Project IST 026850 SUPER
Semantics Utilized for Process management within and between Enterprises

Deliverable 3.6
Advanced Business Process Discoverer
Leading Partner: SAP
Contributing Partner: IBM, PUE, USTUTT
Security Classification: Public (PU)
October, 2008
Version 1.0
## Project Details

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>026850</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acronym</td>
<td>SUPER</td>
</tr>
<tr>
<td>Project Title</td>
<td>Semantics Utilised for Process management within and between EnteRprises</td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://www.ip-super.org">http://www.ip-super.org</a></td>
</tr>
<tr>
<td>EU Project Officer</td>
<td>Werner Janusch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors (Partner)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agata Filipowska (PUE), Monika Kaczmarek (PUE), Zhilei Ma (USTUTT), Ivan Markovic (SAP), Jussi Vanhatalo (IBM), Hagen Voelzer (IBM), Karol Wieloch (PUE)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Owner (Partner)</th>
<th>E-mail</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ivan Markovic (SAP)</td>
<td><a href="mailto:ivan.markovic@sap.com">ivan.markovic@sap.com</a></td>
<td>+49 6227 7 52591</td>
</tr>
</tbody>
</table>
Versioning and Contribution History

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>Initial draft of the deliverable</td>
<td>IM</td>
</tr>
<tr>
<td>0.1</td>
<td>Added input</td>
<td>IBM</td>
</tr>
<tr>
<td>0.2</td>
<td>Added input on auto-completion</td>
<td>PUE, USTUTT</td>
</tr>
<tr>
<td>0.3</td>
<td>Executive summary, conclusion and final formatting</td>
<td>IM</td>
</tr>
<tr>
<td>0.4</td>
<td>Incorporated reviewers comments</td>
<td>PUE, IM</td>
</tr>
<tr>
<td>1.0</td>
<td>Final Version</td>
<td>IM</td>
</tr>
</tbody>
</table>
Table of Contents

Executive Summary 1

1 Introduction 2
1.1 Deliverable Alignment 4
1.1.1 SUPER Architecture 4
1.1.2 SUPER Methodology 5
1.1.3 Modeling Stack Alignment 6
1.1.4 Use Case Alignment 7
1.2 Deliverable Structure 7

2 Semantic Business Process Modeling Methodology 8

3 Overview over the Subtasks 9
3.1 Workflow Graph Parsing Techniques and Applications 9
3.2 Business Process Querying 9
3.2.1 Query relaxation and refinement 9
3.3 Auto-completion for Business Process Models 10
3.3.1 Auto-completion Based on Naming and Functional Annotations of Process Tasks 10
3.3.2 Auto-completion Based on Non-Functional Properties 10
3.4 Goal-driven Business Process Modeling 11
3.5 Guided Naming and Modeling 11

4 Auto-completion for Business Process Modeling 12
4.1 Auto-completion Based on Naming and Functional Annotations of Process Tasks 12
4.1.1 Scenario and User Interface 12
4.1.2 Auto-completion strategies 14
4.1.3 External interface 15
4.1.4 Components 16
4.1.5 Matching procedure 16
4.1.6 Index structure 17
4.2 Non-Functional Properties within Auto-completion – Filtering and Ranking of Tasks using NFP Analyzer 18
4.2.1 Introduction 18
4.2.2 NFP and different perspectives 19
4.2.3 Level of match 22
4.2.4 Implementation 23
4.2.5 Summary 25

5 Conclusions 26

6 References 27
## Appendix: List of Publications

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>The Refined Process Structure Tree</td>
<td>29</td>
</tr>
<tr>
<td>7.2</td>
<td>Automatic Workflow Graph Refactoring and Completion</td>
<td>29</td>
</tr>
<tr>
<td>7.3</td>
<td>A Framework for Querying in Business Process Modeling</td>
<td>29</td>
</tr>
<tr>
<td>7.4</td>
<td>Auto-completion for Executable Business Process Models</td>
<td>29</td>
</tr>
<tr>
<td>7.5</td>
<td>Methodological Extensions for Semantic Business Process Modeling</td>
<td>29</td>
</tr>
<tr>
<td>7.6</td>
<td>Business Functions Ontology and its Application in Semantic Business Process Modelling</td>
<td>29</td>
</tr>
<tr>
<td>7.7</td>
<td>Linking Business Goals to Process Models in Semantic Business Process Modeling</td>
<td>29</td>
</tr>
</tbody>
</table>
Table of Figures

Figure 1. Subtask structure of D3.6 2
Figure 2. SUPER Reference Architecture (Source: D.7.2 Deliverable) 4
Figure 3. SUPER Methodology (Source: D.2.2. deliverable) 6
Figure 4. SUPER Modeling Stack 7
Figure 5. An initial state of the modeling environment for the Auto-completer 12
Figure 6. A list of suggested fragments 13
Figure 7. The result of a single auto-complete action 13
Figure 8. Data flow within the Auto-completer and between other components 16
Figure 9 Design of NFP Analyzer 24

List of Tables

Table 1. Description of NFPs relevant to the service level 21
Executive Summary

This deliverable continues the work in [15] to provide further support to business experts within the business process modeling phase. We first define a semantic business process modeling methodology, which incorporates the usage of the SUPER ontology stack [7, 14] and reasoning technology in the early phases of process design [11]. The methodology is further supported by several techniques, which are described as individual subtasks:

- **Workflow Graph Parsing Techniques and Applications**: the subtask provides a definition of execution semantics for graph-based process languages that is compatible with subprocess definition and detection. Also, it improves the workflow graph parsing technique from [1] and explores the application of new parsing techniques in automatic workflow graph refactoring and completion.

- **Business Process Querying**: this subtask advances the work on querying of business processes [4] with support for query relaxation and refinement.

- **Auto-completion**: this subtask investigates the auto-completion technique in two directions: i) auto-completion based on naming and functional annotations of process tasks and ii) auto-completion based on non-functional properties.

- **Goal-driven Process Modeling**: here we introduce a goal-driven process modeling approach, and illustrate the scenarios for advanced analysis on process models, such as detection of conflicts and redundancies in processes, propagation of goal contributions, facilitated change management and gap analysis.

- **Guided Naming and Modeling**: in this subtask we demonstrate how the Business Functions Ontology, developed within D1.6 [14], may be used for consistent naming of process activities and suggesting appropriate resources in process modeling.

Following the introduction, we give a short summary of the semantic business process modeling methodology and a detailed overview over the contributing subtasks. Then, the subtask focusing on auto-completion is described in a main section, whereas all others are presented in the form of papers attached to the deliverable in the Appendix section.
1 Introduction

Business process (BP) modeling is the first and most important phase in the business process engineering chain. BP models are created by business analysts with an objective to capture business requirements, enable a better understanding of business processes, facilitate communication between business analysts and IT experts, identify process improvement options and serve as a basis for derivation of executable business processes. Designing a new process model is a highly complex, time consuming and error prone task. This is because BP modeling involves several sources of information, models are dynamic and frequently redesigned to adapt to changes, and BP models are often shared by several departments within a company or even between different companies.

In the previous deliverable [15], we presented approaches to subprocess/process fragment detection and business process querying which help in reducing the complexity of process modeling. That is, we targeted the questions of how can the detection of reusable components within existing process models be done in an automated way and how can these components later be reused during process modeling.

This deliverable builds on and advances the efforts reported in D3.3 [15] on process fragment and subprocess detection and business process querying. We further extend the modeling support with additional techniques which flourish the work on business process ontologies presented in [14, 7]. An overview of all subtasks within this deliverable is given in Figure 1.

![Figure 1. Subtask structure of D3.6](image)

Here we briefly describe all subtask extensions depicted in Figure 1:

- **Workflow Graph Parsing Techniques and Applications**: first, we have worked on a definition of execution semantics for graph-based process languages that is compatible with subprocess definition and detection. Second, we improved the workflow graph parsing technique from [1]. Third, we explored the application of new parsing techniques in automatic
workflow graph refactoring and completion. A partner-internal prototype has been created within the exploitation workpackage by IBM.

- **Business Process Querying**: this subtask advances the work on querying of business processes [4] with support for query relaxation and refinement. A partner-internal prototype has been created within the exploitation workpackage by SAP.

- **Auto-completion**: this subtask investigates the auto-completion technique in two directions: i) auto-completion based on naming and functional annotations of process tasks and ii) auto-completion based on non-functional properties. The implementation of the approach is integrated in the BPMO editor. It is based on process fragments identification component reported in D3.3 [15] which additionally was updated to support the latest (2.0.1) BPMO ontology.

- **Goal-driven Process Modeling**: we first show how formal modeling of business goals enables a more intuitive, goal-driven process modeling approach. Second, as a result of explicit linkage of business goal model to business process models, we illustrate the scenarios for advanced analysis on process models, such as detection of conflicts and redundancies in processes, propagation of goal contributions, facilitated change management and gap analysis. An implementation is ongoing within a Master’s thesis at SAP.

- **Guided Naming and Modeling**: in this subtask we demonstrate how the Business Functions Ontology, developed within D1.6 [14], may be used for consistent naming of process activities and suggesting appropriate resources in process modeling, which leads to more readable, comparable, and consistent process models. No implementation is available within SUPER.

All the aforementioned subtasks support the semantic business process modeling methodology [11] (cf. Figure 1), which aims at utilizing the ontologies and reasoning technology developed within SUPER in the early phases of the BPM lifecycle.

Please note that no consolidated implementation of the extensions exists. As of now, it is unclear on the conceptual level in how far the extensions would interfere with each other. The conceptual and prototypical integration of the extensions proposed here is left for future work.

In the remainder of this section, we present the alignment of our work within SUPER and explain how the deliverable is organized.
1.1 Deliverable Alignment

Due to the fact that this deliverable extends the previous one [15], the alignment with other parts of the SUPER project did not change. For the purpose of completeness, we repeat the respective section from [15].

1.1.1 SUPER Architecture

The SUPER Reference Architecture, as depicted in the figure below, consists of four building blocks communicating via the Semantic Service Bus, namely components related to: execution, SUPER services, tools and repositories. The detailed description of the SUPER architecture is provided in the deliverable D.7.2.

![SUPER Reference Architecture](image)

Figure 2. SUPER Reference Architecture (Source: D.7.2 Deliverable)

The discovery component, between such services as Transformation, Composition, Process Mediation and Data Mediation, is one of the core services provided by the SUPER project. It offers the following functionalities:

- **process fragments detection** – dealing with identification of reusable process fragments within the BPMO process description that are to be approved by the Business Analyst,
- **process fragments discovery** – covering the goal based matchmaking when the user formulates his request consisting of a set of criteria using a query template. The user will be able to specify constraints both on static as well as behavioural aspects of the process fragment description. This request is then matched to descriptions of process fragments from
the BP library and a ranked list of fragments matching the specified goal is presented to the user.

- **auto-completion** – supporting modeling of processes. After a user modelled a part of the process, he may launch the auto-completion functionality asking for process fragments (which have been identified by process fragment detection) that complete what he started to model. The modelled part of the process is matched against the descriptions of process fragments from BP library and the user can select the most suitable process fragment/model from the list of results to include it in his design.

In order to assure provision of the above mentioned functionalities the Discovery component collaborates closely with the repository of process descriptions as well as with the modeling tool.

### 1.1.2 SUPER Methodology

The Semantic Business Process Management Methodology developed in SUPER and presented in the figure below consists of four phases, namely:

- Semantic Business Process Modeling (SBP Modeling), encompassing creation of semantically enriched business process models;
- Semantic Business Process Configuration (SBP Configuration) - where based on the outputs from the SBP Modeling phase, the executable description of the process is prepared;
- Semantic Business Process Execution, during which the execution of the business processes, takes place;
- Semantic Business Process Analysis – used in order to monitor business processes and KPIs and provide feedback on their execution for the needs of continuous process improvement.
Functionalities provided by the Discovery component support Business Analysts when modeling processes and therefore all of them are a part of the Semantic Business Process Modeling phase. This deliverable demonstrates how to support modeling of processes by providing auto-completion, querying of process fragments as well as support this functionality by advanced process fragments detection that enables uncovering fragments in processes.

1.1.3 Modeling Stack Alignment
The SUPER Modeling Stack, as depicted in Figure 4, is structured in five layers from business analysis towards implementation of artefacts. Within this deliverable we utilize the organizational ontologies [14] to support the modeller in business process modeling phase through techniques such as business process querying, auto-completion, goal-driven modeling, etc.
1.1.4 Use Case Alignment

All of the components developed in SUPER are built bearing in mind requirements elicited by business partners and therefore are tested on the examples provided by them. That is also the issue with the discoverer component that is to support the business process modeling phase.

1.2 Deliverable Structure

This deliverable is structured as follows. Section 3 shortly describes the extended semantic business process modeling methodology. A general overview of the subtasks depicted in Figure 1 supporting the methodology is given in Section 4. In Section 5 we present the auto-completion technique for process models in detail and discuss its implementation in BPMO editor. Section 6 concludes the document.
2 Semantic Business Process Modeling Methodology

We first propose methodological extensions for the existing business process modeling methods in order to incorporate the usage of the SUPER ontology stack [7, 14] and reasoning technology in the early phases of process design [11]. Note these extensions are also incorporated in the work on quality assurance in business process modeling within WP2. The main aims of the extensions are improving the reuse of business knowledge within process modeling and advanced support in modeling (auto-completion, goal-driven modeling, guided naming and modeling) and therefore creating process models of higher quality, and speeding up the modeling process while reducing its error proneness. The extended methodology consists of 3 phases: Requirements Analysis, Modeling and Model Validation. Each of the main phases is subdivided into respective subphases, and the benefits of using semantic technologies are demonstrated within every subphase by illustrative examples. Our published work on the semantic business process modeling methodology containing further details is included in the Appendix. The extended methodology has been refined and partly realized within the individual subtasks presented in Section 4.
3 Overview over the Subtasks

In this section, we provide an overview over the individual subtasks contributing to the realization of the semantic business process modeling methodology. For the subtasks where we worked on publications, we here refer to the respective papers which can be found in the Appendix. Where paper has been published, the description in this section is only a summary of the contributions.

3.1 Workflow Graph Parsing Techniques and Applications

This subtask advances the efforts reported in D3.3 on process fragment and subprocess detection. Within this deliverable, we have worked on a definition of execution semantics for graph-based process languages (e.g. BPMN) that is compatible with subprocess definition and detection. Some constructs in those languages (e.g. OR-join gateway) currently obstruct subprocess detection, effective process composition, efficient execution and roundtripping between graph-based model and block-based model. We proposed a novel semantics for those constructs that solves the above mentioned problems.

Our proposal is included in the current draft of the response to the OMG RFP for BPMN 2.0 submitted by BEA Systems, IBM, Oracle and SAP. The initial submission to the OMG is available as OMG document bmi/2008-02-06.

This semantics allows now to use workflow graph parsing techniques for BPMN subprocess detection. A parsing technique that we have published previously in this project [1] can be used, but we also developed an improved parsing technique [2] that produces better parsing results.

Besides the subprocess detection, these parsing techniques have many other applications such as BPMN to BPEL translation, static analysis and automatic workflow graph refactoring and completion. The latter is explored in detail in an additional contribution [3].

For more details on refined process structure tree and automatic workflow graph refactoring and completion techniques, we refer the reader to the respective publications in the Appendix.

3.2 Business Process Querying

This subtask advances the work on querying of business processes [4], described in D3.3. Here we utilize the organizational ontologies [14] to classify business processes according to multiple process perspectives [14]. The following extensions of the approach in D3.3 were made:

3.2.1 Query relaxation and refinement

Depending on the user query, there can be too many or too few results coming from the process repository. The user should be able to i) specify further constraints in the query, choosing more refined goals, thus retrieving more precise and shorter list of results or ii) eliminate some constraints from the query and look for more abstract goals or more undefined flow structure of the process, thus retrieving more results.
(i) The refinement is done mainly by the navigation inside sub-concepts, skipping instances connected to super-concepts. Since a concept can have many sub-concepts, the interaction with the user may be necessary to choose which path to follow.

(ii) If there are too few results, the querying framework can automatically search for instances connected to the parent concepts of the requested one in order to relax the query. Another way for relaxing a query is to "skip" target concepts in the query, e.g. using the "OR" statements. The user can specify which concepts are required and which are optional in the querying interface.

For further details on the process querying framework, we refer the reader to the respective publication [4] included in the Appendix.

3.3 Auto-completion for Business Process Models

Within the following subsections, we briefly summarize the approach to auto-completion of business process models as a way of supporting process modelers in the modeling phase. The approach consists of two parts: i) auto-completion based on naming and functional annotations of process tasks and ii) auto-completion based on non-functional properties. Further details on the approach and its implementation within the BPMO editor are discussed in Section 4.

3.3.1 Auto-completion Based on Naming and Functional Annotations of Process Tasks

This subtask investigates the auto-completion of process models using process fragments. After the user has selected the part of the process for auto-completion and the auto-completion strategy (see Section 4), auto-completer responds with a list of matching process fragments, the user can select from for auto-completion. The process fragments are matched using several matching strategies which implement various auto-completion strategies. The matching procedure uses string distance measures ([5]) between task’s names and a distance (using subsumption hierarchy) between task’s functional annotations expressed in the Business Functions Ontology [14] for calculating overall distance value. For more details on matching strategies and their implementation within the BPMO editor we refer the reader to Section 4.

3.3.2 Auto-completion Based on Non-Functional Properties

In order to improve the level of precision of the suggested process artefacts for auto-completion and achieve their more refined ranking according to user preferences, within this subtask we investigate the usage of non-functional properties in auto-completion. First, we shortly present the non-functional properties that may be used within auto-completion of modelled business processes in the context of SUPER. Then we delve into the issue of expressing preferences and constraints as well as how they may be propagated to other layers. Finally, we show how the level of match may be computed and how the synthetic indicator may be used in order to create a ranking of candidate artefacts.

The overall approach on auto-completion for business processes is described in [10] and provided in the Appendix of this document.
3.4 Goal-driven Business Process Modeling

In [12], we take a step further and show how formal modeling of business goals enables a more intuitive, intention-driven process modeling approach. For modeling business goals, we use the Business Motivation Ontology, presented in [14]. As a first step, the higher-level goals are specified and further broken down into sub-goals, where the sub-goals can be more concrete and easily assigned to a business process (AND/OR business goal decomposition). Based on the refined business goal specification, the business analyst can assign a group of measurable operational level goals to be realized by a business process. The selected set of goals serves as a first draft of a high level process, the business process pattern. In the next step, each of the activities in the business process pattern is refined with a subprocess which realizes a given goal. After each of the activities is refined, the business process model is completed. Each business process model can be assigned to a corresponding business process pattern. By explicitly linking the business goal model to business process models, we are able to perform advanced analysis on process models, such as detection of conflicts and redundancies in processes. Furthermore, this is very helpful when e.g. we want to know which processes contribute to a particular business goal, or which processes need to be changed because a certain goal has been modified. Moreover, we can detect if a process does not have an assigned business goal (gap analysis).

3.5 Guided Naming and Modeling

Current business process modeling tools support neither restricting names nor using ontologies to describe process artifacts. This lack results in creating non-consistent process models which are difficult to understand, compare, evaluate and re-use, etc. In [13], we have shown how the Business Functions Ontology, developed within D1.6 [14], may be used for consistent naming of process activities and suggesting appropriate resources in process modeling, which leads to more readable, comparable, and consistent process models. For more details on the approach, we refer the reader to the corresponding publication in the Appendix.
4 Auto-completion for Business Process Modeling

Auto-completion mechanism is supporting modeling of processes. After a user modelled a part of the process, he may launch the auto-completion functionality asking for process tasks or fragments that complete what he started to model.

4.1 Auto-completion Based on Naming and Functional Annotations of Process Tasks

In this section we will describe the principles of the auto-completion mechanism based on naming and functional annotations of process tasks and discuss its implementation within the BPMO editor.

4.1.1 Scenario and User Interface

Consider the business user who models a process. He wants to see if a system could suggest him a task that would follow the one he has just added to the diagram (Figure 5). Because in the modeling phase there might be many open branches of process flow the user must explicitly select the task he wants to be auto-completed. Then he calls auto-completer to do its job. As an additional parameter (to the selected task and the whole process itself) he selects one of a few strategies he wants the auto-completer to use (the strategies are described in the next section). After auto-completer responds with a list of matching process fragments (Figure 6) the user selects the most appropriate fragment. The chosen fragment replaces the selected task and possibly some more preceding ones, see description of strategies (Figure 7).

Figure 5. An initial state of the modeling environment for the Auto-completer
Figure 6. A list of suggested fragments

Figure 7. The result of a single auto-complete action
4.1.2 Auto-completion strategies

Auto-completion strategy defines what part of an already modelled process is used as a query for finding potentially matching fragments and how the matching is performed.

4.1.2.1 Beginning Task Match

This is the simplest auto-completion strategy. The user selects a task he/she wants to be auto-completed. The auto-completer retrieves all tasks whose activities and a name match the selected task (i.e. the similarity is above specific threshold) and ranks fragments they belong to according to their similarity to the user-provided task. After the user selects one of the suggested tasks, the task he initially selected is replaced with the chosen fragment. Effectively the selected task is auto-completed with a remaining part of the matched fragment. Example:

before: StartEvent-Task0-Gateway0-Task1-Task2

auto-completer problem: find all the fragments that begins with the task similar to Task2

suggested fragment: Task2a-Task3

after accepting suggestion: StartEvent-Task0-Gateway0-Task1-Task2a-Task3

4.1.2.2 Beginning Sequence Match

The user selects a task he/she wants to be auto-completed. Auto-completer finds the longest sequence of tasks that ends with the task selected by the user (i.e. it “moves back” until a gateway or process beginning is reached). This pattern sequence is used to find matching fragments, that start with the sequence of task matching user-provided sequence of tasks. After the user decides to use one of the suggested fragments, the initially found pattern sequence of task will be replaced with the chosen fragment. Effectively the task selected by the user is auto-completed with a remaining part of the matched fragment. Example:

before: StartEvent-Task0-Gateway0-Task1-Task2

auto-completer problem: find all the fragments that begins with the sequence similar to Task1-Task2

suggested fragment: Task1a-Task2a-Task3-Task4

after accepting suggestion: StartEvent-Task0-Gateway0-Task1a-Task2a-Task3-Task4

4.1.2.3 Sequence Match

The strategy is a more general version of the “Beginning Sequence Match”. In this strategy the query is constructed in the same way as in previous strategy. The difference is that the auto-completer selects all fragments that contain sequence of tasks matching sequence of tasks provided by user at any location within them. As the result the task selected by the user is followed by a remaining part of the matched fragment and is preceded by the initial part of the selected fragment. Example:
before: StartEvent-Task0-Gateway0-Task1-Task2

auto-completer problem: find all the fragments that contains the sequence similar to Task1-Task2

suggested fragment: Task5-Task1a-Task2a-Task4-Task3

after accepting suggestion: StartEvent-Task0-Gateway0-Task5-Task1a-Task2a-Task4-Task3

4.1.2.4 Set Match
This strategy generalises the “Sequence Match” strategy in a sense, that auto-completer does not pose a sequence restriction on potentially matching fragments. It returns all fragments that contain tasks matching all user-provided tasks, regardless of their positions, order and existence of additional tasks between them.

Example:

before: StartEvent-Task0-Gateway0-Task1-Task2

auto-completer problem: find all the fragments that contain tasks similar to Task1 and Task2

suggested fragment: Task5-Task2a-Task3-Task1a-Task3

after accepting suggestion: StartEvent-Task0-Gateway0-Task5-Task2a-Task3-Task1a-Task3

4.1.2.5 Auto-complete Name
This additional functionality provides a user with suggestions of the full name of a task whose name is only partially typed.

4.1.3 External interface

4.1.3.1 Implemented interfaces

```java
public interface Autocompleter {
    Response getFragment(Process process, Task lastTask);
    ArrayList<String> getSimilarNames(String name);
    void runIndexer();
}
```

```java
public class Response {
    public ArrayList<WorkflowElement> elementsToRemove;
    public WorkflowElement elementToBeReplaced;
    public ArrayList<ProcessFragment> suggestedFragments;
}
```
4.1.3.2 Interactions with other components

![Diagram of data flow within the Auto-completer and between other components]

**Figure 8. Data flow within the Auto-completer and between other components**

4.1.4 Components

4.1.4.1 Fragmenter

Fragmenter is responsible for identifying SESE fragments within processes stored in the BPL. Identified fragments are stored in the BPL and pushed to the Indexer.

4.1.4.2 Indexer

The role of the Indexer is to retrieve all relevant information regarding process fragments that is required during the matching phase. The retrieved information is stored within an internal data structure (index).

4.1.4.3 FragmentMatcher

FragmentMatcher is the main subcomponent of the auto-completer. It is responsible for implementing various auto-completion strategies. Each of matching strategies is implemented as a plugin using the same index structure and implementing the same interface. Thus, a number of additional matching strategies may be implemented.

4.1.5 Matching procedure

Matching consists of calculating distance value on four levels between following objects:

1. task's attribute vs. task's attribute
2. task vs. task
3. set or sequence of tasks vs. other set or sequence of tasks
4. one or more tasks (depends on strategy) vs. process fragment
4.1.5.1 Attributes level

Each level requires different procedure for calculating distance value. At the level of task’s characteristics we decided to consider two attributes: hasName and hasBusinessFunction.

Distance between task names is calculated using one of the standard string distance measures ([5]).

Distance between business functions assigned to a two given tasks is calculated using the following formula:

\[ d(i, j) = \frac{CPL(i, j)}{H(i) + H(j)} \]

where \( d(i, j) \) denotes the distance value, \( CPL(i, j) \) is the length of the common path in the Business Function Ontology subsumption hierarchy (starting from the root) of the two concepts \( i \) and \( j \); \( H(i) \) is the length of the path in subsumption hierarchy from the root to the concept \( i \).

4.1.5.2 Tasks level

Distance between tasks is calculated as the average of the attribute level distances.

4.1.5.3 Sequence/set level

The similarity of two sequences is calculated as the arithmetic sum of similarity between respective pairs of tasks.

The similarity of two sets is calculated as the maximum of similarity of user-provided set and all possible subsets of the same size.

4.1.5.4 Fragment level

The similarity of a user query to a specific process fragment is calculated as the maximum of similarities between set or sequence provided as user query and all possible sets or sequences belonging to specific fragment. For example in case of Beginning Task Match a single task is compared against first task of each fragment, while in case of Set Match set of \( n \) tasks is matched against each \( n \)-element subset of tasks of each fragment and the maximum of obtained similarity measures is used as query-to-fragment similarity.

4.1.6 Index structure

Two structures are used to index process fragments and utilised in answering user queries. First and the simplest structure consists of a list of all tasks present (with IRIs of corresponding process fragments), identified by an IRI. The second structure contains all \( n \)-grams (including 1-grams) of tasks, specifying also if a specific sequence corresponds to the beginning of specific fragment.

The index is used basically to select candidate sequences or tasks that may be similar to a user query. In case of Beginning Task Match and Beginning Sequence Match strategies \( n \)-grams (\( n \) being the number of tasks in user query; \( n=1 \) in case of Beginning Task Match) that correspond to a start of any fragment are selected. In case of Sequence Match all \( n \)-grams are selected (whatever their position in
the task is). Finally, in case of the Set Match, first the search space is limited to all fragments that contain at least n elements (i.e. that contain a sequence of at least n elements); next all subsets of n elements of all these fragments are considered for similarity measurement.

The index is implemented as an SQL database (HSQLDB Java library is used). Each of match strategies is responsible for construction of an SQL query performing selection of IRIs of process fragments. The similarity measure for two tasks in accessible directly in a query (as a stored procedure, implemented in Java). The results of such query are handled by the FragmentMatcher module.

4.2 Non-Functional Properties within Auto-completion – Filtering and Ranking of Tasks using NFP Analyzer

In this section, we will describe how non-functional properties of process artefacts could be used to filter and rank the recommended process tasks in order to increase the level of precision of the suggestions within auto-completion.

4.2.1 Introduction

Within the auto-completion mechanism it is important to take into account not only the functional aspect of an artefact, but also other relevant aspects, including business requirements. The business requirements may be applied to various process artefacts and may be defined as anything that imposes constraints or expresses the preferences of the modeller towards some values of non-functional properties (NFP) of process artefacts. In turn, non-functional properties (NFP) of an artefact may be defined as anything that exhibits a constraint over the functionality offered by the artefact in question [6]. From the SUPER point of view, the artefacts that may have the NFP assigned are processes, process fragments, tasks and finally Semantic Web services that are combined (composed) in order to provide the required functionality.

NFP, being distinctive criteria for the success of many businesses allowing differentiation between processes/services offering similar functionality, may be effectively used within the auto-completion process to perform filtering or create ranking of candidate tasks. Following [10] this step may be performed along with the process-context and functional-based auto-completion provided a matching list of services or tasks. Within this step, the level of match between the desired non-functional properties against those offered by services/tasks obtained through pre-filtering in the previous two steps should be computed.

Within this document, we shortly present the NFPs that may be used within auto-completion of modelled business processes in a SUPER context. Then we delve into the issue of expressing preferences and constraints as well as how they may be propagated to other layers. Finally, we show how the level of match may be computed and how the synthetic indicator may be used in order to create a ranking of candidate artefacts.
4.2.2 NFP and different perspectives

In general, NFPs may be classified in two categories qualitative (like e.g. payment method, authentication) and quantitative ones (they may be expressed as some measurable value.). For some of the quantitative characteristics the desired value is as low as possible (e.g. latency time), for others, on the contrary, as high as possible (e.g. reliability). Keeping this in mind, three kinds of variables may be distinguished: larger-the-better (LTB), smaller-the-better (STB) and nominal-the-best (NTB). Knowing the character of NFP properties is indispensable when one tries to compare two artefacts taking into account multiple properties as is shown later on in the document.

Within SUPER, NFPs are assigned only to SWS and in fact, no values of NFPs are assigned to tasks as the BPMO ontology does not allow us to do that (what a business analyst can do is to express his preferences and apply constraints on the level of process and/or task as is shown later on in this document). Therefore, for the needs to auto-completion, if we want to take into account also NFPs, we have to consider them on the level of SWS and aggregate the values of NFPs of constituting the tasks services in order to compute the values of NFP characterizing a given task. The aggregation methods may be found in [8]. Briefly speaking, while performing computation we need to take into account the control structure of the composed task (all workflow patterns) and aggregate in an appropriate way (e.g. sum up taking into account critical path or multiply) values of NFPs of constituting services.

Therefore, as we focus on services, a table below presents the properties that are relevant to service level. Their characteristic includes definition, the possible values and units as well as character of the NFP concept. For the exact definition of the NFP properties on all levels (i.e. also for task and process level), we refer a reader to 3.5 deliverable [8].

<table>
<thead>
<tr>
<th>Name of the property</th>
<th>Definition</th>
<th>Possible values and units</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service execution cost</td>
<td>the total cost of a service (e.g. price of a service), an amount of money that needs to be paid to the service provider for the service execution</td>
<td>It is usually stated in monetary unit (e.g. 5 euros),</td>
<td>Quantitative, STB</td>
</tr>
<tr>
<td>Service Response Latency (Service Response Time)</td>
<td>a roundtrip time between sending a request and receiving a response. (Ran, 2003) defines service response time as the guaranteed max (or average or min) required to complete a service request. It has two components: Service Latency Time and Service Execution Time.</td>
<td>Value: real, Unit: time units depending on the measurement resolution</td>
<td>Quantitative, STB</td>
</tr>
<tr>
<td>Service Latency</td>
<td>the time needed for the packet of control data to get to the provider’s server (where the service is executed) and then return to the provider</td>
<td>Value: integer, Unit: time units depending on the measurement</td>
<td>Quantitative, STB</td>
</tr>
<tr>
<td>Name of the property</td>
<td>Definition</td>
<td>Possible values and units</td>
<td>Character</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>---------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>requester. In general, the service latency is defined as the difference of time between the instant indicating the beginning of the transmission request and the real beginning of the action generated by this one (Dridi et al., 2007)</td>
<td>resolution (e.g. milliseconds, seconds, minutes).</td>
<td>Quantitative, STB</td>
<td></td>
</tr>
<tr>
<td>Service Execution Time</td>
<td>the time needed for the service to process the service request completely</td>
<td>Value: real, Unit: time units depending on the measurement resolution</td>
<td>Quantitative, STB</td>
</tr>
<tr>
<td>Service throughput</td>
<td>the number of completed service requests over a time period [Ran2003].</td>
<td>Number of possible invocation in some time interval (e.g. 20 invocations per second)</td>
<td>Quantitative, LTB</td>
</tr>
<tr>
<td>Service Availability</td>
<td>says whether the service is present and ready for immediate use. It is computed considering Mean Time before Failure and Mean Time to Repair</td>
<td>Percentage (e.g. 80%)</td>
<td>Quantitative, LTB</td>
</tr>
<tr>
<td>MTTF</td>
<td>Mean Time before Failure</td>
<td>Value: Real, Unit: time unit (e.g. 20 days)</td>
<td>Quantitative, LTB</td>
</tr>
<tr>
<td>MTTR</td>
<td>Mean Time to Repair</td>
<td>Value : Real Unit: time units. (e.g. 5 hours)</td>
<td>Quantitative, STB</td>
</tr>
<tr>
<td>Service execution reliability</td>
<td>a measure on how often the result of a service invocation is correct [Jäger 2007]. It represents the ability of a Web service to perform its required functions under stated conditions for a specified time interval. It is objective – it is defined as the ratio between the number of service executions divided by all executions in a given time period</td>
<td>Percentage (e.g. service executes successfully in 95% of cases)</td>
<td>Quantitative, LTB</td>
</tr>
<tr>
<td>Service owner</td>
<td>the entity offering the given artefact (i.e. resource, service, task or process). It is relevant to all the NFP layers</td>
<td>String (or a concept from an ontology) denoting name</td>
<td>Qualitative, NTB</td>
</tr>
<tr>
<td>Name of the property</td>
<td>Definition</td>
<td>Possible values and units</td>
<td>Character</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>---------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Data confidentiality</td>
<td>it states whether data confidentiality is ensured. Data should be treated properly so that only authorized users (or other services) can access or modify the data</td>
<td>Boolean value states whether or not the data is confident</td>
<td>Qualitative, NTB</td>
</tr>
<tr>
<td>Encryption</td>
<td>states whether data is encrypted or not. It may also inform about which encryption algorithm has been applied</td>
<td>String value - States what kind of encryption is supported.</td>
<td>Qualitative, NTB</td>
</tr>
<tr>
<td>Authentication</td>
<td>users who can access process, service and data should be authenticated</td>
<td>String value - States what kind of authentication mechanism, if any, is supported.</td>
<td>Qualitative, NTB</td>
</tr>
<tr>
<td>Authorization</td>
<td>users (or other services) should be authorized so that they only can access the protected services</td>
<td>String value - States what kind of authorization mechanism, if any, is supported.</td>
<td>Qualitative, NTB</td>
</tr>
<tr>
<td>Non-repudiation</td>
<td>ensures that the origin of a message cannot deny having sent the message (Zhou and Haas, 1999)</td>
<td>String value - States whether non-repudiation is supported</td>
<td>Qualitative, NTB</td>
</tr>
</tbody>
</table>

Table 1. Description of NFPs relevant to the service level

The mapping from a lower to a higher level – the bottom-up approach – allows for counting/aggregating the relevant values describing the whole task or process. However, a matter of utmost importance is to remember that any business analyst is, in general, concerned mainly in the process level. Business analyst operates on the process level and defines constraints and lists preferences. An analyst is able to particularize whether his demands are directed towards specific process fragments, tasks or the whole process itself. Rarely, he expresses directly his requirements towards a SWS to be used. Therefore, within auto-completion the ability to say what are the requirements of a business user towards the tasks used within a process given that the requirements on the level of process are known is of utmost important (the mapping from the higher level to a lower one – the top-down approach - is useful in the auto-completion process to ‘translate’ the business analyst’s requirements and constraints specified for processes or tasks into service requirements and constraints).
4.2.3 Level of match

The following techniques based on [9] can be used as a basis for machine-readable representation of non-functional aspects as well as the matchmaking and ranking mechanism. Its main objective is to create a ranking of relevant artefacts according to the values of their NFP properties and user preferences (mentioned in the previous section). Some proposals to compute utility function over the given parameters can be found, it is also possible to compute a distance measure, but it does not take preferences into account. That is why it was decided to take advantage of a multiple criteria analysis (MCA). The artefacts can be compared according to their characteristics, e.g. cost, response time, accessibility. To each characteristic a weight is assigned, reflecting preferences assigned by a business analyst to a process.

As already mentioned in [8] non-functional properties may be divided into the following groups, namely into qualitative and quantitative variables, as well as larger-the-better (LTB), smaller-the-better (STB) and nominal-the-best (NTB).

The method used to compare phenomena is computation of the synthetic indicator. For example, if two artefacts are given, together with some statistics concerning their characteristics e.g. response time, reliability etc., it may occur that one of the artefacts performs better according to reliability, while the other is more accurate and less expensive. Additionally, one artefact is paid by credit card and another by wire transfer. Synthetic indicator allows for comparison of such artefacts, given the vector of user preferences (if no preferences are expressed by a business user, we may use the default vector of preferences). The first step of computing the indicator is unification of the quantitative parameters - all STB variables are transformed into LTB. The transformation is very intuitive - given the value $x$ of the STB variable, the LTB is obtained by taking its inverse. If each of the artefacts characteristics is associated a letter $x$ with an appropriate index, e.g. cost of the first artefact: $x_{11}$, cost of the second artefact: $x_{12}$, response time of the first artefact: $x_{21}$, accessibility of the second artefact: $x_{32}$, etc., the formula is as follows:

$$a_{ij} = \frac{1}{x_{ij} + a}$$

$a > 0$, an extremely small number guaranteeing that the denominator is not equal to 0,

$i = 1,\ldots,n$ – the ID of the characteristic,

$j = \{1,2\}$, the indexes of compared services.

Afterwards, the normalisation procedure is applied to ensure that all variables are expressed in the same units and are additive. There are many ways of normalisation, e.g. division by the highest value of parameter that assure that all values are higher than 0 and less or equal 1.

$$x'_{ij} = \frac{x_{ij}}{\arg\max \{x_{11}, x_{12}\} + a}$$
At this step both artefacts have their characteristics normalised and unified.

Another step is to assign binary values to qualitative variables. 1 is assigned to those which values are in accordance with user preference, 0 in other situation.

If all parameters are equally important for the user, the value of the synthetic indicator \( (si) \) is counted as a mean of the parameters. If a user expresses his/her preferences as weights, the weighted mean is counted.

\[
si_j = \frac{1}{m} \sum_{i=1}^{m} w_i y^*_i
\]

The weights may be expressed as some fractions which sum up to one, or may just assign values from 1 to \( m \) (\( m \) meaning the number of parameters taken into account) to the most and less important parameters. In the latter case they must be normalised to obtain the weights in the range \(<0;1>\) and summing to 1.

### 4.2.4 Implementation

The implementation of the NFP Analyzer applies the algorithms discussed above and addresses all of the block-oriented workflow patterns, such as deferred-choice-merge, exclusive-choice-merge, multiple-choice-merge, parallel-split-synchronize, repeat, sequence, and while patterns, and some of the graph-oriented patterns, i.e. exclusive choice and simple merge, multiple choice and multiple merge, multiple choice and simple merge, and parallel split and synchronization patterns. For the prototype of the implementation we have taken four quantitative constraints, i.e. cost, duration, availability and throughput, into consideration. The NFP analyzer provides the following methods for computing the aggregated non-functional properties and for ranking process or process fragments based on the non-functional properties.

The method “aggregateConstraints” provides the functionality to compute the aggregated value of the constraints defined for a BPMO process or a BPMP process fragment. The method “computeProcessIndex” is designed for computing the aggregated constraints for each process fragment in the collection of process fragments that defined in a BPMO process. The aggregated constraints collectively can be considered as an index of the parent BPMO process. To support the NFP-based auto-completion, the NFP analyzer provides three methods for ranking of BPMO processes, BPMO process fragments and both of them. The constraintWeightModel contains information on the user-specified weights of each of the constraints, i.e. cost, duration, availability, and throughput.
Figure 9 Design of NFP Analyzer
4.2.5 Summary
By taking into account the quality of service properties when proposing the follow up activities, we increase the level of precision of the suggestions and achieve a more refined ranking according to user preferences.
5 Conclusions

This deliverable continues the work in [15] to provide further support to business experts within the business process modeling phase. We first define a semantic business process modeling methodology, which incorporates the usage of the SUPER ontology stack [7, 14] and reasoning technology in the early phases of process design [11]. The methodology is further supported by several techniques, which are described as individual subtasks. Within the subtasks, this deliverable first builds on and advances the efforts reported in D3.3 [15] on process fragment and subprocess detection and business process querying. We further extend the modeling support with additional techniques which flourish the work on business process ontologies presented in [14, 7], namely: auto-completion, goal-driven process modeling and guided naming and modeling. This set of extensions has been pursued in largely independent subtasks. The contributions of this deliverable are the auto-completion technique described in the body of the text, and the seven scientific publications listed in the Appendix. The application of the techniques presented here improves reuse of business knowledge within process modeling, supports the business expert during modeling and leads to more readable, consistent and higher quality process models.
6 References


7 Appendix: List of Publications

7.1 The Refined Process Structure Tree

This paper has been published at the 6th international conference on Business Process Management (BPM-08) [2], where it received the best paper award.

7.2 Automatic Workflow Graph Refactoring and Completion

This paper has been published in the proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC-2008) [3].

7.3 A Framework for Querying in Business Process Modeling

This paper was published in the proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI-2008) [4].

7.4 Auto-completion for Executable Business Process Models

A shorter version of this paper was published in the proceedings of the Workshop on Advances in Semantics for Web services (semantics4ws-2008) [10].

7.5 Methodological Extensions for Semantic Business Process Modeling

This paper was published in the proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS-2008) [11].

7.6 Business Functions Ontology and its Application in Semantic Business Process Modelling

This paper was published in the proceedings of the 19th Australasian Conference on Information Systems, (ACIS-2008) [13].

7.7 Linking Business Goals to Process Models in Semantic Business Process Modeling

A shorter version of this paper has been published in the proceedings of the 12th IEEE Enterprise Computing Conference, 2008 (EDOC-08) [12].
Abstract. We consider workflow graphs as a model for the control flow of a business process model and study the problem of workflow graph parsing, i.e., finding the structure of a workflow graph. More precisely, we want to find a decomposition of a workflow graph into a hierarchy of sub-workflows that are subgraphs with a single entry and a single exit of control. Such a decomposition is the crucial step, for example, to translate a process modeled in a graph-based language such as BPMN into a process modeled in a block-based language such as BPEL. For this and other applications, it is desirable that the decomposition be unique, modular and as fine as possible, where modular means that a local change of the workflow graph can only cause a local change of the decomposition. In this paper, we provide a decomposition that is unique, modular and finer than in previous work. It is based on and extends similar work for sequential programs by Tarjan and Valdes [11]. We show that our decomposition can be computed in linear time based on an algorithm by Hopcroft and Tarjan [3] that finds the triconnected components of a biconnected graph.

Keywords. Workflow graph parsing, Control flow, Model decomposition, BPMN to BPEL translation/roundtripping, Subprocess detection, Graph theory

1 Introduction

The control flow of a business process can often be modeled as a workflow graph [10]. Workflow graphs capture the core of many business process languages such as UML activity diagrams, BPMN and EPCs. We study the problem of parsing a workflow graph, that is, decomposing the workflow graph into a hierarchy of sub-workflows that have a single entry and a single exit of control, often also called blocks, and labeling these blocks with a syntactical category they belong to. Such categories are sequence, if, repeat-until, etc., see Fig. 1(a). Such a decomposition is also called a parse of the workflow graph. It can also be shown as a parse tree, see Fig. 1(c).

The parsing problem occurs when we want to translate a graph-based process description (e.g. a BPMN diagram) into a block-based process description (e.g. BPEL process), but there are also other use cases for workflow graph parsing. For example, Vanhatalo, Völzer and Leymann [14] show how parsing speeds up control-flow analysis. Küster et al. [6] show how differences between two process models can be detected and resolved based on decompositions of these process models. We believe that parsing also helps in understanding large processes and in finding reusable subprocesses.

For a roundtripping between a BPMN diagram and a BPEL process, it is desirable that the decomposition be unique, i.e., the same BPMN diagram always translates to
the same BPEL process. Consider, for example, the workflow graph in Fig. 1(a). The translation algorithm proposed by Ouyang et al. [9] is nondeterministic. It may produce one of the two parses shown in Fig. 1(a) and (b), depending on whether the if-block or the repeat-until-block is found first by the parsing algorithm.

One idea to resolve some of this nondeterminism is to define priorities on the syntactic categories to be found [9, 7, 8]. For example, if in each step the parsing algorithm tries to find sequences first, then if-blocks and then repeat-until-blocks, we can only obtain the parse in Fig. 1(a) in our example. However, this can introduce another problem. If we change a single block, say, the repeat-until block by replacing it, e.g. by a single task, we obtain the workflow graph shown in Fig. 1(d). Fig. 1(d) also shows the parse we obtain with the particular priorities mentioned above. The corresponding parse tree is shown in Fig. 1(c). It cannot be derived from the tree in Fig. 1(c) by just a local change, viz., by replacing the Repeat-Until subtree. For a roundtripping between a BPMN diagram and a BPEL process, it would be much more desirable that a local change in the BPMN diagram also result in only a local change in the BPEL process. Replacing a block in the BPMN diagram would therefore only require replacing the corresponding block in the BPEL process. We then call such a decomposition modular.

The existing approach to the BPMN to BPEL translation problem [9] is not modular. Furthermore, it does not provide, because of the above problems, a specification of the translation that is independent of the actual translation algorithm.

A unique and modular decomposition is provided by the program structure tree defined by Johnson et al. [4, 5] for sequential programs. It was applied to workflow graphs by Vanhatalo et al. [14] to find control-flow errors. The corresponding decomposition for our first example is shown in Fig. 2. It uses the same notion of a block as Ouyang et al. [9] do, that is, a block is
a connected subgraph with a single entry and a single exit edge. But in contrast to the approach of Ouyang et al. [9], non-maximal sequences are disregarded in the program structure tree. For example, Sequence 2 in Fig. 1(a) [likewise Sequence 3 in subfigure (b)] is non-maximal: it is in sequence with another block.

Another general requirement for parsing is to find as much structure as possible, i.e., to decompose into blocks that are as fine as possible. As we will see (cf. Sect. 4), this allows us to map more BPMN diagrams to BPEL in a structured way. It has also been argued [9] that the BPEL process is more readable if it contains more blocks. Furthermore, debugging is easier when an error is local to a small block rather than to a large one.

In this paper, we provide a new decomposition that is finer than the program structure tree as defined by Johnson et al. [4, 5]. It is based on and extends similar work for sequential programs by Tarjan and Valdes [11]. The underlying notion of a block is a connected subgraph with unique entry and exit nodes (as opposed to edges in the previous approach). Accordingly, all blocks of the previous approach are found, but more may be found, resulting in a more refined parse tree. We prove that our decomposition is unique and modular. Moreover, we show that it can be computed in linear time, based on an algorithm by Hopcroft and Tarjan [3] that finds the triconnected components of a biconnected graph.

The paper is structured as follows. In Sect. 2, we define the refined process structure tree and discuss its main properties. In Sect. 3, we describe how to compute the process structure tree in linear time. Proofs of the main theorems can be found in a technical report [13].

2 The Refined Process Structure Tree

In this section, we define the refined process structure tree (PST, for short). First, we explain our notion of fragments in Subsection 2.1. Fragments have a strong relationship with the triconnected components of the workflow graph, which we explain in Subsection 2.2. Subsection 2.3 defines the process structure tree. Finally, we show that our decomposition is modular.

2.1 Fragments

We start by recalling some basic notions of graph theory. A multi-graph is a graph in which two nodes may be connected by more than one edge. This can be formalized as a triple $G = (V, E, M)$, where $V$ is the set of nodes, $E$ the set of edges and $M$ a mapping that assigns each edge an ordered or unordered pair of nodes—for a directed or undirected multi-graph, respectively. We will use multi-graphs throughout the paper, directed and undirected, but will call them graphs for simplicity.

Let $G$ be a graph. If $e$ is an edge of $G$ that connects two nodes $u$ and $v$, we also say that $u$ and $v$ are incident to $e$, $e$ is incident to $u$ and $v$, and nodes $u$ and $v$ are adjacent.

Workflow graphs are based on two-terminal graphs\footnote{A workflow graph is a two-terminal graph in which each node is labeled with some control flow logic such as AND, OR, etc.}. A two-terminal graph (TTG for short) is a directed graph $G$ without self-loops such that there is a unique source
node $s$ and a unique sink node $t \neq s$ and each node $v$ is on a directed path from $s$ to $t$. The undirected version of $G$, denoted $U(G)$, is the undirected graph that results from ignoring the direction of all the edges of $G$ and adding an additional edge between the source and the sink. The additional edge is called the return edge of $U(G)$. Figure 3 shows examples of (a) a two-terminal graph $G$, and (b) its undirected version $U(G)$, where $r$ is the return edge.

For a subset $F$ of edges, let $V_F$ denote the set of nodes that are incident to some edge in $F$ and let $G_F$ denote the subgraph with nodes $V_F$ and edges $F$. We say that $G_F$ is formed by $F$.

Let $G$ be a TTG and $F$ a subset of its edges such that $G_F$ is a connected subgraph of $G$. A node $v \in V_F$ is a boundary node of $F$ if it is the source or sink node of $F$, or if $G$ has edges $e \in F$ and $e' \notin F$ such that $v$ is incident to $e$ and $e'$. A boundary node $v$ is an entry of $F$ if no incoming edge of $v$ is in $F$ or if all outgoing edges of $v$ are in $F$. A boundary node $v$ is an exit of $F$ if all incoming edges of $v$ are in $F$ or if no outgoing edge of $v$ is in $F$. $F$ is called a fragment of $G$ if it has exactly two boundary nodes, an entry and an exit. Let $\mathcal{F}(u, v)$ denote the set of all fragments with entry $u$ and exit $v$.

Figure 4 shows examples of fragments. A fragment is indicated as a dotted box. It contains all those edges that either are inside the box or cross the boundary of the box. Thus, the box in subfigure (a) denotes the fragment $F_1 = \{a, b, c\}$. Node $u$ is the entry and $v$ is the exit of $F_1$. In subfigure (b), $F_2 = \{a, b, c, d\}$ is a fragment with entry $u$ and exit $v$. In subfigure (c), $F_3 = \{a, b, c, d\}$ has two boundary nodes, $u$ and $v$, neither of them is an entry or an exit of $F_3$. Therefore, $F_3$ is not a fragment.

Note that it can be checked locally whether a boundary node is an entry or an exit. This notion of fragment was proposed by Tarjan and Valdes [11], where a TTG modeled the control flow of a sequential program. When control flows through any of the edges of a fragment, then it must have flown through the entry before and must flow through

![Fig. 3. (a) Two-terminal graph $G$, and (b) its undirected version $U(G)$, where $r$ is the return edge.](image)

![Fig. 4. (a), (b) Examples and (c) counterexamples of entry, exit and fragment.](image)
are adjacent in a pair of nodes. A graph is triconnected if the dashed edge is considered as an ordinary edge.

4 It is often assumed for workflow graphs that no node has both multiple incoming and multiple outgoing edges. In that case it follows that $U(G)$ is biconnected. See also Sect. 4.

2.2 Triconnected Components

Tarjan and Valdes [11] have shown that the fragments of a TTG are closely related to the triconnected components of its undirected version. We have to introduce a few more notions from graph theory.

Let $G$ be an undirected graph. The following notions are also used for directed graphs by ignoring the direction of their edges. Let $W$ be a subset of nodes of $G$. Two nodes $u, v \notin W$ are connected without $W$ if there is a path that contains $u$ and $v$ but no node $w \in W$. For instance, in Fig. 3(a) nodes $s$ and $t$ are connected without $v_6$, but not connected without $v_5$. Two edges $e, f$ are connected without $W$ if there exists a path containing $e$ and $f$ in which a node $w \in W$ may only occur as the first or last element. A graph without self-loops is $k$-connected, $k > 0$, if it has at least $k + 1$ nodes and for every set $W$ of $k − 1$ nodes, any two nodes $u, v \notin W$ are connected without $W$.

We say connected for 1-connected, biconnected for 2-connected and triconnected for 3-connected. A separation point (separation pair) of $G$ is a node $u$ (pair $\{u, v\}$ of nodes) such that there exists two nodes that are not connected without $\{u\}$ (without $\{u, v\}$). Therefore a graph is biconnected (triconnected) if and only if it has no separation point (separation pair). For instance in Fig. 3, $G$ is not biconnected, because $v_5$ is a separation point, whereas $U(G)$ is biconnected, because it has no separation points. $U(G)$ is not triconnected, because $\{v_5, v_7\}$ is a separation pair. In Fig. 5(a), $T_2$ is an example of a triconnected graph if the dashed edge $x$ is considered as an ordinary edge.

We say that a graph is weakly biconnected if it is biconnected or if it contains exactly two nodes and at least two edges between these two nodes. For instance, in Fig. 5(a), $B_1$ is weakly biconnected, but not biconnected.

Throughout the paper, we assume that $U(G)$ is weakly biconnected. This can easily be achieved by splitting each separation point into two nodes, where the only outgoing edge of the first node is the only incoming edge of the second node. 4

Let $\{u, v\}$ be a pair of nodes. A separation class with respect to (w.r.t.) $\{u, v\}$ is a maximal set $S$ of edges such that any pair of edges in $S$ is connected without $\{u, v\}$. A separation class or branch if it does not contain the return edge; $\{u, v\}$ is called a boundary pair if there are at least two separation classes w.r.t. $\{u, v\}$. Note that a pair $\{u, v\}$ of nodes is a boundary pair if and only if it is a separation pair or $u$ and $v$ are adjacent in $G$. For instance in Fig. 3(b), $\{v_5, v_7\}$ and $\{v_6, v_7\}$ are boundary pairs. The pair $\{v_5, v_7\}$ is also a separation pair, but $\{v_6, v_7\}$ is not.

Each weakly biconnected graph can be uniquely decomposed into a set of graphs, called its triconnected components [3], where each triconnected component is either a bond, a polygon or a triconnected graph. A bond is a graph that contains exactly two nodes and at least two edges between them. A polygon is a graph that contains at least three nodes, exactly as many edges as nodes such that there is a cycle that contains all its nodes and all its edges.
Fig. 5. (a) The triconnected components of $U(G)$ in Fig. 3, (b) the tree of the triconnected components of $U(G)$, and (c) the corresponding component subgraphs of $G$.

Part (a) in Fig. 5 shows the six triconnected components of graph $U(G)$ from Fig. 3. $P1$ and $P2$ are polygons, $B1$ and $B2$ are bonds, and $T1$ and $T2$ are triconnected graphs. Each component contains virtual edges (shown as dashed lines), which are not contained in the original graph. They contain the information on how the components are related to each other: Each virtual edge occurs in exactly two components, whereas each original edge occurs in exactly one component. For example, the virtual edge $x$ occurs in components $T1$ and $T2$. In component $T1$, $x$ represents the component $T2$, whereas $x$ represents $T1$ in $T2$. Therefore, we obtain the original graph by merging the triconnected components at the virtual edges (which removes them).

The triconnected components can be arranged in a tree, cf. Fig. 5(b), where two components are connected if they share a virtual edge. The root of the tree is the unique component that contains the return edge. Each original edge is also shown in the tree under the unique component that contains that edge. Therefore, each component $C$ determines a set $F$ of edges of the original graph, namely all the leafs of the subtree that $C$ corresponds to. For example, component $T1$ determines the set $F = \{a, b, c, d, e, f, g, h, i\}$ of edges. We call the subgraph formed by such a set $F$ of edges the component subgraph of $C$. Figure 5(c) shows the component subgraphs of $G$.

Note that the component subgraphs $B1, P1$ and $T1$ are fragments, whereas $B2, P2$ and $T2$ are not. There are also fragments that are not component subgraphs, for instance, $\{j, k, l, m\}$.

The precise definition of the triconnected components is rather lengthy and has therefore been omitted (see [12, 2, 3]). Instead we present here the exact relationship between the triconnected components and fragments we are going to exploit. The proofs
of the following two theorems can be found in [12]. First, we observe that triconnected components are closely related to boundary pairs.

**Theorem 1.** A set \( \{u, v\} \) of two nodes is a boundary pair of \( U(G) \) if and only if

1. nodes \( u \) and \( v \) are adjacent in \( U(G) \),
2. a triconnected component of \( U(G) \) contains a virtual edge between \( u \) and \( v \), or
3. a triconnected component of \( U(G) \) is a polygon and contains \( u \) and \( v \).

We show examples based on \( U(G) \) in Fig. 3(b) and its triconnected components in Fig. 5(a). For instance, the boundary pair \( \{v_6, v_7\} \) contains two adjacent nodes of \( U(G) \), the boundary pair \( \{v_1, v_2\} \) corresponds to a virtual edge \( x \) of \( T_2 \), and the boundary pair \( \{s, v_7\} \) contains two nodes of the polygon \( P_1 \). Boundary pairs are closely related to fragments as follows.

**Theorem 2.** 1. If \( F \in \mathcal{F}(u, v) \), then \( \{u, v\} \) is a boundary pair of \( U(G) \) and \( F \) is the union of one or more proper separation classes w.r.t. \( \{u, v\} \).
2. Let \( \{u, v\} \) be a boundary pair of \( U(G) \) and \( F \) the union of one or more proper separation classes w.r.t. \( \{u, v\} \). If \( u \) is an entry of \( F \) and \( v \) is an exit of \( F \), then \( F \in \mathcal{F}(u, v) \).

For instance, the boundary pair \( \{v_5, v_7\} \) has three proper separation classes \( \{m\}, P_2 = \{j, k, l\}, \) and \( \{n\} \). \( P_2 \) is not a fragment, because \( v_5 \) is neither its entry nor its exit, whereas \( \{m\} \in \mathcal{F}(v_5, v_7) \) and \( \{n\} \in \mathcal{F}(v_7, v_5) \) are fragments. The union of \( P_2 \) and \( \{m\} \) is a fragment, whereas \( P_2 \cup \{n\} \) and \( \{m\} \cup \{n\} \) are not. \( P_2 \cup \{m\} \cup \{n\} \) is a fragment.

Theorem 1 says that the boundary pairs can be obtained from the triconnected components while Thm. 2 says that the fragments can be obtained from the boundary pairs.

### 2.3 Canonical Fragments and the Process Structure Tree

Two fragments \( F_1 \) and \( F_2 \) may overlap, that is, we have \( F_1 \cap F_2 \neq \emptyset \), \( F_1 \setminus F_2 \neq \emptyset \) and \( F_2 \setminus F_1 \neq \emptyset \). Examples of overlapping fragments are shown in Fig. 6. Overlapping fragments give rise to nondeterministic parsing as explained in Sect. 1. We are therefore interested in a subset of fragments that do not overlap with each other. These will be called canonical. We comment on our particular definition of canonical fragments in Sect. 4. We start by defining various types of bond fragments.

![Fig. 6. Examples of overlapping fragments.](image-url)
Definition 1 (Bond fragments). Let $S$ be a proper separation class (i.e., a branch) w.r.t. $[u,v]$. $S$ is directed from $u$ to $v$ if it contains neither an incoming edge of $v$ nor an outgoing edge of $v$. $\mathcal{D}(u,v)$ denotes the set of directed branches from $u$ to $v$. $S$ is undirected if it is neither in $\mathcal{D}(u,v)$ nor in $\mathcal{D}(v,u)$. The set of undirected branches between $u$ and $v$ is denoted by $\mathcal{U}(u,v)$. A fragment $X \in \mathcal{F}(u,v)$ is

1. a bond fragment if it is the union of at least two branches from $\mathcal{D}(u,v) \cup \mathcal{U}(u,v) \cup \mathcal{D}(v,u)$.
2. a directed bond fragment if it is the union of at least two branches from $\mathcal{D}(u,v) \cup \mathcal{U}(u,v)$.
3. a semi-pure bond fragment if it is the union of at least two branches from $\mathcal{D}(u,v) \cup \mathcal{U}(u,v)$, and
   (a) there exists no $Y \in \mathcal{U}(u,v)$ such that $Y \subseteq X$, $Y$ has an edge incoming to $u$, or
   (b) there exists no $Y \in \mathcal{U}(u,v)$ such that $Y \subseteq X$, $Y$ has an edge outgoing from $v$.
4. a pure bond fragment if it is the union of at least two branches from $\mathcal{D}(u,v)$.

Note that the various bond-fragment types form a hierarchy, i.e., each pure bond fragment is a semi-pure bond fragment, each semi-pure bond fragment is a directed bond fragment etc. Fig. 7 shows examples of various classes of bond fragments that do not belong to a lower class. Bond fragments are closed under composition, i.e., we have:

Proposition 1. If $X, Y \in \mathcal{F}(u,v)$ and $F = X \cup Y$, then $F \in \mathcal{F}(u,v)$. If $X$ and $Y$ are bond fragments, so is $F$. If $X$ and $Y$ are directed (semi-pure) [pure] bond fragments, so is $F$.

Proposition 1 assures that a maximal bond fragment, maximal directed, maximal semi-pure, or maximal pure bond fragment is unique if it exists. We are now ready to define canonical fragments.

Definition 2 (Canonical fragment).

1. If $F_0 \in \mathcal{F}(v_0, v_1)$ and $F_1 \in \mathcal{F}(v_1, v_2)$ such that $F_0 \cup F_1 = F \in \mathcal{F}(v_0, v_2)$, we say that $F_0$ and $F_1$ are in sequence (likewise: $F_1$ and $F_0$ are in sequence) and that $F$ is a sequence. $F$ is a maximal sequence if there is no fragment $F_2$ such that $F$ and $F_2$ are in sequence.
2. A bond fragment (directed bond fragment etc.) $F \in \mathcal{F}(u,v)$ is maximal if there is no bond fragment (directed bond fragment etc.) $F' \in \mathcal{F}(u,v)$ that properly contains $F$. A bond fragment $F \in \mathcal{F}(u,v)$ is canonical if it is a maximal bond fragment, a maximal directed, maximal semi-pure, or maximal pure bond fragment such that $F$ is not properly contained in any bond fragment $F' \in \mathcal{F}(v,u)$.
3. A fragment is canonical if it is a maximal sequence, a canonical bond fragment, or neither a sequence nor a bond fragment.

Fig. 8. (a) The non-trivial canonical fragments of $G$, and (b) the process structure tree of $G$.

Note that each edge is a canonical fragment, which we call a trivial fragment. Part (a) of Fig. 8 shows the non-trivial canonical fragments of graph $G$. $S1 \in \mathcal{F}(v5,v7)$ is a maximal semi-pure bond fragment, and $B1 \in \mathcal{F}(v5,v7)$ is a maximal bond fragment. $P1$ is a maximal sequence. $T1$ is neither a sequence nor a bond fragment.

To prove that canonical fragments do not overlap, i.e., two canonical fragments are either nested or disjoint, this claim is proven first for bond fragments that have the same entry-exit pair.

**Lemma 1.** Let $X,Y \in \mathcal{F}(u,v)$ be canonical bond fragments. Then $X \subseteq Y$, $Y \subseteq X$ or $X \cap Y = \emptyset$.

We continue by showing that two canonical bond fragments that share the same boundary pair do not overlap. In general, we can encounter two situations depending on whether the union of all branches with respect to a boundary pair is a fragment. These two cases are shown in Fig. 9.

In Fig. 9(a), the union of all branches with respect to the boundary pair $\{u, v\}$ is the maximal bond fragment from $u$ to $v$ called $B$. Fragments $D,S$ and $R$ are the maximal directed bond, semi-pure bond, and pure bond fragment from $u$ to $v$ respectively. Compared with part (a) of Fig. 9, part (b) has an additional edge outgoing from $u$ that is outside of the union of all branches with respect to the boundary pair $\{u, v\}$. Because of this added edge, neither $u$ or $v$ is an entry of this subgraph. Thus, this set of edges is not a fragment. Fragment $R1$ is the maximal pure bond fragment from $u$ to $v$. Fragment $S$ is the maximal semi-pure bond fragment from $u$ to $v$. As there is no larger bond fragment from $u$ to $v$, $S$ is also the maximal directed bond fragment and the maximal bond fragment from $u$ to $v$. $R2$ is the maximal pure bond...
fragment from \( v \) to \( u \). As there is no larger bond fragment from \( v \) to \( u \), \( R2 \) is also the maximal semi-pure bond, the maximal directed bond and the maximal bond fragment from \( v \) to \( u \). Through an analysis of these two cases, we can prove the following:

**Lemma 2.** Let \( X \in \mathcal{F}(u,v) \) and \( Y \in \mathcal{F}(v,u) \) be canonical bond fragments. Then \( X \subseteq Y \), \( Y \subseteq X \) or \( X \cap Y = \emptyset \).

We are now ready to present the main theorem.

**Theorem 3.** Let \( X, Y \) be two canonical fragments. Then \( X \subseteq Y \), \( Y \subseteq X \) or \( X \cap Y = \emptyset \).

Theorem 3 allows us to define the unique process structure tree of a workflow graph.

**Definition 3 (Process structure tree).** Let \( G \) be a TTG. The process structure tree \( \text{PST} \), for short) is the tree of canonical fragments of \( G \) such that the parent of a canonical fragment \( F \) is the smallest canonical fragment of \( G \) that properly contains \( F \).

Thus, the largest fragment that contains a whole workflow graph \( G \) is the root fragment of the PST. Part (b) of Fig. 8 shows the PST of graph \( G \) in part (a). The child fragments of a sequence \( P1 \) are ordered left to right from the entry to the exit of \( P1 \). For example, the order of child fragments of maximal sequence \( P1 \) is \( T1, B1 \) and \( o \). Moreover, as \( T1 \) has the same entry as \( P1 \), the exit of \( T1 \) (\( B1 \)) is the entry of \( B1 \) (\( o \)), and \( o \) has the same exit as \( P1 \). We use this ordering in Sect. 2.4 to derive all fragments from the canonical fragments. For this, it is not necessary to order the child fragments of a bond or a triconnected graph.

### 2.4 Computing all Fragments from the Canonical Fragments

The following proposition indicates how to derive all fragments from the canonical fragments. This is useful for example if one wants to find the smallest fragment that contains some given set of graph elements.

**Proposition 2.** Let \( F \) be a set of edges in a TTG. \( F \) is fragment if and only if \( F \) is a canonical fragment or \( F \) is

1. a union of consecutive child fragments of a maximal sequence,
2. a union of child fragments of a maximal pure bond fragment, or
3. a union of child fragments of a maximal bond fragment \( B \) such that \( B \) is not a maximal directed bond fragment.

For example, the maximal sequence \( P1 \) in Fig. 8 has \( T1, B1 \) and \( o \) as ordered child fragments. Besides these canonical fragments and the maximal sequence, also the union of \( T1 \) and \( B1 \) (\( B1 \) and \( o \)) is a fragment. However, the union of \( T1 \) and \( o \) is not a fragment, as these are not consecutive child fragments, i.e., they do not share a boundary node.
Part (a) of Fig. 10 shows a maximal pure bond fragment \( R = \{a, b, c\} \). Its child fragments are \( \{a\} \), \( \{b\} \), and \( \{c\} \). It follows from Prop. 2 that \( \{a, b\} \), \( \{b, c\} \), and \( \{a, c\} \) are the non-canonical fragments in \( R \). Part (b) of Fig. 10 shows a maximal bond fragment \( B = \{a, b, c, d\} \in \mathcal{F}(u, v) \) and a maximal directed bond fragment \( \mathcal{R} = \{a, b\} \in \mathcal{F}(u, v) \) such that \( B \neq \mathcal{R} \). It follows from Prop. 2 that \( \{c, d\} \), \( \{a, b, c\} \) and \( \{a, b, d\} \) are the non-canonical fragments in \( B \). Note that \( \{a, c, d\} \) and \( \{b, c, d\} \) are not fragments, because their boundary nodes are neither entries nor exits.

### 2.5 Modularity

Finally, we state what we mean by saying that our decomposition is modular.

**Theorem 4.** Let \( G \) be a TTG and \( X \in \mathcal{F}(u, v) \) be a canonical fragment of \( G \). Let \( G' \) be the TTG obtained by replacing the subgraph that is formed by \( X \) by some other subgraph formed by a set of (fresh) edges \( X' \) such that \( X' \in \mathcal{F}(u, v) \) is again a fragment of \( G' \) (but not necessarily canonical) with the same entry-exit pair as \( X \). Assume that \( A \) is the parent fragment of \( X \) in \( G \) and \( F, X \) is a child fragment of \( A \) in \( G \). Let \( A' = (A \setminus X) \cup X' \) and \( F' = F \). Then \( A' \) and \( F' \) are canonical fragments of \( G' \) where \( F' \) is a child fragment of \( A' \) in \( G' \).

Theorem 4 means that a local change to the TTG, i.e., changing a canonical fragment \( X \), only affects the PST locally. The parent and siblings of a changed canonical fragment remain in the PST in the same place and it follows inductively that this is also true for all canonical fragments above the parent and all canonical fragments below the siblings of \( X \).

### 3 Computing the PST

In this section, we describe an algorithm that computes the PST in linear time. We have extended the algorithm by Valdes [1] to find all the canonical fragments (his algorithm produces a coarser decomposition, cf. Sect. 4). The algorithm has three high-level steps that are illustrated in Fig. 11 and described in Alg. 1. In Step 1, the tree of the triconnected components is computed, using e.g. the linear-time algorithm by Hopcroft and Tarjan [3]. Gutwenger and Mutzel [2] present some corrections to this algorithm. We illustrate the computed triconnected components through the respective component subgraphs in Fig. 11.

In Step 2, we analyze each triconnected component to determine whether the respective component subgraph is a fragment. This can be done in linear time with the following approach that takes advantage of the hierarchy of fragments. We analyze the tree of the triconnected components bottom-up—all children before a parent. For each
Algorithm 1 Computes the PST for a two-terminal graph.

\textbf{buildPST(}Graph $G$\textbf{)}

\textbf{Require: } $G$ is a weakly biconnected TTG.

// Step 1. Compute the tree of the triconnected components of the TTG.
Tree := Compute the tree of the triconnected components of $G$.

// Step 2. Analyze each component to determine whether it is a fragment.
\textbf{for each Component} $c$ in Tree \textbf{in a post-order of a depth-first traversal do}
  \textbf{Count the number of edges (that are children of $c$) incoming to/outgoing from each boundary node. (4 counts)}
  \textbf{Add these edge counts to the respective edge counts of the parent component for each shared boundary node.}
  \textbf{Compare these edge counts to the total number of incoming/outgoing edges to determine whether each boundary node is entry, exit, or neither.}
  \textbf{Based on these boundary node types, determine whether $c$ is a fragment.}
  \textbf{if} $c$ is a polygon \textbf{then}
    \textbf{Count the number of entry and exit nodes of the child components.}
    \textbf{If a child component is a fragment, order the child components from entry to exit.}
  \textbf{if} $c$ is not a fragment \textbf{and} $c$ has at least two child fragments \textbf{then}
    \textbf{Create a maximal sequence (that contains a proper subset of children of $c$).}
  \textbf{if} $c$ is a bond \textbf{then}
    \textbf{Classify each branch of $c$ based on the edge counts of the boundary nodes of the respective child components of $c$.
    \textbf{if} $c$ is a fragment \textbf{then}
      \textbf{Based on the classifications of the branches, create the maximal pure, the maximal semi-pure, and the maximal directed bond fragment, if any exists.}
    \textbf{else}
      \textbf{Based on the classifications of the branches, create the maximal pure bond fragments, the maximal semi-pure bond fragment, if any exists.}
  \textbf{for each Component} $d$ that has been created in this iteration \textbf{do}
    \textbf{Merge the child fragments of each component in the to-be-merged-list of $d$ to $d$.}
  \textbf{if} $c$ is not a fragment \textbf{then}
    \textbf{Add} $c$ \textbf{to the to-be-merged list (a linked list) of its parent component.}
    \textbf{Concatenate the to-be-merged list of $c$ to the to-be-merged list of its parent.}
  \textbf{else}
    \textbf{Merge the child fragments of each component in the to-be-merged list of $c$ to $c$.}
\textbf{return} Tree
Step 1: Detect the triconnected components.
Step 2: Analyze each triconnected component to determine whether the respective component subgraph is a fragment. Step 3: Create the missing canonical fragments and merge the triconnected components that are not fragments.

Fig. 11. The high-level steps of Alg. 1. Step 1: Detect the triconnected components. Step 2: Analyze each triconnected component to determine whether the respective component subgraph is a fragment. Step 3: Create the missing canonical fragments and merge the triconnected components that are not fragments.

child edge of a triconnected component $c$, we check whether it is incoming to or outgoing from one of the two boundary nodes of $c$. We count these edges to determine whether a boundary node is an entry, an exit, or neither. Based on this information, we can determine whether the respective component subgraph is a fragment. Note that when a triconnected component shares a boundary node with its parent, the same edges do not have to be counted twice, because an edge inside a child is also inside its parent.

In Step 3, we create the missing canonical fragments, and merge each component subgraph that is not a fragment to the smallest canonical fragment that contains it. This restructuring is based on the information computed in Step 2. New fragments are created only in those cases where a bond or a polygon contains canonical fragments that are not component subgraphs. Such a fragment is created as a union of at least two (but not all) children of this bond or polygon. We show examples in the following.

We process the tree of the triconnected components bottom-up as in Step 2. Thus, in Fig. 11, we can begin with $T_2$. It contains no new canonical fragments, because it is neither a sequence nor a bond. $T_2$ is not a fragment, because $v_1$ is neither its entry nor its exit. Thus, it will be merged into its parent fragment $T_1$, that is, the children of $T_2$ become children of $T_1$.

The bond $B_2$ is not a fragment, so it will be merged. $B_2$ contains no new canonical fragments, because it has only two children. The same applies to $P_2$. More interestingly, $B_1$ is a fragment and has three children. Each child of a bond is a branch, and we classify them to find out whether they form new canonical bond fragments. $\{m\}$ is a directed branch from $v_5$ to $v_7$, $P_2$ is an undirected branch that has no outgoing edges from $v_7$, and $\{n\}$ is a directed branch from $v_7$ to $v_5$. Note that the branches can be classified based
Fig. 12. Step 3: From the tree of the component subgraphs to the tree of the canonical fragments.

on the counts of edges incident to each boundary node of a branch computed in Step 2. There is a new semi-pure bond fragment \( S1 = \{m\} \cup P2. \) \( B2 \) and \( P2 \) are merged to \( S1. \) \( S1 \) and \( \{n\} \) become the children of the restructured \( B1. \) Finally, \( P1 \) and all its children are fragments, thus there is no need to restructure \( P1. \)

In the following examples we show polygons and bonds in which more restructuring is required. In Fig. 12(a), \( B1 \) and \( P1 \) are not fragments. However, polygon \( P1 \) has two consecutive child fragments \( \{d\} \) and \( \{e\} \) that form a maximal sequence \( P = \{d\} \cup \{e\}. \) To determine whether a polygon contains such a new maximal sequence, we compute the number of entries and exits of its children already at the end of Step 2. A polygon that is not a fragment contains a maximal sequence as a union of its children if and only if its children have at least three entries or at least three exits in total.

In Fig. 12(b), \( B1, P1, B2, \) and \( P2 \) are not fragments and will be merged. Bond \( B \) is a fragment from \( v1 \) to \( v4 \) and has six branches: two edges as directed branches from \( v1 \) to \( v4, \) and one undirected branch, \( P2, \) that has no edge incoming to the entry of \( B, \) one undirected branch, \( P1, \) that has both an edge incoming to the entry of \( B \) and an edge outgoing from the exit of \( B, \) and another two edges as directed branches from \( v4 \) to \( v1. \) The directed branches from the entry to the exit of \( B \) form a new maximal pure bond fragment \( R. \) The union of \( P2 \) and \( R \) is a new maximal semi-pure bond fragment \( S. \) The union of \( P1 \) and \( S \) is a new maximal directed bond fragment. \( D \) and the remaining two directed branches are the children of \( B. \) \( B1 \) and \( P1 \) are merged to \( D, \) and \( B2 \) and \( P2 \) to \( S. \) \( P \) is a maximal sequence.

Figure 12(c) shows an example of a bond \( B \) that is not a fragment, but its children form new canonical fragments. As there are at least two directed branches to each direction, these branches form two new pure bond fragments, \( R1 \) and \( R2. \) The union of \( R1 \) and branch \( P2 \) is a semi-pure bond fragment \( S. \) Thus, \( B2 \) and \( P2 \) are merged to \( S. \) The polygon \( P \) has four children \( \{a\}, \) \( B3, \) \( B, \) and \( \{b\}. \) \( B3 \) and \( B \) not fragments, but the union of these consecutive siblings is a fragment. Thus, \( B \) is merged to \( B3 \) to form a new fragment \( M. \) \( B1 \) and \( P1 \) are also merged to \( M. \) The fragment \( P \) has only three children.
Each step of the algorithm can be performed in linear time. Thus, also the entire algorithm has linear time complexity.

**Theorem 5.** The PST of a TTG $G$ can be computed in time linear in the number of edges of $G$.

### 4 Conclusion

We have presented a modular technique of workflow graph parsing to obtain fine-grained fragments with a single entry and single exit node. The result of the parsing process, the process structure tree, is obtained in linear time. We have mentioned a couple of use cases in Sect. 1. Coarser-grained decompositions may be desirable for some use cases. Those can easily be derived from the refined process structure tree by flattening. One such coarser decomposition, which can be derived and which is also modular, is the decomposition into fragments with a single entry edge and a single exit edge presented by Vanhatalo, Völzer and Leymann [14]. The new, refined decomposition presented here allows us to translate more BPMN diagrams to BPEL in a structured way. As an example, consider the workflow graph in Fig. 13 and (a) its decomposition with the existing techniques [9, 14] and (b) with our new technique. In Fig. 13(a), $X$ cannot be represented as a single BPEL block, whereas in Fig. 13(b) each fragment can be represented as a single BPEL block.

The main idea of the technique presented is taken from Tarjan and Valdes [11, 1]. They describe an algorithm that produces a unique parse tree. However, they do not provide a specification of the parse tree, i.e., a definition of canonical fragments or claim or prove modularity. Moreover, our PST is more refined than their parse tree. Figure 12 shows examples of workflow graphs where this is the case. The fragments that are not identified by them are $P$ in (a), $D$. $S$ and $R$ in (b), and $S$, $R_1$ and $R_2$ in (c).

We have made some simplifying assumptions about workflow graphs. The assumption that we have unique source and sink nodes can be lifted. Also the assumptions that the undirected version of the workflow graph is weakly biconnected and does not contain self-loops can be lifted. The necessary constructions to deal with these cases will be presented in an extended version of this paper. Thus the remaining assumption on workflow graphs will be that each node is on a path from some source to some sink.

![Workflow Graph Diagram](image-url)

Fig. 13. A workflow graph and (a) decomposition presented in [9, 14] and (b) our decomposition.
The reader might wonder what justifies our particular definition of canonical fragments. It can be shown that the canonical fragments are exactly those fragments that do not overlap with any (canonical or non-canonical) fragment. This means, they are exactly the ‘objective’ fragments in the sense that they are compatible with any parse and hence appear in every maximal parse. Any finer decomposition into fragments can only be obtained by arbitrating between overlapping fragments. Our definition is further justified by Prop. 2, i.e., by the fact that all fragments and hence all parses can be derived from the PST in a simple way.

Acknowledgments The work published in this article was partially supported by the SUPER project (http://www.ip-super.org/) under the EU 6th Framework Programme Information Society Technologies Objective (contract no. FP6-026850).

References
Automatic Workflow Graph Refactoring and Completion

Jussi Vanhatalo\(^1, 2\), Hagen Völzer\(^1\), Frank Leymann\(^2\), and Simon Moser\(^3\)

\(^1\) IBM Zurich Research Laboratory, Switzerland, \{juv, hvo\}@zurich.ibm.com
\(^2\) Institute of Architecture of Application Systems, University of Stuttgart, Germany, frank.leymann@iaas.uni-stuttgart.de
\(^3\) IBM Böblingen Software Laboratory, Germany, smoser@de.ibm.com

Abstract. Workflow graphs are used to model the control flow of business processes in various languages, e.g., BPMN, EPCs and UML activity diagrams. We present techniques for automatic workflow graph refactoring and completion. These techniques enable various use cases in modeling and runtime optimization. For example they allow us to complete a partial workflow graph, they provide local termination detection for workflow graphs with multiple ends, and they allow us to execute models containing OR-joins faster. Some of our techniques are based on workflow graph parsing and the Refined Process Structure Tree [10].

1 Introduction

A \textit{workflow graph} shows the control flow of a business process similar to a flow chart as a directed graph. Figure 1(a) shows an example. Workflow graphs form the core of many specification languages, e.g., BPMN, EPCs and UML activity diagrams.

Different workflow graphs can model the same behavior, i.e., the same control flow. For example, the two workflow graphs in Figs. 1(a) and 1(b) model the same behavior. The workflow graph in Fig. 1(b) is \textit{well-structured} in the sense that it consists of matching pairs of a node that splits the flow and a node that joins the flow. Well-structured workflow graphs are often preferred because they are easier to comprehend [9, 2] and analyze [11], and can be represented as a regular expression.

The workflow graph \(G\) can be transformed into the well-structured workflow graph \(G^*\) using local transformation rules that preserve the execution semantics. While those rules are known [7, 8], it is not clear how to apply them to obtain \(G^*\) from \(G\) automatically. This is because a transformation rule can be applied to different parts of the

![Fig. 1. (a) A workflow graph \(G\), (b) A well-structured workflow graph \(G^*\).](image-url)
workflow graph, which can lead to different refactoring results. For example, applying
the same rule that transforms \( G \) into \( G^* \), we can transform \( G \) also into \( G' \) in Fig. 2. The
transformation rule is given in Sect. 3.2. We are not aware of any work that specifies a
desired refactoring result and proposes a concrete algorithm that computes it.

In this paper, we propose such a definition and such an algorithm. Given a work-
flow graph \( G \), the algorithm computes a normal form \( G^* \) of \( G \) that makes the struc-
ture of \( G \) more explicit—and, as a side-
product, making it more well-structured.

Our approach is based on the normal [11] and the refined process structure trees [10]
of a workflow graph.

It is important that a workflow graph can be represented in an easily comprehensible
form, as this makes it better accessible for users that may not be experienced in business
process modeling. After all, workflow graphs are used for communicating business pro-
cesses among different stakeholders, and not only among modeling experts. We hope
that a workflow graph is, in general, easier to comprehend in our normal form than in
its original form, as it is more well-structured.

![Fig. 2. Alternative transformation of \( G \).](image)

![Fig. 3. (a) A workflow graph with multiple end nodes. (b) Its completion.](image)

In the second part of the paper, we show how this refactoring technique can be used
to compute a completion of a workflow graph. Figure 3 shows (a) a workflow graph
with multiple end nodes and (b) its completion, which by definition has a unique end
node. This has multiple use cases, which we discuss in Sect. 4. For example, it can
be used to complete a partial workflow graph at modeling time [4], to provide local
termination detection for workflow graphs with multiple ends, and to execute models
containing OR-joins faster.

The refactoring technique efficiently computes a completion for many workflow
graphs, but not for all. Sometimes a completion does not exist. We characterize these
cases and also provide an algorithm that computes a completion in the general case
when it exists. This algorithm is less efficient than the refactoring-based completion.

We review preliminary notions in Sect. 2. Then, we present our contributions: the
refactoring technique in Sect. 3 and the completion technique in Sect. 4. Section 5
concludes this paper. We include short proofs of some theorems in this paper, whereas
longer proofs of the other theorems can be found in a technical report [12].
2 Preliminaries

This section defines the basic preliminary notions of this paper, which include workflow graphs and their semantics, their extension by inclusive OR-gateways, and the soundness property for workflow graphs.

2.1 Workflow Graphs

A workflow graph $G$ consists of a directed graph $(V, E)$, consisting of a set $V$ of nodes, a set $E \subseteq V \times V$ of edges, and a partial mapping $\ell : V \rightarrow \{\text{AND, XOR}\}$ such that

1. $\ell(x)$ is defined if and only if $x$ has more than one incoming edge or more than one outgoing edge,
2. there is exactly one source and at least one sink,
3. the source has exactly one outgoing edge and each sink has exactly one incoming edge, and
4. every node is on a path from the source to some sink.

The source is also called the start node, a sink is called an end node, $\ell(x)$ is called the logic of $x$. If the logic is AND or XOR, we call $x$ a gateway; if $x$ has no logic, then we call $x$ a task. We use BPMN to depict workflow graphs, i.e., gateways are drawn as diamonds, where the symbol “+” inside stands for AND, whereas no decoration stands for XOR. Tasks are drawn as rectangles or circles. In particular, here start and end nodes are considered special tasks, which are always drawn as circles. A gateway that has more than one incoming edge and only one outgoing edge is also called a join, a gateway with more than one outgoing but only one incoming edge is also called a split. We say that an edge $e$ is incident to a node $n$ if $e$ is incoming to $n$ or outgoing from $n$.

The semantics of a workflow graph is, similarly to Petri nets, defined as a token game. A state of a workflow graph is represented by tokens on the edges of the graph. Let $G = (V, E, \ell)$ be a workflow graph. A state of $G$ is a mapping $s : E \rightarrow \mathbb{N}$, which assigns a natural number to each edge. When $s(e) = k$, we say that edge $e$ carries $k$ tokens in state $s$. The semantics of the various nodes is defined as usual. An AND-gateway removes one token from each of its ingoing edges and adds one token to each of its outgoing edges. An XOR-gateway nondeterministically chooses one of its incoming edges on which there is at least one token, removes one token from that edge, then nondeterministically chooses one of its outgoing edges, and adds one token to that outgoing edge. As usual, we abstract from the data that controls the flow in XOR-gateways, hence the nondeterministic choice.

To be more precise, let $s$ and $s'$ be two states and $x$ a node that is neither a start nor an end node. We write $s \xrightarrow{x} s'$ when $s$ changes to $s'$ by executing $x$. We have $s \xrightarrow{x} s'$ if

1. $\ell(x) = \text{AND}$ or the logic of $x$ is undefined, and
   
   $s'(e) = \begin{cases} 
   s(e) - 1 & \text{if } e \text{ is an incoming edge of } x, \\
   s(e) + 1 & \text{if } e \text{ is an outgoing edge of } x, \\
   s(e) & \text{otherwise.}
   \end{cases}$
2. \( \ell(x) = XOR \), and there exists an incoming edge \( e' \) and an outgoing edge \( e'' \) of \( x \) such that

\[
s'(e) = \begin{cases} 
    s(e) - 1 & e = e', \\
    s(e) + 1 & e = e'', \\
    s(e) & \text{otherwise.}
\end{cases}
\]

The initial state of \( G \) is the state where there is exactly one token on the unique outgoing edge of the start node and no token anywhere else. Node \( x \) is said to be activated in a state \( s \) if there exists a state \( s' \) such that \( s \xrightarrow{x} s' \). A state \( s' \) is reachable from a state \( s \), denoted \( s \xrightarrow{*} s' \), if there exists a (possibly empty) finite sequence \( s_0 \xrightarrow{x} s_1 \ldots s_{k-1} \xrightarrow{x} s_k \) such that \( s_0 = s \) and \( s_k = s' \). A state is a reachable state of \( G \) if it is reachable from the initial state of \( G \).

### 2.2 Inclusive OR-Gateways

A generalized workflow graph is a workflow graph in which a gateway \( x \) may also have OR-logic (inclusive OR), i.e., \( \ell(x) = OR \). OR-gateways are drawn as diamonds with a circle inside. An OR-gateway has non-local join behavior, which is difficult to define if there is a cycle in the graph that contains an OR-join. As the semantics for the OR-join is not settled in that case, we do not consider that case. So, in the following we assume that a generalized workflow graph does not contain a cycle that contains an OR-gateway that has more than one incoming edge.

The OR-gateway behaves as follows. It is activated if for each incoming edge \( e' \) of the gateway that carries no token, and for each edge \( e'' \) of the graph that carries a token, there is no directed path from \( e'' \) to \( e' \). When it executes, it consumes a token from each incoming edge that carries a token and produces a token for each edge of a nonempty subset of its outgoing edges. That subset is chosen nondeterministically.

More precisely, we also have \( s \xrightarrow{x} s' \) if

\[
s'(e) = \begin{cases} 
    s(e) - 1 & e \text{ is an incoming edge of } x \text{ such that } s(e) \geq 1, \\
    s(e) + 1 & e \in F, \\
    s(e) & \text{otherwise.}
\end{cases}
\]

### 2.3 The Soundness Property

We now define what we understand by a “well-behaved” (generalized) workflow graph. A final state is a reachable state \( s \) of \( G \) that has no successor state, i.e., \( s \) activates no node. A final state is a deadlock if it contains a token on some edge that is not an incoming edge of an end node. (It follows that it must then be an incoming edge of a join.) A reachable state \( s \) contains a lack of synchronization if there is an edge \( e \) such
that \( s(e) > 1 \). A (generalized) workflow graph is sound if it contains neither a deadlock nor a lack of synchronization.

A deadlock is clearly undesired. Absence of lack of synchronization is a valuable design principle as it rules out that a task is executed concurrently to itself ("uncontrolled auto-concurrency"). Multiple concurrent instances of tasks can be modeled explicitly with dedicated constructs (e.g. "multiple instance activities" in BPEL and BPMN). Soundness is necessary for translating a (generalized) workflow graph to BPEL in a structured way, but has also an independent motivation: For workflow graphs with a unique end node and without OR-gateways, it is equivalent with the traditional definition of soundness (cf. [9, 11]).

### 3 Automatic Refactoring of Workflow Graphs

In this section, we present our technique that automatically refactors a workflow graph into a normal form that makes the structure of the workflow graph explicit. By structure we mean the decomposition of the workflow graph into logically atomic parts, which we call fragments. We use two alternative ways to do this decomposition, the normal [11] and the refined process structure tree [10]. We review the definitions of these process structure trees in Sect. 3.1. In Sect. 3.2, we present our novel refactoring technique.

#### 3.1 The Refined and the Normal Process Structure Trees

Let \( G = (V, E, \ell) \) be a (generalized) workflow graph. We assume that \( G \) has a unique end node. A detailed discussion on how to extend a workflow graph with multiple end nodes to a workflow graph with a unique end node is given in Sect. 4.

![Figure 4](image)

**Fig. 4.** (a) The canonical R-fragments of a workflow graph \( G \). (b) The refined process structure tree of \( G \). (c) \( F \) is not an R-fragment of \( G \).

Figure 4(a) shows a workflow graph \( G \) and its decomposition into canonical R-fragments. An R-fragment is a connected subgraph with a unique entry and a unique exit node. The canonical R-fragments form a hierarchy, which can be represented as a tree. This tree, called the refined process structure tree (RPST), is shown in Fig. 4(b). These notions can be formally be defined as follows.

For a set \( F \subseteq E \) of edges, let \( V_F \) denote the set of nodes that are incident to some edge in \( F \) and let \( G_F = (V_F, F) \). We say \( G_F \) is formed by \( F \).
Let $F \subseteq E$ be a set of edges such that $G_F$ is a connected subgraph of $G$. A node $v \in V_F$ is a boundary node of $F$ if it is the source or sink node of $G$, or if there exist edges $e, e' \in E \setminus F$ such that $v$ is incident to $e$ and $e'$. A boundary node $v$ is an entry of $F$ if no incoming edge of $v$ is in $F$ or if all outgoing edges of $v$ are in $F$. A boundary node $v$ is an exit of $F$ if all incoming edges of $v$ are in $F$ or if no outgoing edge of $v$ is in $F$. $F$ is called an R-fragment of $G$ if it has exactly two boundary nodes, an entry and an exit.

Figure 4(a) shows examples of R-fragments, which are indicated dotted boxes. They contain all those edges that are either inside the box or cross the boundary of the box. Thus, the box $D$ denotes the R-fragment $D = \{(u, a1), (a1, v), (u, a2), (a2, v)\}$. Node $u$ is the entry and $v$ is the exit of $D$. $E = \{(v, a4), (a4, w), (w, v)\}$ is an R-fragment with entry $v$ and exit $w$. In Fig. 4(c), $F = \{(u, a2), (a2, u), (v, u)\}$ has two boundary nodes, $u$ and $v$, but neither of them is an entry or an exit of $F$. Thus, $F$ is not an R-fragment.

An R-fragment $F$ is canonical if it does not overlap with any other R-fragment, that is, for each R-fragment $F'$ of $G$, we have $F \subseteq F'$ or $F' \subseteq F$ or $F \cap F' = \emptyset$. It follows that canonical R-fragments do not overlap with each other and hence form a hierarchy, which is represented as the refined process structure tree (RPST) of $G$. Note that each leaf node of the RPST represents an edge of the workflow graph, as each edge forms an R-fragment. The boundary nodes of this R-fragment are the two nodes that this edge connects. Vanhatalo, Völzer and Koehler [10] show how the RPST can be computed in linear time.

![Diagram](image)

**Fig. 5.** (a) The canonical N-fragments of a workflow graph $G$. (b) The NPST of $G$.

Figure 5(a) shows an alternative decomposition of the same workflow graph into N-fragments. An N-fragment has unique entry and exit edges (as opposed to nodes). The corresponding tree, shown in Fig. 5(b), is called the normal process structure tree (NPST).

Let $G = (V, E, \ell)$ be a workflow graph. An N-fragment $G_F = (V', E')$ is a nonempty connected subgraph of $G$ such that there exist edges $e, e' \in E$ with $E \cap ((V \setminus V') \times V') = \{e\}$ and $E \cap (V' \times (V \setminus V')) = \{e'\}$; $e$ and $e'$ are called the entry and the exit edge of $G_F$, respectively. An N-fragment is canonical if it does not overlap with any other N-fragment. The tree representing the hierarchy of canonical N-fragments is called the normal process structure tree (NPST). It can also be computed in linear time [5, 1, 11].

We define special kinds of R-fragments and N-fragments as follows. An R-fragment $F$ is trivial if $F$ has exactly one edge. The union of two R-fragments $F, F'$ is an R-
sequence if the exit node \( u \) of \( F \) is the entry node of \( F' \), and each edge incident to \( u \) is in \( F \cup F' \). An N-fragment \( F \) is trivial if the entry and the exit edge of \( F \) are incident to the same node of \( F \). An N-fragment \( F \) is an N-sequence if \( F \) is the union of two N-fragments \( F', F'' \) such that the exit edge of \( F' \) is the entry edge of \( F'' \). An R-fragment (N-fragment) is proper if it is neither trivial nor an R-sequence (N-sequence). A node \( u \) is a child of a canonical N-fragment \( F \) if \( F \) is the smallest canonical N-fragment that contains \( u \).

While the RPST exhibits more structure than the NPST, the NPST shows the structure more explicitly, in a less dense, more readable, form. The NPST produces a decomposition of the edges and nodes, defining a home fragment for each node and each edge—whereas the RPST produces a decomposition of edges only, while nodes may be shared between adjacent fragments. In the following, we show how to compute the normal form of a workflow graph \( G \), which has the best of both worlds: It has all the structure of the RPST of \( G \), but shows it explicitly in the more readable form of the NPST. The normal form is also more well-structured than the original workflow graph.

### 3.2 Refactoring Based on the RPST

Let \( G \) be a (generalized) workflow graph with a unique end node. We want to transform \( G \), maintaining its structure given by its RPST but showing it more explicitly in form of N-fragments. Some R-fragments can be considered as N-fragments. We call them normal:

**Definition 1 (Normal R-fragment, normal (generalized) workflow graph).** A proper canonical R-fragment \( F \) is normal if exactly one edge outside \( F \) is incident to the entry of \( F \) and exactly one edge outside \( F \) is incident to the exit of \( F \). A (generalized) workflow graph \( G \) is normal if every proper canonical R-fragment \( F \) is normal.

Normal R-fragments can be obtained by splitting nodes. Figures 6(a)-(c) show three examples of a valid expansion \((G_0, G_1)\) that splits a node \( u \) into nodes \( v, w \). (d) An invalid expansion.

![Fig. 6.](image)
shown in subfigure (d) is invalid as the simultaneous separation of inputs and outputs does not preserve execution semantics. (The original path from $b$ to $c$ gets lost.) The three valid cases are instances of a single rule that splits a node into two nodes $v, w$:

**Definition 2 (Valid expansion).** Let $G_i = (V_i, E_i, \ell_i), i = 0, 1$ be two (generalized) workflow graphs. The pair $(G_0, G_1)$ is a valid expansion if there exists two nodes $v, w \in V_i$ and a surjective mapping $\phi : V_1 \to V_0$ such that

- $(v, w) \in E_1$ and $(w, v) \notin E_1$,
- $(v, x) \in E_1$ and $(y, w) \in E_1 \Rightarrow x = w$ or $y = v$,
- $\phi(x) = \phi(y) \Rightarrow x = y$ or $[x, y] = \{v, w\}$,
- $(\phi(x), \phi(y)) \in E_0 \Rightarrow (x, y) \in E_1$ and $(x, y) \neq (v, w),$
- $\ell_0(\phi(x)) = \ell_1(x)$, or $\ell_1(x)$ is undefined.

We define two parameters for a valid expansion: 1. the node $u = \phi(v) = \phi(w)$ and 2. a partition of the edges $E_u$ that are incident to $u$ into two sets $E_v = E_v \cup E_w$. We define

$$E_v = \{(u, \phi(y)) \mid (v, y) \in E_1, y \neq w\} \cup \{(\phi(x), u) \mid (x, v) \in E_1, x \neq w\}$$

(1)

$E_w$ can be defined similarly or as $E_w = E_v \setminus E_v$.

Sadiq and Orlowska [8] have shown for workflow graphs that if $(G_0, G_1)$ is a valid expansion that splits a node $u$ into nodes $v$ and $w$, and the logic of $u$ is AND or XOR, then $G_0$ and $G_1$ have the same behavior. An analogous result is also known from Petri nets [7]. The behavior is also the same if the logic of $u$ is OR and $u$ is not in a cycle.

Some valid expansions create redundant nodes that we want to exclude from our refactoring technique. We call these undesired expansions. A desired expansion splits a gateway into two gateways, whereas an undesired expansion creates at least one task. Figure 7 shows two examples of undesired expansions. Each of these valid expansions is undesired, because node $w$ has only one incoming and only one outgoing edge.

**Definition 3 (Desired expansion, undesired expansion).** Let $G_i = (V_i, E_i, \ell_i), i = 0, 1$ be two (generalized) workflow graphs and $v, w \in V_i$ be distinct nodes such that $(G_0, G_1)$ is a valid expansion and $\phi(v) = \phi(w)$. The pair $(G_0, G_1)$ is a desired expansion if

- $v$ has at least two incoming edges or at least two outgoing edges, and
- $w$ has at least two incoming edges or at least two outgoing edges.

Otherwise $(G_0, G_1)$ is an undesired expansion.
The valid expansions in Fig. 6 are also desired expansions. Desired expansions restrict the application of node splitting. However, application of the desired expansion can still lead to different results, as shown by the example in Sect. 1. As we want to maintain the structure of the graph, we apply the expansion only based on the R-fragments. This will lead to a unique result.

Definition 4 (F-based expansion of node u). Let \((G_0, G_1)\) be a desired expansion with parameters \(u, E_v\) and \(E_w\) as in Def. 2 and let \(F\) be an R-fragment of \(G_0\). Furthermore, let \(E_u\) denote the set of edges that are incident to \(u\). We say that the expansion is \(F\)-based if either \(u\) is the entry of \(F\) and \(E_w = F \cap E_u\), or \(u\) is the exit of \(F\) and \(E_v = F \cap E_u\).

The fragment-based expansions can be applied repeatedly until we obtain a normal generalized workflow graph.

Definition 5 (Expansion). Let \(G_0\) and \(G_n\) be (generalized) workflow graphs. \(G_n\) is an expansion of \(G_0\) if there is a sequence \(G_0, \ldots, G_n\) of (generalized) workflow graphs such that \((G_i, G_{i+1})\) is a fragment-based expansion, when \(0 \leq i < n\).

Although a given workflow graph \(G\) allows different sequences of fragment-based expansions, we nevertheless obtain a unique result, which we call the expanded normal form of \(G\), denoted as \(G^*\).

Theorem 1. For every (generalized) workflow graph \(G\), there exists a unique (generalized) workflow graph \(G^*\) such that \(G^*\) is an expansion of \(G\) and \(G^*\) is normal.

Proof. Each \(F\)-based expansion replaces a fragment that is not normal by a sequence of a trivial fragment and \(F\). It is a local change to the original generalized workflow graph \(G\) that preserves the structure of the RPST. This feature, which was named modularity of the RPST, was proved earlier [10]. In fact, if we do not consider trivial fragments and sequences, the RPST stays the same after the expansion. Because expansions can be considered as modular replacements that do not change the RPST up to trivial fragments and sequences, and because fragments are either nested or disjoint, then all fragment-based expansions that can be applied to the original graph \(G\) are mutually independent, i.e. they can be applied in any order to obtain the same result. (Note however, that a particular expansion can be based on different fragments.)

Furthermore, an \(F\)-based expansion can only be applied if either the entry or the exit of \(F\) violates the normality constraint and if \(F\) is neither trivial nor a sequence. An \(F\)-based expansion removes such a violation of a normality constraint and cannot introduce a new one. It follows that we arrive at a unique normal generalized workflow graph after any sequence of all the expansions that are possible in the original graph \(G\).

It is clear from the proof of Thm. 1 that the normal form \(G^*\) arises as the maximal expansion of \(G\). Therefore, Alg. 1 computes \(G^*\). It can be implemented in linear time to the number of edges of \(G\).

Theorem 2. Let \(G\) be a (generalized) workflow graph. Algorithm 1 computes the expanded normal form of \(G\).
Algorithm 1 Computes the expanded normal form of a generalized workflow graph $G$.

```plaintext
computeExpandedNormalForm(G)

Compute the RPST $T$ of $G$, and a list $L$ of proper canonical R-fragments that are not normal.

while $L$ has a proper canonical R-fragment $F$ such that $F$ is not normal do
    Transform $G$ into $G'$ based on an $F$-based expansion $(G, G')$.
    Locally update $T$ to correspond to the RPST of this updated $G$.
    if $F$ is normal, then remove $F$ from $L$.

return $G$
```

Next we show that the NPST and the RPST coincide if $G$ is normal. As R-fragments and N-fragments are different objects, there are subtle differences in the trivial fragments. However, the main structure that is represented by the proper fragments coincides:

**Theorem 3.** Let $G$ be a normal (generalized) workflow graph, $F$ a set of edges and $G_F$ the subgraph formed by $F$. $F$ is a proper canonical R-fragment of $G$ if and only if $G_F$ is a proper canonical N-fragment of $G$.

**Proof.** Presented in a technical report [12].

Figure 8 shows a workflow graph $G$ and its proper canonical R-fragments. The R-fragment $H$ is normal, whereas $D$ and $E$ are not. Thus, we transform $G$ into its extended normal form $G^*$ shown in Fig. 8(b) through $D$-based and $E$-based expansions. It is possible to transform $G$ with $D$-based expansion of $u$ and $v$, and an $E$-based expansion of $v$. The $D$-based and $E$-based expansions of $v$ split $v$ into nodes $v_1$ and $v_2$ the same way. Thus, we can obtain $G^*$ from $G$ through two $D$-based expansions, or through one $E$-based and one $D$-based expansion. The proper canonical R-fragments of $G^*$ are also the proper canonical N-fragments of $G^*$. $G^*$ has two more proper canonical N-fragments ($D^*$ and $E^*$) than $G$.

An N-fragment $F$ is well-structured (c.f. [6, 11]) if $F$ is trivial, an N-sequence, or if $F$ has exactly two gateways as children, a split and a join $j$, that have the same logic and if the entry edge of $F$ is incoming to $j$, then $j$ is an XOR-join. Note that a well-structured fragment cannot be further refactored. However, if a fragment is not well-structured it may become well-structured as shown in the examples of Figs. 1 and 8. In
this sense, the normal form of $G$ is “more well-structured” than $G$. Note that we only use local refactorings. It is known [6] that some workflow graphs can only be transformed into well-structured graphs through non-local transformations, whereas there are also workflow graphs that have no well-structured equivalent.

4 Automatic Completion of Workflow Graphs

In this section, we show how the refactoring technique of Sect. 3 can be used to compute a completion of a workflow graph.

**Definition 6 (Completion).** Let $G = (V, E, \ell)$ be a sound workflow graph (sound generalized workflow graph) and $G' = (V', E', \ell')$ a workflow graph (generalized workflow graph). $G'$ is called a completion of $G$ if

1. $G$ is a subgraph of $G'$ such that if an edge $e \in E' \setminus E$ is incident to some node in $x \in V$, then $x$ is an end node of $G$ and $e$ is outgoing from $x$,
2. $G'$ has a unique end node,
3. $G'$ is sound.

A completion of a generalized workflow graph is easy to construct. We just add an OR-join $j$ and an end node $e$, and connect each original end node to $j$ and $j$ to $e$. Figure 9 illustrates this.

**Proposition 1.** The construction shown in Fig. 9 defines a completion of a generalized workflow graph.

**Proof.** We have to prove soundness of $G'$. We claim that the final OR-join can fire only if $G$ has no more tokens. Suppose the contrary. Then there is a state $s$ that activates the OR-join and there is some token inside $G$, say on edge $e$. As $G$ has no deadlock, we can move the token from $e$ to some end node of $G$. As $G$ has no lack of synchronization, this end node was unmarked in $s$. It follows that in state $s$, there is a path from some token to an unmarked incoming edge of the OR-join. It follows that the OR-join is not activated in $s$, contradicting our supposition. The claim of the proposition follows now directly.

This simple completion can be used to compute a translation from generalized workflow graphs (e.g. BPMN diagrams) with multiple end nodes to BPEL using the refined process structure tree [10], which needs a generalized workflow graph with a unique start and a unique end node as input. However, there are various use cases where we want a completion without using OR-gateways:

1 In most languages, multiple start nodes stand for either an implicit AND-split or an implicit XOR-split. Therefore, we restrict to workflow graphs having exactly one start node.
A user modeling a process has drawn a part of a diagram where she opened a number of parallel and alternative paths, i.e., she used a combination of AND-splits and XOR-splits, possibly also some AND-joins or XOR-joins. Now the user does not know what logic, i.e., combination of AND-joins and XOR-joins, to use to close a given set of paths correctly. In that situation, an OR-join could also be used to close these paths. However, the OR-join may not be available in the language chosen or may be considered too expensive to execute (see next use case). Gschwind et al. [4] present this use case described above, but do not provide a technical solution.

An OR-join was used to close a particular set of paths. In our restricted setting, the evaluation of the OR-join at runtime requires that the whole graph preceding the OR-join be checked for tokens. If we compute a completion of the graph preceding the OR-join, we can replace the OR-join with a combination of merges and joins, all of which can be executed locally. Removing OR-joins also allows us to translate the generalized workflow graph to a Petri net in a simple way [9]. (Petri net transitions have a local semantics.) A translation to Petri nets may not only be useful as an intermediate language between two different business process specification languages, but also to apply some of the various analysis techniques and tools available for Petri nets.

Dead Path Elimination of BPEL [3] is a way to simulate the OR-join by gateways that can be executed locally. There, dead tokens are sent along those paths that are not taken. An OR-join can then wait for a token, dead or alive, on each incoming edge before it executes. If we can replace OR-joins by merges and joins, dead path elimination can be switched off, saving the overhead of sending, propagating and synchronizing dead tokens along potentially long paths.

Checking whether a process has terminated also requires checking the entire generalized workflow graph for tokens. A sound generalized workflow graph with a unique end node will, however, signal termination through a single token on the unique end node. Thus, a completion at compile time provides a local termination detection at runtime.

In the general case, a completion may be difficult to compute or not even exist. In the following section, we show how to efficiently compute a completion based on the refactoring technique from Sect. 3 for many cases. In Sect. 4.2, we characterize for which workflow graphs a completion exists. Finally, in Sect. 4.3, we discuss how to compute a completion in the general case where it exists.

4.1 Completion by Refactoring

Let $G$ be a sound generalized workflow graph with multiple end nodes. We first complete $G$ using the simple OR-join completion described above. This adds a unique end node, which allows us to compute the RPST for the completed graph. We obtain one or more proper canonical R-fragments that contain the added OR-join of $G$ as exit. Such an R-fragment is called an end fragment of $G$. Figures 10(a) & (b) show an example of a workflow graph $G_0$ and its simple completion $G_1$ that has two end fragments $A$ and $B$.

We then split the final OR-join into several OR-joins, one per end fragment according to the refactoring technique presented in Sect. 3.2. This preserves behavior as stated
in Sect. 3.2 and therefore also soundness. We refactor the end fragments until they are normal, and thus also N-fragments. Figure 10(c) shows an example of the resulting expansion $G_2$ of $G_1$. Now we can replace an OR-join $j$ by an XOR-join if the fragment $F$ of $j$ is sequential, that is, if $F$ has no AND-split and no OR-split as a child. On the other hand, an OR-join $j$ can be replaced by an AND-join if $F$ of $j$ is deterministic, that is, $F$ has no XOR-split and no OR-split as a child. This replacement preserves soundness.

**Theorem 4.** Let $F$ be an N-fragment of a sound generalized workflow graph $G$, and an OR-join $j$ be a child of $F$.

1. If $F$ has neither an XOR-split nor an OR-split as children and $j$ is replaced with an AND-join, then the resulting generalized workflow graph $G'$ is sound.
2. If $F$ has neither an AND-split nor an OR-split as children and $j$ is replaced with an XOR-join then, the resulting generalized workflow graph $G'$ is sound.

**Proof.** Presented in a technical report [12].

Applying this theorem repeatedly allows us to replace all OR-joins in such a fragment $F$ with either 1) AND-joins or 2) XOR-joins. Figure 10(d) shows an example of a completion $G_3$ of $G_0$ obtained with our technique. Note that this completion technique requires only linear time. After a review of more than 150 sound workflow graphs created from realistic business processes, we believe that in practice most workflow graphs can be completed using this fast technique. The power of this technique stems from the fact that it abstracts from the interior of subfragments. For example, fragment $B$ in Fig. 10(c) is sequential, although it contains a subfragment with concurrency.

Figure 11(a) shows a workflow graph that can be completed, but not with the fast technique presented here. We discuss these cases in the next section. It is clear that this technique can also replace OR-joins in the middle of a generalized workflow graph, provided that we restrict its application to settings where an OR-join is not in a cycle.
4.2 Existence of Completions

In this section, we present a technique that computes a completion for each workflow graph that has a completion, and we characterize for which workflow graphs a completion exists. Let $G$ be a sound workflow graph. As $G$ is sound, we can represent a final state as a set of end nodes. Let $\mathcal{F}$ be the set of final states of $G$, presented in this way.

**Definition 7.** Two distinct end nodes $x, y$ are mutually exclusive if there is no final state $M \in \mathcal{F}$ such that $x \in M$ and $y \in M$. A test is a set $T$ of pairwise mutually exclusive end nodes such that for each final state $M$, $M \cap T \neq \emptyset$.

Figure 11(a) shows a workflow graph with final states $\mathcal{F} = \{\{e_1, e_3\}, \{e_2, e_3\}\}$. There are two tests, $T_1 = \{e_1, e_2\}$ and $T_2 = \{e_3\}$. Figure 11(b) shows a workflow graph with final states $\mathcal{F} = \{\{e_1, e_2\}, \{e_2\}\}$. There is only one test $T_3 = \{e_2\}$.

**Theorem 5.** A sound workflow graph has a completion if and only if each end node belongs to a test.

**Proof.** Presented in a technical report [12].

This completion can be constructed as shown in Fig. 12. We extend a workflow graph $G$ by creating an XOR-join for each test of $G$ and one final AND-join that is connected to the unique end node. Each end node $x$ is connected to those XOR-joins that correspond to a test $T$ such that $x \in T$. If an end node has to be connected to more than one XOR-join, an AND-split is inserted after the end node. All XOR-joins are connected to the final AND-join.

Figure 13(a) shows the completion of $G_0$ from Fig. 11. Note that gateways of completion that would have only a single incoming and a single outgoing edge can be omitted. Node $y$ corresponds to the test $\{e_1, e_2\}$ whereas the node created by the test $\{e_3\}$ was omitted. Also no AND-split was necessary.

$G_1$ in Fig. 11(b) does not have a completion because $e_1$ is not in any test. Note that it is impossible to obtain a completion if there exist two final states $M$ and $M'$ such that $M \subseteq M'$. Figure 13(b) shows that it is nevertheless possible to “complete” $G_1$ in a more liberal sense; AND-split $w$ added in the middle of $G_1$ allows us to complete $G_1$. 

**Fig. 11.** Two workflow graphs. (a) $G_0$ has a completion and (b) $G_1$ has no completion.

**Fig. 12.** General completion.
Fig. 13. (a) Completion of \( G_0 \) from Fig. 11(a). (b) Completion of \( G_1 \) from Fig. 11(b) after adding AND-split \( w \) in the middle of the graph.

### 4.3 Computation of the Completion in the General Case

The completion technique above requires computing the final states of the workflow graph, of which there can be exponentially many. The computation can be done by a simple state-space exploration. However, using the refactoring in Sect. 3.2 first, allows us to restrict the completion to single end fragments, that is, we also have to compute the state space only for individual end fragments, which are typically much smaller. We transformed our library of more than 150 industrial process models into workflow graphs having, on average, 57 edges. The largest has 203 edges and more than 100,000 states. However, those fragments that are neither sequential nor deterministic have at most 38 states.

Once the final states have been computed, a suitable set of tests can be computed. Sometimes a simple set of tests exists that can be computed in a simple way: An end node \( x \) is called an identifier of a final state \( M \) if \( x \) is contained in \( M \) but not in any other final state. Assume every state \( M \) has an identifier \( x_M \). Let \( y \) be an end node. Then \( T_y = \{ x_M | M \in \mathcal{F}, y \notin M \} \cup \{ y \} \) is a test containing \( y \). So, in that special case, which is easy to check, we get a simple set of tests that suffices.

### 5 Conclusion

This paper presents two new techniques for generalized workflow graphs—an RPST-based refactoring technique that renders their structure more explicit, and a completion technique that builds on this refactoring technique. We also provide a characterization of workflow graphs that have a completion.

Only few related papers on workflow graph refactoring and completion exist and these papers, having different focus, are only loosely related. Sadiq and Orlowska [8] transform a workflow graph preserving its behavior to analyze its soundness. Eder et al. [2] define equivalence of workflow graphs through rewriting rules. Zhang and D’Hollander [13] use hammocks to structure flow graphs of sequential programs, which can all be made well-structured in the absence of concurrency. A hammock is a special case of an R-fragment, but their technique to restructure hammocks differs from ours. Well-structuredness makes workflow graphs more readable [9, 2]. Gschwind et al. [4] identify a use case for a completion technique, but do not provide a solution to solve it.

We believe that our refactoring-based completion technique is also useful for removing OR-joins that occur in cycles. Figure 14 shows an example. Note that the
refactoring produces a well-structured workflow graph. A formal treatment of these examples would, however, exceed the scope of this paper as it requires a semantics for OR-joins in cycles. In EPCs, our refactoring-based technique can also provide a completion in the beginning of a graph, which describes the start node combinations that lead into a sound execution.

Acknowledgments The work published in this article was partially supported by the SUPER project (http://www.ip-super.org/) under the EU 6th Framework Programme Information Society Technologies Objective (contract no. FP6-026850).

References
A Framework for Querying in Business Process Modelling

Ivan Markovic¹, Alessandro Costa Pereira¹, Nenad Stojanovic²

¹SAP Research, Karlsruhe, Germany
ivan.markovic@sap.com
alessandro.costa.pereira@sap.com

²FZI Karlsruhe, Germany
nenad.stojanovic@fzi.de

Abstract: In order to respond quickly to changing market requirements, a business organisation needs to increase the level of agility in all phases of the business process engineering chain. Business process (BP) modelling is the first and most important phase in this chain. Designing a new and redesigning an existing process model is a highly complex, time consuming and error prone task. In this work, we contribute to BP modelling by designing and implementing a framework for querying in business process modelling which i) supports decision making, ii) facilitates reuse of modelling artefacts and iii) helps ensuring compliance of models to relevant regulations.

1 Introduction

In the modern world, businesses constantly strive to reinvent and differentiate themselves under continuous pressures of regulatory and technological change, as well as the increasing time to market requirements. One of the main obstacles for the changes to be agile is the lack of support when incorporating new business requirements into existing information systems as priorities and perspectives change.

Business process (BP) modelling is the first and most important phase in the business process engineering chain. BP models are created by business analysts with an objective to capture business requirements, enable a better understanding of business processes, facilitate communication between business analysts and IT experts, identify process improvement options and serve as a basis for derivation of executable business processes. Designing a new process model is a highly complex, time consuming and error prone task. This is because BP modelling involves several sources of information, models are dynamic and frequently redesigned to adapt to changes, and BP models are often shared by several departments within a company or even between different companies.

In order to simplify BP modelling, models must be highly reusable, favoring process flexibility and minimizing designs made from scratch. Reusing implies the need for querying the process repository in order to find suitable previous work that can be the base for a new design. This can be done only by an expressive and machine-readable
description of relevant aspects of a BP model that will help to retrieve the most relevant parts of a previous work (model).

Therefore, in order to enable expressive querying of BP models, there is a need for a comprehensive formal process model description capturing all relevant dimensions (perspectives) of a process. Following [CKO92] and [JB96], we consider the functional, behavioural, organizational and informational perspective relevant to adequately organize information about a process. Based on these requirements, in our previous work [MP07] we have proposed a formal model for describing business processes which integrates all aforementioned perspectives, as depicted in Figure 1.

Figure 1: Ontology framework: Perspectives for describing a business process

In the following we are briefly describing these perspectives.

For describing the behavioural (dynamic) perspective of a process model we use a process algebra, the $\pi$-calculus. By using the $\pi$-calculus for representing the process behaviour, we are also able to integrate existing tools and techniques for verification and simulation of processes [Pu06, AB06] in our framework. The dynamic perspective of a process model stands for process control- and dataflow, and if we model it using the ontologized $\pi$-calculus, denoted by Business Process Ontology in Figure 1. For more details on this ontology, we refer the reader to [MP07].

For representing the functional, organizational and informational perspective we have proposed a set of ontologies, imported by Business Process Ontology, as shown in Figure 1. Business Functions Ontology provides a structural breakdown of the organization’s business functions. Concepts from this ontology classify process models by their functionality, independent of the business domain. Business Roles Ontology includes concepts representing roles in the organization, e.g. Manager, Engineer, Clerk, Secretary, etc. Business Resources Ontology describes the resources (documents, systems, machines) which are required to operate the activities in processes. Business Goals Ontology models a hierarchy of organization business goals (milestones, objectives) according to which the processes in the organization are designed. Business goals are modeled in such a way that they conflict if they can not be satisfied simultaneously. Moreover, goals can influence positively or negatively other goals [YM94]. Note that we refer to these perspectives as static view of a process in the rest of the text.
Having captured all relevant perspectives of a process in our model, we expose the complete process description to advanced querying and reasoning. In this paper we describe the framework for querying business process models and explore its formal nature in details.

The paper is structured as follows. In Section 2, we present three sample usage scenarios for our framework. Section 3 describes the framework in general. In Section 4, we show how the querying process operates inside our framework. Section 5 discusses related work. We conclude and give an outlook on future research in Section 6.

2 Motivating Examples

To illustrate the need for a querying framework in business process modelling, we discuss several example scenarios of the framework usage in this section.

![Figure 2: Usage scenarios for the querying framework](image)

Scenario 1: Decision making support

The key challenge in decision making is having access to all relevant information which is to be assessed in a particular situation. Such information is scattered in organization processes and has to be manually collected from diverse sources for each individual case. To facilitate this task, we enable the business expert to quickly and expressively query the process artefact\(^1\) repository of an organization (cf. Figure 2, top). Some example queries for this scenario include: “Give me all processes in the fulfillment area”, “Which processes use system x?”, “What resources are needed for running process y?”, “List all processes with conflicting goals.” [He05].

---

\(^1\) Note that we refer to process patterns, models, fragments and modeling guidelines as process artefacts.
Scenario 2: Reuse of process artefacts

This scenario describes how the business expert can query the process artefact repository for reuse of process patterns, models and fragments in process design (cf. Figure 2, center). Since process modelling is a complex activity, reuse of existing models and model components makes sense in all stages of modelling. For instance, when designing a new process the business expert can first query for existing business process patterns, generic high level process designs emphasizing business goals [Ma07], in search for the best modelling practices in the given domain. An example query for business patterns can be: “Give me all business patterns related to Fulfillment Business Function where Business Goals involved are profileObtained and serviceActivated”. The business expert can also query in the same way for existing models or process fragments - self-contained, coherent building blocks of a process model with a clear business meaning. In case that there are existing process models or fragments that are similar to the desired end design, the business expert can use them in his design in order to achieve a higher degree of reuse, compared to reuse of patterns. Moreover, if the user wants to substitute an existing process fragment based on redesign goals or auto-complete an underspecified model, he can make graphical queries by selecting the desired process part for the substitution or auto-completion in the modelling tool. For this purpose we use properties of bisimulation theory for the π-calculus [Sa96](cf. Figure 2, center).

Scenario 3: Querying modelling guidelines

This scenario covers querying for business guidelines – concrete policies defined according to the company strategy, which apply orthogonally to all processes of an organization (cf. Figure 2, bottom). Queries involved in this scenario retrieve all modelling guidelines (both mandatory and conditional) which match context annotations of the model being checked. This reduces the manual effort of creating an inventory of such guidelines for any given model. For checking which guidelines are relevant in a digital content provisioning process, the example query can be: “Give me all modelling guidelines for Digital Asset Management Business Function where clients are minors and Business Goal associated belongs to Fulfillment”.

For more details on different types of queries that business expert can perform in process modelling, we refer the reader to [Ma07].

3 Querying Framework

In order to realize the aforementioned scenarios, we have derived a set of requirements described in [MP07]. Based on the requirements, we developed a framework for querying in business process modeling. This section presents the components of our querying framework, discusses their functionalities, relation and communication with other components.
3.1 Framework architecture

In Figure 3 we illustrate the framework components and their interactions using a block diagram in FMC notation\(^2\). The framework consists of existing components that have been reused, new components that have been implemented, integration techniques and interfaces for external access. The basis for communication between components is the ontological process description based on the ontology framework in Figure 1.

![Figure 3: Framework architecture](image)

A business Modelling tool has the function to provide an environment for modelling business processes. It is a component which is used by the business expert to interact with our framework. In particular, the user interacts through a user-friendly query interface which will be detailed later in this work. We extended the SAP Research modelling tool "Maestro for BPMN" for integration with our querying framework.

The Process artefact library has the function to provide persistent storage for process patterns, models, fragments and guidelines. For this purpose we used the Ontology Representation and Data Integration (ORDI)\(^3\) framework, a middleware component designed to be able to load various ontology languages and allow enterprise data integration via RDF-like data model.

The Ontology reasoner performs ontological reasoning for obtaining the query results. Since behavioural reasoning is computationally more expensive, query answering by the ontology reasoner is performed first and serves as a filtering step to obtain a subset of process descriptions for later behavioural conformance checking (cf. Figure 3). Since our ontologies are represented in the WSML\(^4\) language, we use WSML2Reasoner\(^5\) framework on top of IRIS\(^6\) back-end reasoner to perform ontological reasoning.

---

\(^3\) [http://ordi.sourceforge.net/](http://ordi.sourceforge.net/)
\(^5\) [http://tools.deri.org/wsml2reasoner/](http://tools.deri.org/wsml2reasoner/)
The Behavioural reasoner is a component that performs π-calculus reasoning, i.e. it serves as a query answering mechanism for graphical queries used in process substitution and auto-completion scenarios. At the moment this component implements the strict congruence algorithm, based on the definition of bisimulation from [MPW89]. In the future, we plan to embed Mobility Workbench (MWB) [VM94] as a subsystem in our framework implementing bisimulation for the π-calculus. This will allow us to investigate more advanced notions of process equivalence.

The Ontology API provides methods for creating and manipulating the ontology object model. This API is used by the modelling tool for creating an in-memory representation of the modelled process, based on the ontology framework in Figure 1. For our purposes we utilized WSMO4J, a reference implementation for WSMO\(^7\) and WSML specifications.

The Parser/Serializer component has been implemented to provide the round trip transformation between WSML ontological representation and plain π-calculus representation describing the behavioural perspective of a process.

In the rest of this section we give more details about the user interface component, embedded in the modelling tool.

3.2 User-friendly query interface

The intended users of our framework are business experts. Therefore, they need to be provided with an intuitive and user-friendly query interface to be able to specify their queries in an easy way, not much different from using the applications they are used to. The complexity of ontologies and reasoning needs to be hidden from the user. In the following, we describe how we designed the query interface based on this requirement.

The query input dialog for performing the queries on the static view of a process (static queries) is presented in Figure 4. The user can navigate through tabs for selecting business annotations (business goals, functions, roles and resources) of the processes he wants to retrieve. Note that these characteristics correspond to the perspectives depicted in Figure 1. In the right box, the ontology navigator provides available ontology concepts which the user can browse.

Desired business annotations can be dragged from the ontology navigator to the left box (Business Functions panel) and marked as required or optional for querying. This is important for achieving flexibility in querying, since the concepts marked as optional can be omitted when constructing the query for retrieving more results (cf. Section 4.3). The user can also specify the type of artefacts he is looking for (model, pattern, fragment, guideline) in the upper panel.

In contrast to performing static queries, querying for desired behaviour does not need a new dialog. Querying is performed by selecting a process part directly in the modelling tool, as shown in Figure 5 (shaded area). We consider this way of graphical querying to be intuitive to the user. The behavioural description of the selected part is obtained automatically using the Parser/Serializer component and used as an input for behavioural reasoning (substitution, auto-completion, deadlock/liveness verification).

4 Querying Process

In this section, we discuss how the querying process operates inside our framework. We first discuss the formal query language which has been defined for coupling static and dynamic process characteristics. Further, we shortly describe the mechanism for
processing queries. Third, we explain how we have introduced flexibility in querying for providing better query results.

4.1 Formal query specification

The query must support the description of static and dynamic behaviour. WSML Logical Expression (LE) is used for specifying queries on static properties of a process. The query should reference instances of business annotations, which are materialized by instances of the relations described in [MP07]. An example is given in the following:

\[
\text{bpo#hasBusinessFunction}(?x, \text{Annot1}) \text{ and } \text{bpo#hasBusinessRole}(?x, \text{Annot2}) \text{ and } \\
\text{bpo#hasBusinessResource}(?x, \text{Annot3}) \text{ and } \text{bpo#hasBusinessGoal}(?x, \text{Annot4})
\]

The dynamic behaviour of a wanted process is described as a process definition, i.e., it uses the ontology framework given in Figure 1 for describing a process, its connections, and the annotations of tasks as relation instances. Defining that the behavioural query has the same structure as the process definitions avoids the mismatch problem that would appear from having a different query language for behavioural querying. Therefore, here we use the ontologized $\pi$-calculus to describe the user request (process query). This query is checked against processes stored in the repository using congruence and bisimulation properties.

The query specification must be defined to use the same language used by processes description. This enables the reuse of the ontological process model for representing the user queries. For meeting our requirements, the query specification should be in a format of a template with placeholders. In addition, it should encapsulate both static and dynamic attributes in a unified language.

The use of a BPO ontology instance is the solution to this: the query template corresponds to a pre-defined ontology structure with namespace definition and element descriptions. Table 1 shows the template contents:

<table>
<thead>
<tr>
<th>Namespace</th>
<th><a href="http://www.ip-super.org/ontologies/BPO/extension/query">http://www.ip-super.org/ontologies/BPO/extension/query</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>dc:type</td>
<td>Either “substitution” or “autocompletion”</td>
</tr>
</tbody>
</table>

Axiom

ID: http://www.ip-super.org/ontologies/BPO/extension/query#static definedBy:
\[
\text{bpo#hasBusinessResource}(?x, \_PLACEHOLDER_) \text{ ‘OR’ } \\
\text{bpo#hasBusinessFunction}(?x, \_PLACEHOLDER_) \text{ ‘OR’ } \\
\text{bpo#hasBusinessGoal}(?x, \_PLACEHOLDER_) \text{ ‘OR’ } \\
\text{bpo#hasBusinessRole}(?x, \_PLACEHOLDER_)
\]

Process

ID: http://www.ip-super.org/ontologies/BPO/extension/query#dynamic
A BPO process model description.

Table 1: Query definition using BPO ontology instance
The ontology contains an axiom (containing the static query), and a process definition (containing the process behaviour). The non-functional property dc:type indicates the type of behavioural query being performed.

The ontology submitted as query contains all necessary information for specifying the query request. The approach enables performing ontological reasoning on the static part and π-reasoning on the dynamic part of process description. This approach is also scalable, since adding new concepts or adding new information to the query will not imply changes to the established query definition.

4.2 Query mechanism

In order to reduce the level of complexity, we have divided this task into two subtasks – static and dynamic (behavioural) querying. The querying mechanism operates on the Business Process Ontology (BPO), presented in Figure 1.

The first subtask investigates simple (static) querying, where the user can specify constraints related to the static view of a process. Here we use WSML logical expressions as a query language and ontological reasoning for query answering. The second subtask investigates graphical (behavioural) querying, where the user can specify requirements on dynamic perspective of a process description. This corresponds to autocompletion and substitution scenarios where algorithms from bisimulation theory are used for comparing the processes. In case that the user request contains constraints on both static and dynamic perspective of a process, static querying is performed first since dynamic querying is more computationally expensive. For more details on the framework querying mechanism, we refer the reader to [Ma07].

4.3 Achieving flexibility in querying

Depending on the user query, there can be too many or too few results coming from the repository. The user should be able to i) specify further constraints in the query, choosing more refined goals, thus retrieving more precise and shorter list of results or ii) eliminate some constraints from the query and look for more abstract goals or more undefined flow structure of the process, thus retrieving more results.

(i) The refinement is done mainly by the navigation inside sub-concepts, skipping instances connected to super-concepts. Since a concept can have many sub-concepts, the interaction with the user may be necessary to choose which path to follow. In Figure 6, an example of refinement is represented by a search for the goal “Creation Done”, which has at the moment no instances directly connected, however, its sub-concepts “User Profile Created” and “Catalog Entry Created” do have, and processes associated with these instances are results of a refined query.
(ii) If there are too few results, the framework can automatically search for instances connected to the parent concepts of the requested one in order to relax the query. For example in Figure 6, a query for processes related to the goal “Client Known” has just one instance as a result. If the framework decides to look for instances connected to “User Known”, two instances match the query.

In general, when going deeper into the tree structure of goals one would retrieve fewer results (refinement), while when going upward in the tree structure the relaxation occurs.

A second way for relaxing a query is to “skip” target concepts in the query, e.g. using the “OR” statements. The user can specify which concepts are required and which are optional using the interface in Figure 4. Consider an example when the user wants models achieving goals A, B and C, where A and B are required to be parts of the model and C is optional. The framework can try to find models achieving all goals with the statement:

\[
Q = \{ x \mid x \in M \land \text{ant}(x, A) \land \text{ant}(x, B) \land \text{ant}(x, C) \}
\]

Where \( Q \) represents the resulting set of process models, \( M \) denotes the set of all process models and relation \( \text{ant} \) denotes that a model \( x \) is annotated using the ontology concept \( A \).

In case of no result, the framework decides to relax the query, executing the statement:

\[
Q = \{ x \mid x \in M \land \text{ant}(x, A) \land \text{ant}(x, B) \}
\]

In the ideal case, the framework would submit the following statement, preferring to answer the user queries where all three goals are achieved:

\[
Q = \{ x \mid x \in M \land (\text{ant}(x, A) \land \text{ant}(x, B) \land \text{ant}(x, C)) \lor (\text{ant}(x, A) \land \text{ant}(x, B)) \}
\]
Here we see a clear need for ranking: results that fulfil all goals are matching perfectly the user query, while results obtained for the relaxed query are fulfilling just a part of the user’s need. At present, the ranking of query results is not supported by our framework. That will be a part of the future work.

5 Related Work

The significance of querying business processes has been acknowledged by BPMI\(^8\) that launched a Business Process Query Language (BPQL) initiative [BPM]. However, no standard specification has been published yet.

In [Be06], the authors present a query language for querying business processes, BP-QL. The query language is designed based on the BPEL\(^9\) standard and thus focuses on querying executable processes. Our work focuses on the reuse of higher level business knowledge, i.e. BP artefacts. In addition, our query specification language is more expressive in that apart from constraints on data and control flow, the user can specify additional properties of the models he wants to retrieve.

The approach presented in [MCZ04] discusses a process component model for process knowledge reuse. Here, the process component model is characterized only using static information (domain, function, performance, lifecycle). We defined a more refined notion of BP artefacts which can be reused in different stages of the presented modelling lifecycle. In addition, with this approach the user is not able to specify behavioural queries which is a more intuitive way of specifying his requests.

We have not seen other approaches addressing the problem of querying in business process modelling using a rich formal model for business processes.

6 Conclusion & Outlook

In this work we have presented a framework which enables expressive querying in business process modelling phase. The framework enables the business expert to have a quick and easy access to the library of process artefacts. We have illustrated three key usage scenarios showing the benefits of using our querying framework. Furthermore, we have developed a prototype of the querying framework, based on the Maestro BPM tool. Currently we are performing a use case study and the evaluation of the results is the very next task.

As our next step, we plan to embed MWB in our framework and investigate more advanced notions of process equivalence based on the theory of bisimulation. In the long term, we aim to define similarity measures for process models with the purpose of quantifying the level of similarity between two models, which can be used in ranking of query results.

Bibliography


Auto-completion of Executable Business Process Models

Matthias Born\textsuperscript{1}, Christian Brelage\textsuperscript{1}, Ivan Markovic\textsuperscript{1}, Daniel Pfeiffer\textsuperscript{2}, Ingo Weber\textsuperscript{1}

\textsuperscript{1}SAP Research Center CEC Karlsruhe, SAP AG, Vincenz-Prießnitz-Str. 1, 76131 Karlsruhe, Germany 
{mat.born, christian.brelage, ivan.markovic, ingo.weber}@sap.com

\textsuperscript{2}European Research Center for Information Systems (ERCIS), Leonardo-Campus 3, 48149 Münster, Germany
pfeiffer@ercis.de

Abstract

This work presents an auto-completion mechanism for creating executable business process models. One purpose of business process models is to serve as the basis for enacting process execution (for instance via workflow management systems). Currently, process modeling tools, however, provide only little support to identify the relevant services that are needed to execute the process model. The selection of appropriate services is left to the skills of the modeler. A novel solution technique for this problem is proposed by combining (1) a context-based analysis, (2) by taking pre and post-conditions into account, and by (3) evaluating the non-functional properties of the functionally and context-wise fitting services. A list of suggestions for possible successor activities is provided by means of the actual state of the process model. The approach described herein has the potential to improve the overall efficiency of the modeling process significantly by a) improving the model quality and reduce modeling errors b) speeding up the modeling process by making sensible suggestions and c) allowing earlier identification of possible services needed to implement a given activity.

1. Problem description

Business process modeling has evolved an important mean for the collection, documentation, and analysis of processes in companies and public administrations. A great variety of process modeling tools are offered to support the creation and maintenance of business process models. To name just a few: ARIS Toolset, Bonapart, Adonis, Microsoft Visio, Income BPM Suite, Nimbus Control or Lombardi Blueprint.

More recently, the focus of business process modeling has changed from just being a mean for documentation towards enabling process enactment. The constantly growing interest for service-based architectures (SOA) has moved the identification and composition of services in the focus of research and practice. Business process models are commonly considered as an appropriate vehicle to support this task. All major vendors of ERP software have published SOA-based strategies and have stressed the important role that process modeling plays in it.

However, current process modeling approaches and tools are still very costly and error-prone since users are not guided in any sensible way. Major training investments are needed in order to teach people in modeling business processes. In particular the following problems hinder a more widespread and economical usage of business process modeling:

Transition from “sense-making” models to engineering models is done manually and, thus, error-prone, costly, and slow.

The waterfall-like modeling paradigm of continuous refinement does not reflect the interactive, “going-back-and-forth” nature of the modeling process. In particular, we claim that an early identification of technical artifacts (such as Web services) has the potential to improve the model quality significantly during the modeling activities. The approach proposed herein allows matching (technical) services to conceptual modeling elements as early as possible in the modeling process.
Current process modeling tools provide only little guidance to the user and, thus, resemble a scratchboard rather than a technical approach. If any guidance is provided at all, it is on a purely syntactical level which does not account for business semantics. The approach describe herein allows tools that factor in business semantics and, thus, make suggestions that follow business logic.

Finally, the approach speeds up the modeling process by making sensible suggestions and by implementing trivial tasks (semi)automatically.

In this paper we address the problem of how to find appropriate services and how to present them to the modeler. We introduce a mechanism that offers a list of services whose entries represent possibly following model elements. This list is compiled based on the actual state of the process model. We call this functionality an auto-completion mechanism. It works like this: whenever a certain manual or automated activity has been created or selected on the modeling pane the system suggests a list of services. The entries in this list are potential successor activities which are already linked to a service. Thus, the modeler can choose from a small set of possible items instead of browsing through the entire service repository. This leads to a more efficient and productive construction of executable business process models and improves the usability of process modeling tools.

This paper proceeds as follows: in the next section we present the technical structure of our novel solution technique. Subsequently, based on these results we present the graphical representation of the technique in a process modeling tool. We provide an overview of related work and close with a summary of the main result and a discussion of our contribution.

2. Solution technique

The solution technique we describe in the course of this work consists of three different sources, namely process context-based analysis, pre- and postcondition analysis, and non-functional property analysis. These sources provide sets of possible successor activities and a combined analysis can be used to improve the overall quality of the suggestions. The three different techniques are used to identify appropriate services from a repository and the results of these three independent techniques are weighted and consolidated. Based on a reasoner the suggested services are deviated. Finally, the resulting list of proposed services is presented to the user within the process modeling tool. The general architecture of the proposed solution technique is shown in Figure 1.

In the following subsections we will discuss the three aspects of our technique in more detail and describe a mechanism for consolidating and weighting the individual aspects.

![Figure 1. Architecture of the proposed solution technique](image)

2.1 Process context-based auto-completion

In our opinion, the process context is composed out of the following perspectives: Business Function, Business Domain, and Business Goal, which are modeled in the appropriate process ontologies. These ontologies provide an unambiguous, explicit and machine-readable categorization of processes by three basic aspects: functional, domain, and intentional aspect.

The Business Function Ontology (BFO) provides a structural breakdown of the organization’s business functions. It does so by splitting the domain in two dimensions, namely vertical and horizontal. The vertical dimension classifies processes into various process areas, such as: procurement, manufacturing, warehousing, order fulfillment, etc. Horizontal dimension differentiates the key functional areas across process areas, describing concepts such as Customer Relationship Management, Supplier Relationship Management, Product Lifecycle Management, Supply Chain Management, etc. Concepts from this ontology classify processes by their functionality, independent of the business domain. Business Domain Ontology complements BFO and describes the domain inside the organization where the process is used. Examples of business domain concepts are: product area, client area, localization area, etc. Business Goals indicate what processes need to achieve from the business perspective and provide answers to why processes exist in the organization. The Business Goal Ontology...
models a hierarchy of business goals and provides a set of relations between them. For more details on formal process representation, we refer the reader to [5].

By annotating processes and Web services with the concepts from the aforementioned ontologies, we are able to perform context-based matching in order to find the Web services which contextually match a given process. The matchmaking mechanism that can be used here is described in detail in [6].

When performing matchmaking, we utilize the ontology structure for ranking of the matching results. Here, we measure the semantic distance between the concepts annotating a process model and those annotating services. Services annotated with concepts having the smallest semantic distance obtain the highest rank in the list.

With this approach we can ensure that the discovered services are fitting in the specific context of the process models. However, we cannot guarantee that the proposed services can eventually be executed as the current state of the process might not fulfill the requirements of the service.

2.2. Auto-completion with pre- and postconditions

There is a substantial body of scientific work on the problem of functional discovery, i.e., matching services against a request on the basis of pre- and postconditions. The precondition of a service is a logical formula that describes under which circumstances the service is able to execute: if the state of the (relevant part of) the world satisfies the logical formula when the service is invoked, then its execution is defined. Similarly, the postcondition (sometimes also called effect) describes how the service changes the world, i.e., how the state when invoking the service is altered by the service.

As stated before, substantial prior art is available for this task. Given the pre- and postconditions of an anticipated service (also referred to as a goal) and a repository of services, these approaches can determine the degree to which each service matches the goal. In particular the best match or best matches can be determined.

In principle, any such approach can be used for the purposes here. This part of the invention describes how functional discovery can be used for auto-completion. In short, this can be achieved by coming up with the suitable pre- and postconditions for the part of the process where a service is to be added. The precondition can be derived from the process as modelled so far [2] by propagating the postconditions of process activities [1] that will be executed before the current position in the process will be reached.

Then, all services with a (partially) satisfied precondition can be selected for auto-completion, where the matching degree of the services can be taken into account by the ranking. Further, potential conflicts with parallel process parts can be detected upfront [2] and the auto-completion can either exclude the conflicting services or adapt the ranking accordingly.

Using the pre- and postcondition analysis we can be certain that the proposed services can be executed at runtime as the states of a service are matched against the states of the process. Nevertheless, this approach might list services that are unsuitable in a specific process context.

2.3. Auto-completion with non-functional properties

Non-functional descriptions of Web services, in particular quality parameters such as price, availability, and execution time, can also be used in the auto-completion process. This step is performed after the process-context and functional-based auto-completion provided a matching list of services. In this step, we evaluate the level of match between the desired non-functional properties against those offered by services obtained through pre-filtering in the previous two steps.

The user will have the possibility to specify his preferences regarding the predefined set of quality of service criteria in the modeling tool, as shown in [4]. Techniques presented in [4] can be used as a basis for machine-readable representation of non-functional aspects as well as the matchmaking and ranking mechanism.

By taking into account the quality of service properties when proposing follow up activities, we increase the level of precision of the suggestions and achieve a more refined ranking according to user preferences.

2.4. Auto-completion based on combined criteria

This section describes how the “Consolidation and Weighting Mechanism” from Figure 1 can be implemented and how the weights may be adjusted based on user feedback.

The list of suggestions can be consolidated by making sure that each service appears only once and by ordering the combined list through a unified view on the matching degree. We suggest doing so by calculating a weighted sum of the individual matching
degrees. We hereby assume that the matching degree is expressed as a value > 0. Then the formula for the weighted sum is:

$$MD_{\text{Sum}} := MD_{\text{Context}} * W_{\text{Context}} + MD_{\text{Functional}} * W_{\text{Functional}} + B_{\text{NFP}} * MD_{\text{NFP}} * W_{\text{NFP}}$$

where the right-hand part of the formula comprises $MD$ as the matching degree and $W$ as the weight of either the Context (2.1), the Functional discovery (2.2) or the Non-Functional Properties (2.3). We hereby assume that the absence of a match is signaled by a $MD=0$, and the better a service matches the higher the respective $MD$. $B_{\text{NFP}}$ is a switch: if $MD_{\text{Context}} = 0$ and $MD_{\text{Functional}} = 0$ we set $B_{\text{NFP}} := 0$; otherwise we set $B_{\text{NFP}} := 1$. Thus, if a service is no match in terms of the Context or the Functional Discovery, then the combined $MD_{\text{Sum}}$ is set to 0; otherwise it is the weighted sum. A non-functional match can improve the $MD_{\text{Sum}}$, but if the service is not of interest with respect to functionality or context, then the $MD_{\text{NFP}}$ is of no relevance.

The goal of this calculation is to have a single unified view on the matching degree, $MD_{\text{Sum}}$. Then, the list of matches can be ordered according to $MD_{\text{Sum}}$ and, optionally, also pruned so that, e.g., only the 15 best matches are displayed.

While the weights may be set based on experience or importance of the single technique, they can also be learned from user behavior. Machine-learning approaches such as Reinforcement Learning [3] can learn user preferences from actions. This could be used by allowing the system to adjust the three weights on its own and by interpreting the user selection from the list as the feedback: the higher the ranked service which the user selects, the better. In particular, multiplying the position of the selected service in the list with “-1” can serve as the so-called feedback function. E.g., when the seventh service in the list is selected, then the feedback is -7. This is better than a -10, but worse than a -1. An interesting question arises around the following points:

- List places which are displayed directly, e.g., the top three or top five choices, could be set to the same feedback value, arguing that the top places in the list are all reasonably good.
- If none of the services in the list applies, the presented services may be the wrong ones due to false weights or the information on which the choices are presented may be misleading or inaccurate. Therefore, the question is whether this should result in a strong negative feedback value or not taken into account by the learning system.

A practical evaluation is necessary to give answers to these questions and to obtain meaningful weights.

3. Representation of the solution

The graphical representation of the solution is based on a mouse-over event. The list of suggested services is shown directly when the mouse is placed over an activity, depicted with a rectangle in Fig. 2. The services are represented as small pictograms underneath the selected activity in the BPMN notation for automated activities.

Figure 2. Graphical representation

The list of services is filtered based on the previously described solution technique. Only a specific and very small number of services (e.g. three) is shown directly. Other relevant services can be reached by two navigation buttons, one at the left hand side, one at the right hand side. The pictograms of the automated activities can be dragged onto the modeling pane. When they are dropped they are automatically connected to the previously selected activity via a sequence flow and assigned to their corresponding service.

4. Related Work

Our approach is related to the work of [7], which is also based on semantic business process modeling. The author describes methods and techniques to support
modeling of semantically annotated business processes described in Petri Nets. The work presented by Koschmider focuses on measuring the similarity of process task labels in order to suggest possible process fragments whereas our approach has a clear linkage to the implementation services.

In [8], the authors present an approach for supporting the modeling of business processes using semi-automated Web service composition techniques. The main difference to our work is that it only takes into account the functional part of service descriptions when making suggestions during modeling. We consider the process context and non-functional properties in addition to the functional properties, thus increasing the level of precision of the suggestions.

5. Conclusion

This work presents a novel approach to improve current business process modeling tools. Our solution supports the process modelers actively during their modeling tasks and thus helps them in finding appropriate services more intuitively. Eventually, this will lead to process models with higher quality and reduces the actual modeling time, as the process modeler gets active support from the process modeling tool.

![Figure 3. Overview about the auto-completion process](image)

The figure sketches the auto-completion process which will be briefly summarized in this section. Based on following three different techniques the relevance of services is rated: i) process context-based auto-completion, ii) auto-completion with pre- and post-conditions, iii) auto-completion with non-functional properties.

Using a mechanism to weigh the relevance of the three techniques, the system will eventually suggest meaningful services to the modeler. The results can be displayed using different graphical embodiments which allow integrating the approach into state-of-the-art modeling environments.

In contrast to existing process modeling tools we claim that our solution is a step towards more intuitive and guided process modeling tools. Some of the major benefits are i) early identification of technical artifacts which match to conceptual model elements, ii) sensible suggestions for the users based on business logic and not only on mere syntactical information, iii) improved model quality and faster modeling process by (semi)automating trivial modeling tasks.

6. References


METHODOLOGICAL EXTENSIONS FOR SEMANTIC BUSINESS PROCESS MODELING

David de Francisco, Henar Muñoz, Noelia Pérez, Javier Martínez
Telefonica Research & Development, Valladolid, Spain
{davidfr,henar,npc,eli}@tid.es

Ivan Markovic
SAP Research, Karlsruhe, Germany
ivan.markovic@sap.com

Keywords: Business Process Modeling, Methodology, Reusability, Validation.

Abstract: Semantic Business Process Management (SBPM) aims, among other advantages, to facilitate the reuse of knowledge related to the definition of business processes, as well as to provide an easier transition from models to executable processes by applying semantic technologies to BPM. In this article the authors focus on extending the scope of knowledge being reused from executable processes to semantic models’ scope. This is done by providing some methodological extensions to existing BPM methodologies in order to take advantage of semantic technologies during business process modeling.

1 INTRODUCTION

Business Process Management (BPM) is an approach to manage the execution of information technology-supported business operations [Smith and Fingar2003]. Semantic Business Process Management (SBPM) [Hepp et al.2005] extends the BPM approach by adopting Semantic Web technologies in order to bridge the gap between the business and technical perspective on supporting information systems. In this context, semantics tries to overcome i) the lack of formal representation of process models, which results in a more difficult transition from models to executable processes, ii) the unnecessary complexity of current business process models, and iii) a lack of reutilization which implies important time-to-market increase.

A simplified comparison between BPM and SBPM life cycles is depicted in Figure 1. BPM life cycle (depicted in continuous arrows) starts with process design [Weske et al.2004] made by business analysts, which provides process models (1). These models are further translated by technicians (2) into executable processes (3), which are integrated with previous processes defined within the company by technicians (4). Semantic BPM life cycle (depicted in dashed arrows) designs semantically annotated models (1’), enabling reuse of previous models and the validation of their business context (2’). These models can be translated into executable semantic processes with less (or no) interaction from technicians (3’). The executable semantic processes can be composed (as Web services are) with other semantic processes in a more flexible way due to their uniform annotation (4’).

Figure 1: Comparison between BPM (continuous line) and Semantic BPM Life Cycles (dashed arrows)

Focusing on design steps (upper part of the picture), a need for adaptable methodologies to drive business process (BP) modeling is presented in [Roser and Bauer2005]. Modeling is covered by
several BPM methodologies, such as ARIS [Scheer2000, Scheer1999], IBM [Wahli et al.2007, IBM2006]. However, the lack of a uniform notation in modeling tools brings model inconsistencies. That stimulated new proposals to share BP models [Yamamoto et al.2005], to improve reusability and flexibility in BPM [Yao et al.2006] or to optimize BP design in order to empower process redesign [Zhou and Chen2003].

Semantic technologies can overcome the lack of uniform representation, empowering the reusability of previous models as previously argued. In addition, semantic BP modeling offers some inherent techniques that are obviously not covered in current BPM methodologies, such as semantic annotation [Born et al.2007], discovery [Markovic and Karrenbrock2007], querying [Markovic et al.2007] and composition of processes, in order to favor the reuse of BP models [Markovic and Pereira2007]. That makes a specific coverage of such techniques necessary for semantic BPM methodologies.

The proposal of this paper focuses on the modeling phase, where a big gap in the translation between business requirements to process models still exists [Hepp et al.2005]. Therefore, we believe that a methodological coverage of steps 1' and 2' of Figure 1 is also needed. To address this requirement, we propose some extensions to current semantic Business Process modeling methodologies which can favor the reuse of BP models [Markovic and Pereira2007]. That makes a specific coverage of such techniques necessary for semantic BPM methodologies.

The authors will present a brief state of the art of the art of current semantic and non-semantic BP modeling methodologies (Section 2). In Section 3 we illustrate the proposed methodology extensions with examples from telecommunication sector, as one of the business domains which could obtain more benefits from the better reusability due to the strong time-to-market requirements of this sector. The article concludes with an outlook on the usage of these methodology extensions in Section 4.

2 STATE OF THE ART

BP modeling phase is covered by most of the existing methodologies, e.g. ARIS [Scheer2000, Scheer1999], IBM [Wahli et al.2007, IBM2006]. The modeling life cycle is mainly formed by requirement analysis from business analysts, process modeling by using a formal language, simulation and redesign [IBM2006]. In addition, ARIS defines the five views of the knowledge (mainly focused on IT level) used during BP Modeling activities [Scheer2000]: data (information objects and their relationships), organization (organizational structures and their relations), function (activities), product (input and output produced by activities) and control (process flow). However, the knowledge related to BP modeling is broader than current methodologies and languages can represent [Mou et al.2004]. Extending the representable knowledge of current methodologies and languages can provide several benefits to overcome the aforementioned weaknesses.

Within the SUPER project (ip-super.org), a methodology which covers the complete SBPM life cycle (from modeling to execution and analysis) has been defined [Fantini et al.2006], and a proposal of methodological guidelines for modeling and configuration phases has been made [Weber et al.2007]. This proposal's objective is to facilitate the transition from BP models to executable processes, as well as the reuse of workflow (steps 3' and 4' of Figure 1). However, we argue that focusing BP modeling on getting only implementation benefits i) doesn't allow business analysts to fully define models, ii) doesn't capture all knowledge from a business perspective which is usually very relevant to a company and iii) misses a great opportunity to reuse and extend the business knowledge, as presented in the following section.

3 LIFE CYCLE PROPOSAL FOR SEMANTIC BP MODELING

In this section, authors propose a life cycle for semantic business process modeling. This life cycle, depicted in Figure 2, is based on the spiral life cycle model first proposed by Boehm [Boehm1988], which is widely adopted by BPM community due the dynamic nature of BPs and their consequent redesign cycle. The use of a continuous evaluation and improvement process (such as CMMI [Institute2007]) is recommended in order to define the procedures to ensure the quality of any artifact produced during semantic BP modeling (ontologies, models, patterns, etc.), and thus depicted in the outer ring of the picture. However, the definition of such a process is beyond the scope of this article.
3.1 Requirements Analysis Phase

During the Requirements Analysis phase, business analysts collect all the relevant information for the BP design and further implementation. This information comes from various sources (e.g. the company's strategy, marketing and product departments, final customers, partners, etc.), and has to be collected and documented by analysts.

3.1.1 Requirements Capture

Requirements Capture is the phase in which a business analyst collects all the relevant information for a BP design and extracts requirements for the BP. The information taken into account comes from heterogeneous sources (e.g. meeting minutes, technical and business documents, etc.) and a common understanding between analysts and requirement providers is needed. The study of this information will discover business and technical needs that will be documented as requirements during the next sub-phase. During this phase, while technical specifications are being defined, knowledge is exchanged between business analysts, people from different departments/companies and technicians.

3.1.2 Requirements Documentation

This phase involves the documentation of requirements captured in the requirement analysis. It is composed by the requirement classification to support their traceability, as well as the refinement of the requirements specification, defining the functional and non-functional objectives to be covered by the BP. Output of this sub-phase is a requirement analysis document in which requirements for the BP modeling are documented as stated before.

There is no systematic way of documenting requirements. Most often the business analysts use Microsoft Word, Powerpoint or spreadsheets for documentation purposes. By providing a shared model (see 3.2.2), ontologies have the potential to structure the method of documenting requirements, enable their common understanding and provide the possibility for automated analysis. An ontology for requirements documentation as in [Jinxin et al.1996] would allow e.g. for detecting redundancies and conflicts in the requirements specification. It would also enable gap analysis and easier traceability of requirements across the specification, by answering queries such as: “What is the source of this requirement?”

3.2 Modeling Phase

After requirements have been specified, the business analyst provides several artifacts as a result of the BP modeling. These artifacts are ontologies, BP patterns, goals and KPIs, and finally concrete BPs which will be translated to executable processes and run in further steps not covered in this article.

3.2.1 Ontologies Modeling

Requirement analysis phase produces a list of documented requirements which might affect the business ontologies being used during the modeling and validation phases (e.g. redefinition or new concepts, relationships, etc.). Business ontologies model different perspectives of processes, where we
reuse the ontology framework described in [Markovic and Pereira2007], and domain-specific information (domain ontologies). The output of this sub-phase is the refinement of business ontologies according to the specific scenario. Note that many of the domain ontologies can make use of existing standards, like NGOSS [Reilly and Creaner2006] in the case of telecommunications sector.

3.2.2 Patterns Modeling

BP patterns are abstract BPs which are not fully defined in terms of concrete tasks and services, and thus can not be executed. These patterns capture the first draft of a process, making emphasis on the business goals and leaving the concrete definition of processes to a further step. They represent a solution to a well known problem, and a base to enable the extension to a concrete problem (as software design patterns [Gamma et al.1995] do), providing best modeling practices.

The first step in defining a business process is to define objectives to achieve. This definition is commonly done by the definition of business process patterns, which can be defined from previous patterns or from scratch. BP patterns are implicitly used by business analysts when they want to define what a process should achieve without defining how this will be done. The patterns define high level goals (or milestones) and a simple workflow which connects them.

The objective of the formalization of these high-level models in our methodological extensions is to provide best practices for other business analysts and a common base for the specification of BPs which share a same sub-domain (e.g. all BP of a given product family share common features in their design).

Patterns are annotated with the context of their usage which is comprised of a business function, a business goal and a business domain. These annotations allow us to query for patterns in the early stages of modeling in order to provide modeling guidance. An example query for business pattern can be formulated in the following way: “Give me all business patterns related to Fulfillment Business Function and Client and Product Business Domain where Business Goals involved are profileObtained and serviceActivated”.

3.2.3 Goals and KPI Modeling

After the definition of the objectives which the BP being modeled has to accomplish, a business analyst usually refines those objectives in terms of business goals at the operational level (more concrete than the ones defined for the pattern), and assigns them to corresponding key performance indicators (KPIs). KPIs define the metrics used to monitor and measure the performance of business processes during execution time. Business goals and KPIs can be reused from previous models. The output of this phase will be the formalization of both business goals and KPIs, annotated according to the ontologies defined in the ontology modeling sub-phase.

Formal modeling of goals allows us to perform goal-based analysis of process models in order to detect redundancies and conflicts in process specifications. In addition, we could follow the dependencies (e.g. identifying which goals support or hinder the observed one) in the goal model. It also allows for improved traceability when changes in the environment occur. Through reflecting the changes on business goals, we are able to easily identify the processes which need to be adapted to the new requirements.

3.2.4 Process Modeling

Once business process objectives are clear, and the metrics used to measure its performance defined, the process is defined. Tasks and workflow can be defined by either i) deriving from previous BP patterns, ii) extending or refining previous BP models, iii) editing directly, or iv) any combination of the previous options, since BP modeling is often made in several steps which can mix up previous approaches. Once the process is defined, it is semantically annotated according to the ontologies defined in the ontology modeling sub-phase. Once the process is finished, we will have a complete BP model including the following artifacts: a BP model with semantic annotations, goal(s) and KPI it should fulfill, BP pattern(s) which derives from, if any, and ontologies which formalize the knowledge used to create the aforementioned artifacts.

The business process ontology [Markovic and Pereira2007] and the domain ontologies defined in section 3.2.1 are used in querying for business process artifacts, which supports decision making and facilitates reuse of modeling artifacts [Markovic et al.2008]. Some example queries for the decision making support scenario include: “Give me all processes in the fulfillment area”, “Which processes use system x?”. Since process modeling is a complex activity, the reuse of existing models and model components makes sense in all stages of modeling. When reusing existing modeling artifacts,
the analyst can reuse process fragments (self-contained, coherent building blocks of a process model with a clear business meaning) or existing process models which are similar to the desired model. Moreover, the analyst can substitute an existing process fragment based on redesign goals or auto-complete an underspecified model, by making queries on process annotations in the modeling tool [Markovic et al.2008].

3.3 Validation Phase

Validation of BP models is a crucial phase in which it is ensured that the BP designed effectively covers the requirements captured in the first phase from both a functional and business perspective. Besides, quality of the model must be checked, ensuring that the model designed will derive in an executable process with high performance, and that the model is conceptually correct.

3.3.1 Model Quality Validation

This sub phase checks the model quality according to a set of metrics. These metrics heavily depend on the company and the continuous evaluation and improvement model they are following. Metrics can include complexity of the model, structure of the model, modularization, or cognition [Volker Gruhn2007].

Annotating processes using a common vocabulary (provided by ontologies from section 3.2.1) enables semantic evaluation of process models and leads to improved readability of the models not only for human beings but also for machines. This enables automation envisioned by the Semantic Business Process Management methodology. Furthermore, by using a shared vocabulary the abstraction conflicts are eliminated and problems caused by synonyms and homonyms are avoided, resulting in models of higher quality.

3.3.2 Functional Validation

This sub phase checks the coverage of functional requirements that the BP has to accomplish. Additionally, the workflow performance is usually tested by the simulation of the model, in order to detect possible bottlenecks at early stages of the BPM life cycle. The result of the validation brings new requirements for improving the BP model, which are taken into account in subsequent modeling iterations.

In this phase, we can utilize the formalization of the behavioral (dynamic) perspective of processes from [Markovic and Pereira2007] in order to perform formal verification of the correctness of the model. This includes deadlock and liveness checks on the model by using techniques from [Puhlmann2006] and also the step-by-step simulation of the process behavior [Puhlmann and Bog2006]. Based on the results of the verification, business analysts can perform necessary corrections on process models in the early stages of the BPM life cycle.

3.3.3 Business Validation

Real BPs do not only have to accomplish certain functional requirements, but also are influenced by business policies and rules which are derived from the company strategy. Business policies orthogonally apply to all business processes of the company and can be mandatory or optional. Mandatory policies must be correctly addressed in BP models, while conditional ones are suggestions to improve or complement the model which can or cannot be taken in consideration. Business rules affect the execution of the process, defining conditions which drive the execution of the workflow. Checking both elements usually provide new requirements for later modeling iterations as well.

Business policies can be annotated with the context of usage (business function, goal, domain) in order to retrieve all policies (both mandatory and conditional) which match context annotations of the model being checked. An example query can be formulated as: “Give me all business policies for Digital Asset Management domain for minors where Business Goal associated belongs to Fulfillment”.

Information retrieved includes a textual explanation of the business policy as well as a control definition (subprocess) which explains how this policy should be included or taken into account in the model being checked. This helps us to ensure the correctness of the designed models. In addition, ontologies can be used to formalize business policies and the derived business rules in order to enable semi-automatic compliance checking of existing process models against relevant policies.

4 CONCLUSION AND FUTURE WORK

In this article, the authors have proposed new methodology extensions which emphasize the application of ontologies, by semantically enhanced
tools and techniques with the aim of reusing business knowledge and therefore creating models of higher quality, speeding up business process modeling while reducing error-proneness of the task. These extensions complement current methodologies, adapting them to the use of semantics in business process management. In the near future, the authors will design a validation plan for these extensions, with the aim of extracting some empirical conclusions about the usage of the proposal within the telecommunication sector.

REFERENCES


Abstract

Current business process modelling tools support neither restricting names nor using ontologies to describe process artefacts. This lack results in creating non-consistent process models which are difficult to understand, compare, evaluate and re-use, etc. Within this article we argue that the Business Functions Ontology (BFO) developed within the SUPER project may be effectively used while modelling processes as a mean for annotating them and thus help to avoid some of the above mentioned problems. We show the BFO structure as well as an example of its practical application within a tool for business process development.

Keywords


1 INTRODUCTION

Business process modelling is the first phase of the Business Process Management (BPM) lifecycle. The main goal of the business process modelling is to produce a business process model depicting and describing the activities that contribute to the production of good or delivery of service taking place in an enterprise. Then, the developed process model needs to be linked to the existing IT infrastructure in order to be executed. This linkage is usually done by the IT department.

One of the main problems in business process modelling is the freedom that the business analysts have to name and describe process artefacts. Current business process modelling tools support neither restricting names nor using ontologies to describe process artefacts. Examples of wrong naming approaches are the following: use of the same term with a different meaning (homonym); use of different terms for the same concept (synonym); use of inappropriate expressions. In addition, different definitions of the same term may be used by various business analysts i.e. process creators. This leads to several problems. One of them is that the modelled business process is fully understandable only to its creator. Furthermore, if multiple roles are involved in the process modelling, they often use terms at different abstraction levels what results in non-consistent process models that are difficult to compare. In addition, if one would like to translate a business process model into another language, for instance from English to German, more problems would appear. Some terms are recognized as synonyms and the proper meaning of homonyms cannot be identified. Also, subtle differences in wording are very hard to translate.

Other problems appear when the modelled business process is passed to another phase within the lifecycle. Business experts and IT experts do not speak the same language, do not share the same concepts of processes, or use the same tools. In consequence, the implementation of the process does not necessarily meet the expectation of business experts. It is so because either the IT department does not understand what should be implemented or the business experts model something that cannot be done using the available IT infrastructure.

In order for enterprises to benefit from business process modelling efforts, business analysts require better support in creating process models. Some of the above mentioned problems could be avoided if a controlled vocabulary recommendation mechanism would be available within the modelling tools. This would enable business analysts and IT experts to achieve consistency in naming of business process artefacts when modelling a process and thus also improve the understandability of the business process model during the implementation phase. However, current business process modelling tools provide only little guidance to the user and, thus, resemble a scratchboard rather than a technical approach. If any guidance is provided at all, it is on a purely syntactical level which does not account for business semantics.
One of the initiatives to support business analysts with predefined set of knowledge during the modelling phase is the Semantic Business Process Management (SBPM) (Hepp et al. 2005) concept developed within the SUPER project. SBPM aims at automation of the business process management life cycle with use of semantics and Semantic Web services technology. The key issue to fulfil this aim is to provide an adequate machine-processable representation of both process structure (control- and dataflow) and the process content description. The process content relates to the enterprise and its environment and therefore must rely on a proper and unambiguous organization description represented in a machine-understandable manner. As the main parts of the process model are activities, therefore, the main ontology needed for description of processes should describe functions performed within an enterprise. It should provide a common, widely acceptable foundation for structuring and defining business functions. This ontology together with the ontologies modelling other parts of an enterprise, i.e. business resources ontology, roles ontology, organizational ontology, strategy ontology, goals and policies ontology, provides complete description of the organizational process space.

The main aim of this paper is to show the Business Functions Ontology (BFO) developed within the SUPER project and how it may be used to annotate processes. We argue that BFO derived based on careful studies of current approaches and classification used by ERP systems may become a commonly accepted terminology to name functions and activities that may be used by business analysts to describe a process. In addition, we show how we can support business process modellers more actively and create high-quality business process models using intuitive and self-explained approaches.

The article is structured as follows. First, the BFO is presented. In the following section, we delve into the sources based on which the BFO was created. Further, we show how the BFO can be applied within the business process modelling phase. The article concludes with some final remarks.

2 BUSINESS FUNCTIONS ONTOLOGY

BFO has been developed in order to provide basics for structuring and defining generic business functions in enterprises and thus to enable a more thorough access to organizational aspects within the modelling and analysis phase. BFO may be also used for classification of processes or process fragments when discovering process fragments for the needs of autocompletion (Markovic 2008). For the needs of the BFO, a business function was defined as “a functional area of an enterprise”. The exemplary functions which are in accordance with this definition are e.g. Human Resources Management, Sales Management, etc.

The discussed ontology consists of two parallel structures: the Function and ActivityOrStep structures. The Function structure gathers top-level concepts which reflect the broadly recognizable business functions within most of enterprises. ActivityOrStep structure is designed at far more detailed level of abstraction. The concepts from both structures are mutually linked by transitive isSubPhaseOf attribute of each of top level ActivityOrStep concepts. The suitability of established connections is ensured by constraining axioms.

The final version of the Function tree consists of 40 concepts, 14 of them are top-level ones and the rest is grouped as their sub-concepts. The top level concepts name coherent and autonomous areas of functionalities. Some of the top level concepts, such as e.g. FinanceManagement, are further divided into sub-functionalities. Figure 1 presents the current Function structure of the BFO.

The ActivityOrStep structure supplements the Function structure. Consequently, it contains most of the ontology concepts (about 920), grouped as sub-concepts of 33 top-level concepts. The isSubPhaseOf attribute connection determines which concept from the Function structure is complemented by a particular group of activities. The attribute is inherited by sub-concepts within the ActivityOrStep tree, therefore assigning the function to a given activity is equivalent to assignment of this function to all the concepts of the activity’s sub-tree.

---

1 www.ip-super.org
2 The Business Functions Ontology can be found at http://www.ip-super.org/ontologies
The current version of BFO is to answer the following competency questions:

- **What are the main functional areas of the enterprise?**
- **How can the areas be further divided?**
- **What activities does the X-business function include?**
- **What kind of business function is an X-business sub-function?**

We decided to use the ontology as a knowledge representation technique as to overcome the conflicts mentioned in the introduction that may occur between various actors. The developed information model expressed using ontology ensures that business analysts use a standard mean to describe attributes of processes, tasks as well as functions. Simultaneously, it ensures that within the next phase of the BPM also IT experts use the right mean to express the information about the services and therefore facilitates matching between various phases of BPM lifecycle.

By modelling certain functions as classes in a Semantic Web language, we can classify them into hierarchies and model additional relations that exist between various concepts. Moreover, the ontological representation of the information is expressive and machine-understandable and can be automatically processed. One of the advantages of using a semantic description language is that one can use machine reasoning in particular class subsumption to bridge different levels of abstraction (as mentioned in the following section) that occur when specifying properties of a process or task. In addition, the ontology is a concise and complete representation technique, flexible and easy to understand. Although machine reasoning is sometimes perceived as not so efficient, the state-of-the-art reasoners are proved to be both efficient and effective (Bishop and Fischer, 2008).

### 3 KNOWLEDGE SOURCES USED TO DEVELOP THE BFO

The purpose of the BFO is to provide a common foundation for structuring and defining business functions. Therefore, while creating the ontology we took inspiration from previous initiatives in this area. The best sources to guide the conceptualization process and modelling decisions turned out to be the currently existing approaches to formalize the process space (REA, TOVE and Context-based Enterprise Ontology). To obtain a source of different types of functions and activities we took advantage of the classification used by ERP systems on the example of the SAP Business Maps.

A concise description of the related work taken into account during the conceptualization phase of the ontology development is presented within this section. First, a short overview of the ontology in the area of business functions is discussed, and then the ERP system being an inspiration to the developed ontology is presented.
While presenting each of these approaches we also mention the differences and similarities between them and the developed BFO.

3.1 Enterprise ontologies

This subsection presents the most important initiatives in the field of business and enterprise ontology modelling. For more detailed description of initiatives in this area please see (Filipowska et al. 2007).

The REA (Resource-Event-Actor) enterprise ontology is based on elements of the early REA model (McCarthy 1982) containing only the concepts of resources, events and agents. The theoretical background of REA comes from the field of microeconomics. All REA concepts and definitions are applied to the collaborative space between enterprises where market exchange occurs among two or more trading partners (Geerts and McCarthy 2000). In REA Enterprise Information Systems the economic activities of enterprises (thus business functions and activities) are represented as a top-down decomposition with three layers: the enterprise scripts, processes and low-level tasks. In this case a business function is not perceived as a subject of separate ontology or domain, but it is embodied into comprehensive REA ontology.

Another work in the field of business ontologies is the TOVE initiative (Fox 1992). The goal of the TOVE project was to develop a set of integrated ontologies for modelling of both commercial and public enterprises. The authors aimed at the creation of shared representation of an enterprise, definition of the semantics and symbols which depict defined concepts along with implementation of the semantics in a set of axioms in order to provide automatic deduction capabilities. In later works of the authors some additional aspects like quality (Kim and Fox 1994), organizations integration (Fox and Gruninger 1994), costs of actions and resources (Tham et al. 1994) were also identified. As a result of works conducted in the TOVE project, two foundational ontologies (Activity, Resource) and four business ontologies (Organization, Product and Requirements, ISO9000 Quality, Activity-based Costing) were developed. We find the granularity of respective ontologies inconsistent, what makes them inoperable to use. In this case also a business function is not perceived as a subject of separate ontology or domain, but it was embodied into comprehensive business ontology.

Another initiative close to our work is the Core Enterprise Ontology (CEO) being an upper level ontology defining enterprise concepts proposed by Bertolazzi et al. In this approach, a specific enterprise ontology is built starting from CEO and proceeding top-down in the refinement and decomposition hierarchies.

Another inspiration comes from the Enterprise Ontology (EO) developed by Uschold et al. EO is a collection of terms and definitions relevant to business enterprises (Uschold et al. 1998). It was developed as part of the Enterprise Project, with the aim to provide a framework for enterprise modelling. EO is divided into five parts: i) terms related to processes and planning, ii) terms related to organisational structure, iii) terms related to high-level planning for an enterprise, iv) terms relating to marketing and sales of goods and services and v) terms used to define the terms of the ontology together with a few terms related to time. It offers comparable levels of details, however, as it was first only informally modelled in a natural language format and only afterwards ported to the semi-formal Ontolingua, its efficient application is hampered.

In 2007, Mauri Leppänen (Leppänen 2007) suggested a contextual approach to ontology development in order to gain high flexibility of the enterprise ontology. A context involves seven domains: purpose, actor, action, object, facility, location, and time. Created context-based enterprise ontology provides a unified view of an enterprise as an aggregation of contexts. This ontology can be specialized into task ontologies or domain ontologies to meet particular needs of enterprises, and still maintain connections of the specialized things to their contexts. Within this approach a business function is defined as an action domain that “comprises all those concepts and constructs that refer to deeds or events in a context” (Leppänen 2007). It scopes from the highly abstract work such as strategic planning to physical execution of step-by-step procedure with detailed routines.

The abovementioned initiatives provided an inspiration and foundation for developing the BFO. The structure of REA and Context-based Enterprise Ontology is the closest to our vision, yet it does not separate vocabulary from business process structure, what we are trying to introduce. BFO follows the understanding of business functions presented in Context-based Enterprise Ontology. However, BFO in contrast to the presented initiatives is much more focused and concentrates only on one part of the information space needed to describe the functional decomposition of coarse-grained business functions to fine-granular activities, i.e. the functional perspective of business processes (Weske 2007). The other parts of the information space are covered by other ontologies (Filipowska et al. 2007) developed within the SUPER project. As a consequence, the BFO reflects its domain in a very detailed manner providing a few layers of details and often dozens of sub-concepts which have one or even none equivalents in other knowledge models. BFO is supposed to be one of the reference ontologies used while developing enterprise-specific functional ontologies. Therefore, our intention is not to provide all means needed for defining activities on a very detailed level (as done in (Leppänen 2007)).
In this context it is worth to notice that the BFO model seems to fit into REA’s three-layer architecture, not duplicating any of those layers but rather constituting the fourth one. The BFO concepts are more general than the low-level layer which in the case of REA resembles the process tasks concepts but they are also different than the REA’s middle layer concepts. The middle layer of the REA model is to describe any given economic process as a kind of resource exchange.

3.2. ERP system categorization - SAP Business Maps

As already mentioned we decided to use SAP Business Maps\(^3\) as an example of ERP system categorization to obtain content for the BFO.

The SAP Business Maps provide information on core processes and functions of an enterprise. They were developed based on several decades of experience gained by SAP consultants. The SAP Business Maps are comprehensive and generic, therefore, may be viewed as a kind of reference model for business processes, both industry-specific and cross industry. Their creation allowed increasing ability of companies to compete, strengthen their relationships with partners, and help enterprises to become more customer-oriented by organizational improvements.

There are two main types of Business Maps, each providing different perspective of the organizations’ processes, namely:

- **Solution Maps** - well-defined tools that outline the scope of an organization’s business. The Solution Maps also show how various processes are covered, including the processes that SAP and its partners support. Solution Maps are divided into two levels: an Overview and Business Blueprint.

- **Business Scenario Maps** - a collection of industry-specific and cross-industry process blueprints. They explain state-of-the-art business processes and, in addition, define the activities, roles, system interfaces, as well as the business documents required for inter-enterprise collaboration.

The above mentioned types sum up to a total of about 410 main documents.

Both types of business maps are diversified when it comes to their scope. This results in distinguishing the following map categories:

- Cross-Industry Business Maps,
- Industry-Specific Business Maps,
- Infrastructure and Services Maps.

It is essential to take into account that SAP Business Maps were designed in order to depict and concentrate on how business may create value for their customers and at the same time how SAP components may support business processes. They are, therefore, the resource containing large amount of semi-structured knowledge on many important as well as irrelevant in our case aspects of an enterprise and its processes. The SAP Business Maps can be viewed, customized, and further developed with the use of dedicated SAP tools.

Recently, on top of the SAP Business Maps the SAP Business Maps Ontology was developed offering a common schema for all entities named throughout all Business Maps. Instances of respective entities were generated in the form of separate ontology importing the schema ontology. Several sources provided input to the development process of the Business Map Ontology. On the one hand information material on the SAP Solution Composer\(^4\) gave a rough overview of the major concepts and their meanings. On the other hand, and even more important, the Business Maps as such were analysed on an instance level. Due to the high degree of flexibility the Solution Composer offers in terms of editing options often merely a close look at the delivered Business Maps to clarify the meaning of a concept and its relations. As almost all Business Maps elements may be interrelated with each other only analysing the delivered instances revealed which interconnections were regarded as meaningful by the Business Maps creators and thus have to be integrated into a ‘common understanding’ of the domain of discourse. Thus, the developed ontology has to be understood as a conceptualisation of the Business Maps in their current state. By virtue of the evolutionary process of map refinement further releases of the Business Maps may contain concepts and relations which are not considered in the ontology presented in the following.


BFO used SAP Business Maps (not SAP Business Maps Ontology as it was developed shortly after BFO) as a primary source of information. Extraction of terms from the SAP Business Maps corpus resulted in 34086 terms. This large set however, was characterized by a huge number of redundancies as well as close synonymy or high level terms similarity. As the terms set was extracted without preserving any structure, the process of abstracting over Business Maps comprised a number of tasks: textual-form terms organization and removal of errors; normalization of terms; identification of concepts’ context; removal of redundancies; atomization and splitting of conjunctions; identification of domain terms and exclusion of non-business-function-related terminology; integration and normalization of terminology from other knowledge sources; conceptualization; iterative restructuring, structure adjustment and concepts matching. The final ontology was obtained as a result of all the above mentioned tasks. After performing all preparatory tasks, a reduction in a number of terms was achieved. The potential concepts were further filtered with regard to intended granularities on different levels of abstraction.

Table 1 represents relations between SAP Business Maps Ontology and BFO. It should be highlighted that in this case one cannot use the term “mapping” between these two structures. Mapping assumes the simple subsumption relation between any two concepts and deals always with a given set of concepts limited to one on at least one side of the mapping relation. In Table 1 we point the thematic areas (sets of concepts) rather than observing strictly any single concept. Consequently, it has to be taken into account that some major changes to the SAP Business Maps were introduced before including them into the BFO structure (as mentioned in the already presented list of the development tasks).

<table>
<thead>
<tr>
<th>Source</th>
<th>Terms group</th>
<th>BFO sub-tree</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP Business Maps</td>
<td>Scenario Group</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>SAP Business Maps</td>
<td>Value Chain Element</td>
<td>Function</td>
<td>Supplementary</td>
</tr>
<tr>
<td>SAP Business Maps</td>
<td>Business Process</td>
<td>Function</td>
<td>Supplementary</td>
</tr>
<tr>
<td>Managerial Literature</td>
<td></td>
<td>Function</td>
<td>Initial structure</td>
</tr>
<tr>
<td>Management Handbooks</td>
<td></td>
<td>Function</td>
<td>Initial structure</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td>Function</td>
<td>Analysis of exemplary enterprises</td>
</tr>
<tr>
<td>SAP Business Maps</td>
<td>Activities</td>
<td>ActivityOrStep</td>
<td>Main source</td>
</tr>
<tr>
<td>SAP Business Maps</td>
<td>Process Steps</td>
<td>ActivityOrStep</td>
<td>Additional source (not fully processed)</td>
</tr>
<tr>
<td>SAP Business Maps</td>
<td>Process</td>
<td>ActivityOrStep</td>
<td>Additional source (not fully processed)</td>
</tr>
</tbody>
</table>

From Table 1 one can see that the ActivityOrStep structure was created on the basis of three groups of terms. The entities within those groups (2817 Activities; 6720 ProcessSteps; 8925 Processes) were selected as candidate terms for lower level vocabulary based on the careful scrutinize and expert evaluation. After removal of defects of the entry Business Maps terms set the number of potential concepts in those groups was reduced to more than 3000 terms.

The Function structure as shown in the Table 1 is composed of concepts that reflect mainly the Scenario Group entities from the SAP Business Maps integrated with knowledge from a number of other knowledge sources (handbooks, literature overview, other research results) and combined with the ontology developers’ analysis results done on the basis of exemplary enterprises and their functional organization layer.

Figure 2 gives a presentation of correspondence of granularity level between specified parts of the SAP Business Maps versus BFO. The figure provides only a general visualization. This means that not all elements were necessarily transformed from SAP Business Maps into indicated places in the BFO hierarchy and target BFO areas might contain a limited number of additional concepts of other origin.
As it can be seen in Figure 2 the grouping and fine-tuning of entities represented by different parts of the SAP Business Maps ontology has been quite complicated especially taking into account that the intermediate and bottom levels in both structures of BFO are internally multi-level layers. In order to achieve the best effects and make it possible to handle this amount of entities, a special web system for collaborative ontology concepts alignment was created and deployed. The use of the system resulted in a swift pace of the carried work combined with high quality of the achieved outcomes.

4 APPLICATION IN SEMANTIC BUSINESS PROCESS MODELING

Current business process modelling tools support neither restricting names nor using ontologies to describe process artefacts. Our approach incorporates ontology-based different matchmaking functionalities which are required to bridge the gap between the business process model and the semantic world, in order to allow for description of a graphical business process model with BFO instances.

Utilizing the BFO, the matchmaking functionalities address the problem of deriving a list of proposals for a selected model element that a user has chosen for semantic refinement, based on the previously specified business function for a process or previous task in a process. To solve this problem, we use a combination of different text and name matching methods and process diagram context information. The utilization of the diagram context information of the selected model elements and the BFO knowledge helps to match the model fragments with BFO elements. Therefore, we can derive a list of element proposal using BFO elements which are already specified in the process model.

For instance, let us assume that a modeller has already created an activity “Create contract offer” which is linked to the BFO concept CreateContractOffer. Based on this information we can derive the process context and know that he wants to model a business process for ContractProcessing, as this is the parent concept of CreateContractOffer. Now, using the ContractProcessing concept of the BFO, the system knows which subsconcepts (e.g. UpdateContract, ReviseContract, etc.) are available for this function. Hence, once a modeller starts creating new activities, our recommendation engine provides him with a list of valid names based on the BFO concept hierarchy. Figure 3 sketches the user interface of our prototype. Furthermore, the BFO is linked to the Business Resources Ontology (Filipowska et al. 2007) through the relation requiresAsResource, which relates a process activity and a business resource required for performing that activity. We utilize this link to propose a usage of appropriate resources for the selected process activity. Figure 3 shows that the concept CreateContractOffer (process activity) of the BFO requires as resource the concept ContractOffer of the Business Resource Ontology. Therefore, the resource ContractOffer is suggested to the modeller.
The matching can lead to even better results when using name matching functionalities, where we use a combination of heuristic comparison methods on the strings of characters, well-known string distance metrics (Cohen et al. 2003, Chapman et al. 2005), and matching methods considering synonyms and homonyms. The work concerning name-based matchmaking is related to schema matching efforts. A survey of (Rahm and Bernstein 2001) presents a nice overview of this research area. However, the fundamental difference is that schema matching tries to map between two formalized schemas (e.g. defined in XML), whereas the matching problem addressed in this paper is to find appropriate BFO elements given a free-text name entered by the user and context information derived from the process model, concerning the selected element. Nevertheless, the name matching tasks are similar for the two approaches. A comparison of (Cohen et al. 2003) describes different string distance metrics, some of which are utilized for name matching tasks in this work. Therefore, these established, general approaches for name matching tasks are reused to some extent and are enhanced with new (context-related) matching functionalities.

Using such matchmaking functionalities the engine can reduce the list of possible activities. Figure 4 continues the given example. Once a modeller starts typing the letters re for naming the process activity, the system will only suggest all concepts related to the string re which are subconcepts of ContractProcessing (cf. Figure 4, right).
BFO concepts (the *Function* structure) are also used for annotating complete process models, parts of them (process fragments) or best practices (process patterns). This helps the modellers in retrieving (discovering) existing process models which can be re-used in their design. In our example, the modeller can query for all process models and best practices in the *ContractProcessing* function, in order to re-use and/or adapt previous work. With such support, process modellers are able to create higher quality models in less time. An even more advanced way of supporting the modellers can be realized through the autocompletion mechanism, where the modeller receives suggestions from the modelling tool regarding the subsequent process activities or process fragments to be added in the model, based on the part already modelled.

5 CONCLUSIONS AND FUTURE WORK

This article elaborates on the Business Functions Ontology, modelling the functional perspective of an enterprise. BFO was developed bearing in mind experience of developers of previous terminologies for description of processes as well as SAP Business Maps. We have shown how BFO concepts may be used for consistent naming of process activities and suggesting appropriate resources in process modelling, which leads to more readable, consistent and higher quality process models. We also present how BFO may be used for the description of processes, what would enable business analysts to e.g. re-use already modelled process artefacts or benefit from the autocompletion mechanism suggesting functions that may complete the process being modelled. Our current work focuses on the full integration of our concepts within the SAP Research “Maestro for BPMN” process modelling tool. In general, it is not easy to measure the quality of business process models as they usually reflect the perspective of one person and are based on individual thoughts and feelings. Nevertheless, in (Becker et al. 2003) the authors have developed the “Guidelines for Business Process Modelling” which are the basis for our evaluation. We plan to conduct a case study using the “Maestro for BPMN” tool and complement this by using surveys.

REFERENCES


---

```xml
wsmlVariant = "http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace {
    wsmo _ "http://www.ip-super.org/ontologies/ContractProcessing#",
    wsmoStudio _ "http://www.wsmo.org",
    bpmo _ "http://www.ip-super.org/ontologies/BPMO/20070529#",
    upo _ "http://www.ip-super.org/ontologies/UPO/20070529#",
    bfo _ "http://www.ip-super.org/ontologies/BFO/20070731#",
    bront _ "http://www.ip-super.org/ontologies/BROnt/20070801#",
    bro _ "http://www.ip-super.org/ontologies/BRO#",
}

ontology _ "http://www.ip-super.org/ontologies/ContractProcessing#"
    // (…)

instance ContractProcessingProcess memberOf bpmo#Process
    bpmo#hasName hasValue "ContractProcessingProcess"
    bpmo#hasBusinessFunction hasValue bfo#ContractProcessing
    bpmo#hasWorkflow hasValue ContractProcessingWorkflow
    bpmo#hasBusinessGoal hasValue goals#contractProcessed

instance CreateContractOffer memberOf bpmo#Task
    bpmo#hasHomeProcess hasValue ProductDeliveryProcess
    // (…)
```


ACKNOWLEDGEMENTS

This work has been partially supported by the European Union within the FP6 IST project SUPER (FP6-026850).

COPYRIGHT

[Matthias Born, Agata Filipowska, Monika Kaczmarek, Ivan Markovic, Monika Starzecka, Adam Walczak] © 2008. The authors assign to ACIS and educational and non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The authors also grant a non-exclusive licence to ACIS to publish this document in full in the Conference Papers and Proceedings. Those documents may be published on the World Wide Web, CD-ROM, in printed form, and on mirror sites on the World Wide Web. Any other usage is prohibited without the express permission of the authors.

Ivan Markovic
SAP Research
Vincenz-Priessnitz-Str. 1,
76131 Karlsruhe, Germany
ivan.markovic@sap.com

Marek Kowalkiewicz
SAP Research
133 Mary street
Brisbane, QLD 4000, Australia
marek.kowalkiewicz@sap.com

Abstract

A broad knowledge is taken into account when a business process is modeled by a business analyst. We argue that existing Business Process Management (BPM) methodologies do not consider business goals at the appropriate level of abstraction in business process modeling. In this paper we present an approach to integrate business goals and business process models. We design a Business Goal Ontology for modeling business goals and provide a business goal modeling notation. Furthermore, we devise a modeling pattern for linking the goals to process models and show how the ontology can be used in query answering. In this way, we integrate the intentional perspective into our business process ontology framework, enriching the process description and enabling new types of business process analysis.

1 Introduction

Conceptual modeling aims at representing the knowledge in a particular domain of discourse in order to understand, communicate and reason about the design of an information system. Ontologies are computational artifacts that represent conceptual models of some domain of interest to be used within information systems. Ontologies are a special form of conceptual models, whose primary role is not to serve documentation but to be machine-readable, such that computers can "reason" about the model. The dominant definition of an ontology is given by Gruber [11]: "An ontology is a formal specification of a shared conceptualization of a domain of interest". In the recent years, ontologies have been used in Business Process Management (BPM) in order to address the limited degree of automation in BPM and to provide a unified view on the process space of an organization, yielding a new area of research named Semantic Business Process Management [12].

Semantic BPM approach has already shown added value in all phases of the BPM lifecycle [22, 32, 33, 6]. In this work, we focus on the semantic business process modeling phase and investigate how ontologies can provide additional support in the design and validation of conceptual business process models. Following [17], we consider here the intentional aspect of process model design, i.e. how business goals influence the creation of process models.

Human action is primarily driven by goals [28]. A goal is a desired state of affairs that needs to be attained [14]. Business goals express what the organization wants to achieve from the business perspective. They may be formulated at different level of abstraction - as short term, targeted milestones, or as general, longer term aims and visions. Goals must be measurable in order to track progress. The measure can be defined in quantifiable terms such as profit, time, or volume, or it can be defined in qualitative terms such as "hire top talent". In order to allow us to study organizations from the intentional point of view, business goals need to be captured explicitly [33]. In the business process modeling community, little attention is paid to the value of making goals explicit and incorporating the notion of goal into process modeling methods [29]. Instead, goals are seen as external concepts not integrated with process models. At the same time, in the management communities, specifically in the area of strategic management (organizational studies and human resource management), as well as in project management, goals are considered to be an important element.

Goals, also referred to as objectives, are a central element of Management by Objectives (MBO), a con-
cept popularized by Peter Drucker in 1954 [8]. MBO introduced the SMART criteria, where each goal has to be specified in a way that is Specific, Measurable, Achievable, Relevant, and Time bound [1]. In some areas SMART is replaced by SMARTER, where ER stands for Extendable and Recorded.

The aim of this paper is to enable formal modeling of business goals and to provide a link between business goal specification and process model design. First, we design a richly axiomatized Business Goal Ontology (BGO) and propose a business goal modeling notation for creating goal models. Further, we integrate the BGO into our business process ontology framework [19] and thus allow for more rich semantic annotation of process models. In this way, we enable a more intuitive approach to business process design and lay the foundations for new types of process model analysis.

The remainder of the paper is structured as follows. In Section 2, we briefly present the extended semantic business process modeling lifecycle. Section 3 provides an overview of our ontology framework for modeling business processes. In Section 4 we present the constructs for modeling business goals. Section 5 discusses the Business Goal Ontology in detail. In Section 6 we explain the linking of business goals to process models, illustrate an example scenario and list new types of queries that can be answered with the provided linkage. Section 7 gives an overview of related work and Section 8 concludes the paper.

## 2 Semantic Business Process Modeling Lifecycle

The semantic business process (BP) modeling lifecycle, presented in Fig. 1 is based on the spiral lifecycle model first proposed by Boehm [2], which is widely adopted by BPM community due to the dynamic nature of BPs and their consequent redesign cycle. Phases, depicted as the inner ring, and sub-phases (next ring), are based on current BP modeling and software engineering lifecycles [26]. The major contribution of this life cycle comes from the application of ontologies, semantic tools and techniques, depicted in the third ring of the picture by a triangle, a hammer and a hand respectively. In the following subsections we briefly present each of the phases and sub-phases of the life cycle. For more details we refer the reader to [5].

### 2.1 Requirements Analysis Phase

During the Requirements Analysis phase (cf. Fig. 1 top right), business analysts collect all the relevant information for the BP design and further implementation. This information comes from various sources (e.g. the company’s strategy, marketing and product departments, final customers, partners, etc.), and has to be collected and documented by analysts. Here we differentiate between two sub-phases: i) Requirements Capture sub-phase, where a business analyst collects all the relevant information for a BP design and extracts requirements for the BP and ii) Requirements Documentation phase involves the documentation of requirements captured in the requirement analysis. It is comprised of the requirements classification to support their traceability, as well as a refinement of the requirements specification, defining the quantitative and qualitative goals to be covered by the BPs.

![Figure 1: Semantic business process modeling lifecycle](image)
of this sub-phase is the refinement of business ontologies according to the specific scenario.

Patterns Modeling. BP patterns are abstract BPs which are not fully defined in terms of concrete tasks and services, and thus are not executable. These patterns capture the first draft of a process, making emphasis on the business goals and leaving the concrete definition of processes to a further step. They represent a solution to a well known problem, and a base to enable the extension to a concrete problem (as software design patterns [9] do), providing best modeling practices.

The first step in defining a business process is to define the objectives to achieve. This is commonly achieved through the definition of business process patterns, which can be defined from previous patterns or from scratch. BP patterns are implicitly used by business analysts when they want to define what a process should achieve without defining how this will be done. The patterns define high level goals (or milestones) and a simple workflow which connects them. Patterns are annotated with the context of their usage which is comprised of a business function, a business goal and a business domain. These annotations allow us to query for patterns in the early stages of modeling in order to provide modeling guidance.

Goal Refinement and KPIs Modeling. After the definition of the objectives which the BP being modeled has to accomplish, a business analyst usually refines those objectives in terms of business goals at the operational level (more concrete than the ones defined for the pattern), and assigns them to corresponding key performance indicators (KPIs). KPIs define the metrics used to monitor and measure the performance of business processes during execution time. Business goals and KPIs can be reused from previous models. The output of this phase will be the formalization of both business goals and KPIs, annotated according to the ontologies defined in the ontology modeling sub-phase.

Process Modeling. Once business process objectives are clear, and the metrics used to measure its performance defined, the process is defined. Tasks and workflow can be defined by either i) deriving from previous BP patterns, ii) extending or refining previous BP models, iii) editing directly, or iv) any combination of the previous options, since BP modeling is often made in several steps which can combine previous approaches. Once the process is defined, it is semantically annotated according to the ontologies defined in the ontology modeling sub-phase.

2.3 Validation Phase

Validation of BP models (cf. Fig. 1, top left) is a crucial phase in which it is ensured that the BP designed effectively covers the requirements captured in the first phase from both a functional and business perspective. Besides, quality of the model must be checked, ensuring that the model designed will derive in an executable process with high performance, and that the model is conceptually correct.

Metric-based Model Validation. Within this sub-phase the model quality is validated according to a set of metrics. These metrics heavily depend on the company and the continuous evaluation and improvement model they are following. Metrics can include complexity of the model, structure of the model, modularization, or cognition [31].

Goal-based Model Validation. Here we utilize the formal business goal model presented in this work in order to detect conflicts, redundancies, perform queries and gap analysis on process models. In this way we i) make sure that the conceptual business process models are consistent with the organizational business goals and ii) enable top-down traceability and bottom up linkage from processes to business goals.

Policy-based Model Validation. Real BPs do not only have to accomplish certain functional requirements, but also are influenced by business policies and rules which are derived from the company strategy. Business policies orthogonally apply to all business processes of the company and can be mandatory or optional. Mandatory policies must be correctly addressed in BP models, while conditional ones are suggestions to improve or complement the model which can or can not be taken in consideration. Business rules affect the execution of the process, defining conditions which drive the execution of the workflow. Checking both elements usually provide new requirements for later modeling iterations.

3 Business Process Ontology Framework

In order to enable semantic analysis, expressive querying and validation of BP models in different phases of the SBP modeling lifecycle, there is a need for a comprehensive formal process model description capturing all relevant dimensions (perspectives) of a process. Following [1] and [13], we consider the functional, behavioural, organizational, informational and intentional perspective relevant to adequately organize information about a process. In our previous work [20]...
we have proposed a formal model for describing business processes which integrates all aforementioned perspectives, as depicted in Fig. 2. In the following we are briefly describing these perspectives.

Figure 2: Ontology framework: perspectives for describing a business process

For describing the behavioural (dynamic) perspective of a process model we use a process algebra, the π-calculus. By using the π-calculus for representing the process behaviour, we are also able to integrate existing tools and techniques for verification and simulation of processes in our framework. The dynamic perspective of a process model stands for process control- and dataflow, and we model it using the ontologized π-calculus, denoted by Business Process Ontology in Figure 2. For more details on this ontology, we refer the reader to [20].

For representing the functional, organizational and informational perspective we have proposed a set of ontologies, imported by Business Process Ontology, as shown in Figure 2. Business Functions Ontology models the functional perspective and provides a structural breakdown of the organization’s business functions. Concepts from this ontology classify process models by their functionality, independent of the business domain. Organizational Ontology addresses the organizational perspective and includes concepts representing organizational units, e.g. Cost Center, Distribution Channel, and roles in the organization, e.g. Manager, Engineer, Clerk, Secretary, etc. Process Resources Ontology covers the informational perspective and describes the resources (documents, systems, machines) which are required to operate the activities in processes. In this work, we introduce an additional, intentional perspective on business processes in our formal model by designing a Business Goal Ontology. In the following section, we present a set of constructs for modeling business goals.

4 Business Goal Modeling Notation

In our work we define business goal as the state of the enterprise that an action, or a given set of actions, is intended to achieve. The goal, when achieved, terminates actions that were intending to achieve it. According to the SMART methodology, goals may have additional properties, such as deadline, assigned resources, a description allowing to assess whether the goal has been achieved or not. If the description specifies quantitative measures for assessing goal achievement, an automated function can be used to perform the assessment.

Goals may be quantitative or qualitative. Quantitative goals are specified in a way that allows devising (measuring) the state of the goal by comparing values stated in the goal that express desired state of the world and the actual state. Qualitative goals are using textual descriptions of the desired state, and automated verification of the state of the goal is not easy. Quantitative goals need to have measures and threshold values defined, qualitative goals do not have to have such properties provided.

Modeling constructs for creating business goal models are depicted in Fig. 3. Business goal models contain a hierarchy of an organization’s business goals according to which the processes in the organization are designed. The higher level goals are refined into more specific goals through AND or OR-decomposition, where the goals are decomposed into alternative subgoals (OR) or combinations of subgoals that must be satisfied (AND). Such decomposition occurs e.g. when goals created by one level of management are handed down to a lower organizational level for more detailed planning or implementation. The goal refinement tree provides traceability links from high-level strategic goals to low-level operational objectives. The relation subgoal_of together with the constructs for AND/OR goal decomposition is used when creating the goal hierarchy (cf. Fig. 3 lower right). An example of such a hierarchy is given in Fig. 3.

Any goal at each refinement level describes what needs be done. At the same time this goal can also be considered as an end (why) for a refined goal, as well as means (how) for a goal at a higher level (cf. Fig. 3).

Business goal can have a more detailed description, it has an assigned measure for controlling the progress, deadline for completion and priority. A goal which is achieved has its corresponding attribute value set to true. Fig. 4 summarizes the types of properties which could be assigned to business goals:

Business goals can conflict with each other. More-
over, goals can influence positively (support) or negatively (hinder) other goals [14]. Our modeling notation provides three types of relations for representing such situations: supports, hinders and conflicts (cf. Fig. 5). Support and hindrance may be considered to have either binary values (a subgoal either supports the goal or it does not support it at all, represented by lack of support relation) or continuous values (one subgoal can support the goal more than another subgoal).

![Figure 5: Business goal modeling constructs](image)

In the following section, we present the formal semantics of the concepts, properties and relations defined in the business goal ontology.

5 Business Goal Ontology

Based on a series of interviews with our target user group (executives, managers, business consultants), we have identified a set of requirements which the designed ontology is intended to meet. In the following we give some categories of requirements supported by our ontology.

- **Req. 1: Identify and model relevant concepts, properties and relations** Provide a standard set of concepts, properties and relations to be considered as relevant factors in defining business goal models.
- **Req. 2: Business goal decomposition** Make sure that decomposition of business goals results in a consistent hierarchy. Are there redundant goals generated? Is a goal an explicit requirement of the stakeholder?
- **Req. 3: Provide a generic, reusable and flexible model** Having a common, generic basis is beneficial because it requires modeling of fundamental concepts only once. The constructs of the model should be domain independent.
- **Req. 4: Possibility to identify conflicts** Find conflicting goals to identify contradictions in goal specifications.
- **Req. 5: Providing guidance in modeling** Processes are defined as operationalizations of business goals. The purpose of a process is the achievement of one or more goals. Enable a goal-driven process design methodology.
- **Req. 6: Enabling faster reaction to changes in the environment** Utilizing model relations to determine which business goals are affected by the changes. Utilize model relations to determine what is the extent of the influence.

5.1 Ontology design

Before proposing a design of goal ontology, the concept of goal needs to be formalized. Below we provide a definition of a goal. When compared with BMM specification [24], our definition covers objectives too. Considering the goals of the work, a distinction between goals and objectives does not have to be made here. However, it can be easily introduced further on.
Definition 1 A goal expresses an attainable, measurable and time-bound target that should be achieved or sustained. Attainability means that there is a possibility that the goal is achieved. Measurability means that there is a way to measure, whether the goal has been achieved or not. Time-boundedness means that there is a deadline for achieving the goal. Formally, we say that $G(description, measure, deadline, priority, achieved)$ is a goal, where:

- description is a human understandable description of the goal,
- measure is an attribute, or a set of attributes that is used to express the advancement of the goal,
- deadline is an expression of the latest time for achieving the goal,
- priority is an optional value allowing ranking of different goals,
- achieved is a threshold value of measure that expresses whether the goal has been achieved or not.

Depending on the time horizon, the goals may be classified as operational or strategic. A strategic goal tends to be longer term and defined qualitatively rather than quantitatively. An operational goal is a short-term contribution to a strategic goal. Operational goals provide the basis for measuring the progress toward meeting strategic goals [24].

Quantitative goals need to have measure and achieved specified. Qualitative goals do not have these properties, as they are difficult to describe in measurable terms and human judgement is needed to verify their achievement. Qualitative goals must have a description of the goal and optionally an assigned priority.

Further in this section we describe the set of concepts and their properties, relations and constraints which constitute the formal model for representation of business goals of an enterprise. The definitions of concepts, attributes and relations are specified in first-order logic.

The hierarchy of goals is built using the relation subgoal_of. The relation subgoal_of is transitive, i.e. if a goal $g_1$ is a subgoal of a goal $g_2$ which is a subgoal of a goal $g_3$, then $g_1$ is a subgoal of $g_3$:

$$(\forall g_1, g_2, g_3) \text{subgoal}_of(g_1, g_2) \land \text{subgoal}_of(g_2, g_3) \rightarrow \text{subgoal}_of(g_1, g_3)$$

The relation subgoal_of is also non-reflexive (a goal $g$ cannot be its own subgoal) and anti-symmetric (if $g_1$ is a subgoal of $g_2$, then $g_2$ is not a subgoal of $g_1$), as defined in the following statements:

$$\forall g \neg \text{subgoal}_of(g, g)$$

$$\forall g_1, g_2 \text{subgoal}_of(g_1, g_2) \rightarrow \neg \text{subgoal}_of(g_2, g_1)$$

As indicated before, decomposition of goals into subgoals can be done through an AND- or OR-decomposition [10]. If a goal $g$ is AND-decomposed (OR-decomposed) into subgoals $g_1,...,g_n$, then if all (at least one) of the subgoals are satisfied, so is goal $g$. An atomic goal is a goal that can not be further divided into subgoals. This type of goals occurs at the lowest level of the goal decomposition hierarchy. The fact that an atomic goal can not have subgoals can be axiomatized in the following way:

$$(\forall g) \text{atomic}_goal(g) \equiv \neg (\exists g') \text{subgoal}_of(g', g)$$

An important notion to keep in mind when talking about goals is time - goals have to be achieved in a certain time period (deadline). Consider a relation $is\text{Achieved}(g, t)$ which states that goal $g$ has been achieved in time $t$. In the case of AND-decomposition of a goal $g$ into subgoals $g_1,...,g_n$, the following must hold:

$$is\text{Achieved}(g, t) \equiv is\text{Achieved}(g_1, t) \land ... \land is\text{Achieved}(g_n, t)$$

Similar holds for the OR-decomposition. The content of each goal can be described using a logical expression. Let $goal(g)$ denote the logical expression of goal $g$. In order to detect goal redundancies, we introduce the predicate subsumes which is defined in the following way:

$$(\forall g_1, g_2) \text{subsumes}(g_1, g_2) \equiv goal(g_1) \rightarrow goal(g_2)$$

Here the goal $g_1$ contains $g_2$. If goals $g_1$ and $g_2$ are both direct subgoals of a goal $g$, then the goal $g_2$ is redundant. Goals can have priorities, which allows us to order the goal set according to their importance. Predicates $priority_{ht}$ (higher than), $priority_{eq}$ (equal to), $priority_{lt}$ (less than) are used to denote priority constraint between two goals. The following axiom must hold:

$$(\forall g_1, g_2) \text{priority}_{ht}(g_1, g_2) \rightarrow \text{priority}_{lt}(g_2, g_1)$$

Prioritization focuses attention on key areas, while allowing the modeler to describe goals which are currently perceived as less important or out of scope. Analysts can choose to show only those goals that are in
scope. This can be achieved visually by filtering on the level of priority. The goal hierarchy approach thus makes clear the effects of scoping decisions, and allows trade-offs to be evaluated.

You can only control what you can measure [7]. In order to control the progress of achieving goals, we assign a Measure to each quantitative goal. Each Measure has a defined Unit. Measure has a Current Value in the observed moment and a Desired Value which should be reached by a goal (achieved in goal definition above).

Additional goal influencing (support/hinder) relation types can exist. A support relationship between goals $g_1$ and $g_2$ suggests that achievement of goal $g_2$ assists achievement of goal $g_1$; however achievement of goal $g_2$ is not a necessary condition for achievement of goal $g_1$, or else goal $g_2$ would be a successor goal of goal $g_1$. On the other hand, a hinders relationship between goals $g_1$ and $g_2$ suggests that achievement of goal $g_1$ negatively influences (hinders) achievement of goal $g_2$. Relations supports and hinders are transitive, non-reflexive and anti-symmetric. For the sake of brevity we omit these axiomatizations. Goals can also conflict with each other. We formalize the notion of conflict in the following way:

$$(\forall g_1, g_2) \text{ hinders}(g_1, g_2) \land \text{ priority}_\text{lt}(g_1, g_2) \rightarrow \text{ conflicts}(g_1, g_2)$$

If a goal $g_1$ hinders a goal of higher priority $g_2$, goals $g_1$ and $g_2$ are in conflict.

6 Linking Business Goals to Process Models

A business process exists for a reason - it strives to achieve a set business goal [25]. The more clearly the goal is formulated, the easier is to design a corresponding business process that achieves the goal.

In Fig. 6 we show how we envision the relation between the business goal model, process models and their corresponding IT implementation. As a first step, the higher-level goals are broken down into sub-goals, where the sub-goals can be more concrete and easily assigned to a business process (AND/OR business goal decomposition). Based on the refined business goal specification, the business analyst can assign a group of operational level goals to be realized by a business process. The selected set of goals serves as a first draft of a high level process, the business process pattern. In the next step, each of the activities in the business process pattern is refined with a subprocess which realizes a given goal. After each of the activities are refined, the business process model is completed. Each business process model can be assigned to a corresponding business process pattern. By explicitly linking the business goal model to business process models, we are able to perform advanced analysis on process models, such as detection of conflicts and redundancies in processes. Furthermore, this is very helpful when e.g. we want to know which processes contribute to a particular business goal, or which processes need to be changed because a certain goal has been modified. Moreover, we can detect if a process does not have an assigned business goal (gap analysis).

From an annotated conceptual business process model we can derive its implementation by applying techniques of semantic Web service discovery, composition and validation described in [32]. We are now able to propagate the link from business goal model through process models to the implementation level services (cf. Fig. 6 right). Additionally, we can utilize the link to trace the business requirements (goals) from the IT artifacts (cf. Fig. 6 left).

![Figure 6: Linking different abstraction layers](image)

Figure 6: Linking different abstraction layers

Within our ontology framework (cf. Fig. 2), the semantic annotation of processes is done by four defined relations (shown in Fig. 7): hasBusinessGoal, hasBusinessFunction, hasBusinessRole and hasProcessResource. These relations group pairwise a process or a process fragment and respectively Business Goal, Business Function, Business Role and Process Resource. The semantic annotations can be used for querying and different types of process model analysis.

![Figure 7: Semantic annotation of processes](image)

Figure 7: Semantic annotation of processes
Every business process needs to contribute to at least one business process goal. This constraint is used in gap analysis and improves traceability between goal specifications and business process models.

In order to allow for specifying an orchestration of goals as a first step in defining a process model, we introduce **Goal Activity** - a new type of process activity with an assigned business goal annotation (denoted by a circle in Fig. 8):

![Figure 8: Goal activity](image)

### 6.1 Example Scenario

This section provides an example of goal-based business process modeling in the telecommunications domain, on the example of Digital Asset Management scenario. The aim of this scenario is the provision of digital content to end users.

When designing a new process the business expert can first query for existing business process patterns, generic high level process designs emphasizing business goals [21], in search for the best modeling practices in the given domain. The purpose of the formalization of these high-level models in our methodology is to provide best practices for other business analysts and a common base for the specification of BPs which share a same sub-domain (e.g. all BP of a given product family share common features in their design).

An example business process pattern for digital content provisioning is presented in Fig. 9. It is composed of six goal activities with the assigned business goals that mark different milestones of the end-to-end process (Order Registered, Order Provisioned, etc.). We assume that each activity in an organization is contributing to a goal, therefore only goal activities are modeled in goal-based business process patterns.

The goal-based model is further refined for each specific goal. In Fig. 10 we show a subprocess for order provisioning, obtained as a refinement of the goal activity **Provision Order**.

Each of the goal activities can be refined in a similar manner, resulting in a complete process model. Every subprocess preserves its business goal annotation (cf. Fig. 10 bottom) which allows for improved linkage and traceability between high-level goals and conceptual process models.

### 6.2 Example Queries

Introducing business goal models enables new possibilities in process analysis. Below we provide a number of examples that demonstrate the additional value.

- **Query 1**: Show me all IT implementations that support a specific goal
- **Query 2**: Show me the goal that is supported by the specific implementation
- **Query 3**: Show me all goals that are not completely specified (quantitative goals without given measure and achieved values)
- **Query 4**: Filter goals on the basis of a given deadline and/or priority
- **Query 5**: Show me all goals that have no business process model (and ultimately IT implementation) linked
- **Query 6**: Show me all running/modeled processes that do not support any goal
- **Query 7**: Show me all IT implementations that do not support any goal
- **Query 8**: Show me all processes that hinder the achievement of a specific goal
- **Query 9**: Show me all conflicting/redundant goals

All the aforementioned queries can be answered using the properties and relations defined in the Business Goal Ontology while utilizing the link between business goal and process models discussed earlier in this section.

### 7 Related Work

In [18] the authors present an approach for goal annotation of process models in order to facilitate semantic discovery and reuse of heterogeneous process models. Goal ontology in [18] is modeled rather as a taxonomy of concepts, which in our opinion does not justify use of ontologies. We provide rich axiomatization of the business goal ontology and thus enable different types of automated analysis on business process models. In addition, goals are only seen as semantic enrichment of process knowledge, whereas we see goal specification as an important prerequisite for the design of intention-driven conceptual process models.

In [16], a methodology for relating business process models to high-level stakeholder goals (modeled
using KAOS) is presented. The methodology establishes traceability and satisfaction links between goals and process models in order to identify the satisfaction between the two models. In contrast to ours, the proposed approach is informal (manual) and thus cannot automate any step in establishing and verifying the relationship between goals and processes.

In [14], the authors present the EKD - Enterprise Knowledge Development method for systematic analysis, understanding, developing and documenting an enterprise and its components, by using Enterprise Modeling. The Enterprise Model contains a number of interrelated sub-models, each of them representing some aspect of the enterprise: Goals Model, Business Rule Model, Concepts Model, Business Process Model, etc. The EKD Goal Model defines relevant concepts, properties and relations relevant to goal modeling. However, this model does not provide formal semantics of the relations, which makes it impossible to perform automated analysis on goal model specifications.

8 Conclusions

Formal modeling of goals allows us to perform automated analysis on goal specifications, i.e. identify inconsistencies, redundancies, conflicts. In addition, we can follow the dependencies (e.g. identifying which goals support or hinder the observed one) in the goal model. Formal modeling also allows for improved traceability when changes in the environment occur. By integrating business goals ontology with a process modeling framework, we are able to perform this type of analysis on the process models as well. In addition, the proposed approach enables a more intuitive process modeling approach, starting from high-level goals, through operational goals to underspecified, abstract process models and their later refinement into detailed processes. As we have shown in the example, the proposed approach fulfills the requirements stated in section 5.

Acknowledgments

This work has been partially supported by the European Union within the FP6 IST project SUPER (FP6-026850).

References


