DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
78

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
78

REINA UBA

Merging business process models

TARTU UNIVERSITY PRESS

Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (Ph.D.) in informatics on September 19, 2011, by the Council of the Faculty of Mathematics and Computer Science, University of Tartu.

Supervisors:

Prof. PhD.	Marlon Dumas Institute of Computer Science University of Tartu, Tartu, Estonia
Senior Lecturer. PhD.	Marcello La Rosa Faculty of Science and Technology Queensland University of Technology, Brisbane QLD, Australia

Opponents:

Prof. Dr.	Mathias Weske Hasso Plattner Institute of IT Systems Engineering University of Potsdam, Potsdam, Germany
Prof. PhD.	Marite Kirikova Department of Systems Theory and Design Riga Technical University, Riga, Latvia

Commencement will take place on October 31, 2011, at 16.15 in Liivi 2-404.

The publication of this dissertation was financed by Institute of Computer Science, University of Tartu.

ISSN 1024-4212
ISBN 978-9949-19-858-0 (trükis)
ISBN 978-9949-19-859-7 (PDF)

Tartu Ülikooli Kirjastus
www.tyk.ee
Tellimus nr. 626

Abstract

Companies that have years of experience in business process management often maintain repositories containing hundreds or even thousands of business process models. The models in these repositories usually originate from various sources and are developed by different stakeholders. A common practice is that new process models are created by extending or refining existing models, or by copying and merging fragments from multiple models. As a result, process models tend to accumulate duplicate fragments which, if left unconsolidated, may evolve independently and lead to inconsistencies. Also, it often occurs that organizations manage multiple business processes that have similar goals, but pertain to different customer types, different products, business units or geographical regions. For example, a business process for handling insurance claims for motor accidents shares the same goal as a business process for handling house insurance claims. Naturally, these models will share several common fragments, but will differ from one another at various points. Managing these processes as entirely separate entities leads to redundancy and inefficiency.

In this setting, this thesis addresses the following question: How to identify duplicate fragments in process model repositories, and more generally, how to identify and consolidate commonalities across models in a large process model repository?

The thesis proposes two complementary methods for process model consolidation, namely process model merging and subprocess extraction. Pro-

cess model merging takes as input two or more process models and produces a single consolidated model that analysts can use to manage entire families of similar process models rather than managing them independently. On the other hand, subprocess extraction is about identifying fragments that are shared by multiple process models (also known as clones) and encapsulating these clones as separate subprocesses in order to eliminate redundancies.

The proposed merging and clone detection methods have been prototyped and validated on large process model repositories sourced from different domains. The process model merging tool has also been used to conduct a case study at an insurance company.

Acknowledgements

It is a pleasure to thank those who made this thesis possible. First and foremost, I would like to thank my supervisor professor Marlon Dumas, who has supported me throughout my thesis with his patience and knowledge. Also, I would like to thank the people from the Institute of Computer Science who have helped me during the PhD studies in various ways.

I would like to acknowledge my co-supervisor, Marcello La Rosa, who has been an invaluable research partner in these years and also mentored me during my visit to Queensland University of Technology. In addition, I would like to thank the members of the Queensland University of Technology Faculty of Science and Technology for supporting me and making me feel welcome during my stay.

Also I want to acknowledge all the people with whom I have conducted a research together.

I owe special thanks to Marlon Dumas, Marcello La Rosa and Kersti Tomusk for reading the final drafts of this thesis and advising on improving the quality of the text.

Last but not least, I would like to thank my family and friends for supporting me throughout my PhD studies.

List of Original Publications

I Reina Uba, Marlon Dumas, Luciano García-Bañuelos, Marcello La Rosa. **Clone Detection in Repositories of Business Process Models.** In Proceedings of 9th International Conference on Business Process Management, BPM 2011 (Clermont-Ferrand, Sept. 2011), Lecture Notes in Computer Science, vol. 6896, pages 248-264, Springer, 2011.

II Remco M. Dijkman, Marlon Dumas, Boudewijn F. van Dongen, Reina Käärik, Jan Mendling. **Similarity of business process models: Metrics and evaluation.** In Information Systems, vol. 36(2), pages 498-516, 2011.

III Remco Dijkman, Marlon Dumas, Luciano García-Bañuelos, Reina Käärik, **Aligning Business Process Models.** In 2009 IEEE International Enterprise Distributed Object Computing Conference, EDOC 2009 (Auckland, Sept. 2009), pages 45-53, 2009.

IV Marcello La Rosa, Marlon Dumas, Reina Uba, Remco M. Dijkman. **Merging Business Process Models.** In R. Meersman, D. Dillon, P. Herrero, eds., Proceedings of OTM 2010 Confederated Int. Confs. CoopIS, DOA, IS, and ODBASE 2010 (Hersonissos, Oct. 2010), Lecture Notes in Computer Science, Part I, vol. 6426, pages 96-113, Springer, 2010.

Contents

Part I

Overview

Chapter 1

Introduction

In 1947, Goldstein and Neumann demonstrated the usefulness of flowcharts [Baec 97]. This technique was initially invented to provide a high-level representation of computer programs in order to enable communication between programmers. But due to its generic nature, it quickly gained wider popularity. Specialists started to use it in other application areas as well, including business process modeling [Giag 01]. Over time, business process modeling based on flowchart-like notations grew up in popularity. In recent times, its importance has been further enhanced due to globalization trends, which push companies to make their business processes more efficient and repeatable [McAd 01].

1.1 Problem Area

Business process modeling has been exploited in various domains. It has been used to describe organizations and their operations including business processes, people, business objects, information systems and in general the organizational environment [Giag 01]. The main purpose of process models is to embrace the information that is needed to understand how complex business procedures need to be carried out among various stakeholders [Reij 09]. Business process modeling open up several benefits.

Firstly, models are simplifications of complex systems that help clarify and understand aspects of problems where there is uncertainty, change or assumptions [Lind 03]. Secondly, business process modeling allows the key operations in an organization to be identified. Thirdly, after documenting these operations, it is possible to measure the efficiency of the processes and therefore improve their performance [Lee 98]. Finally, business process models allow organizations to automate everyday work in order to gain further efficiency and reduce errors due to handovers of work between multiple actors.

As organizations undergo constant change, so do their business operations. Organizations are continuously improving their processes, for instance by adopting new work practices [Canf 05]. Business process models must reflect these changes, therefore, the business process models are also in constant change. Additionally, business process models are revised during company mergers and internal consolidation initiatives [Sche 00].

After a long-term business process management experience, organizations often end up managing large business process model repositories containing hundreds or even thousands of models that represent several man-years of effort [Rose 06, Gull 00]. These model collections may contain process models that describe multiple variants of the same process. Such variants arise for example in the context of federated organizations composed of several more or less independent units, like for example an insurance company with multiple business units dealing with different insurance products (e.g. life insurance, motor insurance, travel insurance, etc.). Other times, these variants arise because an organization is composed of multiple relatively independent units, such as a government composed of independent government agencies or departments. Regardless of their provenance, it is generally the case that process models representing variants of the same process share common fragments, while at the same time diverging in various ways.

Maintaining process model repositories in the presence of process variants is a challenge. It is essential to keep track of various models, their invariants, i.e. commonalities, and differences. Ideally, the model fragments that visualize the same part of a process must be changed concurrently to reduce inconsistencies among models. However, in reality the processes in large companies are edited by stakeholders with varying skills, responsibilities and goals [Card 06] resulting in the process models evolving independently.

In this thesis we propose two approaches for managing commonalities among process models. The first approach concentrates on cases when process models share identical single-entry, single-exit regions that can be extracted into subprocesses. In this case, shared fragments are factored out in subprocesses and in the initial models, in which the fragments occurred, the subprocesses are invoked using call-and-return semantics. The second approach is intended to be used when process models share fragments which cannot be refactored out into shared subprocesses. In this case, it is feasible to use aggregate models in order to enable business analysts to maintain shared parts in a synchronized manner.

1.2 Background

Before discussing concrete methods for merging process models, we provide some background on process modeling and similarity measurement in graph-based models, which will allow us to identify commonalities across process models represented by means of graphs.

1.2.1 Process Modeling Standards

Business process modeling standards can be classified according to their main purpose [Ko 09]:

- **Graphical standards** – allow users to express their processes in a diagrammatic way.
- **Execution standards** – enable automate business processes and execute business logic.
- **Interchange standards** – enable, for instance, to translate graphical standards to execution standards and vice versa.
- **Diagnosis standards** – provide administrative and monitoring capabilities.

In this thesis we restrict ourself to graphical modeling notations only. The application of similarity search and merging algorithms in case of other standards is out of the scope of this thesis, although some of the techniques proposed in the thesis (modulo some extensions) may be applied to executable standards as well. Below we review major graphical standards that are most commonly used for modeling business processes among various stakeholders [Ko 09].

1.2.1.1 Business Process Modelling Notation (BPMN)

The Business Process Modelling Notation (BPMN) [1] was first released in 2004 by the Business Process Management Initiative (BPMI) [Business 11b]. The objective of the BPMN was to support business process management by both technical and business users by providing a notation that is intuitive and at the same time able to represent complex process semantics.

The BPMN elements are divided into four basic categories: *Flow Objects*, *Connecting Objects*, *Swimlanes* and *Artifacts*. *Flow Objects* are the

[1] In its most recent version, BPMN was renamed to "Business Process Model and Notation". However, for historic reasons and to be consistent with the publications attached to this thesis, we use the former nomenclature.

main graphical elements of a BPMN model, these elements define the behavior of a process model. The BPMN flow objects – *Process*, *SubProcess*, *Task*, *Event* and *Gateway* – are depicted in Figure 1.1.

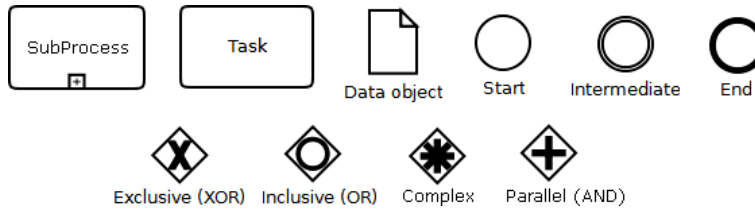

Figure 1.1: BPMN flow objects.

An *Activity* is the main element of BPMN. It represents a unit of work that a company does. Activities can be atomic or non-atomic. The types of activities are: *Process*, *SubProcess* and *Task*. Processes are either unbounded or contained within a *Pool*. A process and a subprocess contain at least one *Task*. An *Event* is something that "happens" during the execution of a business process. There are three types of events based on their effect on the flow: *Start*, *Intermediate* and *End* events. *Triggers* (i.e. *Message*, *Timer*, *Cancel*, etc.) can also be related to events. Triggers define the cause of an event, e.g. a message being received or a timeout that expires. *Gateways* are used to control the divergence and convergence of a sequence flow. Markers within a gateway indicate the type of a flow control behavior. The types of a flow control include:

- *XOR* – exclusive decision and merging.
- *OR* – inclusive decision and merging.
- *Complex* – complex conditions and situations (e.g. 3 out of 5).
- *AND* – parallel forking and synchronization.

Artifacts are used to provide additional information about a process, but they do not affect the message flow. The standard set of artifacts includes *Data Object*, *Group* and *Annotation*. Data objects provide information about the data that activities require to be performed and/or what they produce. Grouping can be used for documentation or analysis purposes. Textual annotations are used to provide additional information for the reader of the model.

There are two ways of grouping the primary modeling elements – *Pools* and by *Lanes*. Pools group a set of activities that have a common characteristic. A lane is a subpartition within a pool.

Flow objects are connected to each other using *Connecting Objects* – *Sequence Flow*, *Message Flow* and *Association*. A sequence flow determines in which order activities will be performed in a process. A message flow shows the flow of messages between two entities – between two activities, between an activity and a pool or between two pools. An association is used to associate data objects with a flow or connect data objects to an activity.

Artifacts, connecting objects and pooling elements are depicted in Figure 1.2.

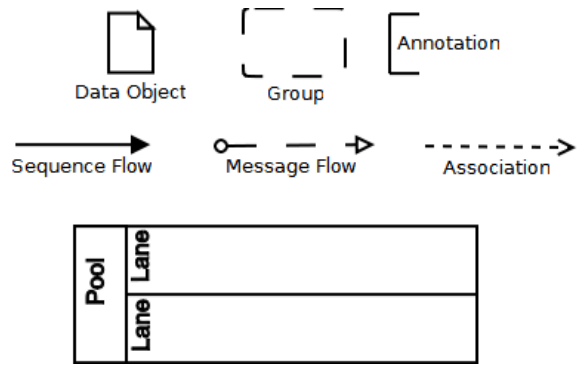

Figure 1.2: BPMN artifacts, connecting objects and pooling elements.

The BPMN main elements are compliant with most flow-chart notations but offer much more precise flow control semantics. The main benefit of BPMN is that this notation allows expressing processes at different granularity levels (using pools, lanes and subprocesses) from the perspective of the key stakeholders or inter-department [Ko 09].

1.2.1.2 Event-Driven Process Chains (EPCs)

The Event-Driven Process Chains (EPCs) notation was developed for modeling business processes with the goal to be easily understood and used by business people. EPCs were developed by the Institute for Information Systems (IWi) at the University of Saarland, Germany. As the name of the notation indicates, the control flow of a process is captured by means of a chain of events and functions [Korh 08].

The main elements of EPCs are [Aals 99]:

- *Functions* – main building blocks representing the activities (tasks, process steps) that need to be executed.
- *Events* – describe a situation before and/or after the functions are executed. Functions are linked by events.
- *Logical connectors* – used for describing logical relationships between elements in a control flow. There are three types of connectors: $\wedge$ (and), $\vee$ (or) and *XOR* (exclusive or).

The basic elements of EPCs are depicted in Figure 1.3

Figure 1.3: The basic set of EPCs elements.

The extended EPCs (eEPCs) notation add the organizational structure and data flow elements like *Organizational Unit*, *Position*, *Data* and *System* to EPCs (see Figure 1.4). These additional elements can be only mapped to a function using a *Relation*. A relation can be directed in case of data elements, indicating that the data is written or read. Additionally, a *Process Link* is introduced indicating the hierarchical or flat link to another process model. This can be used instead of an event or a function [Davi 07].

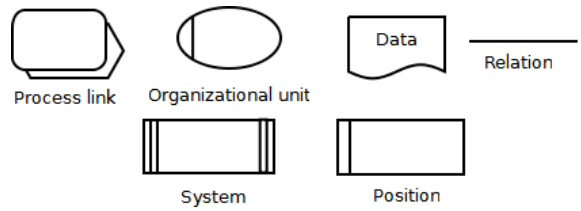

Figure 1.4: Extended set of EPCs elements.

In 2007, Rosemann et al. [Rose 07] introduced the notion of configurable EPCs (C-EPCs) – an extension to the EPCs modeling language which allows capturing the core configuration opportunities than can arise in the context of a business process. In C-EPCs functions and connectors may be configurable. Configurable functions may be included, skipped or conditionally skipped. In order to configure a configurable connector, one or more of the connector's incoming branches (in case of a join) or one or more of its outgoing branches (in case of a split) need to be marked for removal. In addition, configurable connectors may be "restricted" – a configurable OR connector into a regular XOR or a regular AND. This operation is called "restricting" because it reduces the number of possible traces induced by the connector [Aals 06b, Rosa 10]. The C-EPCs notation makes it possible to represent families of business process variants in a consolidated way.

1.2.1.3 Unified Modelling Language(UML) Activity Diagrams

Unified Modelling Language(UML) (version 2.0) was standardized in 2004. It contained 13 object-oriented notations [UML 20 S 11] – six structural diagrams and seven behavioral diagrams. *Activity Diagrams* belong to the behavioral diagrams group and is designed for modelling business processes and flows in software systems.

Figure 1.5 depicts the main elements of UML 2.0 Activity Diagrams.

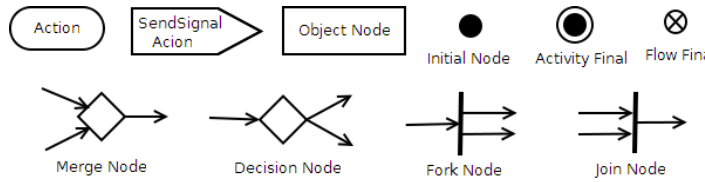

Figure 1.5: The elements of UML Activity Diagrams.

The main elements of UML Activity Diagrams are *Activity Nodes* – *Action*, *Object Node* and *Control Nodes*. An *Action* is the fundamental unit of an Action Diagram that represents a transformation or a process in a modeled system. A *Send Signal Action* is a special case of an action that creates a signal instance from its inputs and transmits it to the target object. An *Object Node* is an abstract activity node that represents an instance of a particular class. *Control Nodes* define the behavior of a process model. Control nodes are: *Initial Node*, *Final Node*, *Fork Node*, *Join Node*, *Decision Node* and *Merge Node*. An *Initial Node* starts an activity. An activity can be related to more than one initial node. There are two types of *Final Nodes* – *Activity Final Node* and *Flow Final Node*. An *Activity Final Node* stops all flows in an activity while a *Flow Final Node* just terminates one flow, the activity remains unaffected. A *Fork Node* splits a flow into concurrent paths and a *Join Node* merges concurrent paths into one outgoing flow. A *Decision Node* is a control node that has multiple

outgoing flows, only one of them is chosen for processing. A *Merge Node* is used to merge alternate flows, it is not used for the synchronization purpose. *Activity Nodes* are connected using *Control Flow* and *Object Flow* edges. A *Control Flow* edge models the flow between actions, an *Object Flow* connects object nodes and actions. Actions that have a common characteristic can be grouped using *Activity Partitions*, which is a notion akin to the notion of pool or lane in BPMN, but more general since an activity may belong to multiple participations, while in BPMN an activity can only be assigned to one pool/lane.

1.2.1.4 Business Process Graph

In this thesis we did not restrict ourselves to one specific modeling notation. Our objective was to develop algorithms that are general, easily applicable and extendible to most popular notations. Therefore, we introduced an abstraction of a business process model – a business process graph (BPG).

Definition 1 (BPG) *Let T be a set of types and Ω be a set of text labels. A BPG is a tuple $(N, E, \tau, \lambda, \alpha)$, in which:*

- *N is a finite set of nodes;*
- *$E : N \times N$ is a finite set of edges;*
- *$\tau : (N \cup E) \to T$ associates nodes and edges with types;*
- *$\lambda : (N \cup E) \to \Omega$ associates nodes and edges with labels;*
- *$\alpha : (N \cup E) \to (T \to \Omega)$ associates nodes and edges with attributes, where an attribute is always a combination of a type and a label;*

A BPG is a directed graph that captures the types of nodes and edges as attributes. This generalization can be performed because of the fact that although there are many modeling notations, most of them are graph based and can be transformed to an abstract format [Rosa 11]. In BPG we focused on the common subset of elements shared by the business process modeling

notations previously introduced – the core elements that are common for all of the modeling notations under observation. Understandably there are elements which are not captured in the abstract graph, but taking all the possible node types into account is out of the scope of this thesis. Also, in this case, the algorithms would get too difficult to comprehend and would need to be specialized to a notation's specific behaviors. The fact that the algorithms introduced by us are easy to extend, for instance, to take into account the objects and roles of a business process model, is evident in the Apromore – Advanced Process Model Repository [1] where our algorithms are integrated.

As previously mentioned, the concept of BPG represents the core functionality of business process notations. Generally, in a BPG we can differentiate three types of nodes – functions, events and routing nodes. Functions represent work nodes in a process model. When comparing different notations, the function nodes of BPG represent the function nodes of EPCs, the activity nodes of BPMN and the actions of UML AD. The events of BPG capture the behavior of the event elements of EPCs and BPMN, and the signals of UML AD. There are different types of routing nodes in a BPG:

- *AND gateway* – executes both of its output branches or waits for all its input branches to finish to continue the process execution. This gateway represents the behavior of a $\wedge$ connector in EPCs, a parallel gateway in BPMN and the fork and join nodes in UML AD.

- *XOR gateway* – executes only one of its output nodes or waits for the input from only one of its input branches before it continues the process execution. This gateway represents an XOR connector in EPCs, an exclusive gateway in BPMN and the merge and decision nodes in UML AD.

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

- *OR gateway* – executes at least one of its outgoing branches or continues after one or many of its input branches reach the gateway. This gateway includes the behavior of an OR connector in EPCs and an inclusive gateway in BPMN. This element is not represented in UML AD.

1.2.2 Graph Matching

In order to compare and merge process models, we need to identify the similarities between models. Transforming business process models to the general graph format enables us to apply the algorithms from the areas of graph isomorphism detection in order to find similarities and detect common regions in them.

Determining if two graphs are the same, if one graph is subsumed in another or if the graphs share a common subgraph has been the focus of intensive research since the end of the 1970s [Mess 95]. In the following sections we introduce various graph matching techniques that have resulted from these research efforts.

1.2.2.1 Graph Isomorphism

Graphs are used for visualization purposes in many areas; for example, in computer vision [Lonc 98], data visualization in scientific applications and computer systems [Herm 00], pattern recognition [Cont 04], etc. In many applications we need to determine if two graph structures are the same. In these cases, graph isomorphism detection algorithms can be used. In particular, graph isomorphism detection algorithms can be used to determine if two business process graphs (or fragments thereof) are identical. This can be useful in the context of refactoring duplicate fragments into shared subprocesses.

Graph isomorphism detection relates to the problem of finding a bijective mapping between input graphs. The mapping must preserve the

structure of edges [Mess 95]. For instance, let us assume we have two unlabeled graphs, $G1$ and $G2$ depicted in Figure 1.6, and we want to know if there exists a mapping between their nodes so that the edge relations are preserved (i.e. these graphs are isomorphic).

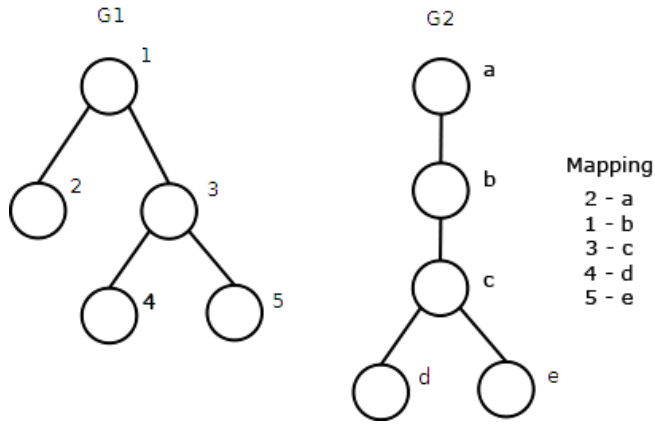

Figure 1.6: Unlabeled graph isomorphism.

For clarity reasons the nodes of the graphs are identified uniquely. As seen from Figure 1.6, such a mapping exists that preserves the node relations. The depicted mapping is not the only one that fulfills the isomorphism requirement. For instance, if the node 5 is mapped with the node d and the node 4 with the node e, we get another mapping that represents the isomorphism between the graphs $G1$ and $G2$.

Note that in this example, the nodes of the graph do not have labels attached to them. In the context of business process graphs, nodes have labels (e.g. names of tasks) and these labels can be taken into account when determining whether or not two nodes should be mapped. For example, if on the one hand the labels of node 4 and node d are the same, and on the other hand the labels of nodes e and 5 are the same, then it becomes clear that the mapping shown in Figure 1.6 is the correct one. In other words, node labels make it easier to identify an isomorphism between two graphs. In the general case however, absence of labels or (equivalently),

duplicate labels are unavoidable and therefore labels cannot be used (alone) to identify an isomorphism. For example, in the context of business process graphs, all XOR gateways are undistinguishable in terms of their labels. Thus, in order to determine how to map the XOR gateways of one graph to those of another graph, we need to take into account the structure of the graphs.

The main drawback of graph isomorphism detection algorithms is their computational complexity. The graph isomorphism problem lies in the NP complexity class. It is not known whether it lies in the P or the NP-complete complexity classes [Mess 95]. Despite decades of active research in this area, all algorithms that have been developed to solve the general graph isomorphism problem require in the worst case exponential time [Mess 95, Peli 99, Derk 10]. There are algorithms that use approximate or continuous optimizations to solve the problem in polynomial time under certain assumptions [Peli 99, Derk 10]. Some algorithms use backtracking and forward checking to prune the search space [Ullm 76, Schm 76]. Other algorithms reduce the complexity by specializing on graphs with special properties [Derk 10, Fort 96, Dick 04] – for instance, there are linear algorithms for finding graph isomorphism in case of planar graphs [Hopc 74] and polynomial time algorithms for graphs with bounded degree [Luks 82] or with bounded color class size[1] [Arvi 06].

Currently, one of the most efficient algorithms for finding graph isomorphism – Nauty – is presented by McKay [McKa 81]. This algorithm is based on canonical labeling of graph vertices, a technique that we rely upon in Chapter 2 and reference [Uba 11].

Graph isomorphism is a subclass of a broader problem – subgraph isomorphism detection.

[1] A color class is a set of nodes in the input graphs that share the same label. A graph has bounded color size if we can put an upper bound to the number of nodes that have identical label.

1.2.2.2 Subgraph Isomorphism

There are occasions when it is not desirable to understand if two graphs are identical. In some cases it is more important to discover if a graph is subsumed by another. For instance, in the area of chemoinformatics, it may be interesting to find if a chemical compound is a subcompound of a further specified compound, given their structural formulas [Corn 70]. A similar problem is that of scene analysis – there is a need to detect if a relationally described object is embedded in a scene [Ullm 76]. Also, in case of business process models, it may be interesting to discover if a model fragment is a subfragment of another model in order to detect most commonly occurring model fragments. In these types of problems, the subgraph isomorphism detection algorithms can be used.

Precisely, given two input graphs $G1$ and $G2$, subgraph isomorphism detection relates to the problem of finding whether a subgraph of $G2$ is isomorphic to $G1$. In Figure 1.7, , graphs $G1$ and $G2$ are depicted. Obviously, the graphs are not isomorphic because the graph $G2$ is larger than $G1$ (i.e. it contains more vertices and edges). Therefore, we might be interested in checking if there exists a subgraph in $G2$ that is isomorphic to $G1$.

In Figure 1.7, one of the possible isomorphic mappings is described. Obviously, this is not the only one. Since the graphs are undirected, the mapping that preserves the isomorphism can also be, for instance, 1 - f, 2 - e, 4 - c, 3 - d, 5 - b. The problem of subgraph isomorphism detection can be extended to directed and labeled graphs such as business process models.

The subgraph isomorphism detection problem belongs to the NP-complete complexity class. Thus, in principle any algorithm to solve this problem has an exponential worst-case complexity, meaning that a candidate solution can be checked in polynomial time, but there is no efficient way to identify a solution [Mess 95].

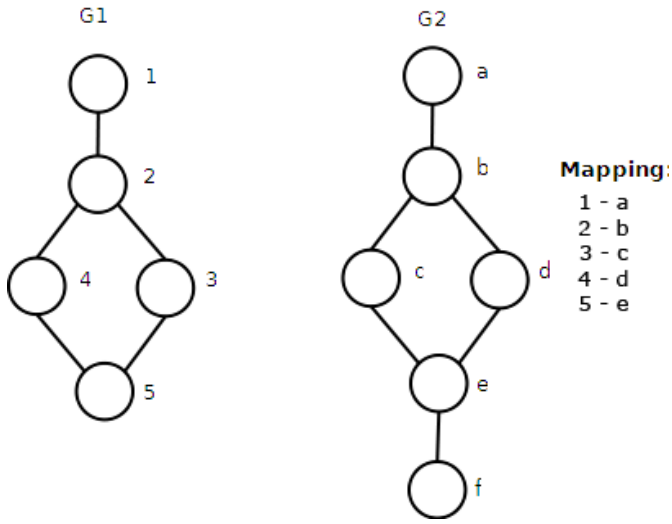

Figure 1.7: Subgraph isomorphism.

There are several algorithms addressing the subgraph isomorphism problem. The most common techniques are based on tree-search algorithms using backtracking [Ullm 76], look-ahead [Hara 80], relaxation [Sche 05] and pruning the search space [Cord 04]. Similarly to the graph isomorphism problem, faster algorithms for graphs with special properties exist, for instance, a linear algorithm in case of planar graphs [Epps 99]. Most of the algorithms take only two graphs as input; however, algorithms that work on a collection of graphs have also been introduced [Mess 00].

1.2.2.3 Maximum Common Subgraph Isomorphism

In many cases, graphs are not identical and one is not subsumed by another, but still they share significant amount of similarity. In these cases, the graphs may share a connected substructure. For instance, in the area of chemoinformatics – where there is a need for the identification of maximal common substructures that occur in many structures [Brin 87]. This problem is also relevant in the field of image and video database retrieval [Shea 01]. The problem of finding maximal common substructures

also occurs in case of process models – when finding the largest common fragment that occurs in many models in order to refactor this out as a standalone subprocess.

Problems described above can be solved using maximum common subgraph isomorphism detection algorithms. Specifically, given two graphs $G1$ and $G2$, the maximum common subgraph isomorphism relates to the problem of finding a maximal subgraph of $G1$ that is isomorphic to a subgraph of $G2$. The problem of maximal common subgraph detection is depicted in Figure 1.8.

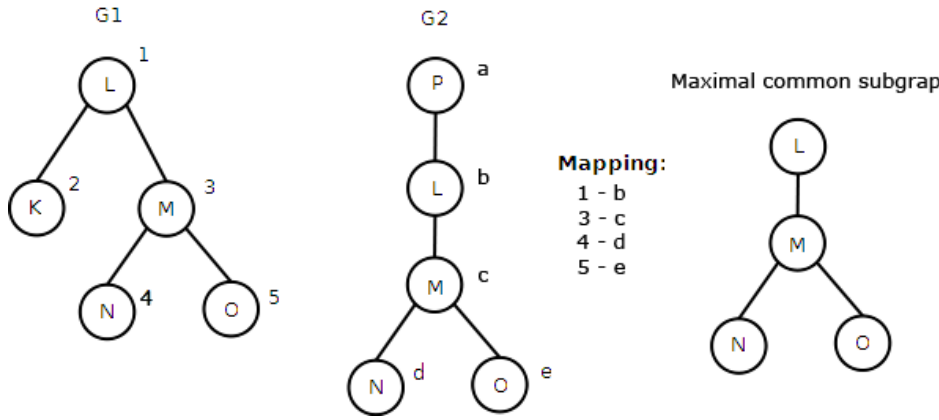

Figure 1.8: Maximal common subgraph isomorphism.

The graphs $G1$ and $G2$ in Figure 1.8 are structurally isomorphic as seen in Figure 1.6. In the case of labeled graphs, such as the one in Figure 1.8, it is feasible to also preserve the node labels. Despite the fact that the graphs are significantly similar, there is no label and structure preserving graph or subgraph isomorphism. Instead, a maximum common subgraph isomorphism exists. Figure 1.8 also represents a maximal common subgraph that appears in both graphs, $G1$ and $G2$.

The maximal common subgraph problem is in the NP-hard complexity class [Kann 92], meaning that it is at least as hard as the hardest of the NP problems. There are approximate and exact algorithms for

maximal common subgraph isomorphism detection. Exact algorithms are based on the maximal clique problem [Bomz 99, Mess 95], clique branching [Sute 05], backtracking [McGr 82] and decision trees [Shea 01]. Approximate algorithms define heuristics in order to estimate a solution within acceptable time complexity. These algorithms are based on genetic algorithms [Cici 00], combinatorial optimization and fragment storage [Raym 02].

1.2.2.4 Error-correcting Graph Isomorphism

In real world applications, imprecisions exist in structural descriptions caused by noise or distortion. These situations occur, for instance, in pattern recognition and image processing [Cont 03]. The algorithms mentioned previously may not give the desired output in these cases.

Error-correcting subgraph isomorphism addresses this problem by taking into account the notion of "error" during graph matching. Usually, a cost is attributed to each type of error and the result of an algorithm is a (dis)similarity measure [Cont 03]. For example, one particular type of error occurs when a node in one graph is mapped to a node in another graph such that the labels of these nodes is slightly different. In the case of business process subgraph, such errors might come from the fact that process models are designed by different stakeholders who use different naming conventions and vocabulary. Yet, despite these errors, we wish to determine if there is a fragment of one process graph that resembles a fragment in another process graph, and more broadly, we wish to determine if two business process graphs are similar, meaning that they share a significant volume of similar fragments.

Similarly to the subgraph isomorphism problem, the problem of error-correcting subgraph isomorphism detection belongs to the NP-hard complexity class [Mess 95].

There are numerous algorithms that address the error-correcting subgraph isomorphism problem. They are based on tree-search algorithms

like the A-star [Nils 82], genetic algorithms [Wang 97], probabilistic relaxation [Chri 95] and neural network training techniques [Neuh 06].

Error-correcting subgraph isomorphism detection can be formulated as a graph edit distance problem. The idea is related to the string edit distance problem where the distance between two strings is described by the number of edit operations that are needed to transform one string to another [Leve 66]. The distance between two graphs can be described as the number of graph edit operations – the insertion, deletion and replacement of nodes and edges [Mess 98]. Similarly to the string edit distance problem [Rist 97], the costs for the edit operations can be obtained automatically using corpus of examples [Mess 00]. In the case of labeled graphs, and under some additional assumptions, the graph edit distance can also be calculated using a Munkres' (a.k.a. Hungarian) algorithm [Munk 57]. In [Ries 07], Munkres' algorithm is extended to be applicable to finding the edit distance between graphs.

1.2.2.5 Graph Matching and Business Process Model Similarity

The methods defined in the area of graph matching give us a foundation for defining notions of similarity between business process models. However, graph matching techniques suffer from scalability problems due to their inherently high computational complexity. By taking into account the specificities of process models, it is possible to design more specialized but at the same time more efficient graph matching heuristics to identify commonalities between business process models.

Graph matching techniques emphasize mainly the structure of models. However, process models contain significant amount of information in their node labels. Therefore, when matching business process models, we need to consider both the information contained in the graph structure, but also the syntactical and semantical information carried by the graph nodes.

1.2.3 Schema Matching

Finding similarities between process models is similar to the database schema matching problem. Numerous techniques have been developed for merging heterogeneous database schema into a unified schema [Do 02a, Rahm 01]. Most of the schema matchers address the problem of $1:1$ matching only because of the difficulty to automatically derive the other types of matches ($1:n$, $n:1$, $m:n$); only some of them cope with $1:n$ matchings [Rahm 01].

The first step behind all the schema matchers is to find an alignment between schema elements using their lexicographical information. Schema matching solutions propose different metrics and instructions for comparing schema elements using their syntactical and semantical information [Do 02b, Madh 01, Mitr 99, Berg 99]:

- **Normalization** – element names are tokenized using punctuation, special symbols, digits, etc. as token separators. Abbreviations are expanded and tokens like articles, prepositions and conjunctions are removed. Tokens are stemmed to their roots. For example, name *PO Lines* will be transformed to {*Purchas, Order, Line*}.

- **Categorization** – to reduce the element-to-element comparisons, the elements are clustered into categories – this allows comparing the elements within the same category. Categorization is done using the element names, data types and associated concepts. For example, the category *Money* includes each element that is associated with money (i.e. the name of an element contains token "money" or elements that are related to money – "price", "cost", "value", etc.).

- The **Syntactic similarity** between tokens can be computed using a range of methods, including:

- *Damerau-Levenshtein edit distance* – the similarity between tokens is computed counting the edit operations necessary to transform one string to another [Leve 66, Dame 64, Lowr 75]. In most of the cases, the normalized edit distance is used – the raw edit distance is divided by the maximum edit distance between two given tokens (i.e. the length of the longest of the two tokens) [Lamb 99] or the weight of the editing path is divided by the length of the editing path [Marz 93]. For example, the edit distance between tokens "value" and "evaluate" is 4, meaning that to transform token "value" to "evaluate", in minimal case, four edit operations are needed (adding 'e' to the beginning of the token, changing 'e' to 'a' and adding 't' and 'e' to the end of the token). The normalized edit distance between these tokens is 0.5 (the edit distance divided by the length of "evaluate").
- *N-grams* – the similarity between tokens is measured based on counting the number of unique n-grams (i.e. substrings with the length of n characters) in the two input strings. The more n-grams the two strings share the more similar these strings are [Ukko 92]. For example, to find 3-gram similarity between tokens "value" and "evaluate", we firstly need to identify unique 3-grams in both of them. In "value", unique 3-grams are "val", "alu", "lue" and in "evaluate", unique 3-grams are "eva", "val", "alu", "lua", "uat", "ate". The similarity between these tokens can be calculated as the ratio of common 3-grams to the all 3-grams in both tokens. In our example, common 3-grams are "val" and "alu", therefore, the similarity between these tokens is 0.45.
- *Affix* – the similarity is calculated using common affixes, i.e. both prefixes and suffixes, between token strings.

- **Semantic similarity** – the semantic similarity may be computed based on loop-ups of synonymy, hypernymy and holonymy relations captured in a thesaurus. Each thesaurus entry is annotated with a coefficient that indicates the strength of the relationship. For example, these semantic relations between tokens can be automatically derived using the Wordnet [Mill 95] lexical system.

- **Name similarity** – the name similarity (ns) of two sets of name tokens T_1 and T_2 may be defined as the average of the best similarity of each token from T_1 with each token from T_2. This measure is calculated as follows:

$$\mathrm{ns}(\mathrm{T}_1, \mathrm{T}_2) = \frac{\sum_{t_1 \in T_1} \left[\max_{t_2 \in T_2} sim(t_1, t_2) \right] + \sum_{t_2 \in T_2} \left[\max_{t_1 \in T_1} sim(t_1, t_2) \right]}{|T_1| + |T_2|}$$

The output of the $ns(T_1, T_2)$ is the similarity score that is used to match the database schema elements.

Some techniques try to represent the database schema as a graph and also use the structural information for schema matching [Do 02b, Meln 02].

There is clearly a lot of room for reusing techniques developed in the context of schema matching to address the problem of process model matching. However, there are fundamental differences between database schema and process models. Firstly, a data schema generally has labeled edges (i.e. associations), edges in a process model usually do not have labels [1]. Secondly, there are fundamental differences in types of nodes and attributes attached to the nodes – for instance, database schema do not have control nodes. Control nodes have a behavioral semantics attached to them. In many cases, different combinations of control nodes may in fact capture the same behavior and should arguably be treated as being equivalent. Thirdly, database schema elements have stricter structural relations – for instance,

[1] This statement applies to process models defined in mainstream process modeling notations such as BPMN or EPCs.

when representing the XML schema as a graph, all the predecessors of a node are describing their successor (i.e. when the type *Personnel* has predecessors *Name*, *Address* and *Telephone Number*, then all these predecessors are describing the *Personnel* type). Therefore, database schema comparison and merging methods are not exactly applicable in case of process models.

1.3 Problem Statement

The management of large process model repositories requires effective techniques in order to find and organize similarities among various business process models. For example, before adding a new process model to a repository, a process analyst needs to check whether similar process models already exist in order to prevent duplications. Similarly, in the context of company mergers, ones need to identify common or similar business processes between the merged companies in order to analyze their overlap and identify areas for consolidation. This leads us to the following problem – after identifying the common parts, how to represent these models in order to reduce redundancy and improve the manageability of the model collection?

More precisely, the management of a business process model collection requires dealing with the following problems:

- Given two process models, how to identify the commonalities between these models. Which elements in these models represent the same process fragments?
- Given two or more business process models, how to find all fragments that are shared among these models effectively in order to refactor these out as subprocesses?

- Given two or more business process models, how to find and represent common fragments which cannot be refactored out as subprocesses. Moreover, how to construct an aggregated model which does not contain duplicate fragments and incorporates all the behavior of the input models?

In this thesis we propose two complementary approaches for merging process models – process merging by refactoring out common subfragments into separate subprocesses and process merging by representing the similar models in aggregate process models using configurable process models.

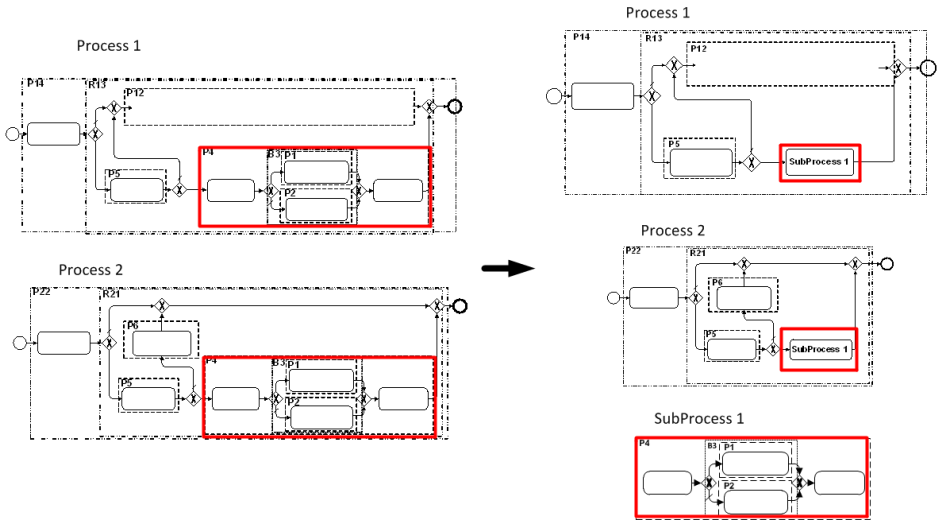

Figure 1.9: Process merging by subprocess refactoring.

In figure 1.9 are depicted two process models, *Process*1 and *Process*2, that share a common fragment. When merging process models by subprocess refactoring, these common fragments are extracted as subprocesses and all the fragment occurrences are replaced by a subprocess call.

Process merging by subprocess refactoring operates on a collection of models. The models are stored in the database so that duplicate fragments are represented only once. Detected fragments are single-entry-single-exit

(SESE) fragments, which enables refactoring using simple call-and-return semantics.

On the other hand, process merging using configurable models enables one to merge process models in case the common fragments are not exact SESE fragments. This approach allows business analysts to manage entire families of similar process models simultaneously. There are two steps in this merging process – identifying common fragments in process models and merging the models into a configurable process model so that the common fragments are represented only once.

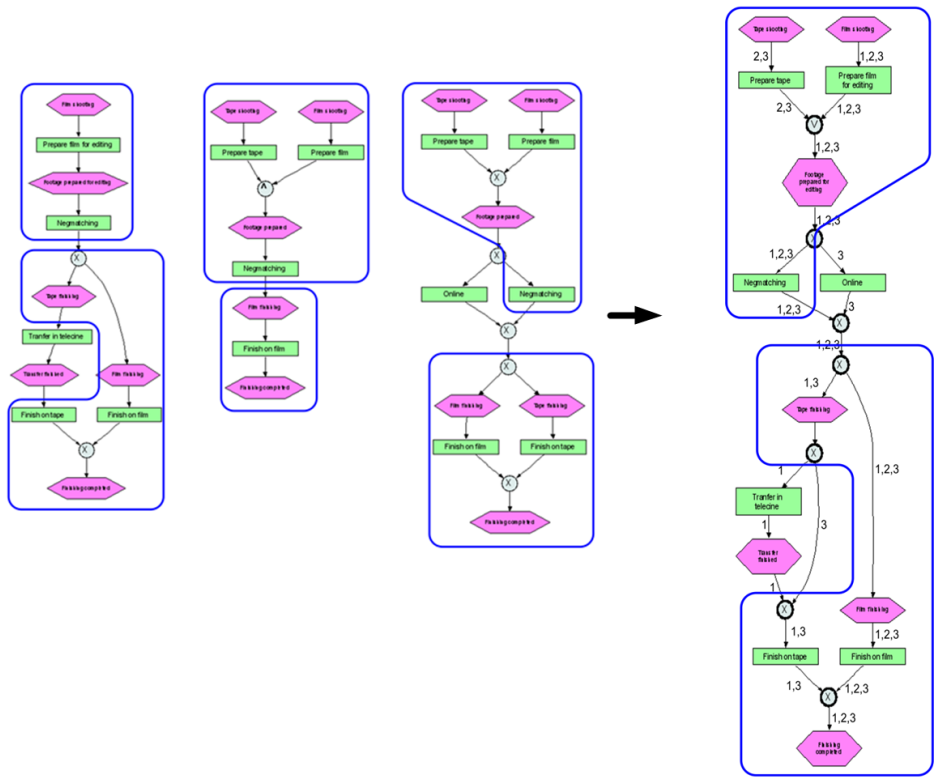

Figure 1.10: Process merging using configurable models.

Figure 1.10 describes a situation where the common fragments are not exact SESE fragments. In the left-hand side we have three input models and

their overlapping fragments. In the right-hand side we have an configurable model that subsumes the behavior of the left-hand side models and in which the common fragments are presented only once.

In this thesis we investigate different metrics for detecting similarities in process models. Additionally, we develop algorithms for merging process models using the identified common fragments.

1.4 Publications and Contributions

This dissertation is based on four articles whose contributions are listed below.

- **Publication 1:** Clone Detection in Repositories of Business Process Models
  - This article concentrates on indexing process models in order to facilitate finding duplicate model fragments that can be factored out as subprocesses. This paper addresses the problem of retrieving all clones in a process model repository that can be refactored into shared subprocesses. Specifically, the contribution of the paper is an index structure, namely the RPSDAG, that provides operations for inserting and deleting models, as well as an operation for retrieving all clones in a repository that meet certain requirements. For this paper, I contributed to the design of the indexing structure. I implemented the prototype, conducted all the experiments, and wrote the Evaluation section of this paper.
- **Publication 2:** Similarity of Business Process Models: Metrics and Evaluation
  - This paper studies three classes of similarity metrics to answer process model similarity queries. The contribution of this paper

is that it presents and validates a collection of similarity metrics. For this paper, I designed one of the three metrics, implemented it and conducted the experiments to compare the three classes of metrics. I also contributed to the write-up of the Evaluation section of the paper.

- **Publication 3:** Aligning Business Process Models
  - Motivated by the previous paper, we investigate techniques, based on lexical matching and error-correcting graph matching, in order to align business process models. The contribution of this paper is that it presents and validates a collection of techniques for automatically matching similar tasks from different processes. For this paper, I designed one of the techniques for aligning process models, implemented this technique and conducted the experiments to compare the three classes of metrics. I also contributed to the write-up of the Evaluation section of the paper.
- **Publication 4:** Merging Business Process Models
  - In this paper we concentrate on merging business process models using the matching techniques that were investigated in the previous papers. The main contribution of the paper is an algorithm that takes as input a collection of process models and generates a configurable process model. For this paper, I contributed to the design of the model merging technique. I implemented the prototype, conducted all the experiments, and wrote the Evaluation section of this paper.

1.5 Structure of the Thesis

This thesis is structured as follows: Chapter 2 corresponds to the publication "Clone Detection in Repositories of Business Process Models". This

chapter analyzes the problem of merging by subprocess refactoring. Specifically, this chapter focuses on the most challenging part of subprocess refactoring, which is that of finding duplicate fragments that can then be refactored as subprocesses. We introduce a model storage method where the models are inserted into a database in such a way that duplicate fragments are stored only once. This also accelerates finding duplicate fragments. The problem with this approach is that we can only deal with exact fragments. However, there are cases when the models in a model collection have high level of similarity, but they do not share exact SESE fragments. In order to deal with these cases, we developed another merging technique where similar models are merged to an aggregate model so that the initial models can be restored from it using a technique called individualization. Chapter 3 corresponds to the publications "Similarity of Business Process Models: Metrics and Evaluation" and "Aligning Business Process Models". In this chapter, we describe the problem of finding commonalities in process models and aligning them accordingly. We compare various algorithms that can be used to determine the commonalities and similarity degree of the process models. Then we use some of these algorithms to align the models and find similar regions. This is the groundwork for the method for process merging using configurable models that is introduced in Chapter 4, which corresponds to the publication "Merging Business Process Models".

Chapter 2

Business Process Merging by Refactoring Common Fragments

The problem of refactoring arises when a process model repository has frequently overlapping regions among various process models. A common practice is that new process models are created by extending or refining existing models, or by copying and merging fragments from multiple models. Therefore, the problem of overlapping fragments is actual in large process model repositories. Managing these fragments individually produces inconsistencies, since fragments that should evolve synchronously start diverging from one another.

2.1 Contributions

In the first publication [Uba 11], we studied the problem of finding frequently occurring exact model fragments in a business process repository. Our aim is not to retrieve all fragments that are isomorphic in the sense of graph isomorphism [Mess 95], but to retrieve the process model fragments that can be factored out into separate subprocesses.

Subprocesses are invoked according to a call-and-return semantics. Hence, a subprocess has a single start point and a single end point. Accordingly, we use the Refined Process Structure Tree (RPST) technique [Vanh 09], that takes a process model as input and computes a tree representing a hierarchy of its single-entry-single-exit (SESE) fragments.

SESE fragments contained in the RPST can be classified into one of four classes [Vanh 09]. A *trivial* (T) fragment consists of a single edge. A *polygon* (P) fragment is a sequence of fragments. A *bond* (B) corresponds to a fragment where all child fragments share a common pair of vertices. Any other fragment is a *rigid* (R).

The RPST is essentially a decomposition of a process model into SESE regions, with larger SESE regions appearing at the top of the RPST, and smaller regions appearing below. Figure 2.1 shows an example of process model decomposition into an RPST. For the sake of illustration, a unique identifier is associated with each model fragment. Identifier starting with "P" refers that this fragment is a "polygon'", "B" refers that this is a "bond" and "R" refers that this fragment is a "rigid". If a model fragment already exists, the existing identifier is given to this fragment.

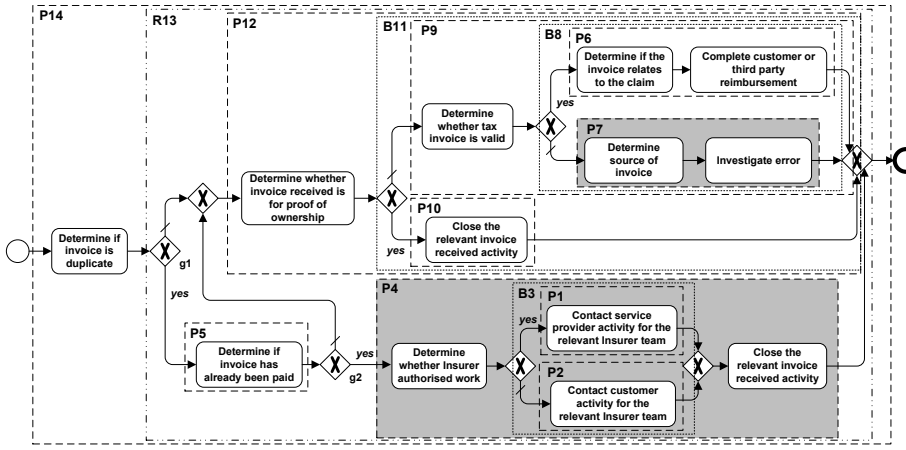

Figure 2.1: Example of RPST decomposition of a model.

Our contribution is introducing the RPSDAG – an index structure that provides operations for inserting and deleting models, as well as an operation for retrieving all clones in a repository that meet the following requirements:

- All retrieved clones must be single-entry- single-exit (SESE) fragments – therefore, these can be extracted to subprocesses and in the initial models the subprocesses can be invoked using call-and-return semantics.
- All clones retrieved must be exact clones so that every occurrence can be replaced by an invocation to a single (shared) subprocess. While identifying approximate clones could be useful in some scenarios, approximate clones cannot be refactored directly into shared subprocesses, and thus, fall outside the scope of this study.
- Maximality – once we have identified a clone, every SESE fragment strictly contained inside this clone is also a clone, but we do not wish to return all such subclones.
- Retrieved clones must have at least two nodes (no "trivial" clones).

For graph indexing, we adapted the graph indexing approach proposed by Williams et al. [Will 07]. This is an indexing technique that assigns a unique canonical code to each graph. Graphs that are isomorphic share a canonical code. This technique allows fast identification of duplicate fragments using string comparison algorithms. Indeed, if we index each SESE fragment using this technique, we can then efficiently determine whether or not a SESE fragment in the RPST of a process model is equal to an already indexed SESE fragment in the RPST of the same model or of another model. When we identify that a SESE fragment is already indexed, we reuse the existing index by making the parent SESE fragment point to the already indexed SESE fragment rather than creating a duplicate of the

SESE fragment. In doing so, we turn a collection of RPSTs into a Directed Acyclic Graph (DAG), since some of the SESE regions in one RPST may point to SESE regions in another RPST. This is the reason for the name RPSDAG.

In [Uba 11], we also describe a representation of the RPSDAG extracted from the RPST as a table structure that allows storing the RPSDAG structure. In this way, all clones can be retrieved by using a simple SQL query.

The implementation of the RPSDAG is available as a standalone application. The program, source code and example models are available for download at `http://apromore.org/tools`. The tool takes a collection of models as input and produces a listing of all clones found.

2.2 Evaluation

We evaluated our technique using four different datasets: the collection of the SAP R3 reference process models [Kell 98], a model repository obtained from an insurance company and two collections from the IBM BIT process library [Fahl 09]. We observed that the construction of a dag and the insertion of a new model to a dag are in acceptable time ranges. Also, the execution time of the SQL query that retrieves all duplicate fragments is in milliseconds, even if the model collection size is more than 500 models. In addition, we observed that real life model collections contain significant amount of duplicate fragments. Thus using our technique can yield a high refactoring gain.

2.3 Related Work

Clone detection in software repositories has been a topic for research for many years already. According to Roy et al. [Roy 09], code clones detection methods can be classified into four main categories: textual, lexical, syntactic and semantic. The last two use the graph-based techniques for

clone detection – more precisely the abstract syntax tree (AST) [Baxt 98] comparison and program dependence graphs (PDG) comparison [Krin 01] accordingly. The AST method [Baxt 98] is not directly applicable in case of business process models because the AST method assumes that the input graph is a tree and applies tree isomorphism detection algorithms. The technique described by Krinke [Krin 01] is based on the PDGs. This algorithm uses the subgraph isomorphism algorithm for clone detection – however, we use the canonical codes that make the subfragment matching faster.

Research on clone detection has also been conducted in the areas of model-driven engineering. In paper by Deissenboeck et al. [Deis 08], a method for detecting clones in large repositories of Simulink/TargetLink models from automotive industry is described. The models are partitioned into connected components and compared pairwise using a heuristic subgraph matching algorithm. The main difference with our approach is that we use canonical codes for fragment comparison instead of the subgraph isomorphism based approach. Another difference is that we use fragment based comparison instead of model pairwise comparison – if one fragment is compared, then this fragment is matched in all of the models in which this fragment occurs.

The problem of clone detection in business process model repositories is also related to the problem of graph database indexing. GraphGrep [Shas 02] is designed to find paths in a graph that match with the regular expression that is given as an input. The indexing is based on the paths that are indexed up to a certain threshold length; therefore the approach is less useful in case of clone detection. Similar approach, named gIndex, is introduced by Yan et al. [Yan 04]. The indexing is based on frequent fragments. Indexed fragments are as small as possible because smaller fragments are contained in more models; also the fragments are in

a predefined size threshold. Similarly to our approach, the canonical labeling is used for fragment hashing. Unlike our algorithm, this approach does not provide returning all clones from the repository. Additionally, the database is indexed in the preprocessing phase and the quality of an index may degrade over time after numerous insertions and deletions.

In the paper by He et al. [He 06], a method based on graph closure trees is introduced. Given a graph G, the closure tree can be used to retrieve all indexed graphs in which G occurs as a subgraph. We could use the closure tree to index a collection of process graphs so that when a new graph is inserted we can check if any of its SESE regions appears in an already indexed graph. However, the closure tree does not directly retrieve the exact set of graphs where a given subgraph occurs. Instead, it retrieves a "candidate set" of graphs. An exact subgraph isomorphism test is then performed against each graph in the candidate set. In contrast, by storing the canonical code of each SESE region, the RPSDAG obviates the need for this subgraph isomorphism testing.

There is a large body of work in the areas of identifying the common substructures in chemical structures databases, for instance papers by Williams et al. [Will 07] and Deshpande et al. [Desh 03]. The article by Williams et al. [Will 07] is the basis of our refactoring article. It introduces the graph decomposition and hashing in order to facilitate common substructures retrieval. The proposed method is not directly applicable in case of business process model refactoring because we are not interested in all subgraphs. However, the basic ideas from Williams et al. can still be adapted to process model repositories. This adaptation is the main contribution of our work. The article by Deshpande et al. [Desh 03] describes the problem of classification of chemical compounds that is conducted by indexing frequent substructures using canonical labeling. We are not interested in this method in case of clone detection because our objective is to retrieve all clones.

2.4 Limitations and Future Work

In [Uba 11] we proposed a method for effectively finding model clones that satisfy the clone retrieval query conditions. Identifying the minimal clone size and occurrence that is reasonable to refactor to a subprocess is up to future work. It is clear that refactoring out all clones is not rational because this reduces the coherence of the model collection.

Another limitation of our work is that it is focused on identifying clones, but it does not address the problem of actually extracting these clones into shared subprocesses. This latter step is dependent on the modeling notation, since different notations used slightly different approaches for representing subprocesses. Since our work was intended to be notation-independent, the refactoring step was left outside the scope of the thesis. We acknowledge however that the subprocess extraction step is necessary in order to apply the technique in a commercial setting.

Finally, another obvious limitation of the approach is that it is limited to identical clone retrieval. It may so happen that two fragments are almost identical except for negligible differences, due for example to slight differences in naming conventions. Addressing this limitation is a direction for future work. In the extreme case, differences between two common fragments might be substantial to the extent that it is unfeasible to refactor these common fragments into shared subprocesses. In this case, an alternative way of consolidating the common fragments is by constructing an aggregate model. This alternative is the subject of the next chapters.

Chapter 3

Business Process Alignment

As announced in Section 1.3, the second technique for merging business process models that we consider in this thesis is that of merging by using an aggregate model. In the case, the fragments to be merged do not need to be identical, but only "similar". To enable this second approach, we need to have a notion of similarity between process models as well as techniques to detect the degree of similarity between two models or fragments thereof. In this chapter, we introduce our work in the area of business process similarity and alignment.

3.1 Node Similarity

Business process model nodes and their labels carry a lot of information about a process model [Mend 10a]. Therefore, the majority of business process model similarity metrics do not concentrate only on the structure of a process model but also on the information that is stored in the node labels.

Usually, models are modified by different stakeholders; therefore, there is a high probability that they use different terms in order to describe the same things [Ehri 07]. When comparing business process elements, it is

not reasonable to assume that model nodes are equivalent only if they have exactly the same labels.

We use four metrics for calculating the similarity between nodes. These metrics have been put forward and evaluated by Dongen et al. [Dong 08]. All node labels are tokenized and stemmed before calculating the similarity. The similarity is calculated using the following metrics:

- *Syntactic similarity* – the similarity between nodes is calculated based on the string edit distance [Leve 66] of the node labels. The edit distance is normalized to the sizes of input strings.
- *Semantic similarity* – the similarity between nodes is calculated using the semantic information of their labels, using synonymes, hyperonymes, etc. For instance, WordNet [Mill 98] can be used for this purpose.
- *Attribute similarity* – the information of the node attributes is also taken into account for calculating similarity between nodes; for example, types and labels of attributes.
- *Contextual similarity* – this similarity metric also takes into account the structure of the process model, capturing the similarities of the nodes that are connected to it. This is particularly useful when computing the similarity between two control-flow nodes, e.g. two splits.

3.2 Model Matching Techniques

Before merging process models, there is a need to determine the similarities and common process parts of process models – the parts that represent the same subprocess which must occur only once in an aggregate model. This leads to the problem of business process alignment – to determine a mapping between business process models and align nodes that might represent the same element in different models.

In the paper "Similarity of business process models: Metrics and evaluation" [Dijk 11] we compared three different model similarity approaches – node matching, structural similarity and behavioral similarity.

3.2.1 Node Matching Similarity

The node matching similarity technique matches nodes lexicographically, using their labels and attributes. The optimal mapping between models is calculated using the Munkres' algorithm [Munk 57]. In our approach, it is not feasible to match nodes that have low amount of similarity. Therefore, we match nodes if and only if their similarity is above a predefined threshold.

Figure 3.1 shows two models from the insurance domain and achieved mappings using the node matching technique. The node matching technique does not take into account the structural information of models, it matches model nodes even if one of them is in the beginning of the model and the other is at the end.

3.2.2 Structural Similarity

The structural similarity metric takes into account the information stored in the nodes as well as the structure of process models. We define the similarity metric based on the graph edit distance [Hart 68] between business process graphs. Analogously to node matching similarity, the initial mapping is calculated based on the information of the nodes only. Then, using the initial mapping, a graph mapping is found – if both nodes of an edge are mapped, then the edge is considered as mapped. After the mapping phase, the edit distance between models can be calculated based on the number of substituted, deleted and added nodes and edges.

Figure 3.2 shows the same models as depicted in Figure 3.1. The mappings are calculated using the graph edit distance. In some cases, using the structural matching technique, less nodes are matched than in case of

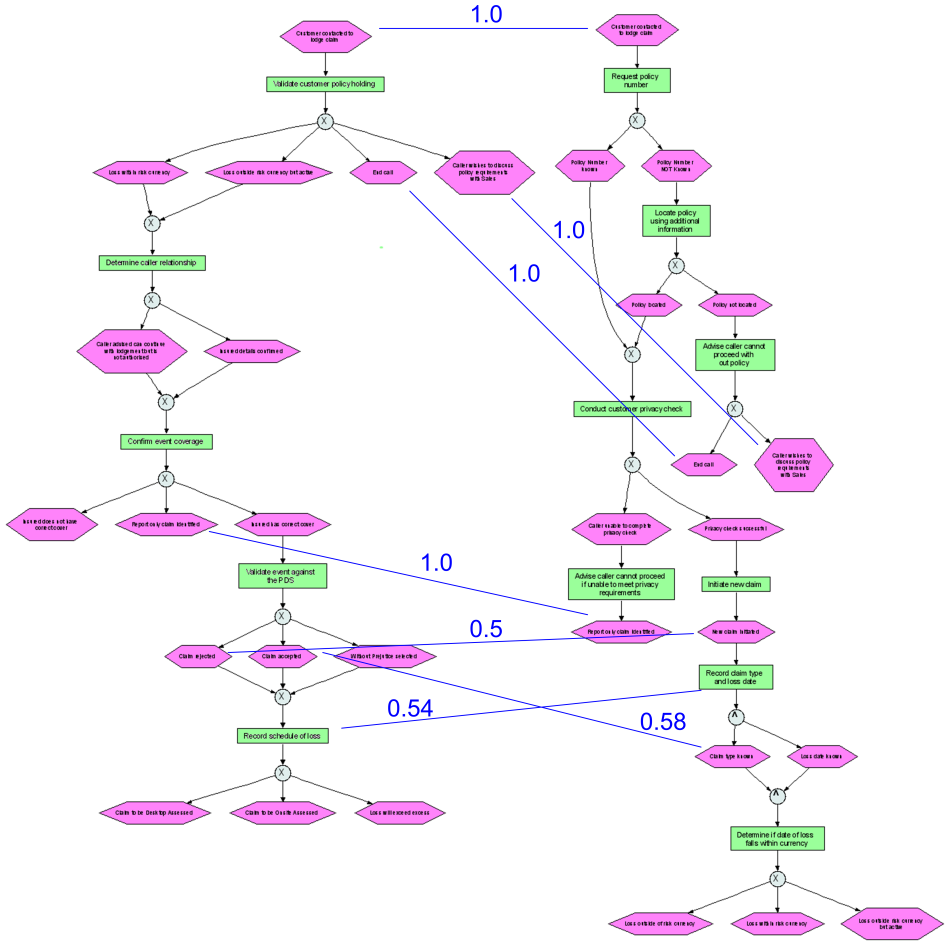

Figure 3.1: Mapping between process models using node matching similarity

pure lexical approach because the structural matching penalizes of matching nodes that are located apart in the process model.

The problem with the structural similarity is that it relies on calculating an error-correcting graph isomorphism which is an NP-complete problem. There are several heuristics to overcome this problem [Dijk 09a]. In paper "Aligning Business Process Models" [Dijk 09b], we focused on two of them that gave the best precision in [Dijk 09a]:

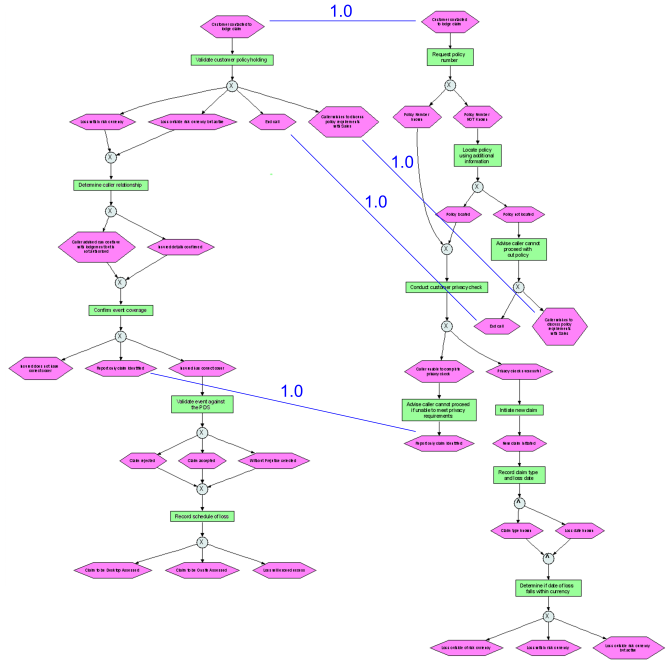

Figure 3.2: Mapping between process models using structural matching similarity

- *Greedy graph matching* – a matching technique that tries to establish a mapping that has an optimal matching score. The score takes into account the similarity of matched nodes as well as the presence or absence of edges between matched nodes. The algorithm starts with an empty mapping. In every step, it adds a node pair to the mapping that increases the mapping score the most. If there are several such pairs, then the pair is selected randomly.

- *A-star graph matching* – the mapping is constructed using the well-known A-star algorithm. The algorithm starts with an empty mapping. In each step, the algorithm selects an existing partial mapping with the lowest edit distance from the partial mapping set (if there are multiple such mappings, the mapping is selected randomly). Then the algorithm selects an unmatched node n_1 from the first graph and

creates mappings with all the nodes n_2, that are not matched in the partial mapping, from the other graph. Additionally, a mapping with the "dummy" node is created – representing the deletion of a node. New mappings are created by adding each pair (n_1, n_2) to the mapping in the partial mapping set that has the smallest graph-edit-distance (i.e. the most "plausible" partial mapping seen so far). This procedure is repeated until all nodes are matched.

3.2.3 Behavioral Similarity

Behavioral similarity metrics are based on the node labels as well as the causal relations between nodes that represent tasks or events. The benefit of using the behavioral similarity over structural similarity is that the behavioral similarity also takes into account indirect causal relations that might might have been introduced during the insertion or deletion of nodes.

Following previous work by van Dongen et al. [Dong 06], we compute the behavioral similarity of two process models by computing their distance in the document vector space constructed from their causal footprints. The causal footprint of a model is a graph, i.e. causality graph, that contains a set of activities and conditions when those activities can occur. A causal footprint describes the process model at a very high level. This does not capture the entire process model but only its control flow. The behavioral similarity between process models is calculated by computing their distance in the document vector space constructed from their causal footprints.

The implementation of these metrics can be found in the "similarity plugin" of the ProM process mining and analysis framework. [1] The similarity search using the greedy and node matching algorithms is also integrated in the Apromore platform [2].

[1] Available at: http://prom.sourceforge.net.

[2] http://www.apromore.org

3.3 Evaluation

We evaluated the above model matching techniques in two ways: how well they perform in the context of model similarity search, and how well they perform in the context of model alignment.

3.3.1 Similarity Search Evaluation

In "Similarity of business process models: Metrics and evaluation" [Dijk 11], we evaluated node matching, structural and behavioral similarity techniques in the context of model similarity search. The term "model similarity search" refers to the following problem, given a process model (called the *query model*) and given a collection of process models (called the *document models*), find those document models that are most similar to the query model. In our experiments we used the SAP reference model collection [Kell 98]. From the model collection we randomly extracted 100 models. From these 100 models 10 models were randomly selected for query models. The labels of the query models were changed in order to investigate the effect of differences in the node labeling. The relevance of all the possible "query model" and "document model" pairs were manually evaluated by multiple subjects with different levels of expertise in process modeling. This manual rating of the similarity was used as the "golden standard" with respect to which the automated similarity search techniques were compared. As the baseline for comparison we used a text-based engine, namely the Indri search engine [Metz 04].

The quality of the process model similarity techniques was evaluated using the notions of precision and recall. Precision is the fraction of relevant instances among all instances retrieved by an algorithm, recall is the fraction of relevant instances retrieved by an algorithm among all relevant instances. Figure 3.3 shows the average precision and recall scores across all the queries. All the similarity algorithms that were under evaluation

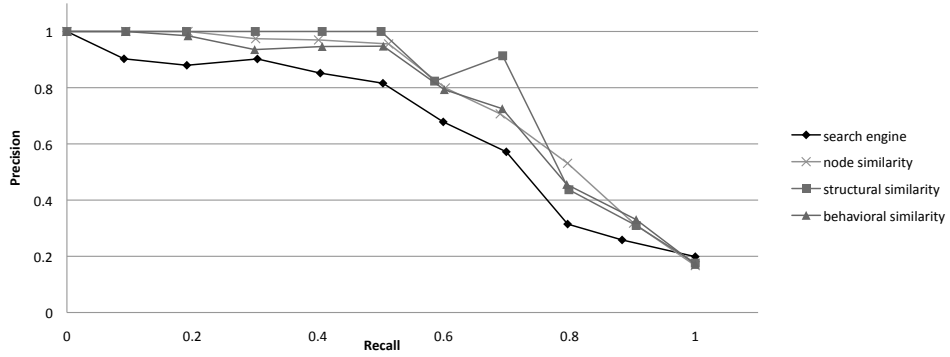

Figure 3.3: Precision-recall curve (precisions are averaged across all 10 queries)

outperformed the pure text-based search engine; the structural similarity yielded the best results.

3.3.2 Model Alignment Evaluation

The previous study [Dijk 11] showed that the structural similarity gives the best results in the case of business process model similarity search. A related problem is that of business process model alignment defined as follows: Given a pair of models M_1 and M_2, find the most adequate mapping between nodes of M_1 and the nodes of M_2. The notion of "most adequate" is subject to expert interpretation. The goal is to come up with automated techniques that mimic as close as possible the judgement of human experts.

In the paper "Similarity of business process models: Metrics and evaluation" [Dijk 11], we only used the A-star algorithm for structural similarity. Therefore, in the paper "Aligning Business Process Models" [Dijk 09b], we also took the greedy matching technique under evaluation and measured their performance in aligning business process models. The pure lexical based technique was also included in the study in order to provide a baseline with respect to which other techniques can be benchmarked.

The algorithms were evaluated using models from the Dutch local government domain. We calculated mappings between 17 process model pairs and compared the results with the mappings determined by human experts. The results were expressed by means of precision, recall and F-Score. F-Score is a measure that considers both the precision and the recall, it can be interpreted as a weighted average of the precision and recall.

Our experiments showed that the pure lexical technique had the lowest precision and F-Score. Also, we denoted that the stemming procedure does not improve the quality of results. A-star algorithm performed moderately better than the naive lexical approach. The greedy algorithm produced the best results, without suffering from the performance bottlenecks of the A-star technique.

3.4 Related Work

There is a large body of work in the areas of model similarity metrics. Most of the proposed business process similarity metrics either remain unvalidated or do not take into account label similarity (they assume that labels are equal in case of node equality) nor behavioral similarity – focusing on structural similarity instead. However, there are works that concentrate on all similarity metrics – label, structural and behavioral similarity – but these works apply their algorithms for computing the similarity between statecharts [Neja 07] or state machines [Womb 06]. In [Neja 07], Nejati et al. proposed a similarity metric for computing similarity between statecharts. The similarity is calculated using the lexical information of the labels of states and behavioral similarity, using the approximation of bi-similarity as well as the nested structure of states in a statechart. Because of the latter feature, the technique is specific to statecharts. Multiple similarity metrics are investigated and evaluated in paper by Wombacher [Womb 06]. The work focuses on workflows modeled using the Finite State Automata. Even

though the business process models can be visualized using reachability graphs (which are basically automata), these can potentially be infinite or exponential in size of the process model [Valm 96].

A body of work has been conducted in the area of business process model similarity. In [Li 08], Li et al. introduced the similarity metric based on high level change operations described in their previous work [Webe 08]. The problem with their approach is that they do not find similarities between processes but transformations from one process to another, assuming that all activities in process models have unique labels. This approach is not applicable in our case when models are not derived from one base model but are originated from different sources. The paper by Lu et al. [Lu 09] introduces the definition of similarity between process variants in terms of their various dimensions. The similarity measure is calculated by comparing process model fragments by means of execution sequences. It is targeted towards querying model variants with certain features from model collections. These features can cover other aspects than tasks and their relations, such as the use of resources and timing characteristics. In paper by Aalst et al. [Aals 06a], process models are compared using their event log containing typical behavior. In our case, the process execution logs are not available. Ehrig et al. [Ehri 07] use syntactic, semantic and structural measures to compute similarity between process model nodes. The overall similarity between models is obtained by aggregating these similarity measures in a combined similarity measure. Another semantical approach is introduced by Brockmans et al. [Broc 06]. These last two approaches are very similar to the semantic matching technique analyzed in our paper, although they do not evaluate their approaches experimentally. Our experiments suggest that approaches based on graph matching are superior to those based purely on linguistic comparison.

Model similarity methods are also used in the areas of gap detection between software capacities and organizational needs. Juntao and Li [Junt 08]

introduced an automatic method for gap analysis. The proposed technique takes two process models as input – one specifying software capacities and the other describing organizational needs. Initially, the similarities between the models are found using lexical similarity and similarity propagation. The mapping between process models is found using the Hungarian algorithm [Munk 57]. Finally, the distance of models is described using high level edit operations – replacement, deletion, movement and insertion of activities.

3.5 Limitations and Future Work

In the experimental evaluation of business process alignment techniques, even the best technique only achieved F-Score in the order of 65%, which is in a way acceptable, but rather low. The reason for this rather low score is that the pairs of models used in this evaluation were in fact quite different. They used very different vocabularies and naming conventions, and in some cases had very different structures. To tackle this heterogeneity, we used tokenization, stemming, Wordnet and even term classification based on the APQC [1]. Therefore, the open question for future work is to develop similarity measures that would perform even if the node labels are very distinct in terms of string edit distance.

The similarity metrics under evaluation were able to match only one node to another. However, in some cases, it would be feasible to match one node to many nodes; for instance, if during process model refinement one node is decomposed into several ones. When finding similarities between these models, it would be more accurate to map one node to several nodes. There are works that define matchers that are able to detect complex correspondences between groups of activities [Weid 10] but do not perform well in terms of precision and recall compared to simple 1-to-1 matchers.

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

Chapter 4

Business Process Merging Using Configurable Models

In the context of company mergers and restructurings, companies often have to manage business process model collections that contain multiple variants of business processes. Usually, these models, originating from different companies or departments, need to co-evolve. Teams of analysts need to analyze the similarities and differences between process models and create integrated process models that can be used to drive the process consolidation effort. This process is time-consuming and error-prone.

4.1 Contributions

In the article "Merging Business Process Models" [Rosa 10], we introduced a semi-automatic approach to aggregate a collection of process models into a single model. The purpose of the merged model is that an analyst can then view the commonalities and differences between multiple variants and manage their co-evolution and convergence – instead of making changes in all models individually, the analyst can change the models in one place in a synchronized manner.

The approach proposed in the article is dictated the following requirements:

- *Behavior-preservation* – the behavior or a merged model must subsume the behavior of all of its input models.
- *Traceability* – an analyst must be able to trace from which process model(s) the element in question originates.
- *Reversibility* – an analyst must be able to derive the initial models from the merged model.

This algorithm takes a collection of models as an input and generates a configurable process model [Rose 07]. A configurable process model captures a family of processes in an aggregated manner and allows analysts to understand what these process models share, what their differences are, and why and how these differences occur. Configurable process models are a suitable output for a process merging algorithm, because they provide a mechanism to fulfill the traceability requirement.

In the first phase, the algorithm finds a mapping between business process models. The mapping can be obtained using the algorithms introduced in Chapter 3. Usually, automatically delivered mappings should be examined by an analyst. In our approach, we use a greedy graph matching algorithm for calculating a mapping between process models because this gave the best results compared to other approaches [Dijk 09b]. Using the calculated mapping between process models, we construct maximum common regions. These maximum common regions are represented in the configurable model only once. In Figure 4.1, three models from the video post production domain are depicted. The figure also shows the common regions among the process models.

The nodes in the common region are connected with the nodes outside of this region using configurable connectors. The edge labels of the configurable merged model indicate the models from which the edge originates.

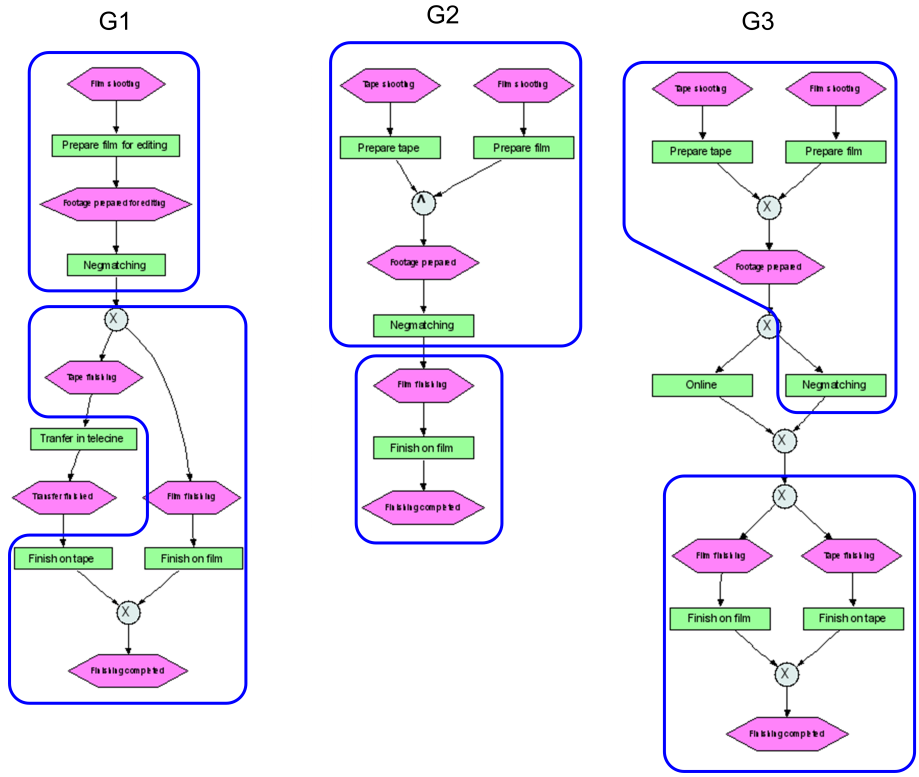

Figure 4.1: Video post production models, common regions.

The merged model of the process models depicted in Figure 4.1 is shown in Figure 4.2. All the common regions are occurring only once in the merged model. Some nodes of the model are annotated with a thicker border to indicate variation points. The initial models can be restored using a procedure known as individualization. Individualization is conducted so that only the edges that contain the identifier of the initial model are retained.

In some scenarios, especially when merging a large number of complex models, it may be convenient to visualize commonalities between models. Therefore, in the extension of the merge paper [Business 11a], we developed a tool to represent most recurrent fragments in the input models. The

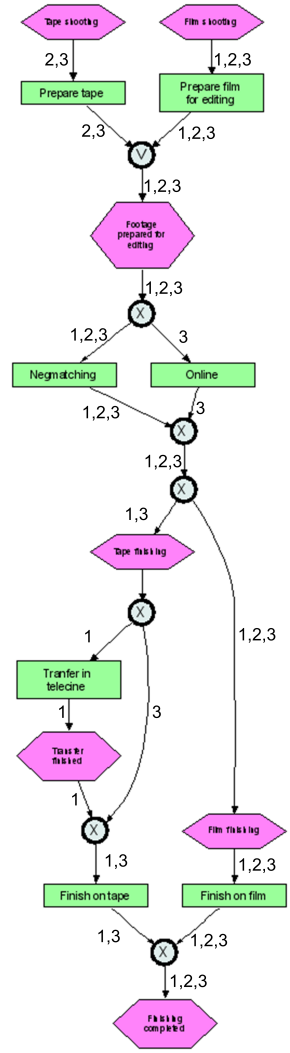

Figure 4.2: Video post production models, merged model.

resulting model is called "digest". Specifically, the digest of a merged graph is a non-configurable process graph that comprises all edges of a merged graph that have a frequency above a given threshold. This algorithm takes a configurable process model as input and retains only the model parts that occur at least a predefined number of times. When removing edges from a

merged graph, we may create a disconnected graph. To avoid disconnected graphs we reconstruct a path where needed and add a special placeholder node, labeled "#", to indicate the presence of nodes in the merged model, which do not satify the digest condition.

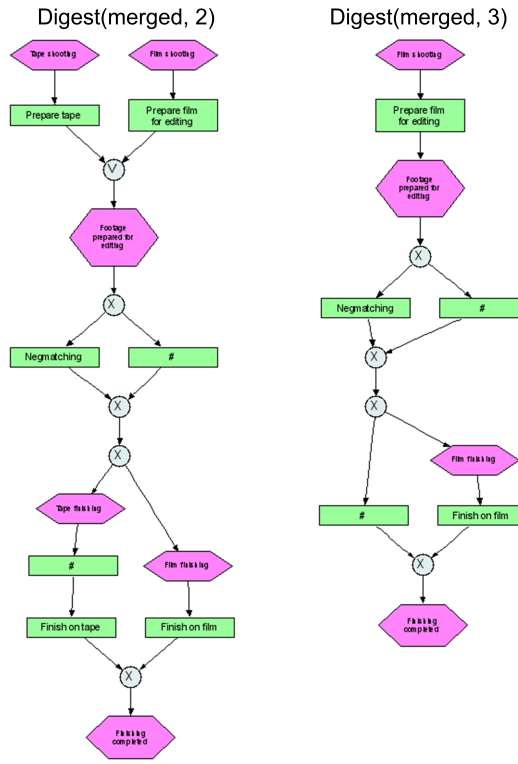

Figure 4.3: Video post production models, digest.

Figure 4.3 depicts a digest view of the merged process model from Figure 4.2. In the left-hand side model, the digest for frequency 2 is visualized. This means that all the edges of process models which occurred at least in two process models are retained. The right hand side model represents the digest with frequency 3 – only these edges which occur in all of the initial process models are retained.

The process merging algorithm has been implemented as a standalone tool, namely Process Merger, that is freely available as a part of the Syner-

gia toolset [1]. The implementation of the algorithm has also been integrated into the Apromore platform [2]. Apromore allows users to store and edit process models in a variety of languages (EPCs, BPMN, YAWL and BPEL). This is made possible via an internal, canonical representation of process models that captures a range of modeling constructs found across multiple process modeling languages. This internal format makes it possible to merge process models that are modeled in different languages. Digests can subsequently be extracted from the merged model.

4.2 Evaluation

We evaluated our algorithm in various aspects – the sizes of the merged models and the overall compression factor as well as the scalability of the algorithm. In addition, we evaluated the usefulness of the algorithm in an industrial setting.

Size is a key factor affecting the understandability of a process model; therefore, it is desirable to have merged models as compact as possible. The sizes of the merged models were evaluated using the SAP reference model collection. The models that had a similarity greater than 0.5 were merged and the compression rate was calculated. The compression rate is the size of the merged model relative to the sizes of the input models. A compression factor close to 0.5 means that the input models are very similar, a compression factor 1 and above shows that the input models are completely different. Our results showed that the average compression factor for this model collection was 0.68, meaning that using the merging algorithm, the average compression rate was 68% compared to the case when the models were just juxtaposed side-by-side.

Additionally, we evaluated the merging algorithm in case of large process models. We considered four model pairs from the domains of land

[1] http://www.processconfiguration.com
[2] http://www.apromore.org

development and insurance. Our experiments showed that the merging operator can handle pairs of models with even around 350 nodes each in a matter of milliseconds.

To evaluate the usefulness of the algorithm, we conducted a case study with a large insurance company. The case study showed that the merge algorithm is useful also in the real life cases. Moreover, after the experiments the insurance company decided to integrate our Process Merger in their development environment to produce batch reports showing the degree of consolidation of their models on a regular basis.

4.3 Related Work

The problem of merging process models has been under investigation in several papers. In [Sun 06], Sun et al. described the problem of merging block-structured workflow nets. The algorithm first finds the mapping pairs – merge points, and then merges the models by applying a set of "merge patterns" (sequential, parallel, conditional and iterative). The merge can be lossless or lossy. The last one refers to the fact that it is not guaranteed that all tasks of initial models remain in the merged model. Therefore, this approach does not satisfy our reversibility requirement. Also, it is not possible to trace from which model the nodes originated. Another drawback is that the proposed method is not fully automated.

Küster et. al. [Kust 08] introduced a method for process merging. Their approach is divided into three steps. In the first step, the differences between models is detected, using correspondences between process models and the SESE fragment technique they present in [Vanh 07]. In the second step, they visualize the differences, and in the last step, the process models are iteratively merged based on the modeler's input. Their main aim is to assist modelers in the merging procedure not conduct the process automatically.

Gottschalk et al. introduced the problem of merging the EPC diagrams [Gott 08]. The focus of their work was on integrating the behavior of the input models to the merged model. In their approach, they use abstraction of the EPC models, namely a function graph, where the original EPC models are reduced to their active nodes – functions and connectors are replaced with edge annotations. The graphs are merged using a set union. The solution proposed by them does not satisfy our traceability and reversibility requirements. Also, their approach does not support approximate node label matching. Finally, they assume that the input models have a single start and a single end node.

Li et al. [Li 10] presents a different approach to the model merging. Given a collection of similar process models (process variants) their goal is to construct a reference model such that the average distance between the reference model and input models is minimal. Intelligibly, the reference model does not subsume the behavior of the initial models; also, the traceability is not provided. Additionally, their approach only works for block-structured process models with AND and XOR blocks.

Mendling and Simon [Mend 06] describe the process of business process model view integration. A process model view is the instantiation of a process model for a specific stakeholder or business object involved in the process. The similarities between models can only be defined in terms of functions and events, connectors and more complex graph topologies are not taken into account. Moreover, a method for finding correspondences is not provided. Models that are merged may be partial views of the process model; therefore, the merged model allows these views to be run in parallel. In other words, the corresponding parts are separated by AND connectors. However, this approach may introduce deadlocks in the merged models. This merging method also fails to correspond to our traceability and reversibility requirements that we stated for our method.

The problem of maintaining a large set of business process models is discussed in paper by Reijers et al. [Reij 09]. Their solution was to maintain the process models in an aggregate manner. The difference from our approach is that the method they proposed is mostly manual and addressed to be applicable in the process designing phase instead of restructuring existing models. Also, their solution is proposed for one modeling notation (EPCs) while our approach is applicable to other modeling notations as well, due to the process graph abstraction.

4.4 Limitations and Future Work

The drawback of the method proposed above is the possibility that the resulting merged models can become relatively large and complex even after applying some optimizations introduced in our paper [Rosa 10]. Several studies have shown [Mend 07, Mend 10b] that large process models are more difficult to maintain and comprehend and they have a higher error probability than small models. Therefore, it might be easier to work with individual modes, but still, in a way that ensures that the individual models are kept synchronized. A direction for future work would be to develop methods for maintaining process model variants individually, but at the same time propagating the changes to all variants in a synchronized manner.

Chapter 5

Conclusions

This research introduced two complementary methods for business process merging. Business process model merging helps to reduce redundancies in a process model repository and enables to modify duplicate fragments in the models in a synchronized manner. Business process model alignment and similarity search techniques based on lexical and graph matching help to identify duplicate or similar fragments for the purpose of process model merging.

The first method concentrated on refactoring recurring model fragments into separate subprocesses. The main restriction of this approach is that the fragments that can be extracted in this way must be identical to one another and must have a single entry point and a single exit point so as to comply with the call-and-return semantics of subprocess invocation. Despite this restriction, our experiments indicate that real life business process model collections contain a significant amount of clones that comply with these requirements.

The second approach introduced the idea of merging two or more process models into a single configurable process model. This merging approach can be separated into two phases. In the first phase, there is a need to determine similarities and aligning parts between the input models. We

proposed and evaluated several algorithms that can be used for this purpose. The performance of these algorithms was measured in the areas of business process similarity search and alignment. The second phase for this merging approach is to aggregate input process models, using the discovered commonalities. The aggregate model must be constructed so that the duplications are eliminated as much as possible, the merged model subsumes the behavior of the input models, the input models are restorable from it and it is possible to trace back which model each edge/node comes from.

All the algorithms introduced in this thesis have been implemented as standalone applications or have been integrated into the ProM framework or the Apromore process model repository.

Appendix A

List of Abbreviations

Abbreviation	Meaning	Page
AST	Abstract syntax tree	44
BPG	Business Process Graph	21
BPMN	Business Process Modelling Notation	15
BPMI	Business Process Management Initiative	15
C-EPCs	Configurable EPCs	19
DAG	Directed Acyclic Graph	43
eEPCs	Extended EPCs	18
EPCs	Event-Driven Process Chains	18
IWi	Institute for Information Systems	18
PDG	Program dependence graph	44
RPSDAG	An index structure for storing RPSTs of the models	42
RPST	Refined Process Structure Tree	40
SESE	Single-entry-single-exit	35
UML AD	Unified Modelling Language Activity Diagrams	20

References

[Aals 06a] W. M. P. van der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters. **Process Equivalence: Comparing Two Process Models Based on Observed Behavior.** In: *Business Process Management*, pp. 129–144, 2006. 56

[Aals 06b] W. M. P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M. H. Jansen-Vullers. **Configurable Process Models as a Basis for Reference Modeling**. *Business Process Management Workshops*, pp. 512–518, 2006. 19

[Aals 99] W. M. P. van der Aalst. **Formalization and verification of event-driven process chains**. *Information & Software Technology*, Vol. 41, No. 10, pp. 639–650, 1999. 18

[Arvi 06] V. Arvind and J. Köbler. **On Hypergraph and Graph Isomorphism with Bounded Color Classes**. In: *STACS*, pp. 384–395, 2006. 25

[Baec 97] R. Baecker, C. DiGiano, and A. Marcus. **Software Visualization for Debugging**. *Commun. ACM*, Vol. 40, No. 4, pp. 44–54, 1997. 12

[Baxt 98] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. **Clone Detection Using Abstract Syntax Trees**. In: *Proceedings of the International Conference on Software Maintenance*, pp. 368–377, 1998. 44

[Berg 99] S. Bergamaschi, S. Castano, and M. Vincini. **Semantic Integration of Semistructured and Structured Data Sources**. *SIGMOD Record*, Vol. 28, No. 1, pp. 54–59, 1999. 31

[Bomz 99] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. **The maximum clique problem**. In: *Handbook of Combinatorial Optimization*, pp. 1–74, 1999. 29

[Brin 87] A. T. Brint and P. Willett. **Algorithms for the identification of three-dimensional maximal common substructures.** *Journal of Chemical Information and Computer Sciences*, Vol. 27, No. 4, pp. 152–158, 1987. 27

[Broc 06] S. Brockmans, M. Ehrig, A. Koschmider, A. Oberweis, and R. Studer. **Semantic Alignment of Business Processes.** In: *ICEIS (3)*, pp. 191–196, 2006. 56

[Business 11a] **Business Process Model Merging : An Approach to Business Process Consolidation**. Accessed: 11 July 2011. `http://eprints.qut.edu.au/38241/`. 60

[Business 11b] **Business Process Modeling Notation (BPMN)**. Accessed: 23 Apr. 2011. `http://www.bpmn.org//`. 15

[Canf 05] G. Canfora, F. García, M. Piattini, F. Ruiz, and C. A. Visaggio. **A family of experiments to validate metrics for software process models**. *Journal of Systems and Software*, Vol. 77, No. 2, pp. 113–129, 2005. 13

[Card 06] J. Cardoso. **Poseidon: a Framework to Assist Web Process Design Based on Business Cases.** *Int. J. Cooperative Inf. Syst.*, Vol. 15, No. 1, pp. 23–56, 2006. 14

[Chri 95] W. J. Christmas, J. Kittler, and M. Petrou. **Structural Matching in Computer Vision Using Probabilistic Relaxation.** *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 17, No. 8, pp. 749–764, 1995. 30

[Cici 00] V. A. Cicirello and S. F. Smith. **Modeling GA Performance for Control Parameter Optimization.** In: *GECCO*, pp. 235–242, Morgan Kaufmann, 2000. 29

[Cont 03] D. Conte, P. Foggia, C. Sansone, and M. Vento. **Graph matching applications in pattern recognition and image processing.** In: *ICIP (2)*, pp. 21–24, 2003. 29

[Cont 04] D. Conte, P. Foggia, C. Sansone, and M. Vento. **Thirty Years Of Graph Matching In Pattern Recognition.** *IJPRAI*, Vol. 18, No. 3, pp. 265–298, 2004. 23

[Cord 04] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. **A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs.** *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 26, No. 10, pp. 1367–1372, 2004. 27

[Corn 70] D. G. Corneil and C. C. Gotlieb. **An Efficient Algorithm for Graph Isomorphism.** *J. ACM*, Vol. 17, No. 1, pp. 51–64, 1970. 26

[Dame 64] F. Damerau. **A technique for computer detection and correction of spelling errors**. *Commun. ACM*, Vol. 7, No. 3, pp. 171–176, 1964. 32

[Davi 07] R. Davis and E. Brabänder. *ARIS Design Platform*. Springer, 2007. 19

[Deis 08] F. Deissenboeck, B. Hummel, E. Jürgens, B. Schätz, S. Wagner, J. Girard, and S. Teuchert. **Clone Detection in Automotive Model-Based Development.** In: *MBEES*, pp. 57–67, 2008. 44

[Derk 10] H. Derksen. **The Graph Isomorphism Problem and approximate categories**. *CoRR*, Vol. abs/1012.2081, 2010. 25

[Desh 03] M. Deshpande, M. Kuramochi, and G. Karypis. **Frequent Sub-Structure-Based Approaches for Classifying Chemical Compounds.** In: *ICDM*, pp. 35–42, 2003. 45

[Dick 04] P. J. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. **Matching graphs with unique node labels**. *Pattern Anal. Appl.*, Vol. 7, No. 3, pp. 243–254, 2004. 25

[Dijk 09a] R. M. Dijkman, M. Dumas, and L. García-Bañuelos. **Graph Matching Algorithms for Business Process Model Similarity Search**. In: *BPM*, pp. 48–63, 2009. 50

[Dijk 09b] R. M. Dijkman, M. Dumas, L. García-Bañuelos, and R. Käärik. **Aligning Business Process Models.** In: *EDOC*, pp. 45–53, 2009. 50, 54, 59

[Dijk 11] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. **Similarity of business process models: Metrics and evaluation**. *Inf. Syst.*, Vol. 36, No. 2, pp. 498–516, 2011. 49, 53, 54

[Do 02a] H. H. Do, S. M., and E. Rahm. **Comparison of Schema Matching Evaluations**. In: *Web, Web-Services, and Database Systems*, pp. 221–237, 2002. 31

[Do 02b] H. H. Do and E. Rahm. **COMA - A System for Flexible Combination of Schema Matching Approaches**. In: *VLDB*, pp. 610–621, 2002. 31, 33

[Dong 06] B. F. van Dongen, J. Mendling, and W. M. P. van der Aalst. **Structural Patterns for Soundness of Business Process Models**. In: *Enterprise Distributed Object Computing Conference, 2006. EDOC '06. 10th IEEE International*, pp. 116–128, 2006. 52

[Dong 08] B. van Dongen, R. Dijkman, and J. Mendling. **Measuring Similarity between Business Process Models**. *Advanced Information Systems Engineering*, pp. 450–464, 2008. 48

[Ehri 07] M. Ehrig, A. Koschmider, and A. Oberweis. **Measuring similarity between semantic business process models**. In: *APCCM '07: Proceedings of the fourth Asia-Pacific conference on Comceptual modelling*, pp. 71–80, 2007. 47, 56

[Epps 99] D. Eppstein. **Subgraph Isomorphism in Planar Graphs and Related Problems**. *CoRR*, Vol. cs.DS/9911003, 1999. 27

[Fahl 09] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. **Instantaneous Soundness Checking of Industrial Business Process Models**. In: *BPM*, 2009. 43

[Fort 96] S. Fortin. **The graph isomorphism problem**. *Department of Computing Science, University of Alberta*, 1996. 25

[Giag 01] G. M. Giaglis. **A taxonomy of business process modelling and information systems modelling techniques**. *Interna-*

tional Journal of Flexible Manufacturing Systems, Vol. 13, No. 2, p. 209, 2001. 12

[Gott 08] F. Gottschalk, W. M. P. van der Aalst, and M. H. Jansen-Vullers. **Merging Event-Driven Process Chains.** In: *OTM Conferences (1)*, pp. 418–426, 2008. 65

[Gull 00] J. A. Gulla and T. Brasethvik. **On the Challenges of Business Modeling in Large-Scale Reengineering Projects**. In: *ICRE*, pp. 17–26, 2000. 13

[Hara 80] R. Haralick and G. Elliot. **Increasing Tree Search Efficiency for Constraint Satisfaction Problems**. *Artificial Intelligence*, Vol. 14, No. 3, pp. 263–313, 1980. 27

[Hart 68] P. Hart, N. Nilsson, and B. Raphael. **A Formal Basis for the Heuristic Determination of Minimum Cost Paths**. *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, pp. 100–107, 1968. 49

[He 06] H. He and A. K. Singh. **Closure-Tree: An Index Structure for Graph Queries.** In: *ICDE*, 2006. 45

[Herm 00] I. Herman, G. Melançon, and M.S. Marshall. **Graph visualization and navigation in information visualization: A survey**. *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 6, No. 1, pp. 24–43, 2000. 23

[Hopc 74] J. E. Hopcroft and J. K. Wong. **Linear Time Algorithm for Isomorphism of Planar Graphs (Preliminary Report)**. In: *STOC*, pp. 172–184, 1974. 25

[Junt 08] G. Juntao and Z. Li. **Detecting Gaps between ERP Software and Organizational Needs: A Semantic Similarity Based Approach**. In: *IEEE International Workshop on Semantic Computing and Systems*, pp. 21–26, 2008. 56

[Kann 92] V. Kann. *On the approximability of NP-complete optimization problems.* PhD thesis, Royal Institute of Technology, Sweden, 1992. 28

[Kell 98] G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley, 1998. 43, 53

[Ko 09] R. K. L. Ko, S. S. G. Lee, and E. W. Lee. **Business process management (BPM) standards: a survey**. *Business Process Management Journal*, Vol. 15, No. 5, pp. 744–791, 2009. 14, 15, 18

[Korh 08] B. Korherr. *Business Process Modelling: Languages, Goals and Variabilities*. VDM Verlag Dr. Mueller e.K., Apr. 2008. 18

[Krin 01] J. Krinke. **Identifying Similar Code with Program Dependence Graphs**. In: *WCRE*, pp. 301–309, 2001. 44

[Kust 08] J. M. Küster, C. Gerth, A. Frster, and G. Engels. **A Tool for Process Merging in Business-Driven Development.** In: *CAiSE Forum*, pp. 89–92, 2008. 64

[Lamb 99] B. L. Lambert, S. J. Lin, K. Y. Chang, and S. K. Gandhi. **Similarity as a risk factor in drug-name confusion errors: the look-alike (orthographic) and sound-alike (phonetic) model**. *Medical Care*, Vol. 37, No. 12, pp. 1214–1225, 1999. 32

[Lee 98] R.G. Lee and B.G. Dale. **Business process management: a review and evaluation**. *Business Process Management Journal*, Vol. 4, No. 3, pp. 214–225, 1998. 13

[Leve 66] V. I. Levenshtein. **Binary Codes Capable of Correcting Deletions, Insertions and Reversals**. *Soviet Physics Doklady*, Vol. 10, No. 8, pp. 707–710, 1966. 30, 32, 48

[Li 08] C. Li, M. Reichert, and A. Wombacher. **On Measuring Process Model Similarity Based on High-Level Change Operations.** In: *ER*, pp. 248–264, 2008. 56

[Li 10] C. Li, M. Reichert, and A. Wombacher. **The Minadept Clustering Approach for Discovering Reference Process Models Out of Process Variants.** *Int. J. Cooperative Inf. Syst.*, Vol. 19, No. 3-4, pp. 159–203, 2010. 65

[Lind 03] Ann Lindsay, Denise Downs, and Ken Lunn. **Business processes–attempts to find a definition**. *Information & Software Technology*, Vol. 45, No. 15, pp. 1015–1019, 2003. 13

[Lonc 98] S. Loncaric. **A survey of shape analysis techniques.** *Pattern Recognition*, Vol. 31, No. 8, pp. 983–1001, 1998. 23

[Lowr 75] R. Lowrance and R. A. Wagner. **An Extension of the String-to-String Correction Problem**. *J. ACM*, Vol. 22, No. 2, pp. 177–183, 1975. 32

[Lu 09] R. Lu, S. W. Sadiq, and G. Governatori. **On managing business processes variants.** *Data Knowl. Eng.*, Vol. 68, No. 7, pp. 642–664, 2009. 56

[Luks 82] Eugene M. Luks. **Isomorphism of Graphs of Bounded Valence can be Tested in Polynomial Time**. *J. Comput. Syst. Sci.*, Vol. 25, No. 1, pp. 42–65, 1982. 25

[Madh 01] J. Madhavan, P. A. Bernstein, and E. Rahm. **Generic Schema Matching with Cupid**. In: *VLDB*, pp. 49–58, 2001. 31

[Marz 93] A. Marzal and E. Vidal. **Computation of Normalized Edit Distance and Applications**. *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 15, No. 9, pp. 926–932, 1993. 32

[McAd 01] R. McAdam and D. McCormack. **Integrating business processes for global alignment and supply chain management**. *Business Process Management Journal*, Vol. 7, No. 2, pp. 113–130, 2001. 12

[McGr 82] J. J. McGregor. **Backtrack Search Algorithms and the Maximal Common Subgraph Problem.** *Softw., Pract. Exper.*, Vol. 12, No. 1, pp. 23–34, 1982. 29

[McKa 81] BD McKay. **Graph isomorphisms**. *Congressus Numerantium*, Vol. 730, pp. 45–87, 1981. 25

[Meln 02] S. Melnik, H. Garcia-Molina, and E. Rahm. **Similarity Flooding: A Versatile Graph Matching Algorithm and**

Its Application to Schema Matching. In: *ICDE*, pp. 117–128, 2002. 33

[Mend 06] J. Mendling and C. Simon. **Business Process Design by View Integration.** In: *Business Process Management Workshops*, pp. 55–64, 2006. 65

[Mend 07] J. Mendling, H. A. Reijers, and J. Cardoso. **What Makes Process Models Understandable?** In: *BPM*, pp. 48–63, 2007. 66

[Mend 10a] J. Mendling, J. Recker, and H. A. Reijers. **On the Usage of Labels and Icons in Business Process Modeling.** *IJISMD*, Vol. 1, No. 2, pp. 40–58, 2010. 47

[Mend 10b] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst. **Seven process modeling guidelines (7PMG).** *Information & Software Technology*, Vol. 52, No. 2, pp. 127–136, 2010. 66

[Mess 00] B. T. Messmer and H. Bunke. **Efficient Subgraph Isomorphism Detection: A Decomposition Approach.** *IEEE Trans. Knowl. Data Eng.*, Vol. 12, No. 2, pp. 307–323, 2000. 27, 30

[Mess 95] B. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs.* PhD thesis, University of Bern, Switzerland, 1995. 23, 24, 25, 26, 29, 40

[Mess 98] B. T. Messmer and H. Bunke. **A New Algorithm for Error-Tolerant Subgraph Isomorphism Detection**. *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 20, No. 5, pp. 493–504, 1998. 30

[Metz 04] D. Metzler and W. B. Croft. **Combining the language model and inference network approaches to retrieval.** *Inf. Process. Manage.*, Vol. 40, No. 5, pp. 735–750, 2004. 53

[Mill 95] G. A. Miller. **WordNet: A Lexical Database for English**. *Commun. ACM*, Vol. 38, No. 11, pp. 39–41, 1995. 33

[Mill 98] G. A. Miller and C. Fellbaum. *WordNet: An electronic lexical database.* MIT Press, 1998. 48

[Mitr 99] P. Mitra, G. Wiederhold, and J. Jannink. **Semi-automatic Integration of Knowledge Sources**. In: *2nd International Conference on Information Fusion (FUSION 1999)*, 1999. 31

[Munk 57] J. Munkres. **Algorithms for the assigment and transportation problems**. *J. Soc. Ind. Appl. Math.*, Vol. 5, pp. 32–38, 1957. 30, 49, 57

[Neja 07] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. **Matching and merging of statecharts specifications**. *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pp. 54–63, 2007. 55

[Neuh 06] M. Neuhaus and H. Bunke. **A Convolution Edit Kernel for Error-tolerant Graph Matching**. In: *18th International Conference on Pattern Recognition*, pp. 220–223, 2006. 30

[Nils 82] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer, 1982. 30

[Peli 99] M. Pelillo. **Replicator Equations, Maximal Cliques, and Graph Isomorphism**. *Neural Computation*, Vol. 11, No. 8, pp. 1933–1955, 1999. 25

[Rahm 01] E. Rahm and P. A. Bernstein. **A survey of approaches to automatic schema matching**. *VLDB J.*, Vol. 10, No. 4, pp. 334–350, 2001. 31

[Raym 02] J. W. Raymond and P. Willett 0002. **Maximum common subgraph isomorphism algorithms for the matching of chemical structures.** *Journal of Computer-Aided Molecular Design*, Vol. 16, No. 7, pp. 521–533, 2002. 29

[Reij 09] H. A. Reijers, R. S. Mans, and R. A. van der Toorn. **Improved model management with aggregated business process models**. *Data Knowl. Eng.*, Vol. 68, No. 2, pp. 221–243, 2009. 12, 66

[Ries 07] K. Riesen, M. Neuhaus, and H. Bunke. **Bipartite Graph Matching for Computing the Edit Distance of Graphs**. In: *GbRPR*, pp. 1–12, 2007. 30

[Rist 97] E. S. Ristad and P. N. Yianilos. **Learning String Edit Distance.** In: *Proceedings of the International Conference on Machine Learning*, 1997. 30

[Rosa 10] M. La Rosa, M. Dumas, R. Uba, and R. M. Dijkman. **Merging Business Process Models.** In: *OTM Conferences (1)*, pp. 96–113, 2010. 19, 58, 66

[Rosa 11] M. La Rosa, H. A. Reijers, W. M. P. van der Aalst, R. M. Dijkman, J. Mendling, M. Dumas, and L. García-Bañuelos. **APROMORE: An advanced process model repository**. *Expert Syst. Appl.*, Vol. 38, No. 6, pp. 7029–7040, 2011. 21

[Rose 06] M. Rosemann. **Potential Pitfalls of Process Modeling: Part A**. *Business Process Management Journal*, Vol. 12, No. 2, pp. 249–254, 2006. 13

[Rose 07] M. Rosemann and W. M. P. van der Aalst. **A configurable reference modelling language**. *Inf. Syst.*, Vol. 32, No. 1, pp. 1–23, 2007. 19, 59

[Roy 09] C. K. Roy, J. R. Cordy, and R. Koschke. **Comparison and evaluation of code clone detection techniques and tools: A qualitative approach.** *Sci. Comput. Program.*, Vol. 74, No. 7, pp. 470–495, 2009. 43

[Sche 00] A.-W. Scheer and F. Habermann. **Making ERP a Success**. *Communications of the ACM*, Vol. 43, No. 4, pp. 57–61, 2000. 13

[Sche 05] C. Schellewald and C. Schnörr. **Probabilistic Subgraph Matching Based on Convex Relaxation.** In: *EMMCVPR*, pp. 171–186, 2005. 27

[Schm 76] D. C. Schmidt and L. E. Druffel. **A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices**. *J. ACM*, Vol. 23, No. 3, pp. 433–445, 1976. 25

[Shas 02] D. Shasha, J. T. Wang, and R. Giugno. **Algorithmics and Applications of Tree and Graph Searching.** In: *PODS*, pp. 39–52, 2002. 44

[Shea 01] K. Shearer, H. Bunke, and S. Venkatesh. **Video indexing and similarity retrieval by largest common subgraph detection using decision trees.** *Pattern Recognition*, Vol. 34, No. 5, pp. 1075–1091, 2001. 27, 29

[Sun 06] S. Sun, A. Kumar, and J. Yen. **Merging workflows: A new perspective on connecting business processes.** *Decision Support Systems*, Vol. 42, No. 2, pp. 844–858, 2006. 64

[Sute 05] W. H. Suters, F. N. Abu-Khzam, Y. Zhang, C. T. Symons, N. F. Samatova, and M. A. Langston. **A New Approach and Faster Exact Methods for the Maximum Common Subgraph Problem.** In: *COCOON*, pp. 717–727, 2005. 29

[Uba 11] R. Uba, M. Dumas, L. García-Ba nuelos, and M. La Rosa. **Clone Detection in Repositories of Business Process Models**. In: *Proceedings of the 9th International Conference on Business Process Management (BPM)*, Springer, Clermont Ferrand, France, September 2011. To appear. 25, 40, 43, 46

[Ukko 92] E. Ukkonen. **Approximate String Matching with q-grams and Maximal Matches**. *Theor. Comput. Sci.*, Vol. 92, No. 1, pp. 191–211, 1992. 32

[Ullm 76] J. R. Ullmann. **An Algorithm for Subgraph Isomorphism**. *J. ACM*, Vol. 23, No. 1, pp. 31–42, 1976. 25, 26, 27

[UML 20 S 11] **UML 2.0 Superstructure Specification**. Accessed: 01 May 2011. `http://www.omg.org/spec/UML/2.0/Superstructure/PDF/`. 20

[Valm 96] A. Valmari. **The State Explosion Problem.** In: *Petri Nets*, pp. 429–528, 1996. 56

[Vanh 07] J. Vanhatalo, H. Vlzer, and F. Leymann. **Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition.** In: *ICSOC*, pp. 43–55, 2007. 64

[Vanh 09] J. Vanhatalo, H. Vlzer, and J. Koehler. **The refined process structure tree.** *Data Knowl. Eng.*, Vol. 68, No. 9, pp. 793–818, 2009. 41

[Wang 97] Y. Wang, K. Fan, and J. Horng. **Genetic-based search for error-correcting graph isomorphism.** *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, Vol. 27, No. 4, pp. 588–597, 1997. 30

[Webe 08] B. Weber, M. Reichert, and S. Rinderle-Ma. **Change patterns and change support features - Enhancing flexibility in process-aware information systems**. *Data Knowl. Eng.*, Vol. 66, No. 3, pp. 438–466, 2008. 56

[Weid 10] M. Weidlich, R. M. Dijkman, and J. Mendling. **The ICoP Framework: Identification of Correspondences between Process Models.** In: *CAiSE*, pp. 483–498, 2010. 57

[Will 07] D. W. Williams, J. Huan, and W. Wang. **Graph Database Indexing Using Structured Graph Decomposition.** In: *ICDE*, pp. 976–985, 2007. 42, 45

[Womb 06] A. Wombacher. **Evaluation of Technical Measures for Workflow Similarity Based on a Pilot Study.** In: *OTM Conferences (1)*, pp. 255–272, 2006. 55

[Yan 04] X. Yan, P. S. Yu, and J. Han. **Graph Indexing: A Frequent Structure-based Approach.** In: *SIGMOD Conference*, pp. 335–346, 2004. 44

Kokkuvõte (Summary in Estonian)

Äriprotsessimudelite ühildamine

Ettevõtted, kellel on aastatepikkune kogemus äriprotsesside haldamises, omavad sageli protsesside repositooriumeid, mis võivad endas sisaldada sadu või isegi tuhandeid äriprotsessimudeleid. Need mudelid pärinevad erinevatest allikatest ja need on loonud ning neid on muutnud erinevad osapooled, kellel on erinevad modelleerimise oskused ning praktikad. Üheks sagedaseks praktikaks on uute mudelite loomine, kasutades olemasolevaid mudeleid, kopeerides neist fragmente ning neid seejärel muutes. See omakorda loob olukorra, kus protsessimudelite repositoorium sisaldab mudeleid, milles on identseid mudeli fragmente, mis viitavad samale alamprotsessile. Kui sellised fragmendid jätta konsolideerimata, siis võib see põhjustada repositooriumis ebakõlasid – üks ja sama alamprotsess võib olla erinevates protsessides erinevalt kirjeldatud. Sageli on ettevõtetel mudelid, millel on sarnased eesmärgid, kuid mis on mõeldud erinevate klientide, toodete, äriüksuste või geograafiliste regioonide jaoks. Näiteks on äriprotsessid kodukindlustuse ja autokindlustuse jaoks sama ärilise eesmärgiga. Loomulikult sisaldavad nende protsesside mudelid mitmeid identseid alamfragmente (nagu näiteks poliisi andmete kontrollimine), samas on need protsessid mitmes punktis erinevad. Nende protsesside eraldi haldamine on ebaefektiivne ning tekitab liiasusi.

Doktoritöös otsisime vastust küsimusele: kuidas identifitseerida protsessimudelite repositooriumis korduvaid mudelite fragmente, ning üldisemalt – kuidas leida ning konsolideerida sarnasusi suurtes äriprotsessimudelite repositooriumites?

Doktoritöös on sisse toodud kaks üksteist täiendavat meetodit äriprotsessimudelite konsolideerimiseks, täpsemalt protsessimudelite ühildamine üheks mudeliks ning mudelifragmentide ekstraktimine. Esimene neist võtab sisendiks kaks või enam protsessimudelit ning konstrueerib neist ühe konsolideeritud protsessimudeli, mis sisaldab kõikide sisendmudelite käitumist. Selline lähenemine võimaldab analüütikutel hallata korraga tervet perekonda sarnaseid mudeleid ning neid muuta sünkroniseeritud viisil. Teine lähenemine, alamprotsesside ekstraktimine, sisaldab endas sagedasti esinevate fragmentide identifitseerimist (protsessimudelites kloonide leidmist) ning nende kapseldamist alamprotsessideks.

Meetodid protsesside ühildamiseks ning neist kloonide leidmiseks on prototüpiseeritud ning allalaetavad eraldi rakendustena ja/või integreeritud protsessijuhtimise süsteemidesse. Meetodid on valideeritud kasutades suuri äriprotsessimudelite repositooriume erinevatest domeenidest. Protsesside ühildamise tööriista on kasutatud kindlustusfirma juhtumianalüüsis.

Dissertatsioon koosneb viiest peatükist. Esimene peatükk on sissejuhatus, milles antakse ülevaade probleemist ja selle üldisest taustast – populaarsematest modelleerimise notatsioonidest ning algoritmidest, mida saab kasutada graafi kujul olevate mudelite võrdlemiseks. Peatükid 2, 3 ja 4 on seotud avaldatud artiklitega.

Artiklite kokkuvõtted on jagatud järgmisestesse teemadesse:

- Peatükk 2: **Protsessimudelite ühildamine kloonide ekstraktimise teel** – refereerib artiklit “Clone Detection in Repositories of Business Process Models”, kus kirjeldasime meetodit äriprotsessimudelite indekseerimiseks, mis aitab kaasa protsessimudelite repositooriumist kiirele korduvate protsessifragmentide leidmisele.

- Peatükk 3: **Protsessimudelite sarnasuse leidmine** – kirjeldab artiklite "Similarity of Business Process Models: Metrics and Evaluation" ja "Aligning Business Process Models" tulemusi. Artiklites uurisime ning võrdlesime erinevaid meetodeid äriprotsessimudelite sarnasuste leidmiseks ning joondamiseks.

- Peatükk 4: **Protsessimudelite ühildamine konfigureeritud mudelisse** – kirjeldab artiklis "Merging Business Process Models" tutvustatud meetodit äriprotsesside ühildamiseks agregeeritud mudelisse.

Artiklite kokkuvõtetele järgneb inglisekeelne dissertatsiooni kokkuvõte, lühendite nimekiri, kirjanduse loetelu ning eestikeelne dissertatsiooni kokkuvõte.

Part II

Papers

Curriculum vitae

General

Name:	Reina Uba
Date and places of Birth:	26.11.1981, Estonia
Citizenship:	Estonian

Education

2007 – 2011	University of Tartu, Faculty of Mathematics and Computer Science, doctoral studies, Specialty: informatics
2006 – 2007	University of Tartu, Faculty of Mathematics and Computer Science, master studies, Specialty: informatics change of curriculum
2001 - 2006	University of Tartu, Faculty of Mathematics and Computer Science, bachelor studies, Specialty: informatics
... - 2001	Elva Gymnasium

Work experience

06/2011 - ...	Webmedia, system analyst
05/2007 – 01/2008	University of Tartu, software developer
03/2006 - 06/2011	Playtech Estonia, software developer

Elulookirjeldus

Üldandmed

Nimi:	Reina Uba
Sünniaeg ja koht:	26.11.1981, Eesti
Kodakondsus:	Eesti

Haridus

2007 – 2011	Tartu Ülikool, Matemaatika-informaatikateaduskond, doktoriõpe, Eriala: informaatika
2006 – 2007	Tartu Ülikool, Matemaatika-informaatikateaduskond, magistriõpe, Eriala: informaatika (õppekava vahetus)
2001 - 2006	Tartu Ülikool, Matemaatika-informaatikateaduskond, bakalaureuseõpe, Eriala: informaatika
... - 2001	Elva Gümnaasium

Teenistuskäik

06/2011 -	Webmedia, süsteemianalüütik
05/2007 – 01/2008	Tartu Ülikool, takvara arendaja
03/2006 - 06/2011	Playtech Estonia, tarkvara arendaja

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

1. **Mati Heinloo.** The design of nonhomogeneous spherical vessels, cylindrical tubes and circular discs. Tartu, 1991, 23 p.
2. **Boris Komrakov.** Primitive actions and the Sophus Lie problem. Tartu, 1991, 14 p.
3. **Jaak Heinloo.** Phenomenological (continuum) theory of turbulence. Tartu, 1992, 47 p.
4. **Ants Tauts.** Infinite formulae in intuitionistic logic of higher order. Tartu, 1992, 15 p.
5. **Tarmo Soomere.** Kinetic theory of Rossby waves. Tartu, 1992, 32 p.
6. **Jüri Majak.** Optimization of plastic axisymmetric plates and shells in the case of Von Mises yield condition. Tartu, 1992, 32 p.
7. **Ants Aasma.** Matrix transformations of summability and absolute summability fields of matrix methods. Tartu, 1993, 32 p.
8. **Helle Hein.** Optimization of plastic axisymmetric plates and shells with piece-wise constant thickness. Tartu, 1993, 28 p.
9. **Toomas Kiho.** Study of optimality of iterated Lavrentiev method and its generalizations. Tartu, 1994, 23 p.
10. **Arne Kokk.** Joint spectral theory and extension of non-trivial multiplicative linear functionals. Tartu, 1995, 165 p.
11. **Toomas Lepikult.** Automated calculation of dynamically loaded rigid-plastic structures. Tartu, 1995, 93 p, (in Russian).
12. **Sander Hannus.** Parametrical optimization of the plastic cylindrical shells by taking into account geometrical and physical nonlinearities. Tartu, 1995, 74 p, (in Russian).
13. **Sergei Tupailo.** Hilbert's epsilon-symbol in predicative subsystems of analysis. Tartu, 1996, 134 p.
14. **Enno Saks.** Analysis and optimization of elastic-plastic shafts in torsion. Tartu, 1996, 96 p.
15. **Valdis Laan.** Pullbacks and flatness properties of acts. Tartu, 1999, 90 p.
16. **Märt Põldvere.** Subspaces of Banach spaces having Phelps' uniqueness property. Tartu, 1999, 74 p.
17. **Jelena Ausekle.** Compactness of operators in Lorentz and Orlicz sequence spaces. Tartu, 1999, 72 p.
18. **Krista Fischer.** Structural mean models for analyzing the effect of compliance in clinical trials. Tartu, 1999, 124 p.

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
20. **Jüri Lember.** Consistency of empirical k-centres. Tartu, 1999, 148 p.
21. **Ella Puman.** Optimization of plastic conical shells. Tartu, 2000, 102 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
25. **Maria Zeltser.** Investigation of double sequence spaces by soft and hard analitical methods. Tartu, 2001, 154 p.
26. **Ernst Tungel.** Optimization of plastic spherical shells. Tartu, 2001, 90 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 p.
28. **Rainis Haller.** *M(r,s)*-inequalities. Tartu, 2002, 78 p.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
30. **Eno Tõnisson.** Solving of expession manipulation exercises in computer algebra systems. Tartu, 2002, 92 p.
31. **Mart Abel.** Structure of Gelfand-Mazur algebras. Tartu, 2003. 94 p.
32. **Vladimir Kuchmei.** Affine completeness of some ockham algebras. Tartu, 2003. 100 p.
33. **Olga Dunajeva.** Asymptotic matrix methods in statistical inference problems. Tartu 2003. 78 p.
34. **Mare Tarang.** Stability of the spline collocation method for volterra integro-differential equations. Tartu 2004. 90 p.
35. **Tatjana Nahtman.** Permutation invariance and reparameterizations in linear models. Tartu 2004. 91 p.
36. **Märt Möls.** Linear mixed models with equivalent predictors. Tartu 2004. 70 p.
37. **Kristiina Hakk.** Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 137 p.
38. **Meelis Käärik.** Fitting sets to probability distributions. Tartu 2005, 90 p.
39. **Inga Parts.** Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.
40. **Natalia Saealle.** Convergence and summability with speed of functional series. Tartu 2005, 91 p.
41. **Tanel Kaart.** The reliability of linear mixed models in genetic studies. Tartu 2006, 124 p.
42. **Kadre Torn.** Shear and bending response of inelastic structures to dynamic load. Tartu 2006, 142 p.

43. **Kristel Mikkor.** Uniform factorisation for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.
44. **Darja Saveljeva.** Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 117 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
46. **Annely Mürk.** Optimization of inelastic plates with cracks. Tartu 2006. 137 p.
47. **Annemai Raidjõe.** Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. **Olga Panova.** Real Gelfand-Mazur algebras. Tartu 2006, 82 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
50. **Margus Pihlak.** Approximation of multivariate distribution functions. Tartu 2007, 82 p.
51. **Ene Käärik.** Handling dropouts in repeated measurements using copulas. Tartu 2007, 99 p.
52. **Artur Sepp.** Affine models in mathematical finance: an analytical approach. Tartu 2007, 147 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
54. **Kaja Sõstra.** Restriction estimator for domains. Tartu 2007, 104 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. **Evely Leetma.** Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. **Ants Kaasik.** Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 139 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
60. **Indrek Zolk.** The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
63. **Marek Kolk.** Piecewise Polynomial Collocation for Volterra Integral Equations with Singularities. Tartu 2010, 134 p.

42

64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
65. **Larissa Roots.** Free vibrations of stepped cylindrical shells containing cracks. Tartu 2010, 94 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
68. **Olga Liivapuu.** Graded q-differential algebras and algebraic models in noncommutative geometry. Tartu 2011, 112 p.
69. **Aleksei Lissitsin.** Convex approximation properties of Banach spaces. Tartu 2011, 107 p.
70. **Lauri Tart.** Morita equivalence of partially ordered semigroups. Tartu 2011, 101 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.
74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
75. **Nadežda Bazunova.** Differential calculus $d^3 = 0$ on binary and ternary associative algebras. Tartu 2011, 99 p.
76. **Natalja Lepik.** Estimation of domains under restrictions built upon generalized regression and synthetic estimators. Tartu 2011, 133 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.