. Step S2 initializes mp with the pattern of the active member m of the modifications set. Next, in steps S3 to S7, depending on the type of action, the link will be either added to (steps S3, S4) or deleted from (steps S5, S6, S7) mp's goal model building block. Then, if m.oldbst is not **null**, it will be deleted from mp in steps S8 to S11. Finally, if m.bst contains a new business strategy, then it is added to mp (steps S12 to S14).

5.3.3 Example: Modification of a Pattern

The Modification Mechanism can be used as a standalone mechanism for modifying a pattern within the family or it can be used through other mechanisms to update the family. This example illustrates the details of how the modification algorithm modifies the PF when it is used via the Extension Mechanism in the example of section 5.2.4. The inputs of the modification algorithm in this example are:

(I1) – pf, which is the initial PF. This input is provided to the modification algorithm by step S3 of the extension algorithm in section 5.2.4. The left part of Figure 46 (with grey background) represents this input.

(I2) – modifications set is the second input provided in S3 of the extension algorithm in section 5.2.4, which is equal here to {(p1, link_I_T, *#Add*, bs1_3, **Null**)}

Applying the Modification Algorithm

The steps (S2 - S14) must be taken for all elements of the modifications set, which in this case has only one member (called m). S2 initializes mp with p1, i.e., the first element of

m. Because m.action is equal to *#Add*, steps S5 to S7 add link_I_T to the goal model building block of p1. No action is taken in steps S8, S9, S10, S11 because m.oldbst is null (there is no old business strategy to be deleted). The underlying object model at this point is illustrated in Figure 48.

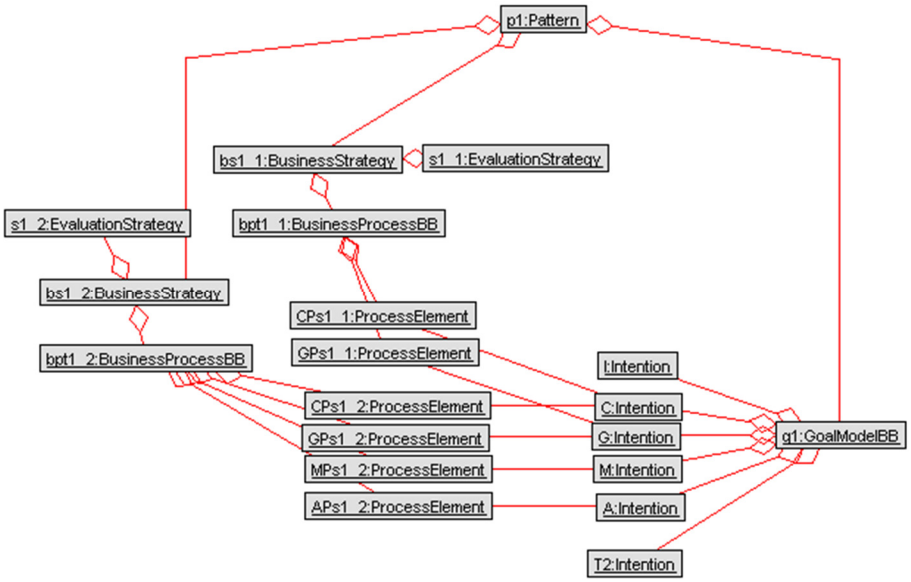

Figure 48 Object model of the p1 pattern after modification of its goal model building block

Steps S12 to S14 add the new business strategy in m.bst to p1. This business strategy represents another solution to the problem (Increase Patient Safety) in p1. This solution contains T2 (Take Action) along with other intentions for improving the underlying procedures. Figure 44 on page 85 illustrates the business process building block of this solution. Figure 49 represents p1 after the whole modification. Figure 46 shows the complete output of the modification algorithm (O1). In this particular example, the modified PF is returned to step S3 of the extension algorithm in section 5.2.4, which initially invoked the modification algorithm.

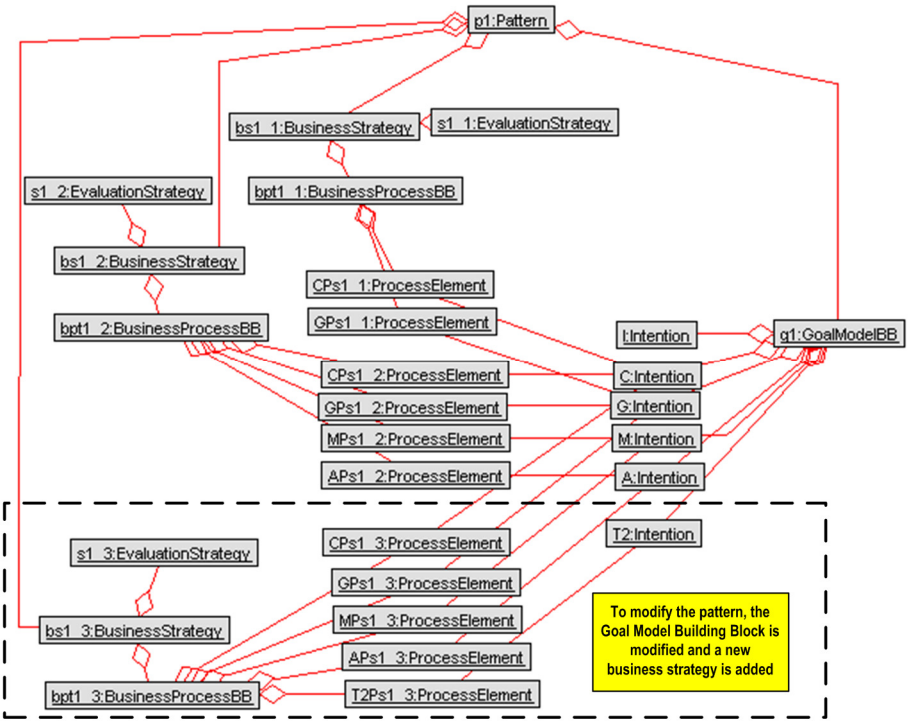

Figure 49 Object model of the p1 pattern after modification

5.4 Elimination

When the PF analyst observes that a pattern in the PF is no longer needed by the PF users and does not solve current problems of the stakeholders, it is considered obsolete. Consequently, it must be removed from the pattern family. This section describes the algorithm for evolving a PF by eliminating its obsolete patterns. Elimination is composed of three steps (Figure 50). First, all pattern refinement (patternDef) links from those patterns that were refined by the obsolete pattern are removed. Next, related patterns in the family that are being affected by elimination are modified. Then, the obsolete pattern is deleted from the family. The elimination algorithm uses the modification algorithm for modifying the related patterns in the family. The Elimination Mechanism does not propagate elimination of patterns, so if another pattern becomes obsolete as a result of removing one pattern from the family, the PF analyst must explicitly use the Elimination Mechanism to eliminate the other obsolete pattern.

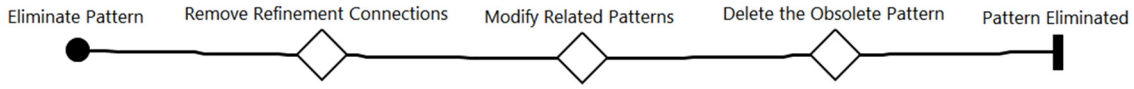

Figure 50 Main steps of the Eliminating Mechanism

5.4.1 Elimination Algorithm

Algorithm 3 provides inputs, output, and steps of the elimination mechanism.

Inputs of the elimination algorithm
I1. pf:*PF*
I2. op:*Pattern* /* an obsolete pattern that must be eliminated from pf */
I3. modifications: **ordered set of** (rp:*Pattern*, link:*ElementLink*, action ∈ {#Add,#Delete},
bst:BusinessStrategy , oldbst:BusinessStrategy)
where (link.toLinks.isEqualTo(rp.GMBB.mainGoal()) ∧
(action == *#Delete* ⇒ link.fromLinks.isEqualTo(op.GMBB.mainGoal()))

Output of the elimination algorithm
O1. **modified** pf:*PF* /* the original pattern family where op is eliminated*/

Steps of the elimination algorithm
S1. **foreach** (p:*Pattern* **in** pf.patternCollection() **where not** p.isEqualTo(op))
S1.1.**foreach** (i:*Intention* **in** p.GMBB.leafCollection())
1.1.1. **if** (i.patternDef .isEqualTo(op))
1.1.2. i.patternDef = **null**
1.1.3. **endif**
S2. **if** (**not** modifications.isEmpty()) **then**
S3. modify (pf, modifications) /* related patterns that will be effected are modified */
S4. **endif**
S5. pf.delete(op) /* op is removed from pf */

Algorithm 3. Elimination of PF

5.4.2 Applying the Elimination Algorithm

This algorithm takes three inputs: pf is an initial PF (I1), op is an obsolete pattern that must be removed from the initial PF (I2), and the modifications set (Table 6) highlights the effects that eliminating op from pf has on other patterns in the family (I3).

Table 6 Elements of the modifications set in the elimination algorithm

Element	Description
rp	A related pattern that is affected by elimination of obsolete pattern and must be modified
link	A link between two intentions that highlights the part of the goal model building block that must be modified
action	An indication of what must be done on the link
bst	A new business strategy that represents a new solution
oldbst	An old business strategy that must be eliminated from rp

The PF analyst locates the obsolete pattern by observing pattern usage of PF users in a specific domain and by using their feedback about patterns in the PF. She then analyzes and determines the effects of this elimination, which in turn, leads to the creation of a modifications set.

As different types of modifications may be necessary, the second, forth, and fifth elements of modifications set (see Table 6) may be **null**. The toLinks side of the link must always point to the main goal of rp. If the action is *#Delete*, then the fromLinks side must point to an intention that is refined by op. These two preconditions prevent using the elimination algorithm for merely changing an unrelated pattern in pf. Isolated modifications of patterns must use the modification algorithm described in section 5.3. pf is the modified pattern family in which the obsolete pattern is eliminated (O1).

Through steps S1 to S5, this algorithm carries out the three major activities illustrated in Figure 50. Step S1 and its sub-steps remove those refinement links in the family that are pointing to the obsolete pattern. Steps S2 to S4 modify those patterns that are affected by elimination of the obsolete pattern. Finally, step S5 deletes the obsolete pattern from the pattern family.

5.4.3 Example: Elimination of an Obsolete Pattern

This example illustrates how the elimination algorithm eliminates an obsolete pattern from PF. In order to use the Elimination Mechanism, the PF analyst prepares and provides pf (I1), op (I2), and modifications set (I3).

For the purpose of simplicity, this example uses the pf created in section 5.2.4 and eliminates the Action Taking pattern with which the PF in that section was extended. Figure 51 illustrates the FMM-based UML object diagram of the mentioned pf (I1). In this PF, the Action Taking pattern is considered obsolete (I2). In this example, the Increases Patient Safety pattern (p1) is the only pattern affected by the elimination because the PF analyst considers Take Action as an element that does not contribute to Increases Patient Safety. Therefore, modifications is set to {(p1, link_l_T, *#Delete*, **null**, bs1_3)}. link_l_T indicates the obsolete goal (Take Action) and its contributions that must be removed from p1's business goal model building block. It also indicates the old business strategy that represents an alternative solution that must be removed from p1.

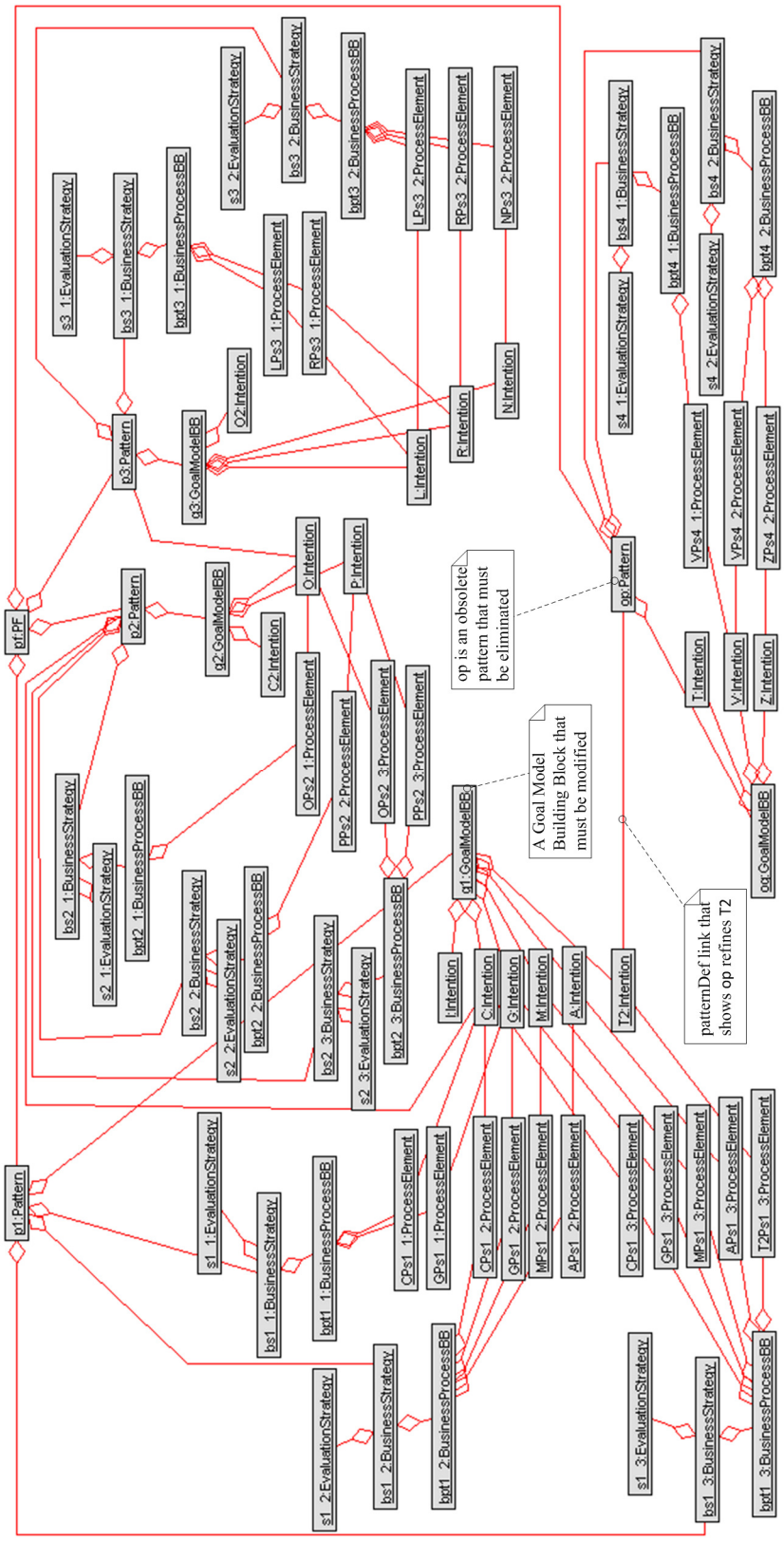

Figure 51 Object model of initial PF with an obsolete pattern (op)

Applying the Elimination Algorithm

Step S1 and its sub-steps set patternDef of T2 to **null**. This removes the refinement link between T2 and op (see Figure 51). The next three steps (S2, S3, and S4) invoke the modification algorithm with pf and the modifications set as its inputs. The modification algorithm (section 5.3) applies these changes to the related patterns, i.e., p1, as it is the only related pattern in pf. Finally, S5 removes op from the pf. The modified pf will be equal to the pattern family illustrated in the left part of Figure 46 (with grey background).

5.5 Combination

This section describes the Combination Mechanism, which targets the combination (merging) of two pattern families. When PF analysts observe that two pattern families describe similar areas of a domain, they may use this algorithm to merge the two families. The output of this algorithm is a pattern family that includes the patterns from both initial PFs. This algorithm also maintains the integrity of patterns and their relationships in the output. The PF analyst can then further modify the resulting PF with the Extension, Elimination, and Modification Mechanisms.

The Combination Mechanism empowers the PF analyst to define the *starting point* of the combination of two pattern families. This feature makes it possible to use the algorithm for two distinct purposes. First, selecting the highest-level common pattern as starting point will lead to creating a pattern family that includes all the patterns that refine the start point. Second, by selecting a common pattern in the middle of pattern family hierarchy, this algorithm creates a common subset of both pattern families. The latter, less apparent usage of this algorithm, is particularly useful to create a pattern family from a common subset of patterns in two PFs that would otherwise have little similarities.

The Combination Mechanism can be used to combine two PFs from a particular start point. Having a start point is valuable because it enables the analyst to merge the overlapping parts of two slightly different pattern families into a new pattern family that includes the more specific knowledge scattered over two original families.

5.5.1 Combination Algorithm

Algorithm 4 provides inputs, output, and steps of the combination mechanism.

Inputs of the combination algorithm
I1. pf1:*PF* /* this PF is the first pattern family */
I2. pf2:*PF* /* this PF is the second pattern family */
I3. startPattern:*Pattern* /* this is a common pattern in both pf1 and pf2 and is used as the start point for combination */

Output of the combination algorithm
O1. pf:*PF* /* a pattern family that contains the patterns from both pf1 and pf2 */

Preconditions and post-condition of the combination algorithm
Pre 1: pf1.patternCollection()->**exists**(p|p.isEqualTo(startPattern))
Pre 2: pf2.patternCollection()->**exists**(p|p.isEqualTo(startPattern))
Post 1: **let** pf1subset = pf1.pattern->**any**(p|p.isEqualTo(startPattern))->**first**().DefiningPatternSet() **in**
let pf2subset = pf2.pattern-> **any**(p|p.isEqualTo(startPattern))->**first**().DefiningPatternSet() **in**
let pf1pf2subset = pf1subset->**union**(pf2subset)->**union**(startPattern) **in**
pf.patternCollection()->**forAll**(p1|pf1pf2subset.->**exists**(p2|p2.isEqualTo(p1))) **and**
pf1pf2subset->**forAll**(p1|pf.patterncollection()->**exists**(p2.isEqualTo(p1)))

Steps of the combination algorithm
S1. pf = **new** *PF* /* initialize pf */
S2. pf.insert(startPattern)
S3. toMerge: **ordered set of** (*Pattern*)= {startPattern}
S4. allPatterns: **set of** (*Pattern*) = pf1.patternCollection() ∪ pf2.patternCollection()
S5. **while** (toMerge **is not empty**)
S5.1. nextPattern:*Pattern* = toMerge.nextElement()
S5.2. nextLeafGoals:**set of** (*Intention*) = nextPattern.GMBB.leafCollection()
S5.3. **foreach** (g:*Intention* **in** nextLeafGoals)
5.3.1. **foreach** (p:*Pattern* **in** allPatterns)
5.3.2. **if** (p.GMBB.mainGoal().isEqualTo(g)) **then**
5.3.3. pf.insert(p)
5.3.4. g.patternDef = p
5.3.5. toMerge.append(p)
5.3.6. endif
S5.4. toMerge.removeElement(nextPattern)

Algorithm 4. Combination of PFs

5.5.2 Applying the Combination Algorithm

When two pattern families represent the knowledge in one domain and have similar patterns, the PF analyst may decide to combine two families in order to have a new PF that better captures the overall knowledge in the domain. The Combination Mechanism can be used to combine two PFs from a particular start point.

In order to use the Combination Mechanism, the PF analyst prepares and provides two pattern families, pf1 (I1) and pf2 (I2), along with the starting point startPattern (I3). The output of the algorithm represents the new pf (O1). The combination algorithm assumes that common patterns (i.e., patterns with the same name) are the same in both PFs. In those cases where patterns are slightly different but can be combined into one common pattern, the analyst must first use the Modification Mechanism (Section 5.3) in order to eliminate the differences among patterns that must be considered as equivalent. Precondi-

tion 1 (Pre 1) and precondition 2 (Pre 2) respectively limit the inputs I1 and I2 so they contain the startPattern (I3). The post-condition 1 ensures that the output pattern family (pf) includes all the patterns in each of pf1 and pf2 that can potentially refine startPattern. There are other well-formedness constraints such as "pf1 and pf2 must include no more than one pattern equivalent to startPattern" or "all patterns in pf must directly or indirectly refine startPattern". It should be noted that such well-formedness constraints are implied by the OCL constraints provided in Chapter 3.

Step S1 defines pf as an empty pattern family. After the complete execution of the algorithm, pf will contain the combined pattern family. Step S2 inserts the pattern from startPattern into pf. Step S3 defines toMerge as an ordered set of patterns and initializes it with startPattern as its first element. Step S4 defines allPatterns as a set of patterns and initializes it with the patterns in pf1 union those in pf2. Step S5 is a loop that adds patterns from the toMerge list to the combined pattern family. Step S5.1 assigns the first element of the toMerge list to nextPattern. Step S5.2 then assigns all the leaf intentions of the nextPattern's goal model building block to nextLeafGoals. These intentions may potentially be refined with other patterns in the allPatterns set. Step S5.3 and its sub-steps form a nested loop that checks all the elements of allPatterns to find patterns that refine intentions in nextLeafGoals. Once a refining pattern is found, steps 5.3.3 to 5.3.5 add that pattern to pf, connect it to the relevant intention in nextLeafGoals, and add it the toMerge list. After all iterations of S5.3, the pattern in nextPattern is connected to all those patterns from both pf1 and pf2 that could refine its leaf intentions. Next, step S5.4 removes this pattern from the toMerge list. Subsequent iterations of step S5 examine the next patterns in toMerge for finding their refining patterns and adding them to pf. The iterations of step S5 continue until all the refining patterns in both pf1 and pf2 are included in pf.

5.5.3 Example: Combination of two Pattern Families

This section illustrates the application of the combination algorithm by combining two patterns families that cover part of the case study's patient safety domain (i.e., prospective surveillance). In order to avoid unnecessary complexity, unlike the examples in previous sections, patterns are represented here in a more abstract and compact way using stereotyped UML packages. Figure 52 shows the Increase Patient Safety pattern in this format and illustrates how it hides the details of this pattern.

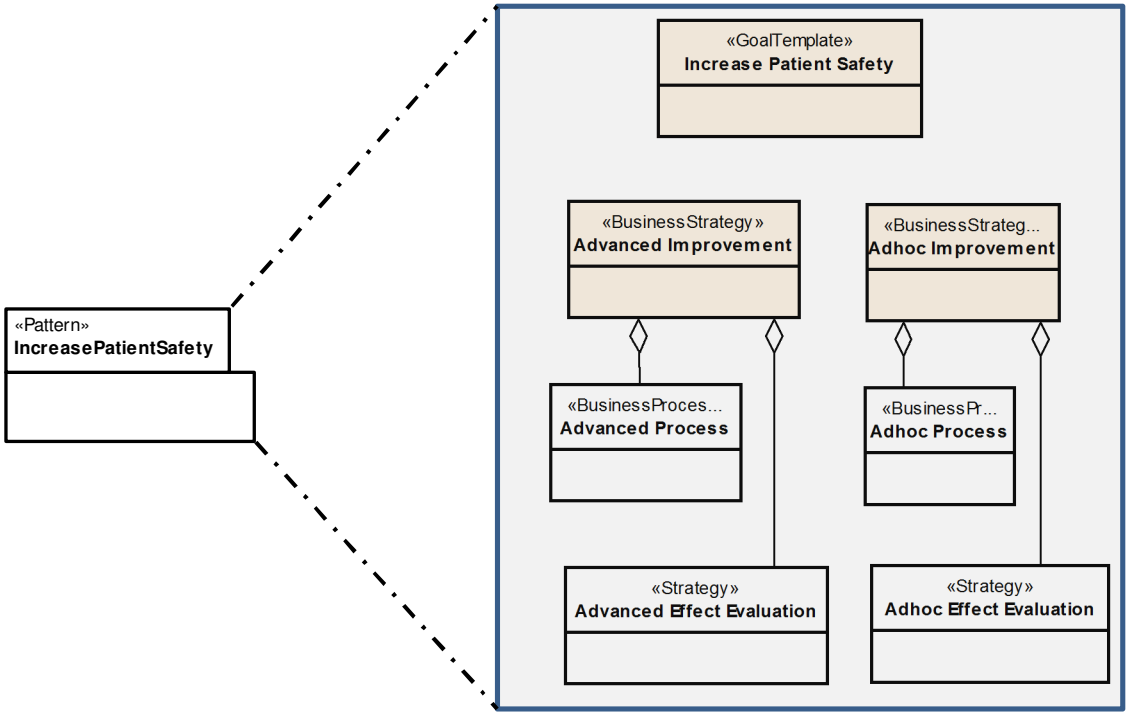

Figure 52 Using a stereotyped UML package to represent the Increase Patient Safety pattern

Some of the patterns in these two families are common, yet each of them contains unique patterns that are not included in the other family. Figure 53 illustrates pf1, which is the first input of the algorithm (I1). This pattern family includes patterns for improving the patient safety using ad hoc approaches. Figure 54 presents pf2, which is the second input of the algorithm (I2). pf2 includes needed patterns for systematic approaches but it does not include the detailed refinement patterns included in pf1.

In this example, the PF analyst sets startPattern (I3) to Increase Patient Safety pattern. This input indicates the common pattern, which will be the highest-level pattern of the output pattern family (O1).

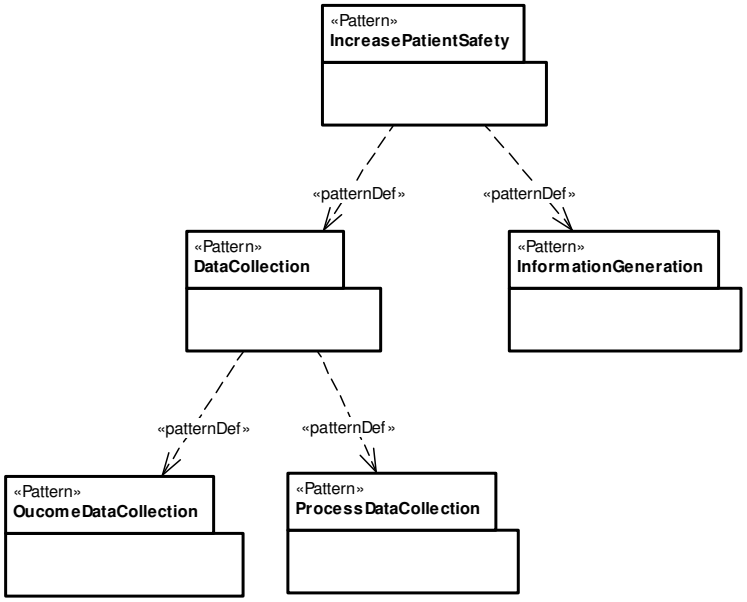

Figure 53 pf1 is a PF that contains patterns for ad-hoc approaches to improve patient safety

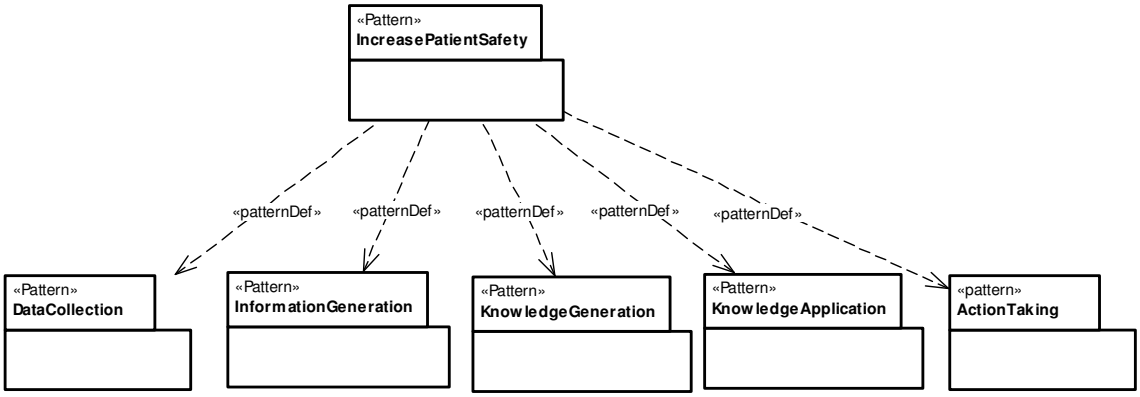

Figure 54 pf2 is a PF that contains patterns for both ad-hoc and systematic approaches for improving patient safety

Applying the Combination Algorithm

Step S1 creates pf, which is an empty PF that eventually will contain the combined pattern family. Step S2 inserts the Increase Patient Safety pattern into pf. Step S3 defines toMerge as an ordered set of patterns and initializes it with startPattern. Therefore, the Increase Patient Safety pattern is set as the first element of toMerge. Steps S4 defines allPatterns and assigns to it all the patterns of pf1 union those of pf2, that is:{Increase

Patient Safety, Data Collection, Information Generation, Outcome Data Collection, Process Data Collection, Knowledge Generation, Knowledge Application, Action Taking}.

In the first iteration of step S5, toMerge is equal to {Increase Patient Safety}. Consequently, step S5.1 assigns Increase Patient Safety pattern to nextPattern. Step S5.2 assigns to nextLeafGoals the set of all of nextPattern's intentions. After this step, nextLeafGoals is set to {Collect Data, Generate Informative Outcome Information, Make Safety Decision, Adopt Decision, Take Action}. This set contains all the leaf intentions of Increase Patient Safety pattern. Step S5.3 compares each intention in the nextLeafGoals to the main goal of all the patterns in allPatterns set. In the first iteration of S5.3, the algorithm will find that the main goal of Data Collection pattern is Collect Data from nextLeafGoals. This implies that the Data Collection pattern refines the Collect Data goal in the Increase Patient Safety pattern. Therefore, steps 5.3.3 to 5.3.5 add the Data Collection pattern to pf, connect Collect Data from Increase Patient Safety pattern to Data Collection pattern through patternDef refinement link, and finally add Data Collection pattern to the toMerge list. In the subsequent iterations of S5.3, the following actions take place:

- Information Generation, Knowledge Generation, Knowledge Application, Action Taking patterns are inserted in pf.
- Generate Informative Outcome Information, Make Safety Decision, Adopt Decision, Take Action goals from the Increase Patient Safety pattern are connected to relevant inserted patterns through patternDef links.
- Information Generation, Knowledge Generation, Knowledge Application, Action Taking patterns are appended to the toMerge list.

Step S5.4 then removes Increase Patient Safety pattern from the toMerge list. At the end of the first iteration of step S5, toMerge is equal to {Data Collection, Information Generation, Knowledge Generation, Knowledge Application, Action Taking}. Figure 55 shows pf at the end of this step.

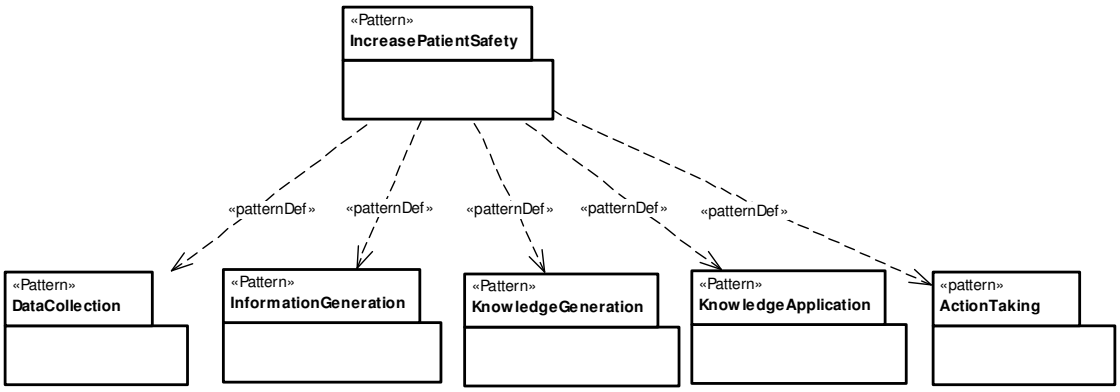

Figure 55 pf after first iteration of step S5

Because toMerge is not empty, step S5 will be executed a second time. Step S5.1 sets nextPattern to the Data Collection pattern. Step S5.2 sets nextLeafGoals to {Collect Outcome Data, Collect Process Data}. Iterations of step S5.3 and its sub-steps insert Outcome Data Collection and Process Data Collection patterns into pf, set them as refining patterns of Collect Outcome Data and Collect Process Data goals of Data Collection pattern, and add them to the toMerge list. Step S5.4 removes the Data Collection pattern from the toMerge list. Figure 56 represents pf at the end of the second iteration of S5. At this point, toMerge is equal to {Data Collection, Information Generation, Knowledge Generation, Knowledge Application, Action Taking, Outcome Data Collection, Process Data Collection}.

Because toMerge is again not empty, the execution of step S5 will continue. However, in these iterations, no refining pattern can be found and step S5 removes the patterns from toMerge at each iteration until it becomes empty. After complete execution of the combination algorithm, pf represents a PF that contains the combination of pf1 and pf2. Figure 56 illustrates this PF.

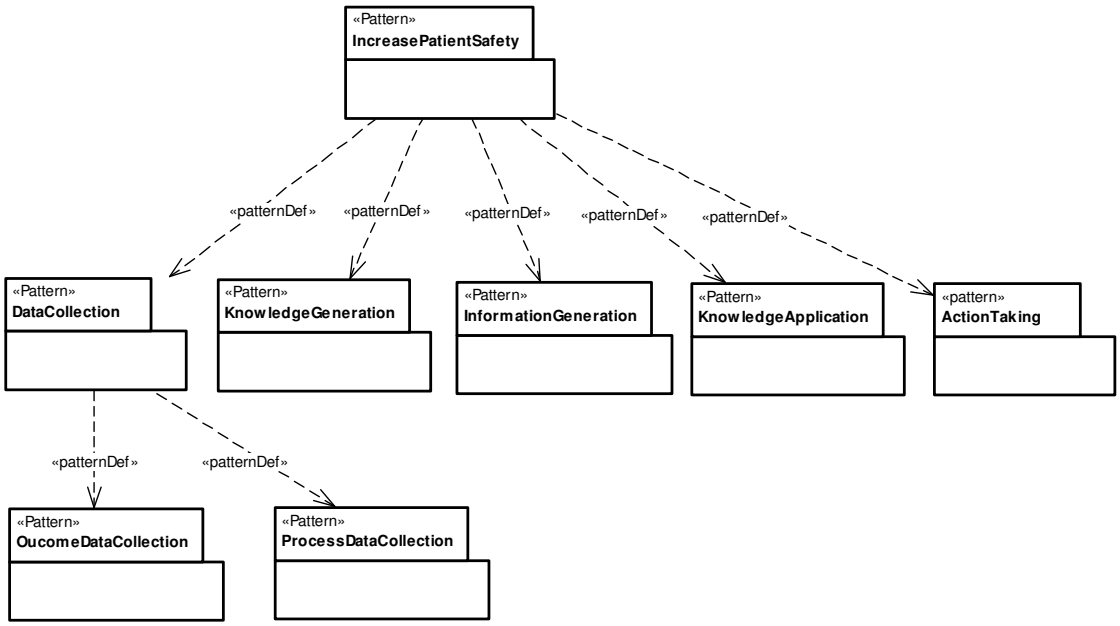

Figure 56 Combined pattern family

5.6 Summary

A pattern-based framework that lays down a foundation for capturing knowledge about business goals and processes is valuable. However, the problems and solutions within a domain are always changing. Consequently, such framework can be useful only if it can evolve over time. This chapter introduced and formalized four evolution mechanisms for extending and modifying PFs or for modifying or eliminating specific patterns within the PFs. The feasibility and utility of these mechanisms was demonstrated with examples from the patient safety domain.

The next chapter will present another method that focuses on how an up-to-date pattern family can be used to extract suitable business processes (with links to business goals) in the context of a given organization.

Chapter 6. ORGANIZATION-DRIVEN CUSTOMIZATION AND EXTRACTION METHOD (OCEM)

The knowledge captured in the form of patterns and organized as the family can help target organizations achieve their objectives. However, the relevant part of this knowledge must first be extracted and customized. A *PF user* is an analyst in charge of using the patterns and of creating the customized models that represent the requirements of stakeholders and corresponding business processes that can realize them. Figure 57 gives an overview of the steps taken for building such models.

This chapter introduces OCEM, a goal-driven algorithm that formalizes the extraction and customization of business processes by adapting instances of solutions for particular organizations within the domain targeted by a pattern family (e.g., patient safety). This algorithm currently does not formalize the interactions of PF users and stakeholders beyond receiving the initial organizational goal model. OCEM uses a PF as input and assesses the impact of alternative solutions for achieving the high-level goals of a given organization in a systematic, top-down approach. Another input is a partial business goal model where only some of the high-level goals of an organization need to be identified. OCEM's main output is a more complete organizational goal model combined with business processes aligned with the identified goals, as well as additional traceability links between the two views. The usage of OCEM complies with the spirit of MDE while including goal models in the chain of transformations. Finally, this chapter shows how pattern families, whose development and evolution were discussed in previous chapters, can be reused to create business goals and processes for individual healthcare organizations while taking into consideration the specifics of their context, including their own organizational goals and capabilities.

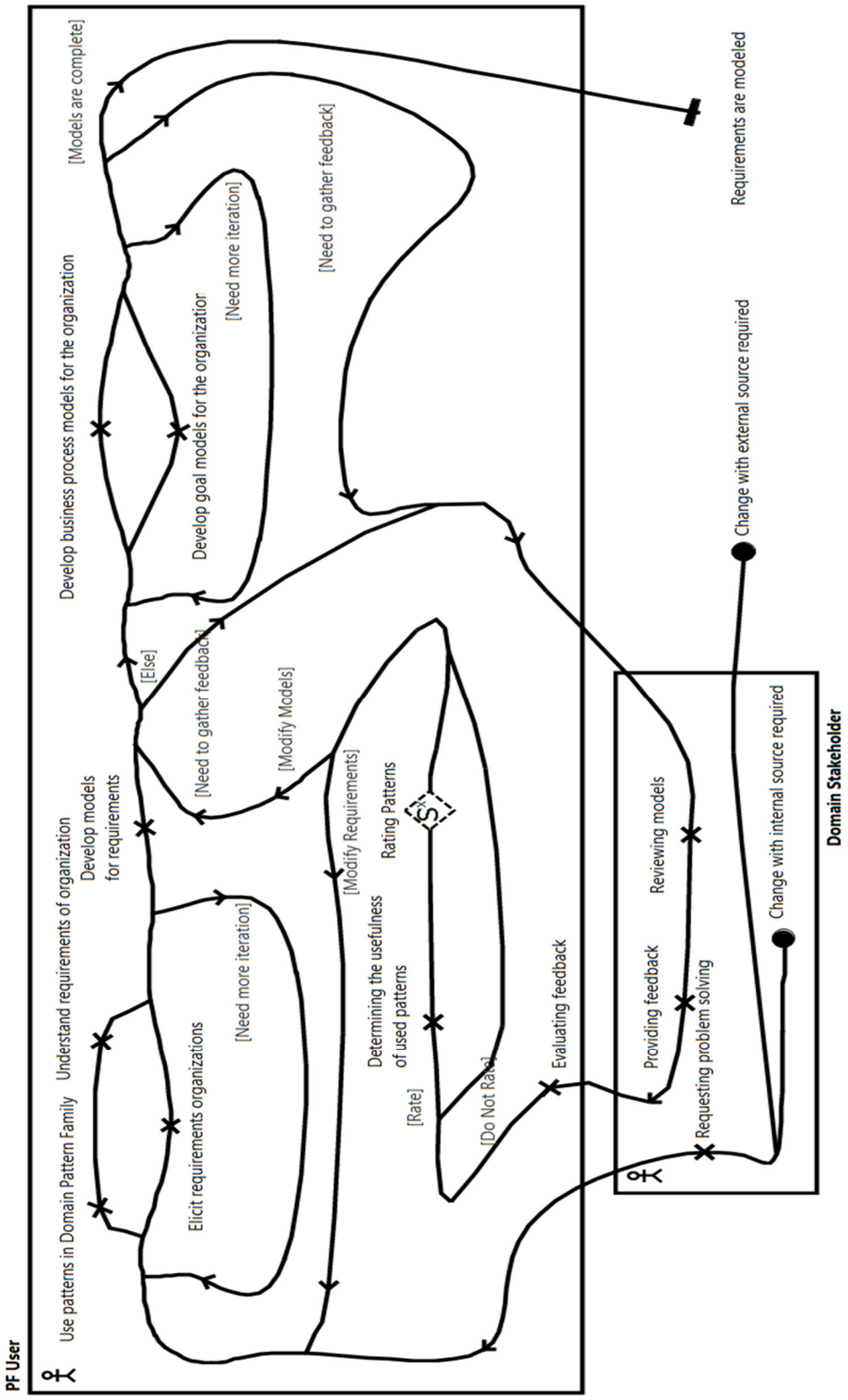

Figure 57 Using a pattern family to build requirements models for a specific stakeholder in a domain

6.1 Algorithm

Algorithm 3 provides the inputs, outputs, and main steps of the OCEM algorithm.

Inputs of the OCEM algorithm
I1. orgGM:*GRLgraph* /* initial organizational goal model */
I2. pf:*PF* /* pattern family of the domain */
I3. ev:*EvaluationStrategy* /* organization as-is evaluation */

Output of the OCEM algorithm
O1. orgGM:*GRLgraph* /* customized and extracted organizational goal model */
O2. orgBPM:*UCMmap* /* customized and extracted organizational business process model */
O3. URN links describing how the goals are realized by the process element.

Steps of the OCEM algorithm
S1. mainGoal:*Intention* = pf.GMBB.Intention **where** pf.GMBB.Intention.isEqualTo(orgGM.mainGoal)
S2. initBP:*UCMmap* = a simple process that contains only one stub (targetStub:*Stub*)
S3. toRefine:*List* = {(mainGoal, targetStub)}
S4. **foreach** (i:*Intention* **in** orgGM)
S5. **if** (i.isEqualTo(mainGoal)) **then**
S6. add a GRL contribution of weight 100 from mainGoal to i
S7. **endif**
S8. **foreach** (i:*Intention* **in** orgGM)
S9. **foreach** (p:*Pattern* **in** pf)
S10. **foreach** (ip:*intention* **in** p **where** i.isEqualTo(ip))
S11. **if** (ip.externalGoal ∧ ip.sideEffectGoal) **then**
S12. add a GRL contribution of weight 100 from ip to i
S13. **elseif** (ip.externalGoal ∧ ip.dependencyGoal) **then**
S14. add a GRL contribution of weight 100 from i to ip
S15. **endif**
S16. **while** (toRefine ≠ **null**)
S16.1. (NextGoal, NextStub) = next element of toRefine
S16.2. remove the element from toRefine
S16.3. strat:*BusinessStrategy* = best business strategy **in** p.businessStrategyCollection()
where (p:*Pattern* **in** pf.patternCollection() ∧ p.GMBB.mainGoal.isEqualTo(NextGoal))
/* selecting this business strategy leads to the highest satisfaction of the main actor in orgGM compared to ev */
S16.4. add NextGoal, strat.BPT.ProcessElement.intention (set of intentions), and their links to orgGM
S16.5. add strat.BPT as a plug-in to NextStub
S16.6. add realization URN link from NextGoal to its realization process element
S16.7. add realization URN link from NextGoal to its corresponding BPT
S16.8. add realization URN links from the intentions selected in strat to the corresponding process elements in BPT
S16.9. **foreach** (i:*Intention* initialized **in** strat)
S16.10. **if** (i ≠ task) **then**
S16.11. toRefine.add(i, i's realization stub)
S16.12. **endif**

Algorithm 5. OCEM

6.2 Application

A *PF user* is a modeler who uses the PF of a particular domain for creating the goal models and business processes that address the problems of an organization within the domain. The PF user prepares the inputs of the OCEM algorithm and then applies it.

This algorithm takes three inputs: orgGM as the initial organizational goal model (I1), pf as the pattern family of the domain (I2), and ev as the as-is evaluation strategy of

the organization (I3). This last input (a GRL strategy) provides initial satisfaction values to some of orgGM's intentional elements. These values capture the current status of the organization in need of a business process to be built from the pattern family. The OCEM algorithm extracts the most appropriate pieces of goal and business process models and builds a customized goal model accompanied with business process models that realize its goals.

Step S1 initializes the mainGoal with the intention in the domain that corresponds to the main stakeholder goal stated in the orgGM. Then, step S2 assigns a simple process with only one stub (targetStub) to initBP, which contains the initial business process of the organization. Next, step S3 initializes toRefine to a list that contains the pair (mainGoal, targetStub) as its only member. Steps S4 to S7 add a contribution with weight 100 from the mainGoal to the corresponding goal in orgGM. Steps S8 to S15 link corresponding goals in orgGM and patterns in pf. Step S10 looks for the "equality" between an intention from the pattern family and one from the organizational goal model, but in fact this equality could be weaken to some sort of equivalence (whose nature and handling are outside the scope of this thesis). Steps S11 and S12 add contributions with weight 100 from side-effect intentions in the patterns to the corresponding goals in the orgGM, while steps S13 and S14 add contributions with weight 100 from the intentions in orgGM to those external goals, which other goals in the pattern depend on. Not all intentional elements from the organization goal model need to be linked to an element of the pattern.

Step S16 is an iterative step that contains the following sub-steps. S16.1 and S16.2 initialize the pair (NextGoal, NextStub) with the next element on the toRefine list and remove the element from the list. Step S16.3 applies all strategies to find the best solution. Depending on the preferences of the stakeholders, the best strategy is the solution that better satisfies the priorities provided in orgGM. Therefore, it is possible that different stakeholders with different initial organizational goal models end up selecting different strategies as their best solution, which then shapes their businesses differently.

Step S16.4 adds the goals and links of the pattern to the organizational goal model, while step S16.5 adds the business process building block related to the chosen strategy as a plug-in to the stub of the simple process. Then, steps S16.6, S16.7, and S16.8 establish realization links between the main goal and the business process building block as

well as the sub-goals of the pattern and the stubs of the business process building block. Finally, step S16.9, S16.10, S16.11, and S16.12 add all sub-goals and linked stubs to the toRefine list, to be evaluated in the next iterations of step S16. This is done for all sub-goals that are required by the chosen strategy. Step S16 then iterates until all appropriate goals in the pattern are added to orgGM and business process building blocks have been chosen for all goals. In other words, continuous iteration of this step uses the knowledge captured in the pf and refines the business goals and processes to the desired level of detail.

The output of the OCEM is a refined GRL model of the organization (O1) with a model of the chosen business process options (O2), together with links that represent the rationale for their selection (O3).

6.3 Example

This patient safety case study illustrates how OCEM (Algorithm 5) is applied for customizing and extracting goal and business process models for a particular hospital, at the highest level of abstraction. In order to apply OCEM, the PF user prepares and provides its inputs: orgGM as the initial organizational model for Hospital A (I1), pf as the PF of the patient safety domain (I2), and ev as the as-is evaluation strategy of the Hospital A (I3).

Figure 58 illustrates the organizational goal model of Hospital A (orgGM, I1), which identifies the main goal (Increase Patient Safety) and three high-level softgoals related to quality, cost, and research concerns. Importance values are added to some of these intentional elements in the organizational goal model. In this example, the importance of Increase Patient Safety to its containing actor is deemed to be 100 while the importance of Decrease Cost is 25, which means that decreasing cost is less of an issue to Hospital A than increasing care and safety. Furthermore, Figure 58 depicts ev (I3) that is the current evaluation strategy of the organization, describing the as-is situation. Initial satisfaction levels, shown by the presence of a star (*), are provided: 60 to the task, indicating that although some advanced infrastructure is available, there is still room for improvement, and 40 to describe the current level of support to research on adverse events at that hospital. Considering the hospital has an average initial safety system, the satisfac-

tion levels of Increase Patient Safety and Increase Quality of Care are equal to 0, which consequently results in a general satisfaction of the stakeholder of -1. Color feedback is provided to show the satisfaction level of the intentions (the greener, the better).

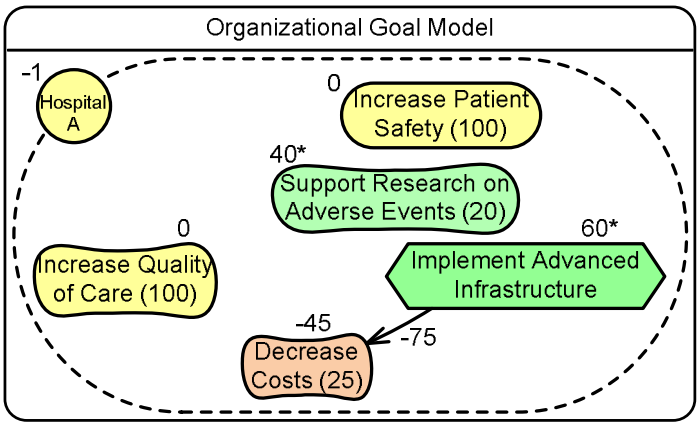

Figure 58 Original organizational goal model (I1, I3)

The patient safety PF, which targets the improvement of patient safety, has been built using the FDM described in Chapter 4 and Chapter 5 by observing the models created for several departments of a teaching hospital. This pattern family has 32 patterns that include goal model building blocks (organized in ten layers) together with 79 strategies and business process building blocks. This patient safety PF is used as the second input of OCEM (I2). Figure 43 illustrates a subset of the patient safety PF (3 patterns are shown).

Increasing patient safety is an abstract, recurring problem in different hospital departments and other healthcare organizations. The Increase Patient Safety pattern in the PF captures this problem and its solutions. The goal model building block of the pattern shown in Figure 33 represents the contributions of Collect Data, Generate Informative Outcome Information, Make Decision, and Apply Knowledge to the realization of Increase Patient Safety together with side-effects (e.g., on Increase Quality of Care in Long Term) and dependencies (e.g., on Deploy Advanced Infrastructures). Two strategies have been defined for this pattern. The first one (A) includes only the sub-goals Collect Data and Generate Informative Outcome Information. The second strategy (B) includes also the two other sub-goals, Make Decision and Apply Knowledge and adds corresponding activities to its business process building block. UCM models (see Figure 34) represent business process building blocks of these two strategies, which describe the ordering

of the activities (that are further refined in other patterns of the PF). All these models together constitute the Increase Patient Safety pattern (p1 in Figure 43).

OCEM establishes links between the initial organizational goal model and the goal model building block of the pattern (left half of Figure 59). First, a contribution with weight 100 is added from the Increase Patient Safety goal in the pattern to the Increase Patient Safety of the organization (showing the equivalence between these two goals, which could have had different names too as long as they are deemed "equivalent"). Then, two contributions with weight 100 are added from the quality/cost softgoals in the pattern to the quality/cost softgoals of the organizational model. Finally, a contribution with weight 100 is added from the task in the organizational model to the softgoal with the dependencies in the pattern. Not all intentional elements from the organization goal model need to be linked to an element of the pattern (e.g., Support Research on Adverse Events is not addressed by the current pattern).

Next, all alternative strategies are compared automatically for finding the best solution. Two strategies have been defined for this pattern, as shown in the left half of Figure 60. The first one (A) includes only the sub-goals Collect Data and Generate Informative Outcome Information. The second strategy (B) includes also the two other sub-goals, Make Safety Decision and Adopt Decision and adds corresponding activities to its business process building block. UCM models represent these two strategies as business process building blocks, which describe the ordering of the activities (that are further refined in other patterns of the Pattern Family, not shown here).

Figure 59 shows the result of the second strategy (B) because it yields better results than the first strategy (A), given that the organizational goal model places more value on quality than on cost and already has some advanced infrastructure available. A different healthcare institute with more focus on cost than quality and no advanced infrastructure available would see the first strategy (A) win over the second strategy (B). This evaluation is automated with OCEM, as it builds on GRL's quantitative evaluation algorithm, which propagates the known satisfaction levels to other intentional elements in the GRL models through their links.

The goals in the pattern and the links are then added to the organizational goal model, while the business process building block related to the chosen strategy is added as a sub-model to the Increase Patient Safety stub (indicated by the long-dash-dot-dotted line in the right half of Figure 60). Figure 59 and Figure 60 together represent the output of applying the OCEM method: a refined GRL model of the organization with a model of the chosen business process options, together with the rationale for their selection. Applying the patterns in the PF can further refine the four sub-goals and linked stubs of Increase Patient Safety pattern. In order to support a satisfactory level of detail, the patterns of the patient safety family include ten layers of decomposition (of which only the top one is discussed here).

Applying the OCEM Algorithm

The first three steps of OCEM initialize the mainGoal with Increase Patient Safety (step S1) and then the initBP and toRefine variables (steps S2 and S3). Steps S4 to S7 add a contribution with weight 100 from the Increase Patient Safety goal in the pattern to the Increase Patient Safety of orgGM (see Figure 59). The next corresponding goals in orgGM and the Increase Patient Safety pattern are linked (steps S8 to S15). In steps S11 and S12, two contributions with weight 100 are added from the quality/cost softgoals in the pattern to the quality/cost softgoals of the organizational model. Then, a contribution with weight of 100 is added from the Implement Advanced Infrastructure task in the organizational model to the Deploy Advanced Infrastructures (the softgoal with the dependencies) in the pattern (steps S13 and S14). It possible that some elements in orgGM do not have links to the intentions in the pattern (e.g., Support Research on Adverse Events is not addressed by the current PF).

At this point, the toRefine list contains the Increase Patient Safety goal and a simple process with one stub, which are extracted to NextGoal and NextStub in step S16.1 and removed from toRefine in step S16.2. The next step, S16.3, applies all strategies to find the best solution for Hospital A. In this case, there are two strategies defined for Increase Patient Safety pattern as shown in Figure 33 on page 71. Figure 60 (right) shows the result of the second strategy (B) because it yields the better result than the first strategy (A), given that the organizational goal model places more value on quality than on cost and already has some advanced infrastructure available. For example, the results

are 98 vs. 98 for the main goal Increase Patient Safety, 50 vs. 98 for Increase Quality of Care, -45 vs. -75 for Decrease Cost, and 59 vs. 75 for the main stakeholder for strategy (A) vs. (B), respectively. A different healthcare institute with more focus on cost than quality and no advanced infrastructure available would likely see the first strategy (A) win over the second strategy (B). Note how the satisfaction values of the goals Make Decision and Apply Knowledge are not the same as the satisfaction values of Collect Data and Generate Informative Outcome Information as defined by the second strategy (B), because the dependencies restrict the satisfaction values of Make Decision and Apply Knowledge to lower values (in a dependency, the depender cannot be more satisfied than the dependee).

Step S16.4 then adds NextGoal, intentions of the Increase Patient Safety pattern that are chosen regarding the selected strategy, and their links to the goal model of Hospital A. Then, step S16.5 adds the business process building block of the selected strategy (B) as a plug-in to the stub of the simple process (indicated by the long-dash-dot-dotted line in Figure 59). Step S16.6, S16.7, and S16.8 establish realization links between the Increase Patient Safety goal and the corresponding business process building block (shown in Figure 60) as well as between Collect Data, Generate Informative Outcome Information, Make Decision, and Apply Knowledge as the sub-goals of the pattern and the stubs of the business process building block (not shown in Figure 60 for reasons of simplicity). Finally, steps S16.9, S16.10, S16.11, and S16.12 add all four sub-goals and linked stubs to the toRefine list to be evaluated in the next iterations of the loop. This is done because all sub-goals are required by strategy (B). In the next iteration of step S16, the three strategies of the Collect Data (see Figure 35 and Figure 36) are assessed and the best strategy is chosen (not shown in Figure 60). Continuing this iteration further refines the goal model of Hospital A and selects the business process building blocks for all of its goals.

Figure 59 and Figure 60 together represent the output of first iteration of OCEM method for the highest-level pattern in the patient safety PF: a refined GRL model of the organization (O1) with a set of the chosen business processes (O2) and realization links between them (O3).

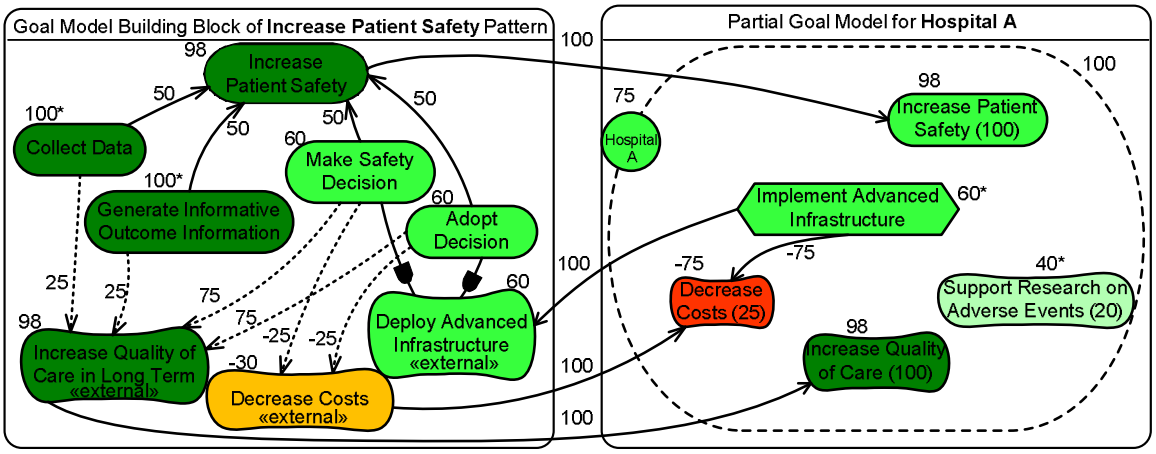

Figure 59 Using OCEM: linking goal model building blocks to the organizational goal model, with evaluations

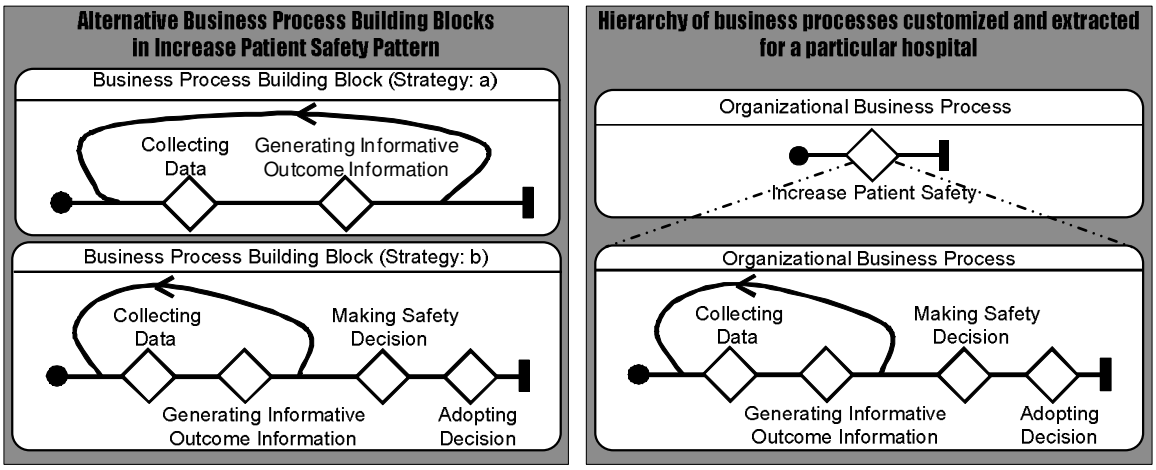

Figure 60 Using OCEM: business process building blocks, with strategy "B" being selected

6.4 Summary

This chapter introduced OCEM, a method that enables the selection of appropriate solutions from a pattern family in the context of a particular organization within the domain. The OCEM algorithm provides a systematic approach for customizing and extracting models based on the knowledge embedded in pattern families. The outcome of this algorithm is a goal model customized to the requirements of stakeholders and accompanied by a suitable business process where traceability links between goals and processes are

documented. An example related patient safety illustrated how OCEM is used. OCEM is important as it enables knowledge reuse across organizations, while taking into consideration the specifics of their context.

The next chapter presents and discusses the evaluation of the Goal-oriented Pattern Family framework based on case studies and a comparison with related work, together with limitations and threats to the validity of this evaluation.

Chapter 7. EVALUATION

For evaluating the GoPF framework, two extensive pattern families were created for the patient safety and the aviation security domains. Healthcare institutes interested in patient safety and organizations that need to regulate aviation security can benefit from these applications of the framework. The GoPF framework (i) lays down a foundation for capturing knowledge about business goals and processes, (ii) provides methods for reusing this knowledge within one organization or across similar ones by extracting and customizing models for specific stakeholders, and (iii) enables evolution of the knowledge when new problems and solutions emerge.

The patient safety case study was used as an ongoing example throughout this thesis. Therefore, section 7.1 is limited to a summary of the experience gained with the GoPF framework in the patient safety domain. Section 7.2 provides more detailed insight on the experience gained using GoPF in the aviation security domain, including the use of indicators in the patterns. The evaluation also includes an assessment of the GoPF framework and a comparison to closely related work against nine dimensions (section 7.3). Finally, section 7.4 discusses several limitations and threats to the validity of this work.

7.1 Case Study 1: Patient Safety Domain

The patient safety domain was selected as one of the case studies for the evaluation of the GoPF framework because of the recent push for healthcare reform that has caused healthcare organizations to focus on better ways to provide high quality and safer treatments while reducing related costs [134]. This case study was developed as part of a collaborative project involving the University of Ottawa and a teaching hospital in Ontario, which made it possible to access the necessary information and stakeholders for doing research on pattern families in the patient safety domain. The patterns were mainly collected between September 2008 and September 2009 and were then refined as GoPF became more formal.

Healthcare institutions, which manage hundreds of clinical and other types of business processes, strive to improve the safety of their patients. Yet, every year, thousands of patients suffer from adverse events, which are defined as undesirable outcomes caused by healthcare business processes. Decreasing adverse events by improving these processes forms the scope of the patient safety domain targeted here.

In [39], we showed that goal and business process modeling with URN could be used effectively in this domain in order to capture problems and their solutions. Through applying the methods of GoPF at different units of the hospital (Cardiac Surgery Intensive Care, Intensive Care, and General Internal Medicine), a collection of 32 patterns, grouped into a pattern family, were discovered and documented. This pattern family contains 32 GRL diagrams with 145 intentional elements, together with 82 UCM diagrams with 176 stubs, all of which being organized in a structure that is 10 levels deep.

For this thesis, the following activities were completed:

- Documentation of the Patient Safety PF based on observations and domain expert interviews in different departments of a real teaching hospital. Chapter 4, Chapter 5, and Chapter 6 provide extracts of this pattern family, and the complete PF can be found online at [135];
- In order to support the well-formedness of pattern families, constraints in OCL (Appendix A) were developed and then tested using a formalized subset of the Patient Safety PF implemented in the USE environment [128][129]; and
- In order to validate the algorithms that formalize the methods of the framework and demonstrate their potential for automation, the four evolution mechanisms described in Chapter 5 were implemented in Java [136] and tested against a subset of the Patient Safety PF described with data objects. These Java programs constitute a reusable library of transformations for manipulating pattern families.

The feasibility of the methods provided in Chapter 4, Chapter 5, and Chapter 6 for capturing patterns, creating pattern families, evolving them, creating customized models for

healthcare organizations based on patient safety pattern family was illustrated through the examples found in those chapters.

As an example of another (and somewhat unexpected) concrete impact of this work, part of the resulting pattern family was used as the foundation for creating an *adverse event management system*, a software application that supports the documentation of potential healthcare adverse events through prospective surveillance done by specialized observer nurses and the classification and analysis of events by a committee of physicians. The prototype system was re-implemented at the hospital as part of a larger incident reporting system, and is now fully deployed.

7.2 Case Study 2: Aviation Security Domain

Some regulators in the aviation security domain are exploring outcome-based approaches toward regulations. Outcome-based regulations focus on measurable goals rather than on prescriptive ways of achieving these goals [137]. As regulators start evolving existing prescriptive regulations towards an outcome-based approach, it becomes important to reuse knowledge about existing problems and solutions. However, these organizations face some challenges for establishing the new approaches. The GoPF framework can be used to address some of the challenges faced in this domain.

This second case study takes the opportunity of using GoPF to create a pattern family that targets a new domain, namely aviation security regulatory compliance. This PF was created as part of a collaborative research project involving the University of Ottawa and a regulator for aviation safety and security, on the construction of performance framework. Interactions with different stakeholders having similar yet different objectives enabled exploring the feasibility of using the framework's methods for creating patterns in this new context. Note that the sensitive and confidential nature of this security work prevents the discussion of the details of the patterns, but generic examples and characteristics will be provided in the following subsections.

Using this case study helped to apply the full framework's infrastructure (refined based on the experience gained through the first case study) as well as the process of eliciting requirements leading to the creation of patterns and families [137]. Given the importance of measures in this outcome-based regulatory compliance context, the concept

of indicator from the framework is further emphasized as it enables the reuse of compliance measurement, in context.

7.2.1 Introduction to the Aviation Security Domain

Many existing regulatory compliance approaches are prescriptive, which means that they impose specific ways for regulated parties to comply. However, in some domains, regulators are now trying to focus on the regulation intentions (e.g., the goals) that matter most while enabling regulated parties to choose the business processes and implementation strategies that best suit their context. In such outcome-based or goal-oriented regulatory approaches, regulators must ensure that solutions chosen by regulated parties effectively satisfy the intent of the regulations, e.g., by measuring whether the outcome is satisfactory.

Carrying out an outcome-based approach usually depends on the capability to capture requirements from two angles. First, the regulator must define a hierarchy of goals and related measures needed for evaluating their satisfaction. Second, a hierarchy of business processes representing the regulatory strategies and best practices for achieving these goals at the organization's end needs to be captured.

As an example, the regulator for aviation security in Canada is reviewing some of its policies and regulations in order to see if moving from the current prescriptive style to an outcome-based style is warranted. Such changes lead to three major challenges for an effective and efficient implementation of this approach. First, it is necessary to capture the knowledge about the hierarchy of intentions and related processes, which represent problems and existing solutions. This is the knowledge that can be reused for achieving the desired outcome of regulatory responsibilities. Moreover, it is difficult to measure the satisfaction of regulatory goals for various reasons, such as a mismatch between indicators (when they exist) and supposedly corresponding goals.

Second, requirements in this domain are complex and difficult to model from scratch. In addition, there are different stakeholders with similar areas of responsibilities (e.g., those in charge of different aspects of aviation security). These stakeholders need different yet similar hierarchies of intentions. Despite the fact that they cannot reuse complete models of intentions and related processes, it is possible and beneficial to reuse

parts of existing goal and business process models as building blocks for new models. For this to happen, it is necessary to capture such knowledge in the form of reusable building blocks of goal and business process models. Moreover, ignoring the similarities of problems in the domain may lead to inconsistencies in goal and business process models that represent them. These issues underline the importance of reusing knowledge about problems (goals) and solutions (business processes) in particular regulatory domains.

Third, knowledge about problems and solutions in a given domain is gradually and constantly changing. Furthermore, regulators and regulated parties need to be accountable for their actions. Therefore, there is need to (i) retain the known problems and solutions at given times in the past, and (ii) enable evolving the knowledge and tracking such evolution when new lessons are learned.

In order to address these challenges, the GoPF framework is to target commonalities across objectives of many regulations. Organizations in the aviation security domain can benefit from using this framework by capturing knowledge in the form of goal and process model building blocks, enabling regulatory parties to build a hierarchy of goals and related processes that suits their context.

7.2.2 Outcome-based versus Prescriptive Approaches in Regulatory Compliance

Currently, regulators prescribe the solutions that fit the highest number of regulated parties (e.g., airports and airlines). These prescribed solutions fit the average conditions but may not suit other conditions outside a narrow average.

The outcome-based approach in the aviation security domain, an alternative to the current prescriptive approach, lifts the burden of implementing prescribed solutions from regulated parties. Instead, it delegates the details to those who implement solutions, yet it ensures that objectives of regulators are effectively achieved. This approach encourages regulated parties to comply to regulations with innovative solutions. It is also more flexible and well aligned with the spirit of the oversight responsibility of regulatory organizations. With the desired outcome-based approach, regulators are able to (i) define the important objectives of aviation security screening at different levels of abstraction and for different regulated parties, and (ii) create a system that takes available evidence data

(e.g., from audit information or from the execution of business processes) as input, evaluates indicators, and ensures proper satisfaction of goals. In this approach, the effectiveness of solutions is what will matter to security organizations. Compliance happens when the implemented solutions are effective toward achieving the required objectives.

A typical performance framework [138] can aim to (among other objectives) help develop performance-based standards, define measurable and traceable compliance goals, and define and assess performance expectations. Modeling was recently started of individual outcome-based regulations [139] with the User Requirements Notation. This work underlined the importance of reusing the domain knowledge and inspired capturing the knowledge in the form of patterns and families.

7.2.3 Areas of Aviation Security Screening Domain

In order to apply GoPF and create an aviation screening pattern family, collaborating with stakeholders and domain experts is necessary to understand the domain requirements.Requirements for screening come from multiple regulations. An important objective of a regulator in the aviation security domain is to oversee the following aspects of screening processes [140]:

- *Pre-board screening* includes the screening of (i) *passengers* and (ii) their *carry-on baggage*;
- *Hold-baggage screening* includes screening of checked bags; and
- *Non-passenger screening* is applied to non-passengers (e.g., employees) entering restricted areas.

For each of these aspects, the regulator needs to ensure that the quality of screening complies with the regulations and effectively mitigates relevant risks. Many commonalities among these three aspects can be exploited to create patterns.

7.2.4 Motivation for Using GoPF and Creating a Pattern Family

Shamsaei *et al.* demonstrated that GRL supplemented with indicators can be used to model regulations and organizational objectives, and to measure the compliance of these organizations and their processes against regulations [141]. This work is at the basis of the modeling approach presented by Tawhid *et al.* [139]. However, the lack of a hierar-

chical goal model of the regulator's objectives accompanied by suitable indicators is one of the challenges normally faced while implementing outcome-based approaches. The screening oversight is complex and leads to a large hierarchy of intentions that represents the objectives and concerns of a typical aviation security regulator at different levels of abstractions. The goals in this hierarchy can be achieved through different potential solutions. However, the best solution is not the same under all conditions. Depending on a given context, the best solution is the one that results in higher levels of satisfaction of goals in the hierarchy. Hence, it is difficult to build from scratch goal and process models that focus on the outcome.

In order to successfully adopt the outcome-based approach in this domain, stakeholders and domain experts need ways of capturing different problems, solutions, contexts and their relationships. In order to reuse these problems and solutions within one organization or across similar regulatory organizations, there is a need for means of selecting the best solution depending on a context. The dynamic nature of this domain (e.g., due to the discovery of new security threats) also highlights the need for systematic evolution of captured problems and solutions. Moreover, there is a need to keep the history of gradual changes over time. On one hand, this helps regulators to be accountable and to understand the rationale of the past decisions. On the other hand, past versions of pattern families can be used for learning lessons and turning them into new effective strategies. Finally, according to informal feedback from stakeholders and experts in the domain, addressing the above concerns is critical in the successful adoption of outcome-based approaches that will be applied for improving the oversight screening process.

In this case study, GoPF is used for creating patterns and a pattern family that enables the modeling and reuse of the commonalities and differences of measurable outcome-based regulations. This can lead to the successful adoption of an outcome-based approach where regulated parties are both compliant and effective. GoPF lays down the infrastructure for (i) capturing different screening-related activities that lead to achieving similar goals from different departments and aspects of regulations (capturing reusable knowledge), and (ii) measuring the impact of solutions on the hierarchy of requirements. The next subsection illustrates the results of applying the process introduced in Chapter 4

to the task of building a GoPF-based pattern family for the aviation security screening domain.

7.2.5 Building an Aviation Screening Pattern Family

While interacting with a few stakeholders and domain experts involved in our collaborative research project, we were able to observe and analyze several requirements. GoPF's family development method (FDM in Chapter 4 with its resulting artifacts illustrated in Figure 26) was used for building the aviation screening pattern family. Following the enumerated steps, requirements elicitation sessions composed of stakeholders were first performed, followed by individually interviews. This resulted in the creation of goal models at different levels of abstraction. This process was repeated for oversight of screening in three different regulation units: passenger, carry-on bag, and hold-baggage screening.

The similarity of goals and responsibilities in these areas enabled the recognition of the repetitive and reusable goal and business process models. These models are captured in the form of GRL graphs and UCM diagrams, respectively. In order to improve correctness and accuracy, the models were discussed and validated with stakeholders and domain experts.

In this pattern family, each goal model building block is linked with the corresponding business process building blocks. For instance, the sample goal model building block illustrated in Figure 61(c) together with business process building blocks shown in Figure 61(d) form a pattern called *ComplianceToProcedureX*. As illustrated in Figure 61(a), each discovered pattern is associated with patterns that refine it. Although the details of this non-trivial pattern family cannot be discussed, its structure can briefly be characterized: 13 patterns, 4 levels of depth, 58 goals, 36 indicators, and 28 business processes.

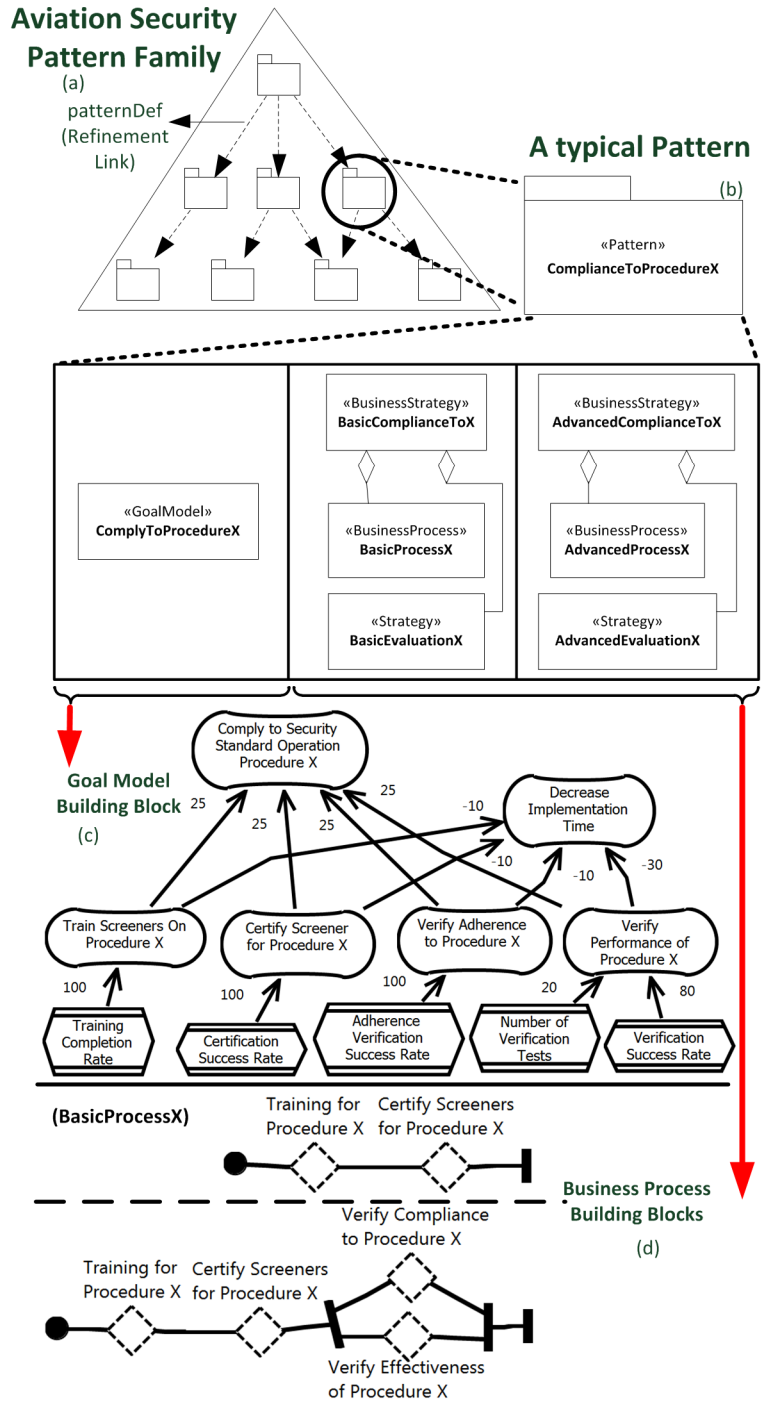

Figure 61 Artificial example : (a) Pattern family, (b) Pattern and its internal structure, (c) Goal model building block, (d) Business process building blocks

Note the additional presence of indicators in goal model building blocks, as shown in Figure 61(c). Relevant indicators (which are GRL intentional elements) from the pattern family are transferred to the goal model of the target organization or department after the

application of the OCEM method. This enables the organization or department to get a first set of indicators that can be used to measure objectives (which are related to compliance in this case study). Indicators in URN models can similarly improve reuse in the context of performance management of medical processes such as the palliative care processes recently documented by Kuziemsky *et al*. [143], which include indicators.

7.2.6 Evaluation of GoPF in the Aviation Security Domain

The GoPF framework was used with a typical regulator for building a pattern family that captures the intentions, expected outcomes, and processes of a given domain (screening for aviation security). One aspect of using GoPF for this case study is the modeling of indicators, which can help regulators measure compliance and effectiveness.

This pattern family was presented to and validated with stakeholders and domain experts. The feedback was encouraging and provided ad hoc evidence of interest and potential benefits in using GoPF in a regulatory domain. First, the aviation screening pattern family captures knowledge about problems and solutions at a given time. This helps regulatory parties by enabling the reuse of goal and business process model building blocks. It is also helpful to regulatory organizations by keeping track of the evolution of the domain's problems and solutions on one hand and by shedding light on the rationales of past decisions on the other hand. The latter is particularly helpful for the accountability of such organizations.

Second, the pattern family was successfully evolved on a small scale to address the concerns and new requests received from stakeholders when creating the pattern family. Such continuous evolution helps maintain the accuracy of the contained knowledge in a pattern family. Finally, stakeholders confirmed that patterns in this family are valuable vessels for reusing the knowledge for goal and business process modeling in other areas of screening. For instance, non-passenger screening, which was outside the scope of the case study, can benefit from using the patterns modeled in the screening pattern family. Moreover, it is also expected that the knowledge captured here can be potentially reused in other similar domains such as aviation safety or screening domains outside of aviation.

7.3 Comparison with Related Work

This section compares the GoPF framework against similar alternative approaches, which are introduced in the literature reviews in Chapter 2. Table 7 represents nine important dimensions of comparison between GoPF and other related work.

Table 7 Dimensions of comparison between GoPF and related work

Comparison Dimensions	Descriptive Questions
Requirements Models Used For Model Transformation	Are requirements models being used to begin creating a chain of model transformations?
Formalized Pattern Specification	Is there any formal foundation for specifying problems, solutions, and forces in the pattern?
Goal Model Inclusion	Are the intentional requirements of stakeholders being captured in the form of goal models within patterns?
Business Goals and Processes Linkage	Are the reusable business goals and processes connected?
Pattern Organization	What is the mechanism for organizing patterns and capturing their relationships?
Pattern and Family Evolution	Is it possible to change patterns and collections of patterns to reflect the changes that may happen in the corresponding domain?
Goal-oriented Solution Customization and Extraction	Are the requirements of stakeholders being used for extracting the knowledge from the pattern in order to extend the organization goals build relevant business process models?
Domain Specialization	Do patterns capture the domain-specific knowledge about its recurring problems and solution?
Pattern and Family Creation	Are there any guidelines for building patterns and organizing them in collections?

Table 8 provides a summary of the assessment of the various approaches (including the GoPF framework) against these dimensions, hence enabling a comparison between GoPF and related work. The following subsections provide a more detailed assessment of the various approaches per dimension and contrast how the GoPF framework stands out.

Table 8 Summary of comparison between GoPF and related work

Dimension / Related Work	Require-quire-ments Models Used For Model Transfor for-mation	Formal-ized Pattern Specifi-cation	Goal Model Inclu-sion	Business Goals and Process-es Linkage	Pattern Organi-zation	Pattern and Family Evolu-tion	Goal-oriented Solution Custom-ization and Extrac-tion	Domain Speciali-zation	Pattern and Family Creation
MDA [58]	No	No	No	No	No	No	No	No	No
Mussbacher *et al.* [71]	No	Partially	Yes	Yes	Informal	No	No	No	No
Gross and Yu [77]	Partially	Partially	Yes	No	Informal	No	Partially	No	No
Chung *et al.* [78]	No	No	Partially	No	No	No	Partially	No	No
Konard and Cheng [70]	No	Partially	No	No	Informal	No	No	Yes	No
Andrade [79] [80]	No	Partially	No	No	Informal	No	No	Yes	Partially
Markovic and Kowalkiewicz [99]	Partially	No	Yes	Yes	No	No	Partially	Yes	No
Rimassa *et al.* [100][101]	Partially	No	Partially	Yes	No	No	No	Yes	No
Stirna *et al.* [94]	Partially	Partially	Yes	Partially	Informal	No	Partially	Yes	Yes
Zhao *et al.* [85]	No	No	No	No	No	Partially	No	No	No
Dong *et al.* [86][87]	No	No	No	No	No	Partially	No	No	Partially
Aoyama [88]	No	Partially	No	No	Informal	Partially	No	No	No
Lapouchnian *et al.* [13]	Yes	No	Partially	Partially	No	No	Yes	No	No
Yu *et al.* [114]	Yes	No	Partially	Partially	No	No	Yes	No	No
Liaskos *et al.* [115]	No	No	Partially	No	No	No	Partially	No	No
Hui *et al.* [19]	Partially	No	Partially	Partially	No	No	Partially	No	No
Wang *et al.* [20][111][112]	Partially	Yes	Yes	Yes	Formal	No	Partially	Yes	No
Čiukšys and Čaplinskas [16]	Partially	Yes	Partially	Yes	No	No	Partially	Yes	No
Filipowska *et al.* [113]	Partially	No	No	No	No	No	No	Yes	No
Wen *et al.* [69]	Partially	Partially	Yes	Partially	Informal	No	Partially	Yes	Partially
GoPF framework	***Partially***	***Yes***	***Yes***	***Yes***	***Formal***	***Yes***	***Yes***	***Yes***	***Yes***

7.3.1 Requirements Models Used For Model Transformation

The GoPF framework introduces patterns and pattern families that capture and organize requirements. It puts the emphasis on models that capture stakeholder intentions and link them to models that specify their operationalizations in terms of business processes. GoPF provides steps that take the initial organizational goal model as an input and iteratively build hierarchical business goal and process models. These models can then be used as inputs for building more concrete models required for creating software applications using a MDE-based approach. This can be considered to some extent as an extension of MDA [58] by including the requirements models at the beginning of the model transformation chain. However, GoPF does not include any automated model transformation mechanism for this purpose. Table 9 highlights other significant related work in this dimension (with a focus on the approaches that do well against this criterion).

Table 9 Summary of comparison based on requirements models used for model transformation

Related Work	Assessment of Related Work
MDA [58]	**No.** Modeling requirements of stakeholders are outside the scope of MDA.
Lapouchnian *et al.* [13]	**Yes.** These two approaches begin with goal models that represent the requirements of stakeholders and produce a model that illustrates the better configuration. Note that these approaches differ from GoPF as they do not reason about business processes.
Yu *et al.* [114]	

7.3.2 Formalized Pattern Specification

The GoPF framework includes a metamodel for capturing patterns. Patterns are described with URN by formalizing:

- The description of the problem and the forces that are involved with GRL's intentional elements as well as contribution links as reusable goal models (goal model building blocks);
- The reusable business solutions composed of:

- The UCM models representing business processes that provide a more detailed description of the behavior and structure of the solution (business process building block); and
- The URN evaluation strategies representing effects of corresponding business processes.

The approach taken in GoPF for presenting ways of reasoning about patterns is similar to approaches suggested by Mussbacher *et al.* [71], Gross and Yu [77], and Chung *et al.* [78] (see Table 10). However, there are important differences. GoPF is more concerned with formalizing, through a metamodel, both problem and solution sections of the patterns along with the forces in the context, which leads to capturing the knowledge of the domain.

Note that formalization of pattern specifications comes at the price of complexity of pattern building. For example, as opposed to frameworks where patterns have a natural language description, PF analysts in GoPF must be familiar with URN and with the steps required to build business goal and process models. This characteristic of GoPF can be seen as a disadvantage in terms of usability compared to approaches that have a less formal structure for capturing knowledge. Therefore, the framework would be suitable to those users who can accept such challenge.

Table 10 Summary of assessment based on the formalized pattern specification dimension

Related Work	Assessment of Related Work
Mussbacher *et al.* [71]	**Partially.** This approach uses URN models for formally representing problems, solutions, and forces. However, it does not provide a formal foundation such as metamodel for capturing the elements of patterns, nor does it include well-formedness rules.
Gross and Yu [77]	**Partially.** This approach represents the goals of patterns using Non-Functional Requirements Goal Graphs. However, there is no suggestion for an underlying foundation for formally specifying both problems and solutions.
Chung *et al.* [78]	**No.** This approach uses object-oriented, goal-driven, and pattern-based methodologies for developing "good" software architectures. Although the Non-Functional Requirements framework used here provides a formal notation for capturing problems, the paper does not suggest a formalization of patterns.
Wang *et al.* [20][111][112]	**Yes.** They suggest a metamodel that formalizes their O-RGPS framework for capturing domain knowledge. This work, unlike GoPF, does not modularize and organize this knowledge in the form of pattern families.
Čiukšys and Čaplinskas [16]	**Yes.** They leverage an ontology and provide a metamodel for capturing domain knowledge. However, their primary objective is to reuse business processes and not to extend the initial goal model of the stakeholders.

7.3.3 Goal Model Inclusion

GRL's ability to represent a hierarchy of organization goals (and their interactions) enables the GoPF framework to capture the relevant reusable fractions of this hierarchy within the patterns. Furthermore, patterns are also formalized with URN in a way such that one pattern groups similar solutions that address a recurring problem, with their trade-offs.

Konard and Cheng's approach [70] uses UML models for capturing structural and behavioral aspects of requirements patterns. However, it does not include a formal foundation, such as goal models, for capturing intentions and connecting them to the behavioral models that realize them. Similarly, Andrade [79][80] uses UCM for formally capturing the commonalities of solutions in her approach toward requirements patterns in

wireless systems. Again, this approach does not include a formal description of the goals that accompany the suggested patterns.

Patterns in the approaches suggested by Mussbacher *et al.* [71] and Gross and Yu [77] include goal models while the approach in and Chung *et al.* [78] link patterns to such models. Table 11 presents relevant related approaches that include goal models.

Table 11 Summary of assessment based on the goal model inclusion dimension

Related Work	Assessment of Related Work
Konard and Cheng [70]	**No.** This approach uses the UML notation for formally defining behavioral and structural aspects of patterns but does not include goal models.
Andrade [79][80]	**No.** Andrade's approach for analysis patterns in the mobile systems domain does not include goal models.
Mussbacher *et al.* [71]	**Yes.** This approach uses URN-based goal models. It differs from GoPF in the sense that it neither includes traceability links nor a systematic approach that helps capturing, maintaining, and using patterns.
Gross and Yu [77]	**Yes.** This approach suggests using Non-Functional Requirements Goal Graphs for better describing the problems that are being addressed by patterns.
Chung *et al.* [78]	**Partially.** Although the Non-Functional Requirements framework is used to capture intentions of requirements, goal models are not part of patterns.
Markovic and Kowalkiewicz [99]	**Yes.** They use goal models to improve Business Process Management methodologies. This approach enables querying the knowledge. However, this knowledge is not captured in the form of patterns.
Stirna *et al.* [94]	**Yes.** Their Enterprise Knowledge Patterns include goal models to represent recurring problems. This approach does not provide a formal foundation for the models used in their suggested patterns. In addition, the organization of patterns is informal and done through textual references.
Wang *et al.* [20][111][112]	**Yes.** The metamodel for O-RGPS includes a layer for capturing goals (G-net). However, this work does not capture and organize the domain knowledge in the form of patterns and families.
Wen *et al.* [69]	**Yes.** In their analysis framework, *i** is used for modeling intentions as part of creating requirements patterns. This knowledge is then used to address the problems in a specific domain, namely medical information security. However, no guideline is suggested for capturing the patterns and for using them.

7.3.4 Links between Business Goals and Processes

Problems and solutions are two sides of the same coin. They can be best understood when goal and business process views are captured together. In the GoPF framework, such integration is formalized by two means. First, the metamodel defines a pattern as a container of a goal model building block on one side and a set of pairs of <business process building block, evaluation strategy> on the other side. This structure is helpful because patterns include both perspectives. In addition, the evaluation strategies capture the effect of relevant business processes on the corresponding intentions in the goal model. Second, the metamodel includes realization links between the contributing elements of a problem depicted with GRL and elements of solutions represented in UCM diagrams with process stubs and activities.

Markovic and Kowalkiewicz [99] do not use patterns, but they provide an approach that includes a business process ontology and a goal modeling notation. The links between these views then enable the integration of the intentional and business process perspectives. This ontology is used for performing automated analysis and for querying the contained knowledge. This is also feasible in the GoPF framework (e.g., by creating OCL queries on the URN model). However, unlike for the GoPF framework, no method or framework is provided to enable the creation of patterns in order to reuse solutions to recurring problems.

Rimassa *et al.* [100][101] leverage agent technology and introduce the goal-oriented business process modeling notation (GO-BPMN), which is a BPMN-based visual language for business process modeling enriched by goal modeling. However, unlike GoPF, the relationships between goals and business process are limited and the notation does not support different levels of abstraction. Table 12 highlights some of the closest related work and how they link business goals and processes.

Table 12 Summary of assessment based on the links between business goals and processes dimension

Related Work	Assessment of Related Work
Mussbacher *et al.* [71]	**Yes.** This approach uses GRL graphs in connection with UCM diagram. However, their approach does not include a systematic way of maintaining patterns.
Markovic and Kowalkiewicz [99]	**Yes.** Their approach suggests linking goal models to business processes for improving Business Process Management methodologies. This enables querying the captured knowledge. However, this knowledge is not captured in the form of patterns. Furthermore, it does not include steps needed for capturing such knowledge.
Rimassa *et al.* [100][101]	**Yes.** They provide limited linking between goals and business processes through their visual language (GO-BPMN). However, the links are limited and do not support organizing hierarchical knowledge.
Wang *et al.* [20][111][112]	**Yes.** Their metamodel for ontology-based O-RGPS framework enables lining the goals in G-net layer to the processes in P-net layer. However, this work does not capture and organize the domain knowledge in the form of patterns and families. In addition, it does not provide steps for capturing the domain knowledge.
Čiukšys and Čaplinskas [16]	**Yes.** They leverage an ontology and provide a metamodel that enables linking goals to business process. However, their primary objective is to reuse the business processes and do not extend the initial goal model of the stakeholder.

7.3.5 Pattern Organization

The framework metamodel introduces pattern families, which organize patterns at different levels of abstraction and enable navigation from more abstract problems and solutions to more concrete ones. Furthermore, this organization of patterns addresses different interests of stakeholders where intentions, while being considered as elements of solution for one stakeholder, are also viewed as main goals for another stakeholder. The organization of a pattern family is captured through refinement links between individual patterns when one refines the other.

Stirna *et al.* in [94][95][96] propose Enterprise Knowledge Patterns for capturing patterns and for managing knowledge. Enterprise Knowledge Patterns contain interrelated goal, business process, and concepts models. They describe reusable solutions for en-

terprise problems. This approach however does not provide a formal foundation for the models and their relationships. In addition, it does not provide a formal foundation for relationships between patterns, and domain knowledge is managed by capturing textual references. The GoPF framework, on the other hand, is built upon a metamodel expressed as a profile of the URN standard, along with a method that enables automated transformations. Moreover, the business strategies in the framework enable different solutions to be used for the same pattern depending on the forces and goals characterizing the target organization. Although patterns are not explicitly used by Wang *et al.* [20][111][112], their approach has a formal foundation for capturing domain knowledge (see Table 13).

Table 13 Summary of assessment based on the pattern organization dimension

Related Work	Assessment of Related Work
Stirna *et al.* [94]	**Informal.** Enterprise Knowledge Patterns intend to capture and manage domain knowledge using patterns. However, the pattern organization is informal and takes place through textual references. In addition, this approach does not provide a formal foundation for patterns.
Wang *et al.* [20][111][112]	**Formal.** Their metamodel for the ontology-based O-RGPS framework lays down a formal foundation for organizing goals, processes, roles, and services. This approach does not use patterns explicitly, which may negatively affect reusing the goals and processes. GoPF on the other hand explicitly captures the domain knowledge in the form of reusable blocks.

7.3.6 Pattern and Family Evolution

GoPF's evolutionary mechanisms are designed to maintain the knowledge captured in pattern families and manage change.

Zhao *et al.* [84][85] propose and approach for the evolution of design patterns and pattern-based designs. Similarly, Dong *et al.* [86][87] propose an approach for the evolution of design patterns. Aoyama [88] also proposes an evolutionary mechanism based on a more formal representation of patterns provided in his research. They are different from the evolutionary mechanisms in GoPF in two ways: (i) these approaches focus on design patterns, and (ii) evolution is limited to variations of the initial pattern. Table 14 summarizes these approaches.

Table 14 Summary of assessment based on the pattern and family evolution dimension

Related Work	Assessment of Related Work
Zhao *et al.* [85]	**Partially.** They provide transformation approaches for evolving design patterns. However, they are limited to evolving design patterns to variations of initial ones. GoPF on the other hand provides mechanisms that enable evolution of patterns and pattern families so that they reflect the current knowledge of a given domain.
Dong *et al.* [86][87]	
Aoyama [88]	**Partially.** He proposes a set of graphical notations, and collects evolved patterns into families. This approach focuses on the evolution of design patterns. It also introduces the concept of collection as a set of variations for an initial pattern.

7.3.7 Goal-oriented Solution Customization and Extraction

Traditional process-oriented software development approaches put little emphasis on using goal models for eliciting and analyzing stakeholder requirements. Consequently, no links will be systematically established between business goals and processes at the time of finding solutions that realize the requirements of stakeholders. In order to address this disadvantage, the GoPF framework provides the OCEM method for extracting and customizing solutions in the context of an initial organizational goal model. This method is carried out iteratively and gradually extends the goal model of organization while considering the forces and conditions of the stakeholders and organization. Furthermore, this method also builds a hierarchy of business process that is realizing the corresponding goals in the extended goal model.

Lapouchnian *et al.* [13] also propose an approach for capturing the needs of stakeholders in goal models and for annotating them. This enables reasoning and selecting processes and finding customization alternatives that best accommodates stakeholders' goals. In another research, Yu *et al.* [114] propose a two-step approach for reasoning and selecting configuration alternatives by utilizing reverse-engineered goal models. Similarly, Liaskos *et al.* [115] provide an approach for configuring software applications. In this approach, a goal model captures the requirements of stakeholders and is used for reasoning about the best configuration alternatives. Yu's approach and Liaskos' enable configuration of the software application for realizing the goals of stakeholders, yet they have two limitations. First, these approaches are mainly using goal models and do not

include reasoning about business processes. Second, in order for these approaches to configure software applications, they must be accompanied by a goal model representing the alternatives.

Hui *et al.* [19] also propose a framework that customizes software applications in the context of goals, skills, and preferences of stakeholders. In her approach, a goal model represents stakeholder requirements while the leaf intentions are mapped to class diagrams that capture the possible alternatives. This approach then uses the goal model for selecting the alternative that satisfies stakeholders in a given context.

These requirements-driven approaches tackle the gap between goals and their realization, with an emphasis on goal models. Similarly, this thesis aims to bridge the gap between business goals and business processes, but the solution presented here is different from the above approaches in three important ways.

First, unlike the mentioned approaches that create specific goal models for particular software applications, GoPF uses goal models to capture domain knowledge. Second, unlike approaches in [19][114][115] that configure implemented business processes, GoPF provides different ways of achieving stakeholder goals at the business process level rather than at the configuration level. Third, unlike [13], GoPF captures goal models and business process models separately along with traceability links between them. Separation of goal models and business process models increases the maintainability of patterns as well as their comprehensibility. In summary, GoPF aims to reuse domain knowledge to enable business process development in context while the approaches mentioned in this subsection are fine-tuned for requirement-driven customization of behaviors of applications after they are developed. Table 15 summarizes those approaches that attempt to extract and customize solutions based on goals of stakeholders.

Table 15 Summary of assessment based on the goal-oriented customzation and extraction dimension

Related Work	Assessment of Related Work
Lapouchnian *et al.* [13]	**Yes.** This approach suggests the augmentation of goal models so they represent the underlying business processes configuration alternatives. This is then used to tailor the business process for better addressing the requirements of stakeholders. This approach differs from GoPF in two ways: (i) it does not attempt to capture reusable knowledge across different organizations and (ii) it strictly couples the knowledge about process alternatives in the goal models.
Yu *et al.* [114]	**Yes.** This approach uses the goal model that represents the domain requirements and reasons about a configuration of an application (e.g., Firefox browser) that is best suited for a particular user. This approach is somewhat limited and mainly focuses on configuration.
Liaskos *et al.* [115]	**Partially.** This approach parameterizes the goal models and uses them to find the best alternative. The main objective of this solution is to enable automatic personalization of software applications (e.g., Mozilla Thunderbird email client).
Hui *et al.* [19]	**Partially.** In this approach, goal models are linked to class diagrams and are used for the customization of applications.

7.3.8 Domain Specialization

The GoPF framework provides structures and methods for capturing the knowledge specific to a domain in the form of pattern families, where patterns highlight recurring problems and solutions. Furthermore, evolutionary mechanisms enable pattern families to change over time and better reflect current problems and solutions in a specific domain. In this sense, the objective of the GoPF framework, similar to that of domain engineering, is to provide a source of reusable knowledge about the problems in the domain and facilitate reusing the solutions, while teaching and communicating with stakeholders.

Wang *et al.* have suggested a domain modeling framework for networked software applications [20][111][112]. This framework uses five layers of ontologies for encapsulating the domain knowledge in different views. However, the main objective of this approach is to bridge the gap between process descriptions and web services by reusing the functionalities that are captured in the form of services. Similarly, an ontology-based method suggested by Čiukšys and Čaplinskas [16] facilitates reusing business pro-

cesses in a domain. Filipowska *et al.* [113] also suggest an approach that leverages ontologies and captures the domain knowledge about business processes. However, the primary objective of the approaches of Filipowska *et al.* and of Čiukšys and Čaplinskas is to capture the domain knowledge about business processes. The GoPF framework on the other hand captures the knowledge about goals/requirements of stakeholders and links them with processes that realize those requirements. Table 16 compares the approaches that enable capturing domain-specific knowledge.

Table 16 Summary of assessment based on the domain specialization dimension

Related Work	Assessment of Related Work
Konard and Cheng [70]	**Yes.** The suggested approach captures can be specialized for a domain. However, it does not include goal models.
Andrade [79][80]	**Yes.** Andrade's suggested approach for analysis patterns in mobile system domain does not include goal models.
Markovic and Kowalkiewicz [99]	**Yes.** Their approach attempts to improve Business Process Management methodologies and enables querying the domain knowledge. However, this knowledge is not captured in the form of patterns.
Rimassa *et al.* [100][101]	**Yes.** Their provided visual language (GO-BPMN) can be used for capturing the goals and business processes for specific domains. However, the models are limited and the approach does not support organizing hierarchical knowledge.
Stirna *et al.* [94]	**Yes.** They proposed Enterprise Knowledge Patterns for capturing the enterprise knowledge in the form of patterns. This is similar to GoPF but their approach does not provide a formal foundation for the suggested pattern. In addition, the organization of patterns is informal and no mechanism is suggested for evolution of patterns.
Wang *et al.* [20][111][112]	**Yes.** This approach can be used for specific domains. However, this work does not capture and organize the domain knowledge in the form of patterns and families.
Čiukšys and Čaplinskas [16]	**Yes.** Their approaches attempt to reuse the domain knowledge about business processes. However, this is the primary objective and the approach does not attempt to extend the initial goal model of the stakeholder.
Filipowska *et al.* [113]	
Wen *et al.* [69]	**Yes.** Requirements patterns in this analysis framework capture the domain knowledge by using *i** models in combination with problems frames. This knowledge is then used to address the problems in a specific domain, namely medical information security. However, no guideline is suggested for capturing the patterns and for using them.

7.3.9 Pattern and Family Creation

The GoPF framework provides a systematic way of eliciting goals/problems and capturing recurring solutions of a specific domain. This method involves interactions with stakeholders and domain experts as well as previous knowledge of PF analysts. In addition, this approach provides steps for building patterns from the elicited goal models and related pairs of business processes and their effects. Finally, the approach takes the new patterns as inputs for building new pattern families. Evolutionary mechanisms also enable maintenance of a pattern family for a specific domain when new patterns in that domain are built.

Wen *et al.* [69] suggest an approach for capturing requirements patterns containing the knowledge about the security problems in medical information system based on the *i** framework and on problem frames. However, they did not include systematic guidelines for capturing patterns but underlined such guidelines as a desired extension of their work. Stirna *et al.* [94] proposed guidelines for capturing patterns but the latter lack the steps necessary for systematically organizing patterns into a collection. Table 17 summarizes these approaches.

Table 17 Summary of assessment based on the pattern and pattern family creation dimension

Related Work	Assessment of Related Work
Stirna *et al.* [94]	**Yes.** Enterprise Knowledge Patterns include guidelines for capturing the domain knowledge in the form of patterns. These guidelines do not include systematic steps for organizing patterns into a family. Furthermore, this approach does not include a formal foundation for patterns.
Wen *et al.* [69]	**Partially.** Their analysis framework highlights the general steps needed for capturing requirements patterns by combining *i** models with problems frames. However, no guideline is provided for systematically capturing the patterns and using them.

7.4 Limitations and Threats to Validity

This section discusses several important limitations related to the GoPF framework and threats to the validity of the work surrounding its development and evaluation.

7.4.1 Case Studies and Pattern Family Construction

The patient safety pattern family is based on goals and actual processes observed in several *units* of one teaching hospital in Ontario [39]. However, the patterns would get stronger by taking into account the PF users' feedback when the patterns are used and by observing how relevant healthcare processes are done at *different hospitals*, perhaps even across provinces or countries, so that best practices spanning multiple organizations in different healthcare contexts can be shared. Note that, in order to mitigate this threat to some extent, informal discussions with adverse event researchers at another hospital in Quebec (involved in the same research project) were held to ensure some level generality in the pattern family.

For the second case study, the aviation security pattern family was built by observing the objectives and business processes of *different departments* of one Canadian regulatory organization contributing to screening security. Similarly, the patterns in this family would be stronger by observing *other organizations* contributing to screening security.

The patterns in both these families would also benefit from being collected and reused by more than one person, to avoid common bias issues. At the moment, the thesis' author is the only person who has created and used pattern families. Having other PF analysts would also help demonstrate the usability of the approach prescribed by the framework. The framework is also meant to be usable by third-party, intermediary organizations (e.g., consultants) as they are more likely to be observing goals and processes across organizations in a domain, but again it is premature to claim that this is beneficial to such intermediary organizations.

Finally, the framework was applied to only two (albeit very different) domains so far. More pattern families need to be created and used in other contexts in order to claim generality in terms of the domains where the framework is applicable.

7.4.2 OCEM Limitations

The current algorithm for Organization-driven Customization and Extraction Method (OCEM), namely Algorithm 5, does not necessarily guarantee that the best solution for the target organization will be extracted from the pattern family, even if that family is

perfectly modeled. The selection of goal strategies (and hence of corresponding business processes) depends on the quality of the input organizational goal model, and how sensitive the algorithm is to that input has not been studied. In addition, the selection of the best strategies for a pattern is based on local optimizations (in a step-by-step or decision-by-decision basis) rather than based on a global optimization across all patterns at once. The latter approach might indeed lead to better global solutions for the organization, but it would be more computationally challenging and less modular than the current OCEM approach.

OCEM was also only exercised for the patient safety pattern family, and this was not done in the context of a real organization. More validation is hence required on that side.

7.4.3 Precision of Goal Models in Patterns

Like for any goal model, one can question the precision and correctness of the contribution weights as well as initial satisfaction values in strategies. These can however be refined and validated with time and experience. Moreover, the strategy selection method is a comparative one (e.g., the algorithm chooses the "best" strategy amongst a few), so precise numbers are not always required in models. URN also allows modelers to start with qualitative values, when knowledge is sparse and fuzzy, and then to refine these links with more granular quantitative values as more precise knowledge is acquired.

7.4.4 Automation and Tool Support

Because the GoPF framework is using a profile mechanism to extend URN, current URN-based tools such as jUCMNav can be used to model patterns and pattern families through modeling their elements. However, modelers need to have a fair understanding of the URN language. In addition, the available tools lack the ability to help modelers evolve PFs (through the various evolution mechanisms seen in this thesis) and this currently limits the usability of the framework. The framework could benefit from GoPF-customization of current URN modeling tools (e.g., jUCMNav) and from seeing its various algorithms (Algorithms 1 to 5) integrated in them. The availability of Java implementations for the evolution algorithms [136] is a step towards this vision, but much work remains to be done in order to have a complete modeling and analysis environment.

7.5 Summary

The GoPF framework was used for capturing knowledge in two different real-world domains. This chapter evaluated the GoPF and its feasibility based on these two representative case studies. The patient safety case study, which has been an ongoing example throughout the thesis, was briefly reviewed while the second case study about outcome-based compliance in the aviation security domain was discussed with more details, especially in view of the use of indicators to measure goal satisfaction. In addition, different dimensions of comparison were introduced, and the framework was compared to related work along these dimensions. This comparison highlights that the GoPF framework is quite unique in its formalization, manipulation, and usage of patterns and families that exploit goals, business processes, and their relationships. Finally, the limitations of this thesis as well as threats to its validity were discussed. Following the approach of Hevner *et al.* [35] for the research methodology, the intentions of this chapter was to illustrate the usefulness and applicability of the GoPF framework.

The next chapter will summarize the intentions and structure of the GoPF framework and provide concluding remarks. It will also highlight future areas of research that can extend or improve this thesis.

Chapter 8. CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The value of software applications to an organization is based on how well business goals are satisfied through their use. Successful software development for an organization involves an accurate understanding of requirements, business goals, and (automated) business processes that can satisfy them. Therefore, it is necessary to build goal and process models that appropriately represent the requirements of stakeholders. However, defining such models from scratch is challenging. In spite of similarities between problems and solutions in a given domain, organizations often have difficulties in properly identifying, documenting, organizing, and reusing goals, business processes, and the links between these two views. In other words, capturing and reusing the domain knowledge is challenging and it is becoming increasingly difficult to ignore the benefits of knowledge reusability. Reusing domain knowledge captured in the form of *patterns* can often help address this issue.

Furthermore, there is a gap between goals and business processes. Whereas much attention has been devoted to the transition from business processes to supporting software products, the gap between business goals and business processes has received far less attention. Most approaches do not address the latter gap properly, and many even ignore goals altogether. A significant number of software development projects, in healthcare for example, yield disappointing results or are simply canceled because business processes are not aligned properly with business goals. Modeling business goals and processes separately is not sufficient to bridge this gap, and hence traceability must be taken into account.

Motivated by the above challenges, this thesis introduced the *Goal-oriented Pattern Family* (GoPF) framework, which is a pattern-based framework for capturing knowledge about a domain, structuring it, and reusing it in other contexts. The patterns contain the knowledge about recurring problems and solutions in the domain. They also include the potential effects of alternative solutions on the goals. This knowledge is cap-

tured in the form of reusable *goal model building blocks*, *business process building blocks*, *links* that define the realization relationships between them, and *evaluation strategy* for alternative solutions. This structure of patterns indicates which business processes alternatively realize particular business goals. Using the knowledge in the pattern along with contextual conditions and requirements of a given organization enables selecting and customizing known solutions for the organization. Hence, patterns help bridging the gap between requirements of a particular organization and corresponding (existing) solutions. These patterns are then organized in *pattern families*, i.e., collections of related patterns, where refinement links capture potential refinement relationships amongst patterns. The Framework Metamodel (FMM) lays down the foundation of pattern families and formalizes the patterns that capture knowledge. FMM is formalized as a profile of the standard URN modeling notation, which combines goals, processes, realization links between them, and potential effects of solutions, as well as refinement links in between patterns.

The GoPF framework includes methods for capturing the domain knowledge by locating recurrences and building patterns and pattern families through interaction with stakeholders and domain experts. However, changes in the requirements and context of stakeholders in a domain are unavoidable and therefore the problems and solutions within a domain are always evolving. Furthermore, the knowledge about business requirements in a domain is changing over time at a more rapid pace than for conventional software design patterns. One benefit of using the GoPF framework for capturing knowledge is that the patterns encapsulate recurring problems and solutions into loosely-coupled modules. Still, rapid changes in technologies, business environments, and concerns of stakeholder have highlighted the need for evolving patterns and pattern families. This is essential for maintaining the accuracy and usefulness of patterns. Introducing evolutionary mechanisms that systematically help maintaining pattern families is hence a necessity. Adding new patterns, removing obsolete ones, modifying patterns, and combing pattern families are core aspects of family evolution. GoPF is equipped with four automatable evolutionary mechanisms for the *extension*, *modification*, *elimination*, and *combination* of pattern families. According to the literature reviewed, this is the first attempt at describing and formalizing evolution mechanisms for patterns that integrate business goals

and processes. In order to demonstrate the feasibility of these mechanisms, a Java prototype application was developed that implements the four evolution algorithms. This application was used to test the algorithms against various manipulations of the patient safety pattern family.

The GoPF framework also provides a customization and extraction method that enables the selection of appropriate solutions in the context of a particular organization. The framework facilitates extracting stakeholders' business goals in the form of a hierarchical model and designing business processes with the help of reusable knowledge captured as patterns. This method uses the domain knowledge in the pattern families and an initial organizational goal model as inputs and then assesses the impact of alternative solutions for achieving the goals of stakeholders in a given organization with a step-by-step, top-down approach. The result is a URN model where the business goals have been refined based on the knowledge embedded in the patterns, where a suitable business process was constructed, and where traceability links between the two views are documented.

This thesis made the hypothesis that one can reuse and maintain, in a rigorous way, the knowledge about business goals, business processes and the links between them, captured as patterns to create suitable business processes in the context of a different organization. This hypothesis was demonstrated through the main contributions of this work, which include:

- The GoPF framework itself, which includes a Family Metamodel that formalizes patterns and families using a URN profile. Pattern families capture the knowledge of a given domain by including:
  - Goal model and business process building blocks representing recurrent problems (including indicators to measure them) and solutions, as well as realization links between them;
  - Effects of alternative solutions on stakeholders at the level of each pattern; and
  - Refinement links between patterns, hence capturing their hierarchy.

- A Goal-driven Method (GDM) that includes processes, mechanisms and algorithms for the creation and evolution (through extensions, modifications, eliminations, and combinations) of pattern families;
- An Organization-driven Customization and Extension Method (OCEM) providing steps for building customized goal and process models for specific organizations by reusing the captured knowledge in the families; and
- Two case studies in which the GoPF framework was used for capturing and reusing their relevant domain knowledge.

The validation was done through the above two realistic case studies (where real problems, organizations, and stakeholders were involved), but also through a comparison with related work, which highlighted the uniqueness of the framework and its potential. The GoPF framework is expected to have a positive impact on the scientific community through the formalization, evolution, and reuse of patterns in domain-specific business domains. From an industrial viewpoint, this framework will also help intermediary organizations (such as consulting firms) who are required to repeatedly create and document goal and process models for other organizations in their business domain.

8.2 Future Work

Several limitations of the GoPF framework and threats to the validity of the work were identified in the thesis, especially in section 7.4. These issues lead to different opportunities for improving and completing this work in the future.

8.2.1 Customization and Extraction of Models (and Propagation)

OCEM currently uses a "locally optimal" approach (in the context of a pattern) for selecting the most satisfying solution for the stakeholders. This implies a breadth-first evaluation of one level of strategies in the context of a specific pattern. However, alternative algorithms could also be explored, including depth-first or hybrid traversals, the evaluation of combined strategies from *N* levels deep in a look-ahead way, etc. There are also research opportunities for changing OCEM and assessing it against the current method. For instance, a constraint-oriented solving approach to goal evaluations suggested by Luo and Amyot [144] may enable a "globally optimal" approach. Such approach can find the

best solution in the context of the organization based on the possible evaluation strategies outside the scope of a particular pattern.

Furthermore, OCEM currently starts from an empty business process, but another possible start point could be the as-is organization business process, which PF users would like to refine or extend through the framework.

8.2.2 Usage of GoPF in Different Domains and by Different People

The GoPF framework was used on two real-life examples. However, it would be valuable for improving the framework to use it for capturing and reusing knowledge in other domains. Analysis of its usefulness and difficulties will help improving GoPF's metamodel and methods, especially in terms of generality.

The framework should also be used by different people to avoid bias and enable studying its usability. The experiments could be conducted by different PF analysts and PF users, who respectively will be in charge of capturing and reusing domain knowledge.

8.2.3 Usage of GoPF in Different Organizations of a Given Domain

GoPF's feasibility and usefulness was evaluated against different departments of two large organizations. The departments in each of these organizations were also large and had similar yet different requirements, which made them suitable candidates to play the role of individual organizations in their domains. Domain experts underscored the benefits of using the pattern families outside the scope of the examined organizations. Yet, a broader experiment would be valuable. Therefore, one next step could be to use the GoPF framework across multiple organizations (and not just departments) of a given domain. Such experiments, in addition to helping with the assessment of the usefulness and generality of the framework, may improve the generality of the patterns and families themselves, leading to higher benefits for future users.

8.2.4 Evolution of Pattern Families

Another future work item is to monitor a domain and evolve its pattern family over longer periods (possibly years). Such research would examine the sufficiency of the evolutionary mechanisms and incorporate learned lessons and emerging best practices. This may potentially highlight the need for new or improved evolutionary mechanisms.

8.2.5 Other Modeling Languages

At the time this research was started (in 2007), URN had advantages over similar alternatives for goal and process modeling because it integrates these two views, and tool support was readily available. However, it would be interesting to evaluate the sufficiency and completeness of URN-based goal and process models for organizations in different domains and compare the results to similar experiments with alternative standards and notations such as OMG's BPMN [49] and/or BPDM [53]. Using such alternative languages might also enable transferring the benefits of GoPF outside of the URN world.

8.2.6 GoPF for Building Goal Models

This thesis focused on capturing and reusing the knowledge about business goals and business processes. Currently, the GoPF framework is designed to reuse such domain knowledge by extending the initial goal model to a full-blown goal model that fit stakeholder requirements. The method in charge of building the goal model (OCEM) is simultaneously building a hierarchy of related business processes. Having both goal models and relevant business processes is the desired outcome for many stakeholders. However, there are domains, such as outcome-based compliance in general, in which some stakeholders are mainly concerned with goal models, and not so much in business processes. Thus, a future research item would be to customize the framework, and especially OCEM, for building customized goal models.

8.2.7 Product Line Software Engineering

In this thesis, reusable business goals and processes are combined in patterns as the underlying means of reusing domain knowledge. An interesting future work is to assess the possibility of customizing some GoPF methods (e.g., evolutionary mechanisms) to be used in product line software engineering. The research by Brown *et al.* [145], which augments feature models with UCM diagrams, could be a start point for this purpose.

8.2.8 Tool Support

Currently, jUCMNav provides tool support for creating goal model and business process building blocks in the form of GRL and UCM diagrams, respectively. Evaluation strategies can also be captured. The refinement links and realization links are specified using URN links. Finally, jUCMNav helps PF users build hierarchical goal and business pro-

cess models for the stakeholder reusing pattern captured by PF analysts. The case studies for patient safety and aviation security both made use of jUCMNav for capturing and reusing knowledge. However, jUCMNav currently does not provide support for visualization of the structure of patterns and families, and evolutionary mechanisms and OCEM are not integrated either. A tool more specialized for developing and using GoPF-based patterns would benefit both PF analysts and PF users.

8.2.9 Run-Time Approach

The GoPF framework is mainly concerned with assisting requirements engineers and business analysts by capturing domain knowledge and reusing it. Currently, reusing goals and selecting the solutions that can better satisfy stakeholders happens at design time. The framework is not attempting to dynamically reuse goals and processes at run-time, in an adaptive way, but this could be the topic on interesting research. One challenge here is providing a mechanism for applying the selected solution. For example, instead of having only one sub-process (plug-in map) selected per stub and resolving everything at design time, many sub-processes may be kept in dynamic stubs (which would allow for many sub-processes) and be resolved at run-time according to the current context of the organization as monitored by the indicators. Pourshahid *et al.* [146] have recently published a systematic literature review on business process adaptation that discusses work in that area, including work involving URN and aspect-oriented extensions to URN. In addition, the current GoPF framework captures the knowledge about the problems and solutions for the stakeholders in the organizations. However, run-time approaches may also need to capture patterns about actors outside the scope of organizations.

REFERENCES

[1] E. Simchi-Levi and P. Kaminsky, *Designing and managing the supply chain: concepts, strategies, and case studies*. Irwin/McGraw-Hill, 2003.

[2] T. H. Davenport, *Process innovation: reengineering work through information technology*. Harvard Business School Press, 1993.

[3] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure", *Distributed and Parallel Databases*, vol. 3, no. 2, pp. 119-153, 1995.

[4] R. K. Ko, S. S. Lee, and E. W. Lee, "Business process management (BPM) standards: a survey", *Business Process Management Journal*, vol. 15, no. 5, pp. 744-791, 2009.

[5] F. Alencar, B. Marín, G. Giachetti, O. Pastor, J. Castro, and J. H. Pimentel, "From i* Requirements Models to Conceptual Models of a Model Driven Development Process", in *The Practice of Enterprise Modeling*, LNBIP 39, Springer, pp. 99-114, 2009.

[6] Y. Huang, S. Kumaran, and K. Bhaskaran, "Platform-independent model templates for business process integration and management solutions", in *IEEE International Conference on Information Reuse and Integration (IRI)*, IEEE CS, pp. 617-622, 2003.

[7] D. Amyot and G. Mussbacher, "Bridging the requirements/design gap in dynamic systems with use case maps (UCMs)", in *Proceedings of the 23rd International Conference on Software Engineering*, IEEE CS, pp. 743-744, 2001.

[8] OASIS, "Web Services Business Process Execution Language (WSBPEL), version 2.0", April 2007.

[9] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.

[10] D. C. Schmidt, "Model-driven Engineering", *IEEE Computer*, vol. 39, no. 2, pp. 25-31, 2006.

[11] M. Bensaou and M. Earl, "The right mind-set for managing information technology", *Harvard Business Review*, vol. 76, pp. 118-129, 1998.

[12] T. Hoffman, "Study: 85% of IT departments fail to meet biz needs", *Computer World*, vol. 11, p. 24, Oct. 1999.

[13] A. Lapouchnian, Y. Yu, and J. Mylopoulos, "Requirements-Driven Design and Configuration Management of Business Processes", in *Business Process Management,5th International Conference, BPM 2007*, LNCS 4714, Springer, pp. 246-261, 2007.

[14] G. Mussbacher, D. Amyot, J. Araújo, A. Moreira, and M. Weiss, "Visualizing Aspect-Oriented Goal Models with AoGRL", in *2nd Int. Workshop on Requirements Engineering Visualization*, IEEE CS, p. 1, 2007.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[16] D. Čiukšys and A. Čaplinskas, "Ontology-based approach to reuse of business process knowledge", *Informacijos mokslai*, no. 42, pp. 168-174, 2007.

[17] J. Bézivin, "In search of a basic principle for model driven engineering", *Novatica Journal, Special Issue on UML* , vol. 5, no. 2, pp. 21-24, 2004.

[18] J. Greenfield, K. Short, S. Cook, and S. Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, 1st ed*. Wiley, 2004.

[19] B. Hui, S. Liaskos, and J. Mylopoulos, "Requirements Analysis for Customizable Software Goals-Skills-Preferences Framework", in *11th IEEE International Conference on Requirements Engineering*, IEEE CS, p. 117, 2003.

[20] J. Wang, K. He, B. Li, W. Liu, and R. Peng, "Meta-models of Domain Modeling Framework for Networked Software", in *6th Int. Conf. on Grid and Cooperative Computing*, IEEE CS, pp. 878-886, 2007.

[21] G. Button, *Technology in Working Order: Studies of Work, Interaction and Technology, 1st ed*. Routledge, 1992.

[22] K. Schneider and I. Wagner, "Constructing the 'Dossier Représentatif'", *Computer Supported Cooperative Work (CSCW)*, vol. 1, no. 4, pp. 229-253, Dec. 1993.

[23] J. Bowers, "Making it work: a field study of a CSCW network: Computer-supported cooperative work", *The Information Society: An International Journal*, vol. 11, no. 3, pp. 189-207, 1995.

[24] R. Atkinson, "Project management: cost, time and quality, two best guesses and a phenomenon, it's time to accept other success criteria", *International Journal of Project Management*, vol. 17, no. 6, pp. 337-342, 1999.

[25] M. Berg, "Implementing information systems in health care organizations: myths and challenges", *International Journal of Medical Informatics*, vol. 64, no. 2, pp. 143-156, Dec. 2001.

[26] M. Berg, *Rationalizing Medical Work: Decision-Support Techniques and Medical Practices*. MIT Press, 1997.

[27] M. F. Collen, *A History of Medical Informatics in the United States, 1950 to 1990,* 1st ed. American Medical Informatics Association, 1995.

[28] E. B. Steen, D. E. Detmer, and I. O. Medicine, *The Computer-Based Patient Record: An Essential Technology for Health Care,* Rev Sub. National Academy Press, 1997.

[29] W. Hasselbring, "Information system integration", *Communications of the ACM*, vol. 43, no. 6, pp. 32-38, 2000.

[30] S. Herold, A. Metzger, A. Rausch, and H. Stallbaum, "Towards Bridging the Gap between Goal-Oriented Requirements Engineering and Compositional Architecture Development", in *2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, IEEE CS, p. 7, 2007.

[31] S. S. Ostadzadeh, F. S. Aliee, and S. A. Ostadzadeh, "An MDA-Based Generic Framework to Address Various Aspects of Enterprise Architecture", in *Advances in Computer and Information Sciences and Engineering*, Springer, pp. 455-460, 2008.

[32] ITU-T – International Telecommunications Union, *Recommendation Z.151 (10/12) User Requirements Notation (URN) – Language definition*, Geneva, Switzerland, 2012.

[33] "URN Wiki", *http://www.usecasemaps.org/*, Dec-2009. [Online]. Available: http://www.usecasemaps.org/. [Accessed: 14-Sep-2009].

[34] S. T. March and G. F. Smith, "Design and natural science research on information technology", *Decision Support Systems*, vol. 15, no. 4, pp. 251-266, Dec. 1995.

[35] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research", *MIS Quarterly*, vol. 28, no. 1, pp. 75-105, 2004.

[36] "URN Virtual Library", *http://www.usecasemaps.org/urn/*, May-2009. [Online]. Available: http://www.usecasemaps.org/urn/. [Accessed: 11-May-2009].

[37] D. Amyot and G. Mussbacher, "User Requirements Notation: The First Ten Years, The Next Ten Years", *Journal of Software (JSW)*, Vol. 6, No. 5, Academy Publisher, pp. 747-768, May 2011.

[38] ITU-T – International Telecommunications Union, *Recommendation Z.150 (02/11), User Requirements Notation (URN) – Language Requirements and Framework*. Geneva, Switzerland, 2011.

[39] S. A. Behnam, D. Amyot, A. J. Forster, L. Peyton, and A. Shamsaei, "Goal-Driven Development of a Patient Surveillance Application for Improving Patient Safety", in *4th Int. MCeTech Conf. on eTechnologies*, LNBPI 26, Springer, pp. 65-76, 2009.

[40] M. Weiss and D. Amyot, "Business process modeling with URN", *International Journal of E-Business Research*, vol. 1, no. 3, pp. 63-90, 2005.

[41] Z. Cai and E. Yu, "Addressing Performance Requirements Using a Goal and Scenario-Oriented Approach", in *Proceedings of 14th international Conference on Advanced Information Systems Engineering*, LNCS 2348, Springer, pp. 706-710, 2002.

[42] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, "Evaluating Goal Models within the Goal-oriented Requirement Language", *International Journal of Intelligent Systems (IJIS)*, vol. 25, issue 8, pp. 841-877, 2010.

[43] G. Mussbacher, D. Amyot, P. Heymans, "Eight Deadly Sins of GRL", in *Proceedings of 5th International i* Workshop (iStar 2011)*, Trento, Italy, CEUR-WS, Vol-766, pp. 2-7, 2011

[44] G. Mussbacher, "Evolving Use Case Maps as a Scenario and Workflow Description Language", in *10th Workshop of Requirement Engineering (WER'07)*, pp. 56-67, Toronto, Canada, 2007.

[45] OMG, "Object Constraint Language", *OMG Specification, Version 2.0, formal/06-05*, vol. 1, pp. 06-05, 2006.

[46] D. Amyot, J. Horkoff, D. Gross, and G. Mussbacher, "A Lightweight GRL Profile for i* Modeling", *3rd Int. Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGiM 2009), ER Workshops*, LNCS 5833, Springer, pp. 254-264, 2009.

[47] "jUCMNav v5.1.0", *http://softwareengineering.ca/jucmnav/*. [Accessed: 9-Jul-2012].

[48] A. Pourshahid, D. Amyot, L. Peyton, S. Ghanavati, P. Chen, M. Weiss, and A. J. Forster, "Business process management with the user requirements notation", *Journal of Electronic Commerce Research*, vol. 9, no. 4, pp. 269-316, Dec. 2009.

[49] Stephen A. White, Introduction to BPMN, July 2012 *http://www.omg.org/bpmn/Documents/Introduction_to_BPMN.pdf*.

[50] Stephen A. White, BPMN Tutorial, Sep 2009, *http://www.bpmn.org/Documents/OMG BPMN Tutorial.pdf*.

[51] P. Wohed, W. M. van der Aalst, M. Dumas, A. H. Hofstede, and N. Russell, "On the suitability of BPMN for business process modelling", in *Business Process Management 2006*, LNCS 4102, Springer, pp. 161-176, 2006.

[52] C. Ouyang, M. Dumas, A. H. ter Hofstede, and W. M. van der Aalst, "From BPMN process models to BPEL web services", in *Web Services, 2006. ICWS'06. International Conference on*, IEEE CS, pp. 285-292, 2006.

[53] Object Management Group, *Business Process Definition MetaModel (BPDM),* version 1.0, formal/2008-11-03, November 2008.

[54] Object Management Group, *BPDM - Common Infrastructure*, September 2009, *http://www.omg.org/cgi-bin/doc?dtc/08-05-07*.

[55] C. Atkinson and T. Kuhne, "Model-Driven Development: A Metamodeling Foundation", *IEEE Software*, vol. 20, no. 5, pp. 36-41, 2003.

[56] J. M. Favre, "Towards a basic theory to model model driven engineering", in *3rd Workshop in Software Model Engineering, WiSME*, 2004.

[57] S. Kent, "Model Driven Engineering", in *Integrated Formal Methods, Third International Conference*, LNCS 2335, Springer, pp. 286-298, 2002.

[58] Object Management Group, *Model Driven Architecture - Architecture Board - ORMSC*, Nov 2009, *http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01*,.

[59] A. G. Kleppe, J. Warmer, J. B. Warmer, and W. Bast, *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley, 2003.

[60] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press US, 1977.

[61] J. Naish and L. Zhao, "Towards a generalised framework for classifying and retrieving requirements patterns", in *Requirements Patterns (RePa), 2011 First International Workshop on*, IEEE CS, pp. 42 -51, 2011.

[62] M. Fowler, *Analysis Patterns: reusable object models*. Addison-Wesley, 2000.

[63] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. John Viley & Sons Ltd, Chirchester, England, 1996.

[64] J. Adams, S. Koushik, G. Galambos, and G. Vasudeva, *Patterns for e-business: A Strategy for Reuse*. IBM Press, 2001.

[65] "Cloud Computing Patterns", July 2012, http://cloudcomputingpatterns.org/

[66] R. P. Gabriel, *Patterns of Software: Tales from the Software Community*. Oxford University Press, USA, 1998.

[67] M. Weiss, "Modelling Security Patterns Using NFR Analysis", *Integrating Security and Software Engineering: Advances and Future Visions*, IGI Global, pp. 127-141, 2006.

[68] M. Weiss, "Pay to play or requirements prioritization in collectives", in *Requirements Patterns (RePa), 2011 First International Workshop on*, pp. 28 -31, 2011.

[69] Y. Wen, H. Zhao, and L. Liu, "Analysing security requirements patterns based on problems decomposition and composition", in *Requirements Patterns (RePa), 2011 First International Workshop on*, IEEE CS, pp. 11 -20, 2011.

[70] S. Konrad and B. H. C. Cheng, "Requirements Patterns for Embedded Systems", in *10th Anniversary IEEE Joint International Conference on Requirements Engineering*, IEEE CS, Washington, DC, USA, pp. 127-136, 2002.

[71] G. Mussbacher, D. Amyot, and M. Weiss, "Formalizing Patterns with the User Requirements Notation", *Design Pattern Formalization Techniques*, IGI Global, pp. 302-322, 2007.

[72] J. Coplien, *Software Patterns*. SIGS, 1996.

[73] T. Taibi and D. C. L. Ngo, "Why and how should patterns be formalized", *Journal of Object-Oriented Programming (JOOP)*, vol. 14, no. 4, pp. 8-9, 2001.

[74] I. Araújo and M. Weiss, "Linking non-functional requirements and patterns", in *Conference on Pattern Languages of Programs (PLoP)*, 2002.

[75] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Kluwer Academic Publishers, 2000.

[76] H. Y. Ong, M. Weiss, and I. Araújo, "Rewriting a pattern language to make it more expressive", *Hot Topic on the Expressiveness of Pattern Languages, ChiliPLoP*, 2003.

[77] D. Gross and E. Yu, "From Non-Functional Requirements to Design through Patterns", *Requirements Engineering*, vol. 6, no. 1, pp. 18-36, 2001.

[78] L. Chung, S. Supakkul, and A. Yu, "Good Software Architecting: Goals, Objects, and Patterns", in *Information, Computing & Communication Technology Symposium (ICCT-2002)*, UKC'02, pp. 8-11, 2002.

[79] R. Andrade and L. Logrippo, "Reusability at the Early Development Stages of Mobile Wireless Communication Systems", in *4th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2000)*, pp. 11-16, 2000.

[80] R. Andrade, *Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems*, School of Information Technology and Engineering (SITE), University of Ottawa, 2001.

[81] E. Billard, "Patterns of agent interaction scenarios as use case maps", *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 34, no. 4, pp. 1933-1939, 2004.

[82] G. Mussbacher and D. Amyot, "A collection of patterns for Use Case Maps", in *First Latin American Conference on Pattern Languages of Programming (PLoP 01)*, 2001.

[83] S. Henninger and V. Corrêa, "Software pattern communities: current practices and challenges", in *14th Conference on Pattern Languages of Programs*, ACM, pp. 1-19, 2007.

[84] C. Zhao, J. Kong, J. Dong, and K. Zhang, "Pattern-based design evolution using graph transformation", *Journal of Visual Languages and Computing*, vol. 18, no. 4, pp. 378-398, 2007.

[85] C. Zhao, J. Kong, and K. Zhang, "Design pattern evolution and verification using graph transformation", in *40th Annual Hawaii Int. Conf. on System Sciences*, IEEE CS, p. 290a, 2007.

[86] J. Dong, S. Yang, and Y. Sun, "A Classification of Design Pattern Evolutions", *International Journal of Object Technology*, vol. 6, no. 10, pp. 95-109, Nov. 2007.

[87] J. Dong, Y. Zhao, and Y. Sun, "Design pattern evolutions in QVT", *Software Quality Journal*, vol. 18, no. 2, pp. 269-297, 2010.

[88] M. Aoyama, "Evolutionary Patterns of Design and Design Patterns", in *Principles of Software Evolution, International Symposium on*, IEEE CS, pp. 110-116, 2000.

[89] T. Kobayashi and M. Saeki, "Software development based on software pattern evolution", in *6th Asia-Pacific Software Engineering Conference (APSEC99)*, Takamatsu, Japan. IEEE CS, pp. 18-25, 1999.

[90] R. E. Johnson and B. Foote, "Designing reusable classes", *Journal of Object-Oriented Programming*, vol. 1, no. 2, pp. 22-35, 1988.

[91] H. Iida and Y. Tanaka, "A Compositional Process Pattern Framework for Component-based Process Modeling Assistance", in *1st Workshop on Software Development Process Patterns (SDPP'02)*, TUM-I0213, Munich, Germany, 2002.

[92] H. Iida, "Pattern-Oriented Approach to Software Process Evolution", in *Int. Workshop on the Principles of Software Evolution*, Fukuoka, Japan, pp. 55-59, 1999.

[93] H. Tran, *Modélisation de Procédés Logiciels à Base de Patrons Réutilisables*, Thèse de doctorat, Université de Toulouse-le-Mirail, France, 2007.

[94] J. A. Bubenko Jr, A. Persson, and J. Stirna, *User Guide of the Knowledge Management Approach Using Enterprise Knowledge Patterns*, Deliverable D3, IST Programme project HyperKnowledge–Hypermedia and Pattern Based Knowledge Management for Smart Organisations, Stockholm, Sweden: Department, 2001.

[95] J. Stirna, A. Persson, and L. Aggestam, "Building Knowledge Repositories with Enterprise Modelling and Patterns-from Theory to Practice", in *European Conference on Information Systems (ECIS)*, Göteborg, Sweden, 2006.

[96] A. Persson, J. Stirna, and L. Aggestam, "How to Disseminate Professional Knowledge in Healthcare: The Case of Skaraborg Hospital", *Journal of Cases on Information Technology*, vol. 10, no. 4, pp. 41-64, 2008.

[97] S. Ghanavati, D. Amyot, and L. Peyton, "A Systematic Review of Goal-oriented Requirements Management Frameworks for Business Process Compliance", in *4th Int. Workshop on Requirements Engineering and Law (RELAW).* IEEE CS, pp. 25-34, Aug. 2011.

[98] A. Shamsaei, A. Pourshahid, and D. Amyot, "A Systematic Review of Compliance Management Based on Goals and Indicators", in *3rd Workshop on Governance, Risk and Compliance-Applications in Information Systems (GRCIS 2011), CAiSE 2011 Workshops*. LNBIP 83, Springer, pp. 228-237, June 2011.

[99] I. Markovic and M. Kowalkiewicz, "Linking Business Goals to Process Models in Semantic Business Process Modeling", in *Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference*, IEEE CS, pp. 332-338, 2008.

[100] G. Rimassa and B. Burmeister, "Achieving Business Process Agility in Engineering Change Management with Agent Technology", in *Workshop dagli Oggetti agli Agenti WOA*, Genova, Italy, pp. 1-7, 2007.

[101] D. Greenwood and G. Rimassa, "Autonomic Goal-Oriented Business Process Management", in *3rd International Conference on Autonomic and Autonomous Systems*, IEEE CS, pp. 43-48, 2007.

[102] A. Pourshahid, G. Mussbacher, D. Amyot, and M. Weiss, "An Aspect-Oriented Framework for Business Process Improvement", in *4th International MCeTech*

Conference on eTechnologies (MCeTech09). LNBIP, vol. 26, Springer, pp. 290-305, 2009.

[103] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing", *International Journal of Human Computer Studies*, vol. 43, no. 5, pp. 907-928, 1995.

[104] M. Sabetzadeh and S. Easterbrook, "View merging in the presence of incompleteness and inconsistency", *Requirements Engineering*, vol. 11, no. 3, pp. 174-193, 2006.

[105] M. Hepp, F. Leymann, J. Domingue, A. Wahler, E. Wahler, and D. Fensel, "Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management", in *IEEE International Conference on e-Business Engineering*, IEEE CS, pp. 535-540, 2005.

[106] H. Kaiya and M. Saeki, "Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach", in *5th International Conference on Quality Software*, IEEE CS, pp. 223-230, 2005.

[107] H. Kaiya and M. Saeki, "Using Domain Ontology as Domain Knowledge for Requirements Elicitation", in *14th IEEE International Requirements Engineering Conference,* IEEE CS, pp. 186-195, 2006.

[108] R. Falbo, G. Guizzardi, and K. C. Duarte, "An ontological approach to domain engineering", in *14th international conference on software engineering and knowledge engineering*, ACM, pp. 351-358, 2002.

[109] R. S. Pressman, *Software engineering: a practitioner's approach, 5th ed*. McGraw-Hill New York, 2000.

[110] G. Arango and R. Prieto-Diaz, "Domain Analysis Concepts and Research Directions", in *Domain analysis and software systems modeling*, IEEE CS, pp. 9-32, 1991.

[111] L. Wei, H. Ke-Qing, W. Jiang, and P. Rong, "Heavyweight Semantic Inducement for Requirement Elicitation and Analysis", in *Semantics, Knowledge and Grid, Third International Conference on*, IEEE CS, pp. 206-211, 2007.

[112] Y. He, K. He, J. Wang, and C. Wang, "Toward a Context Driven Approach for Semantic Web Service Evolution", in *3rd International Conference on Convergence and Hybrid Information Technology*, vol. 2, IEEE CS, pp. 1089-1094, 2008.

[113] A. Filipowska, M. Kaczmarek, M. Kowalkiewicz, I. Markovic, and X. Zhou, "Organizational ontologies to support semantic business process management", in *4th International Workshop on Semantic Business Process Management*, New York, NY, USA, pp. 35-42 , 2009.

[114] Y. Yu, S. Liaskos, J. Mylopoulos, and A. Lapouchnian, "Requirements-driven configuration of software systems", in *Reverse Engineering To Requirements (RETR'05)*, Pittsburgh, USA. p. 18, 2005.

[115] S. Liaskos, A. Lapouchnian, Yiqiao Wang, Yijun Yu, and S. Easterbrook, "Configuring common personal software: a requirements-driven approach", in *13th IEEE International Conference on Requirements Engineering*, IEEE CS, pp. 9-18, 2005.

[116] L. Northrop and P. C. Clements, *A framework for software product line practice, version 5.0*, Software Engineering Institute, Pittsburgh, USA, 2007.

[117] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns.* Addison-Wesley, 2002.

[118] K. Lee, K. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering", *Software Reuse: Methods, Techniques, and Tools*, LNCS 2319, Springer, pp. 62-77, 2002.

[119] T. H. Davenport, *Process innovation: reengineering work through information technology*. Harvard Business School Pr, 1993.

[120] M. Penker and H. E. Eriksson, *Business Modeling With UML: Business Patterns at Work*. John Wiley & Sons, 2000.

[121] M. Dumas, W. van der Aalst, and A. Ter Hofstede, *Process-aware information systems: bridging people and software through process technology*. Wiley-Blackwell, 2005.

[122] M. Hammer and J. Champy, *Reengineering the corporation: A manifesto for business revolution*. Collins Business, 2003.

[123] M. D. McGinnis and I. U., Bloomington, *Polycentric games and institutions*. University of Michigan Press, 2000.

[124] L. M. Cysneiros, V. Werneck, and E. Yu, "Evaluating Methodologies: A Requirements Engineering Approach Through the Use of an Exemplar", in *7th Workshop on Requirements Engineering*, 2004, pp. 40-55.

[125] R. E. Freeman, *Strategic Management: A Stakeholder Approach*. Pitman Publishing, 1984.

[126] "USE: UML-based Specification Environment", *http://www.db.informatik.uni-bremen.de/projects/USE/*, Sep 2011. [Online].

[127] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL", *Science of Computer Programming*, vol. 69, no. 1-3, pp. 27-34, 2007.

[128] S.A. Behnam, *FMM metamodel implementation in USE*, http://www.eecs.uottawa.ca/~damyot/pub/Behnam/FMM/, July 2012.

[129] S.A. Behnam, *Sample pattern family implementation in USE*, http://www.eecs.uottawa.ca/~damyot/pub/Behnam/PF-USE, July 2012.

[130] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*, 2nd ed. Addison-Wesley Professional, 2003.

[131] L. Eggertson, "Hospitals to report C. difficile and MRSA". *Canadian Medical Association Journal (CMAJ)*, vol. 176 no. 10, pp. 1402-1403, May 2007. http://www.cmaj.ca/content/176/10/1402.full

[132] S.A. Behnam and D. Amyot, "Evolution Mechanisms for Goal-driven Pattern Families used in Business Process Modeling". *Int. Journal of Electronic Business*, Inderscience Publishers, 2012 *(to appear)*.

[133] S.A. Behnam and D. Amyot, "Evolution of Goal-driven Pattern Families for Business Process Modeling", in *5th Int. MCETECH Conference on eTechnologies*, Les Diablerets, Switzerland, January 2011. LNBIP 78, Springer, pp. 46-61, 2011.

[134] D. Alexandrou and G. Mentzas, "Research Challenges for Achieving Healthcare Business Process Interoperability", in *eHealth, Telemedicine, and Social Medicine, eTELEMED'09. International Conference on*, IEEE CS, pp. 58-65, 2009.

[135] S.A. Behnam, *Patient Safety Pattern Family*, http://www.eecs.uottawa.ca/~damyot/pub/Behnam/PF, July 2012.

[136] S.A. Behnam, *Implementation of Evolutionary Mechanisms*, http://www.eecs.uottawa.ca/~damyot/pub/Behnam/EM-Java, July 2012.

[137] S.A. Behnam, D. Amyot, G. Mussbacher, E. Braun, N. Cartwright, and M. Saucier, "Using the Goal-Oriented Pattern Family Framework for Modelling Outcome-Based Regulations", in *Second International Workshop on Requirements Patterns (RePa)*, Chicago, USA, IEEE CS, pp. 35-40, September 2012.

[138] Government of Canada, *Audit of Aviation Security Regulatory Oversight*, April 2011. http://www.tc.gc.ca/eng/corporate-services/aas-audit-870.htm

[139] R. Tawhid, M. Alhaj, G. Mussbacher, E. Braun, N. Cartwright, A. Shamsaei, D. Amyot, S.A. Behnam, and G. Richards, "Towards Outcome-Based Regulatory Compliance in Aviation Security", in *20th IEEE Int. Requirements Eng. Conf (RE'12)*, IEEE CS, pp. 267-272, 2012.

[140] Canadian Air Transport Security Authority, *What We Do*, June 2012. http://www.catsa.gc.ca/Page.aspx?ID=31

[141] A. Shamsaei, A. Pourshahid, and D. Amyot, "Business Process Compliance Tracking Using Key Performance Indicators", in *6th Int. Workshop on Business Process Design (BPD 2010)*, LNBIP 66, Springer, pp. 73-84, 2010.

[142] S. A. Behnam, D. Amyot, and G. Mussbacher, "Towards a Pattern-Based Framework for Goal-Driven Business Process Modeling", in *8th Int. Conf. on Software Engineering Research, Management and Applications (SERA2010),* IEEE CS, pp. 137-145, 2010.

[143] C. Kuziemsky, X. Liu, and L. Peyton, "Leveraging Goal Models and Performance Indicators to Assess Health Care Information Systems", in *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, IEEE CS, pp. 222 -227, 2010.

[144] H. Luo and D. Amyot, "Towards a Declarative, Constraint-Oriented Semantics with a Generic Evaluation Algorithm for GRL", in *5th Int. i* Workshop*, CEUR-WS, Vol-766, pp. 26-31, 2011.

[145] J. Brown, R. Gawley, I. Spence, P. Kilpatrick, C. Gillan, R. Bashroush, and others, "Requirements Modelling and Design Notations for Software Product Lines", in *1st Int. Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, Limerick, Ireland, 2007.

[146] A. Pourshahid, D. Amyot, A. Shamsaei, G. Mussbacher, and M. Weiss, "A Systematic Review and Assessment of Aspect-oriented Methods Applied to Business Process Adaptation", *Journal of Software (JSW)*, Vol. 7, No. 8, Academy Publisher, pp 1816-1826, August 2012.

Appendix A. OCL CONSTRAINTS FOR FMM

This appendix defines all the necessary OCL constraints implemented to ensure the integrity of FMM-based models. These constraints are implemented along with FMM using USE [126][127] and can be downloaded at [128]. Furthermore, the Increase Patient Safety pattern family introduced as an example in Section 5.2.4 is also used as an example for evaluating the effectiveness of OCL constraints. Figure 62 illustrates the result of applying the OCL constraints on this example in the USE environment. The implementation of this example in the USE environment can be downloaded from [129]. In the following, constraints are categorized by the FMM classes they target (i.e., the OCL `context`), and a brief description of their purpose is also provided.

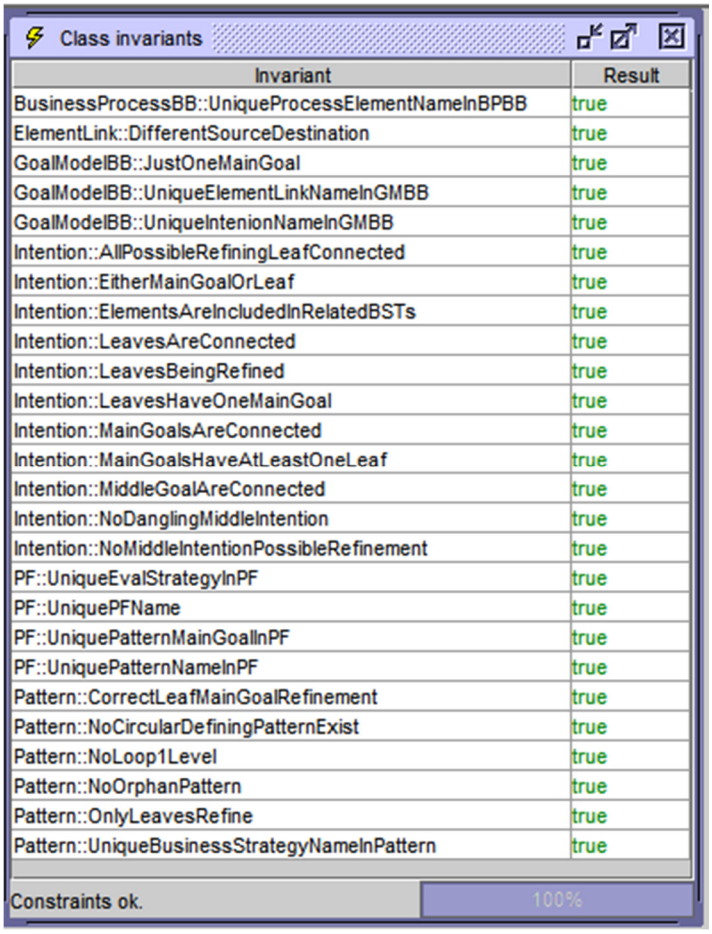

Figure 62 Results of checking the OCL constraints on the Increase Patient Safety PF example

A.1 Constraints on PFs

Table 18 represents the four invariant constraints that must be evaluated to true for all instances of PF. UniquePFName ensures that different patterns families are unique. Considering that, currently, the name is the unique identifying key of objects in GoPF models, there cannot exist two PFs with the same name. Similarly, UniquePatternNameInPF ensures that two patterns that have the same name cannot exist in a particular PF. Next, UniquePatternMainGoalInPF ensures that each pattern in the family has a unique mainGoal in their GoalModelBuildingBlock. Finally, UniqueEvalStrategyName ensures that instances of EvaluationStrategy are unique within a pattern family.

Table 18 Invariant OCL constraints in the context of PatternFamily

OCL Constraints
inv UniquePFName: PF.allInstances()->**isUnique**(name)
inv UniquePatternNameInPF: **self**.pattern->**isUnique**(name)
inv UniquePatternMainGoalInPF: **self**.pattern->**isUnique**(GT.mainGoal().name)
inv UniqueStrategyNameInPF: **self**.pattern.businessStrategy.evaluationStrategy->**isUnique**(name)

A.2 Constraints on Patterns

Table 19 contains the implemented invariant constraints on Patterns. UniqueBusinessStrategyNameInPattern guards the uniqueness of BusinessStrategies in a particular pattern. OnlyLeavesRefined prevents a pattern to refine intentions, which are not leaves of their respective GoalModelBuildingBlocks. CorrectLeafMainGoalRefinement limits the refinement of intentions only to those patterns whose GoalModelBuildingBlocks's mainGoal is the same as the refined intention. NoOrphanPattern ensures that only one root pattern exists in each pattern family. This means that all the patterns are connected and therefore their usage in the extraction process is possible. NoCircularDefiningPatternExist ensures refinement relationship between patterns is not circular. This prevents both direct circular refinements and multi-level circular refinements.

Table 19 Invariant OCL constraints in the context of Pattern

OCL Constraints
inv UniqueBusinessStrategyNameInPattern: **self**.businessStrategy->**isUnique**(name)
inv OnlyLeavesRefined: **self**.refined->**forAll**(i|i.leaf)
inv CorrectLeafMainGoalRefinement: **self**.refined->**forAll**(i|i.name = **self**.GMBB.mainGoal().name)
inv NoOrphanPattern: Pattern.allInstances()->**one**(p| p.refined->**size**()=0)
inv NoCircularDefiningPatternExist: **self**.DefiningPatternSet()->**excludes**(**self**)

NoCircularDefiningPatternExist uses two operations, namely DefiningPatternSet() and indirectly DownPatternSet() defined in Table 20, in order to support closure in circular refinements.

Table 20 OCL operations of Pattern

```
businessStrategyCollection(): Set( BusinessStrategy ) =
        self.businessStrategy
```

```
isEqualTo(p:Pattern):Boolean =
       ( (self.name = p.name) and
         (self.GMBB.isEqualTo(p.GMBB)) and
         (self.businessStrategy->forAll(b1|p.businessStrategy->
                                exists(b2|b2.isEqualTo(b1)))) and
         (p.businessStrategy->forAll(b1|self.businessStrategy->
                                exists(b2|b2.isEqualTo(b1)))) )
```

```
DownPatternSet(s:Set(Pattern)):Set(Pattern) =
       if s->includesAll(s.GMBB.intention.patternDef->asSet())
       then s
       else DownPatternSet(s->union(s.GMBB.intention.patternDef->
                                                        asSet()))
       endif
```

```
DefiningPatternSet():Set(Pattern) =
       if self.GMBB.intention.patternDef->asSet()->size()>0
       then        DownPatternSet(self.GMBB.intention.patternDef-
>asSet())
       else null
       endif
```

Table 21 represents the post-conditions that make sure the result of businessStrategy-Collection(), replaceGT(), addBizS(), and removeBizS() are as expected by the FMM. These post-conditions are relatively simple, however their definition in OCL clarifies the expected results and guides the correct implementation of these operations.

Table 21 Post-condition and preconditions in the context of Pattern

Context	OCL Constraints
context Pattern:: businessStrategyCollection(): **Set**(BusinessStrategy)	**post** businessStrategyCollectionisDone: **result** = **self**.businessStrategy
context Pattern:: replaceGMBB(g:GoalModelBuildingBlock): **Boolean**	**post** addGMBBisDone: **result** = (**self**.GMBB=g)
context Pattern:: addBizS(b:BusinessStrategy):**Boolean**	**post** addBizSisDone: **result** = (**self**.businessStrategy = **self**.businessStrategy@**pre**->**including**(b))
context Pattern:: removeBizS(b:BusinessStrategy): **Boolean**	**post** removeBizSisDone: **result** = (**self**.businessStrategy = **self**.businessStrategy@**pre**->**excluding**(b))

A.3 Constraints on GoalModelBuildingBlocks

Table 22 defines the invariant constraints that guard the well-formedness of FMM-based models. UniqueIntenionNameInGMBB and UniqueElementLinkNameInGMBB respectively ensure the uniqueness of Intentions and ElementLinks within a particular GoalModelBuildingBlock. JustOneMainGoal forces that only one of the Intentions in a particular GoalModelBuildingBlock be a mainGoal.

Table 22 Invariant OCL constraints in the context of GoalModelBuildingBlock

OCL Constraints
inv UniqueIntenionNameInGMBB: **self**.intention->**isUnique**(name)
inv UniqueElementLinkNameInGMBB: **self**.EL->**isUnique**(name)
inv JustOneMainGoal: **self**.intention->**one**(i|i.mainGoal)

In Table 23, the operations of GoalModelBuildingBlock are represented.

Table 23 OCL operations of GoalModelBuildingBlock

```
leafCollection(): Set(Intention) =
       self.intention->select(i|i.leaf)
```

```
mainGoal(): Intention =
       self.intention->select(i|i.mainGoal)->asSequence()->at(1)
```

```
isEqualTo(g:GoalModelBuildingBlock) : Boolean =
       ((self.name = g.name) and
        (self.intention->forAll(i1|g.intention->
                  exists(i2|i2.isEqualTo(i1)))) and
        (g.intention->forAll(i1|self.intention->
                  exists(i2|i2.isEqualTo(i1)))) and
        (self.EL->forAll(e1|g.EL->
                  exists(e2|e2.isEqualTo(e1)))) and
        (g.EL->forAll(e1|self.EL->exists(e2|e2.isEqualTo(e1)))))
```

Table 24 represents the post-conditions that make sure the results of leafCollection() and mainGoal() are as expected.

Table 24 Post-condition and preconditions in the context of GoalModelBuildingBlock

Context	OCL Constraints
`context GoalModelBuildingBlock::` `leafCollection(): Set(Intention)`	`post leafCollectionisDone:` `result = self.intention->select(i|i.leaf)`

`context GoalModelBuildingBlock::`
`mainGoal(): Intention`
	`post mainGoalisDone:`
	`self.intention->`
	`forAll(i|i.mainGoal implies result = i)`

A.4 Constraints on Intentions

Table 25 defines the invariants in the context of Intention. LeavesBeingRefined ensures that in patterns, only leaf intention may be refined. ElementsAreIncludedInRelatedBSTs guarantees that when an intention is realized by a ProcessElement, the ProcessElement is related to at least one BuisnesProcessTemplate from the holding pattern. Either-MainGoalOrLeaf ensures that an intention cannot be both leaf and mainGoal at the same time. The LeavesAreConnected, MainGoalsAreConnected, and MiddleGoalsAre-Connected constraints make sure that intentions at different levels of hierarchy are well connected and that their respective links are also included in the GoalModelBuilding-Block.

AllPossibleRefiningLeafConnected ensures that all potential refinement relations are established. This is done by making sure that, when a pattern whose GoalModelBuild-ingBlock's mainGoal is equal to a leaf intention, then they are connected. NoMiddleInten-

tionPossibleRefinement ensures that just leaf intentions may be potentially refined by other patterns. NoDanglingMiddleIntention ensures that intentions in the middle of a hierarchy are connected to one mainGoal though a chain of links. It also ensures that they are connected to at least one leaf through a chain of links. LeavesHaveOne-MainGoal ensures that every leaf intention is connected to one mainGoal intention through a chain of links. Similarly, MainGoalsHaveAtLeastOneLeaf ensures that through a chain of links, each mainGoal is connected to at least one leaf intention.

Table 25 Invariant OCL constraints in the context of Intention

OCL Constraints
inv LeavesBeingRefined: **self**.patternDef<>**null implies self**.leaf
inv ElementsAreIncludedInRelatedBSTs: **self**.GMBB.pattern.businessStrategy.BPT.PE ->**includesAll**(**self**.realizingElement)
inv EitherMainGoalOrLeaf: **not** (mainGoal **and** leaf)
inv LeavesAreConnected: **self**.leaf **implies self**.GMBB.EL->**exists**(e|e.fromLink()=**self**)
inv MainGoalsAreConnected: **self**.mainGoal **implies self**.GMBB.EL->**exists**(e|e.toLink()=**self**)

inv MiddleGoalsAreConnected:
(**not self**.mainGoal **and not self**.leaf)
  implies
  self.GMBB.EL->**exists**(e1, e2|e1.toLink()= **self and** e2.fromLink() = **self**
)

inv AllPossibleRefiningLeafConnected: **self**.leaf **implies** Pattern.allInstances()->**forAll**(p|(**self**.name = p.GMBB.mainGoal().name **and self**.GMBB.pattern<>p **and self**.GMBB.pattern.family.name=p.family.name) **implies** **self**.patternDef = p)
inv NoMiddleIntentionPossibleRefinement: Pattern.**allInstances**()->**exists**(p|**self**.name=p.GMBB.mainGoal().name) **implies** (**self**.leaf **or self**.mainGoal)

inv NoDanglingMiddleIntention:
not (**self**.leaf **or self**.mainGoal) **implies**
  (UpIntentionSetFromSelf()->**one**(i|i.mainGoal) **and**
  DownIntentionSetFromSelf()->**exists**(i|i.leaf))

inv LeavesHaveOneMainGoal: **self**.leaf **implies** UpIntentionSetFromSelf()->**one**(i|i.mainGoal)
inv MainGoalsHaveAtLeastOneLeaf: **self**.mainGoal **implies**   DownIntentionSetFromSelf()->**exists**(i|i.leaf)

Table 26 introduces the operations defined in the Intention class.

Table 26 OCL operations of Intention

```
isEqualTo(i:Intention) : Boolean =
        (self.name = i.name and
         self.mainGoal = i.mainGoal and
         self.leaf = i.leaf)
```

```
UpIntentionSet(s:Set(Intention)):Set(Intention) =
        if s ->includesAll(s.ELf.toLink()->asSet())
        then s
        else UpIntentionSet(s->union(s.ELf.toLink()->asSet()))
        endif
```

```
UpIntentionSetFromSelf():Set(Intention) =
        UpIntentionSet(Set{self})
```

```
DownIntentionSet(s:Set(Intention)):Set(Intention) =
        if s ->includesAll(s.ELt.fromLink()->asSet())
        then s
        else DownIntentionSet(s->union(s.ELt.fromLink()->asSet()))
        endif
```

```
DownIntentionSetFromSelf():Set(Intention) =
        DownIntentionSet (Set{self})
```

A.5 Constraints on BusinessProcessBuildingBlocks

Table 27 represents an invariant for BusinessProcessBuildingBlock. UniqueProcess-ElementNameInBusinessProcessBuildingBlock ensures that ProcessElements are unique in the context of BusinessProcessBuildingBlock.

Table 27 Invariant OCL constraint in the context of BusinessProcessBuildingBlock

OCL Constraint
`inv UniqueProcessElementNameInBusinessProcessBuildingBlock:` `self.PE->isUnique(name)`

A.6 Constraints on ElementLinks

Table 28 represents an invariant for ElementLinks. DifferentSourceDestination ensures that instances of ElementLink are not linking intentions to themselves.

Table 28 Invariant OCL constraints in the context of ElementLinks

OCL Constraint
`inv DifferentSourceDestination:` `self.toLink<>self.fromLink`