Adaptation of Process Models – A Semantic-based Approach

Thomas Eisenbarth

Programming Distributed Systems Lab
University of Augsburg
Augsburg, Germany
eisenbarth@ds-lab.org

Florian Lautenbacher

Programming Distributed Systems Lab
University of Augsburg
Augsburg, Germany
lautenbacher@ds-lab.org

Bernhard Bauer

Programming Distributed Systems Lab
University of Augsburg
Augsburg, Germany
bauer@ds-lab.org

Semantic Business Process Management is an emerging research area to support enterprises achieving economic and strategic objectives and improving their daily business. However, the magnitude of changes which are required for models that capture the business processes is a challenge today. The necessity to change process models in order to stay synchronized with the reality is due to changing requirements within enterprises or external events like mergers and acquisitions, changing laws and reactions to changing markets. The approach we describe in this paper shows a way to automatically adapt existing process models when parts of a process have changed. We apply semantic technologies to the automatic planning approach for the adaptation.

Categories and Subject Descriptors: D.2.2 (Software Engineering): Design Tools and Techniques; D.2.13 (Software Engineering): Reusable Software – Reuse models; H.4.1 (Information Systems): Automation – Workflow Management

1. INTRODUCTION

Business Process Management (BPM) has been one of the main topics in commercial information technology for many years and is becoming even more important today. Formally defined process models establish the basis for automatic execution of processes in enterprises. This becomes increasingly important in business competition because of the possibility to gain shorter time to market, increased customer satisfaction, changing laws and so on. But the graphical modeling of business processes and their execution in software requires so much human work that the software life-cycles can hardly comply with the fast changing demands of today's global markets. Even though techniques like model transformation enable the generation of low level implementation

Manuscript received: 28 February 2010
Communicating Editor: Georg Grossmann

code, it is common to change only the implementation level. Process models get outdated this way and become worthless after a while.

These demands require that processes in a company as well as between several companies need to be adapted frequently. Since the process models are adapted by hand, which is a time-consuming job, a mechanism to automatically adapt process models is required when the process that is executed has been changed or the implementation has been modified. With such an automatic adaptation a business analyst only needs to review the computed models and can save time and money.

The research area of Semantic Business Process Management (SBPM) (Hepp *et al*, 2005) transfers the concepts and technologies of the Semantic Web to BPM. Thereby, it aims to achieve a higher level of automation regarding the querying, manipulation, and management of business processes and the usage and development of corresponding process descriptions. This requires a machine-accessible representation of the terms used in process descriptions and queries. In the context of process modeling this means that the terms in process models are technically described with concepts of an ontology. Using these semantic annotations an automated planning of the control flow of process models is possible as introduced in Henneberger *et al* (2008).

In this paper we build on this automatic planning and introduce an approach for the adaptation of existing process models using semantic technologies. When process actions have been changed (either in their implementation as e.g. discovered by process mining techniques, or as stated by a management decision), the actions need to be identified in all process models and automatically adapted.

The contribution of this paper is a framework for an automatic adaptation of process models which is indispensable for enterprises that consider to be exposed to intense competition.

There are several variants of possible necessary adaptation. Beginning with the substitution of a complete process model by a re-planned version as an extensive adaptation method, there are more fine-grained ones, such as adaptation of single process actions. As we are interested in adapting only parts of a model, our approach expects a set of process actions that require adaptation.

The adaptation process first searches through all process models for process actions that have been changed. For these process actions the surrounding process fragments are identified. After these steps that are syntax-based only, we now utilize the specified semantic annotations and compute the start state and end state of these fragments which are needed in the following planning step. The planner computes a complete process model from the start state to the end state considering all process actions which are stored in a process library. The result of the planner is integrated into the process model and validated afterwards.

The remainder of this paper is organized as follows. In Section 2 we summarize some basics about (semantic) business process modeling and the automated planning of process models using our planner SEMPA. Section 3 describes the tasks that are necessary in order to adapt an existing process model which is exemplified in a case study in Section 4. We name some related work in Section 5, before we conclude with directions for further research.

2. AUTOMATIC PLANNING OF PROCESS MODELS

For the automatic planning of new process models as well as the adaptation of existing models we are only interested in the control flow perspective of a process (van der Aalst *et al*, 2003). Other perspectives normally related to process models (roles, applications, etc.) are neglected in this paper, but could easily be integrated.

We describe a process (or workflow) model formally as a directed graph G which is denoted by (N, E), where N is the set of nodes and E the set of edges. N consists of the disjoint subsets: N_{start},

N_{stop}, N_{action}, N_{fork}, N_{join}, $N_{decision}$ and N_{merge}. The terms *action*, *decision*, etc. are thereby used in analogy to UML activity diagrams (OMG, 2007).

Each node $n \in N$ has a set of incoming and outgoing edges denoted as $E_{in}(n)$ and $E_{out}(n)$ respectively. Furthermore, the graph G has to satisfy the following conditions:

- N_{start} (resp. N_{stop}) has exactly one element n_{start} (resp. n_{stop}), such that $|E_{in}(n_{start})| = 0 \wedge |E_{out}(n_{start})| = 1$ (called *entry edge* e_{entry}) and $|E_{out}(n_{stop})| = 0 \wedge | E_{\text{in}}(n_{stop})| = 1$ (called *exit edge* e_{exit}).
- $\forall n \in (N_{fork} \cup N_{decision})$: $|E_{in}(n)| = 1 \wedge |E_{out}(n)| >= 2$, $\forall n \in (N_{join} \cup N_{merge})$: $|E_{in}(n)| >= 2 \wedge |E_{out}(n)| = 1$ and $\forall n \in N_{action}$: $| E_{in}(n)| = 1 \wedge |E_{out}(n)| = 1$.
- $\forall n \in N : \exists$ path $p = (n_{start}, \ldots, n_{stop}) \subseteq G$: $n \in p$ (all nodes are reachable from the start).

As pointed out in Thomas and Fellmann (2007), the semantics of meta-model elements for process modeling and their relations are defined already by well established approaches for process modeling. However, the terms used to specify individual model elements (e.g. the name of a particular function or the name of an input parameter) and their semantics are still left to the modeler. It is quite common that different people tend to use different terms for the same real world concepts. Problems in comprehension or ambiguities are the consequence of inconsistently used terms in these models which makes them difficult to understand.

By means of ontologies, terms in process models are conceptualized and their relations are technically defined. This allows for an advanced and automatic processing of semantically annotated process models and their elements.

A *Semantic Business Process Model* describes a set of activities including their functional, behavioural, organizational, operational as well as non-functional aspects. These aspects are not only machine-readable, but also "machine understandable" i.e. that they are either semantically annotated or already in a form which allows a computer to infer new facts using the underlying ontology.

We define *semantic annotation* formally as a function that returns a set of concepts from the ontology for each node and edge in the graph, $SemAn : N \cup E \rightarrow C_{Onts}$. *SemAn* describes all kind of semantic annotations which can be input, output, metamodel annotation, etc. The semantic annotation can either be done manually or computed automatically considering word similarities, etc. We can now define a semantic annotated graph $G_{sem} = (N_{sem}, E_{sem}, Onts)$ with $N_{sem} = \{(n, SemAn(n))|n \in N\}$ and $E_{sem} = \{(n_{sem}, n'_{sem}) \,|n_{sem} = (n, SemAn(n)) \wedge n'_{sem} = (n', SemAn(n')) \wedge (n, n') \in E\}$. C_{Onts} is a set of concepts of (possibly different) ontologies of the set of ontologies *Onts* ($C_{Onts} \subseteq Onts$).

An *ontology* $Ont \in Onts$ is a "(formal) explicit specification of a (shared) conceptualization" (Gruber, 1993) and in our context defined as a quadruple $Ont := (C, R, I, A)$ which consists of different classes C and relations R between them. A relation connects a class either with another class or with a fixed literal. It can define subsumption hierarchies between classes or other relationships. Additionally, classes can be instantiated with a set of individuals I. An ontology might also contain a set of axioms A which state facts (what is true) in a domain. Please note that (Gruber, 1993) actually speaks of "classes, relations, functions and other objects", whereby current languages of the semantic web such as OWL (Grau *et al*, 2008) also include individuals and axioms.

In SEMPRO, a project funded by the German Research Foundation, a (semi-)automatic creation of process models with AI planning algorithms is envisioned which uses these semantic annotated process actions. We speak of semiautomated because the planned process models (which are created fully automatically) are considered as proposals that afterwards need to be assessed by an expert regarding business aspects.

For the automatic planning of process models we require that each node $n \in N$ is at least annotated with some kind of semantic input and output data. $SemIn(n) : N_{sem} \rightarrow C_{Onts}$ ($SemOut(n)$ analogous) is a filter function on the semantic annotation ($SemAn(n)$) to return only the input (resp. output) data. Using semantic annotation allows us to exploit advantages such as e.g. checking for equivalence of concepts, inference techniques and so on.

Therefore, we use parameters that are defined as (*label*, *domain*, *restriction*) for each input and output. All possible parameters constitute the set of parameters *P*. The *label* provides the name of this parameter and the *domain* specifies the ontological class that is the basis. Each semantic input or output similar to Degwekar *et al* (2007) can be further refined by *restrictions* in order to specify specific values (e.g. an *order* has been submitted, *true*, etc.) or ranges (e.g. *order amount* between 0 and 500).

The developed planning algorithm SEMPA (SEMantic-based Planning Approach) proceeds in three steps. First, each semantic input *SemIn* and output *SemOut* of all process actions stored in a process library (called lib_A) are semantically matched (recursively, starting with one or more specified goal parameters) and dependencies between them are calculated and stored for the following steps. The matching between parameters uses reasoning on the ontology concepts and additionally considers the restrictions in order to evaluate whether two parameters $SemIn(n_i)$ vs. $SemOut(n_j)$) match completely, partial or not at all.

Goal parameters belong to a state that shall be reached at the end with one or more goal states being defined. Each parameter in a goal state must be fulfilled by the $SemOut(n)$ of precedent process actions or by parameters that are part of the initial state.

In the second step a forward-search collects all applicable process actions from the stored graph that can help to achieve such a goal state. Therefore, a planning algorithm computes the state after the execution of each action considering the semantic outputs and calculates which other actions can be executed in this state (using *SemIn*). This is performed until all specified goal states are attained or the planning is stopped, because a state has already been computed and does not need to be evaluated again or a goal state will never be achieved.

The result is then used in the last step to create the process model. Therefore, the action-state-graph computed in the step before is extended with different control structures and finally a specific modeling notation, e.g. UML activity diagram (OMG, 2009b), is returned. An overview about these planning steps can also be seen in Figure 1.

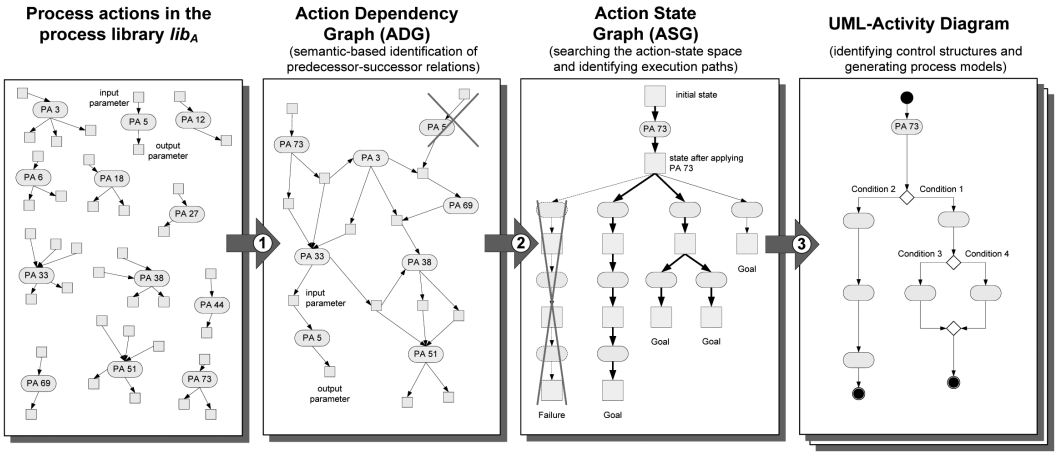

Figure 1: Three steps of SEMPA

We stated before that all process actions are assumed to be annotated with parameters. The first step of the planning algorithm is based on these parameters which is why a lack of the annotation could lead to the impossibility to achieve planning results.

3. TASKS FOR PROCESS MODEL ADAPTATION

The adaptation of process models as well as their execution is necessary for companies in order to be flexible and therefore to have an advantage over competitors. The term flexibility has been widely discussed in information systems (IS) and two areas of flexibility are commonly distinguished (cf. Hanseth *et al*, 1996 or Gebauer and Schober, 2006): flexibility in the pattern of use and flexibility for further changes. Flexibility-to-use is thereby the range of process requirements that is supported by the IS without requiring a major change of the IS. On the other side, flexibility-to-change requires a major change of the IS considering the flexibility of the IT personnel, the integration of data and functionality and the modularity of system components (Gebauer and Schober, 2006).

For process models we can transfer these definitions: a process model has an internal flexibility-to-use, if different kinds of processes that only vary slightly are covered. It has an inherent flexibility-to-change, if most parts of the process model stay the same for major changes of the business process and only some parts need to be adapted. A process model is not flexible at all, if the complete model needs to be redesigned, when changes of the underlying process appear. In this paper we focus on flexibility-to-change, i.e. only some parts of the process model need to be adapted to fit to the business process again.

After transferring these definitions we would name a process model inflexible, if all process actions of the model have been changed: $\forall n \in N : n \in lib_{CA}$ where elements stored in the library lib_{CA} ($\forall n \in lib_{CA} \Rightarrow n \in lib_{CA}$) are those that have been changed. This would result in a replacement of the complete process model.

In contrast, we would name a process model flexible-to change if single process actions or fragments changed and possibly need adaptation:

- *Deleting* a single process action without further adaptation
- *Replacing* a process action with another action out of lib_A
- *Marking* a process action to be re-planned and adaptation that possibly leads to deletion or replacement of the process action or process fragments according to the re-planning

Deleting and replacing single process actions does not necessarily induce a complex replanning if input and output are identical or a produced output is not necessary any longer. Given a process action n_{del} to be deleted, an incoming edge to this node ($E_{in}(n_{del})$) and the successor process action (or another arbitrary process model node) n_{succ} of n_{del} such that $E_{out}(n_{del}) = E_{in}(n_{succ})$. As defined in Section 2 all process actions in our graph have at most one incoming and one outgoing edge. To perform the deletion, the incoming edge to $E_{in}(n_{del})$ is connected to n_{succ} and n_{del} is removed from the model.

The replacement of single process actions is even more easy. The incoming edges to the node n_{old} to be replaced are connected to the new node n_{new}. The outgoing edges of n_{old} are connected to n_{in} and finally n_{old} is deleted. After integrating the new one no replanning is necessary. Please note that this obvious deletion and replacement of process actions is possible only if the following assumptions apply:

- Output produced by the deleted or replaced process action is not required at any other node in the process model

- Input and Output fit flawlessly from predecessor to successor nodes when deleting nodes
- Input and Output of a replacement node is semantically identical to the node to be replaced

As those assumptions require quite significant human effort to identify suitable alternative process actions, we look into the last and most interesting possibility when it comes to process model adaptation. This is when the adaptation of certain process actions is necessary but it is not now how this adaptation could look like e.g. by replacing a process fragment with another new one. This is the usual case as we suppose the set of process actions in enterprises to be extraordinary large and complex. Furthermore there might be two or more process actions that need to be adapted due to legal and/or business driven changes. As a result, the most appropriate way would be to mark all process actions that need adaptation. More technically we collect all those process actions in a library $lib_{CA} \subseteq lib_A$ as introduced above.

In order to automatically adapt a process model to changing requirements we assume that the process model has already been planned and therefore all process actions are described at least with their semantic inputs and outputs using ontology concepts (*SemAn*). Furthermore, we assume that we know the process actions that have been changed and how they have changed. Despite this assumption we will describe how this identification could take place and how the used semantics could help to improve this manual process. Since new regulations or customer requests can mostly be reduced to changes on a few actions this assumption is not too restrictive anyway. After the changed actions have been identified in the existing process models (which is a simple search according to their name), the adaptation process can start.

Following up the described requirements, the tasks for process model adaptation are the following:

1. Identification and collection of process actions that have been changed (Section 3.1)
2. Computation of the fragments that need to be adapted[1] (Section 3.2)
3. Identification of the initial state of each process fragment (Section 3.3)
4. Calculation of the goal state of each process fragment (Section 3.4)
5. Re-planning the process fragments considering the changed process actions (Section 3.5)
6. Integration of the planning result into the process model (Section 3.6)
7. Validation of the adapted process model (Section 3.7)

We will now elaborate each task in further detail.

3.1 Identification of Changed Process Actions

As explained in the section before, our approach expects process actions that have changed to be in a library called lib_{CA}. The first step in our approach is how the collection of process actions is managed, and the library is filled at the end.

The most obvious way to fill lib_{CA} is simply to put the process actions out of lib_A manually, i.e. the human modeler needs to look through all process actions that have been stored in the process library lib_A, select those that have been changed and copy them to the library lib_{CA}. As this might be a time consuming task depending on the power of lib_A we will show another more efficient way.

As our approach bases on semantic annotations, these can be used for the identification. We assume that we do not know each task that has been changed, but at least what the changes are about. Concretely this means that we are able to identify the concept *c* in our ontology that is used

1 Where a fragment can be more than the known process action that changed. Basically, it could be made up of several process actions around the changed one.

for annotation and is subject of the change. Using this concept and the underlying ontology it becomes possible to automatically identify process actions that take the concept as input or output using the filter functions for semantic annotations: $\forall n \in lib_A : SemIn(n) \cap C \neq \emptyset \vee SemOut(n) \cap C \neq \emptyset \Rightarrow n \in lib_{CA}$ with C being the set of changed concepts. Using this method it is possible to identify process actions that might have changed and need adaptation. Those are added to lib_{CA} therefore to be considered in the following phases.

3.2 Computation of Fragments to be Adapted

All process actions that have been changed (stored in the process library lib_{CA}) need further processing. In the next step we need to search for the entailing fragments of these process actions in order to re-plan these fragments. The calculation of the surrounding fragment allows us later to start planning with one initial state and a single goal state and makes the integration of the planning result easier afterwards.

A process fragment (or Single-Entry-Single-Exit fragment, short: SESE fragment) can be either a single process action or a part of the model that has only one incoming (the entry edge e) and one outgoing (the exit edge e'). It can be defined as a nonempty subgraph of G with $N' \subseteq N$ and $E' = \mathrm{E} \cap (N' \times N')$ such that there exist edges $e, e' \in E$ with $E \cap ((N \backslash N') \times N') = \{e\}$ and $E \cap (N' \times (N \backslash N')) = \{e'\}$ (cp. Vanhatalo *et al*, 2007). In our further work we are only interested in canonical process fragments, i.e. fragments that do not overlap and are either nested or disjoint.

We calculate the (canonical) process fragments as well as the strongly connected components (SCC) of the marked process actions that have been changed. SCCs of a directed graph G are the maximal strongly connected subgraphs, i.e. there is a path p from each node in the subgraph to every other node in the same subgraph. The computation of fragments and SCCs is done using a token-flow algorithm that has been introduced in (Götz *et al*, 2009). We shortly summarize our algorithm here.

This algorithm builds on a token propagation mechanism. Tokens and token algorithms have already been described elsewhere, e.g. in (OMG, 2009a). The token-flow is calculated in two steps: first, single tokens propagate through the graph and second, tokens from the same origin are recombined.

In the first step, tokens are created at the out-flow of splitting gateways carrying information on their origin. They propagate along the flow and can re-combine with other tokens. To each edge a subset of tokens called *token labeling* is assigned.

For a single token, the propagation through the graph is calculated by tracking its route along the edges. When tokens arrive at a gateway with several outgoing edges (either $N_{decision}$ or N_{fork}), all of the gateway's outgoing edges $e \in E_{out}$ are labeled with the same token: At nodes without degree > 1, new tokens are created. The outgoing edges are labeled with the union of the arriving token sets and the newly generated tokens. At merging gateways (N_{merge} or N_{join}), E_{out} is labeled with the union of all incoming tokens.

Calculating the flow for each token separately is inefficient, because edges have to be visited several times, once for each token. It becomes more efficient when handling complete sets of tokens, by successively calculating the out-flow at nodes where all entering flow has been labeled.

In the second step, the recombination of tokens is calculated. When all tokens belonging to the same gateway have arrived at one edge, they are removed from the labeling (re-combination). Components can be derived by matching pairs of edges with equal token sets.

Different to (OMG, 2009a), the process does not stop whenever a component is encountered but continues until all the edges have been labeled. The procedure does not have to start again, and also enables the recognition of more advanced, interleaved structures.

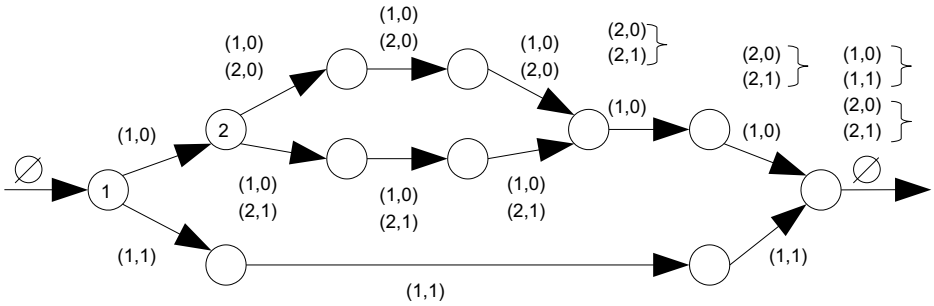

Figure 2: Example of Converging Token Flow

We now define the set of *tokens* $\mathbb{T}$. A token carries information on the parallelization for which the token was generated, i.e. the origin node n and a number i referring to the corresponding outgoing edge:

$$\mathbb{T}(_{N,E}) := \{(n, i) \mid n \in N \wedge i \in \mathbb{N} \wedge i < |E_{out}(n)|\}.$$

Each edge in the graph is assigned a subset of $\mathbb{T}$ by the *token labeling function* $t : E \rightarrow P(\mathbb{T}) \cup \{\perp\}$. Token creation occurs at nodes with $|E_{out}(n)| > 1$. Tokens propagate and unite at nodes with $|E_{in}(n)| > 1$. Figure 2 illustrates the flow of tokens created at nodes 1 and 2. After all edges have been labeled, tokens originating from the same node *converge* and are removed from the labeling (indicated in the figure by the curly brackets).

Finally, the labeling is used to determine the components of the graph. If for two edges $e_1, e_2 \in E, e_1 \neq e_2 : t(e_1) = t(e_2)$ holds, they mark the beginning and the end of a component C, i.e. $\{e_1, e_2\} = \{\text{source}(C), \text{sink}(C)\}$.

The resulting components can be overlapping, ambiguous, and include trivial components. In Figure 2 the converged labeling $\{(1, 1)\}$ appears at four edges. Each pair of these edges marks a valid component, but not all of them are desirable. This can be avoided by a strategy which shows how to derive the correct partitioning, covering sequences, and further component types.

Algorithm 1 summarizes the Token Flow procedure. For an acyclic graph with start node N_{start}, call the *tokenFlow* function with an initial labeling $t(N_{start}) := \emptyset$. The function $pick(e \in E \mid prop : e \rightarrow \mathbb{B})$ nondeterministically selects an element e from E, that meets a required property, i.e. $prop(e) = true$. The function *timestamp* assigns an incrementing index to the visited elements, thus imposing an ordering on them. The main loop of the algorithm in line 14 picks an arbitrary node, for which all incoming edges have been labeled with tokens, and calls the *processNode* function for it. This function computes the leaving flow from the entering flow by first merging all token sets from the entering edges (line 2), and then propagating the resulting set to the outgoing edges. For multiple outgoing edges, a new token is assigned to each edge in line 8. After processing, to each edge a timestamp (line 18) is assigned, which can later be used to identify sequence-boundaries. Once the main loop in *tokenFlow* terminates, converged tokens are removed (line 22).

Please note that cycles in process models need a special treatment (which is why SCCs are also computed) and are only considered in an extended version of this algorithm which is elaborated in further detail in (Götz *et al*, 2009).

Regarding the complexity of the Token Flow algorithm we assume a constant runtime of the functions *pick*, *t*, *timestamp* and *first*. For the remaining parts of the algorithm the complexity is $O(|N| + |E|)$.

Algorithm 1: Calculate Token-flow

Require: $t : E \rightarrow P(\mathbb{T}) \cup \{\perp\}$ // initial making
$O \subseteq N$ // to-do set of nodes
Ensure: $t : E \rightarrow P(\mathbb{T})$
1: **processNode**($n \in N$) {
2: $P(\mathbb{T})\ M := \cup_{u \in t(E_{in}(n))} u$;
3: **if** $|E_{out}(n)| = 1$ **then**
4: $t(\text{elt}(E_{out}(n))) := M$;
5: **else**
6: **int** $i := 0$;
7: **for all** $e \in E_{out}(n)$ **do**
8: $t(e) := M \cup \{(n, i++)\}$;
9: **end for**
10: **end if**
11: }
12:
13: **tokenFlow**(t, O) {
14: **while** $O \neq \emptyset$ **do**
15: **pick**($n \in O \mid t(E_{in}(n)) \not\ni \perp$);
16: **processNode**(n);
17: $O = O \backslash n$;
18: **timestamp**(n);
19: **end while**
20: **for all** $e \in E$ **do**
21: **for all** $n \in \text{first}(t(e))$ **do**
22: **if** $t(e) \supseteq \{(n, i) \mid i < |E_{out}(n)|\}$ **then**
23: $t(e) = t(e) \backslash \{(n, i) \mid i < |E_{out}(n)|\}$
24: **end if**
25: **end for**
26: **end for**
27: }

By using the described algorithm we found the components that contain changed process actions and therefore need to be adapted. We set these as the process fragments and calculate the initial and goal state of each fragment for replanning in the further steps.

This algorithm walks through the process model graph and processes each node once. Thus, the complexity of this part is linear ($O(n)$).

3.3 Initial States of Process Fragments

In this step we need to identify the initial state of the process fragment that shall be actualized. A state s is a subset of the set of parameters P, $s \subseteq P$. For this identification we have two possibilities:

The first one would be to compute the state that has been reached where the process fragment starts. This state can then be used as initial state for the re-planning. Therefore, we build a planning graph starting with the first action of the process model until the beginning of the fragment has been reached. The disadvantage of this solution is, that, if we have a rather big process model, probably

Algorithm 2: Compute Initial State

```
Require: Strongly connected component C
1:  computeInitialState(C) {
2:  n := getFirstNode(C)
3:  init := SemIn(n) // initial state
4:  removeList := SemOut(n)
5:  for all n := getFollowingNodeInComponent(n, C) do
6:     for all Parameter P ∈ SemIn(n) do
7:        if ¬ (P ∈ init) ∧ ¬(P ∈ removeList) then
8:           init = init ∪ P
9:        end if
10:    end for
11:    removeList := removeList ∪ SemIn(n)
12: end for
13: return init
14: }
```

hundreds of process actions and states need to be computed again. Additionally, we face the problem how to compute the initial state if the process action that has been changed is the first one of the whole process that has been executed. Then, there is no chance to determine this state.

The second possibility only looks at the process fragment that needs to be re-planned. This process fragment is probably much smaller which results in a faster computation than for the whole process model. Here, we compute all input parameters of the actions of this fragment. We remove the generated output parameters again, because those had been created as part of the fragment before and therefore are not available for the initial state anymore during re-planning.

This leads to a set of parameters that was necessary before for the execution of the process actions and should be sufficient for the re-planning, too. Formally, we take as initial state $init := SemIn(n_1) \cup \bigcup_{\forall i=2..|N|}(SemIn(n_i) \backslash SemOut(n_{i-1}))$ (compare Algorithm 2). The complexity of this algorithm is $O(|N| + |P|)$.

3.4 Goal States of Process Fragments

Goal states = $G_1, G2,\ldots, G_k$ specify the set of $k \in \mathbb{N}$ different (sets of) parameters $G_x \subseteq P$ (with $x = 1, \ldots, k$) which shall be reachable in a feasible solution. The calculation of the goal state work analogous to the computation of the initial state. Again, we have two possibilities: The first possibility computes the initial state of the rest of all process actions in the process model that follow the fragment. The other possibility considers only the process fragment that should be adapted and calculate the sum of all outputs of all actions in this fragment ($\bigcup_{i=1..|N|} SemOut(n_i)$).

For performance issues we again take the second possibility, thus the complexity is equal to the step before: $O(|N| + |P|)$.

3.5 Re-planning Process Fragments

Now that we have the initial state and the goal state of the process fragment we can re-plan the process fragment. We described the automatic planning of business processes already in Heinrich *et al* (2008). Therein, we described the planning but did not specify how the adaptation of process models can be realized.

In order to better understand this important phase we will summarize the steps of our semantic planning approach in the following.

Note that all existing process actions $n \in N_{action}$ that have been stored in the process library lib_A are considered. Please note, that the library lib_{CA} was only used for identification of the process fragments, but for re-planning all process actions available in the library are required.

In order to identify dependencies between process actions regarding their input and output parameters, SEMPA computes an Action Dependency Graph (ADG) through backwards traversing beginning from the goal state. This ADG includes process actions and corresponding parameters as nodes. We use semantic reasoning for this part of the algorithm. This means input and output parameters of the process actions stored in the process library and their relationships in the ontology are analyzed (*SemIn*(*n*) and *SemOut*(*n*) resp.).

As the ADG does not describe direct sequences of process actions, a forward search algorithm is used to determine all sequences of process actions leading from the initial to the goal state. As a result, we obtain an Action State Graph (ASG) which has two partitions: the process actions N_{action} and states which capture the state of the world after the execution of the action considering *SemIn* and *SemOut*. Thereby, the algorithm performs a nondeterministic planning (Ghallab *et al*, 2004) with initial state uncertainty (Bonet and Geffner, 2001). This graph comprises all feasible solutions to the corresponding planning problem.

The action-state-graph is the basis to build the (syntactically correct) process model in the last step and to identify control structures such as e.g. $N_{decision}$ or N_{join}.

There is no estimation regarding complexity of the aforementioned algorithm as it is not examined in further detail here. We refer to Henneberger *et al* (2008) for further details.

3.6 Integrating the Planning Results

The result of the planning is first put into an own embedded subprocess to enable an isolated consideration and possible further (manual) changes of the new planned part. This is performed utilizing a StructuredActivityNode in UML in order to create a logical, closed group of the new process actions. The two nodes N_{start} and N_{stop} of the subprocess are not used in further steps and can therefore be removed. The remaining subprocess is integrated into the original model as follows: As defined in Section 3.2 the parts of the original process we identified for re-planning have only one entry edge e and one exit edge e' (due to the fact that they are SESE fragments). Those are connected to the entry edge e_{entry} of the new planned fragment. The exit edge e_{exit} is connected respectively.

It is possible that the planning algorithm returns more than one result. There are several possibilities to deal with this. Based on specified properties one of the fragments could be chosen automatically as the one to be inserted in the original model. Those properties and thereon based decision strategies can range from straightforward to more complex approaches.

A straightforward way would be to define the number of process actions in the new planned fragments as parameter for the decision. The strategy and this parameter could be to chose and integrate the fragment that has the fewest process actions as this might indicate the fastest execution time or least complexity. More complex strategies could include economic parameters in the decision. Processing time or the cost of the planned fragments could affect the strategy which fragment to implement. Calculating (including the definition of) the cost of process model or process model fragments is out of the scope of this paper. Therefore, we present a more pragmatic approach that does not exclude any possible results: If SEMPA computes more than one result that could be integrated, then all planning results are integrated in copies of the original process model

and the different alternatives are shown to the user who must then decide which one conforms best to the business requirements.

If SEMPA does not return a result, e.g. because not all actions for achieving the goal states exist in the process library, then the planning is stopped and the user is notified which parameter could not be fulfilled. This part of the algorithm that locates the position where to integrate the new fragment can be achieved by a depth-first search basically. This results in complexity of $O(|N| + |E|)$. Another possibility would be to store the identified SESE nodes in an efficient data structure like a hash map.

3.7 Validating the Adapted Process Model

The resulting process model needs to be validated in the end in order to ensure that still all dependencies have been fulfilled. First, it is validated automatically: therefore, it is evaluated whether each process action has all necessary input parameters and whether there is any deadlock or lack of synchronization. Therefore, we applied the algorithm presented in Götz *et al* (2009) again that is using the already existing components as well as the re-planned fragments. For computing deadlocks and lack of synchronization we applied a method similar to Vanhatalo *et al* (2007) that has been adapted to conform to the token analysis algorithm introduced before.

If no errors have been detected, then the resulting process is shown to the business analyst who can then decide whether it should be further refined or it can be enacted directly as it has been planned. The business analyst might for example identify parts of the process model that can be further simplified: since the adaptation only computes process fragments again but not the whole process, there could now be optimization potential in the process model.

The overall complexity of our method is composed of the discrete parts we described in this section. Although we analyzed most of the steps regarding complexity not all parts have been detailed as some of them are outside the scope of this paper. That is why we cannot give the exact costs for the entire process.

4. CASE STUDY

For a case study we chose an example from the financial services industry. This industry seems to be dedicated for the adoption of technologies facilitating the automatic planning of business processes. Electronic commerce has radically changed the competitive landscape in the financial services industry and provides new business opportunities (Malhotra and Malhotra, 2006). The internet constitutes e.g. a flexible delivery channel for selling financial products. Product life cycles have been accelerated dramatically and at the same time financial products have become more varied and complex. This determines more difficult and permanently changing processes that need to be managed including external suppliers and partners. The ability to rapidly adapt processes to new business requirements constitutes a significant competitive advantage.

We envision financial services companies that are able to design new products and at the same time can automatically configure their processes reducing time-to-market. The basic idea for this example has been extracted from a real life case which has been conducted in the SEMPRO-project at a large financial service institution. However, it has been highly simplified for this paper.

The process depicted in Figure 3 describes the processing of an order (e.g. stock order). After an order has been submitted to a financial institute, the responsible employee checks the competencies and validity of the customer and routes the order, if it is valid, to the stock market. If it is not valid, then the customer is contacted first and afterwards the validity is checked by a second person again (4-eyes authorization). If the competencies and validity are fulfilled now, then some

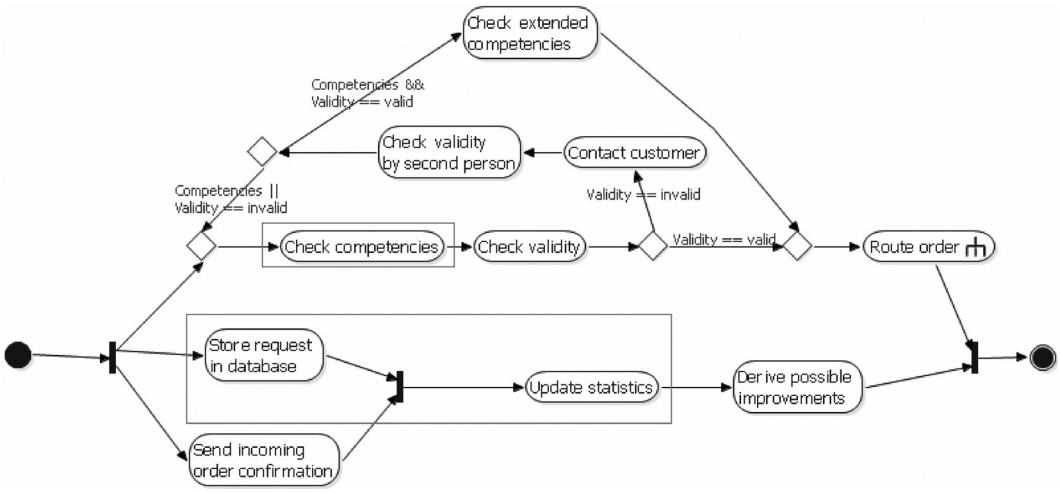

Figure 3: Proof of Concept from the Financial Area with Marked Process Actions

extended competencies are checked, otherwise the basic checks are started again. In parallel to this process, the order request is stored in a database and the customer gets a confirmation email stating that the order request has been received. After that, some statistics are updated which allow the derivation of possible improvements later on. The following changes have now been made to the process actions (marked in a rectangle in Figure 3):

- *Store request in database* had before as input an *order request* and now requires additionally the *person* that works on the *order request*. This is not covered by the process model yet and therefore other process actions of the process library lib_A need to be integrated.
- *Update statistics* first required only an *order request* and gave a *statistics* as output, now it requires a *valid order* and gives an *order statistics* as output. The management has decided that they only want to see statistics about valid orders.
- *Check competencies* is not allowed anymore and has been deleted from lib_A. Now always the extended competencies must be checked (not only whether the *employee* has the rights to work on the *order*, but also whether the *customer* has the authority to buy or sell the *order* corresponding to the assigned *risk class*).

The ontology that is used here (an excerpt is shown in Figure 5) defines an *order* as a composite parameter which includes an order state (e.g. *valid* or *invalid*), an order amount (typically a positive numeric value) and the order type (should the order be bought or sold). Orders are integrated in *order statistics* and have one requesting *customer*. A customer is a person which might also be an employee. Each order has an associated risk class.

After identifying the changed process actions, we need to compute the fragments that need to be adapted. Therefore, our token-flow algorithm analyzes the process model and discovers several components (including loops in the process model, cf. Figure 4). For the calculation of fragments it is required to split some control nodes (like the first fork node) into two different ones in order to compute the process fragments. These can be seen in Figure 6. In this figure C_2 and C_6 need to be adapted as some of their entailing actions have been changed.

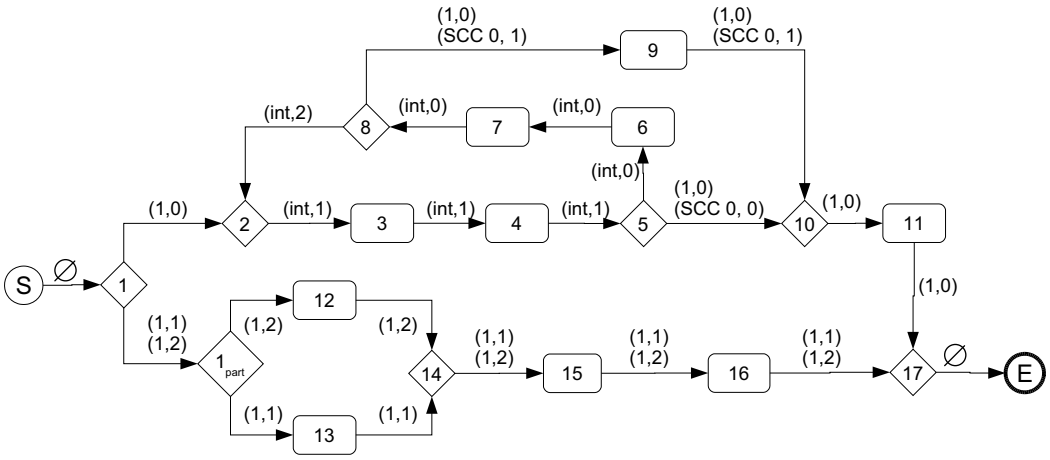

Figure 4: Tokens in the Graph

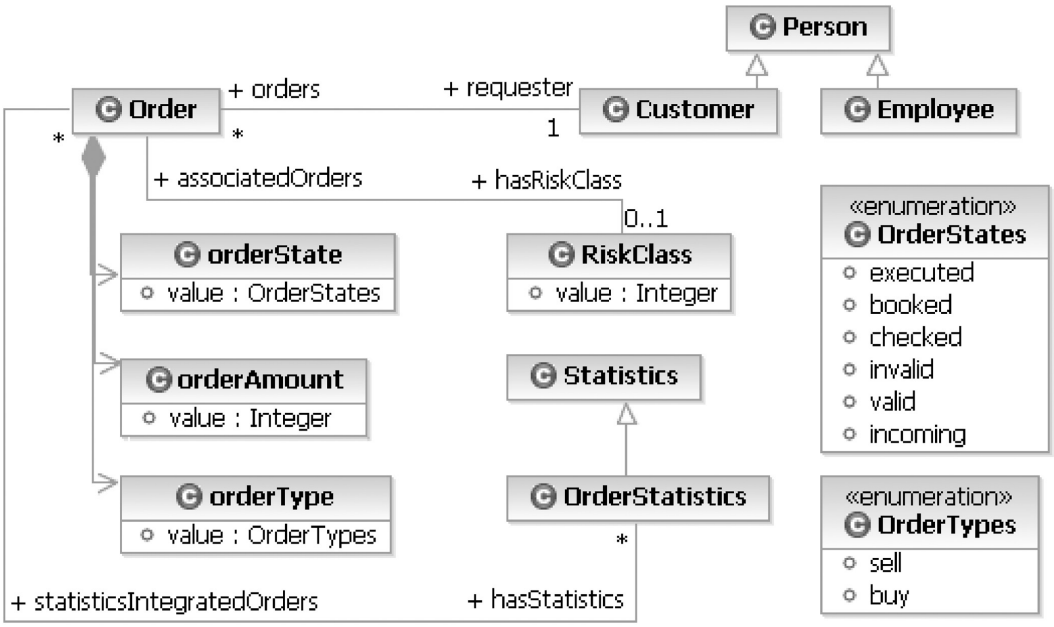

Figure 5: Ontology in the Proof of Concept

As third task we compute the initial states of component C_2 and C_6 (both are shown in Figure 7 again). The initial state of C_2 is defined as *SemIn*(*Check competencies*) ∪ *SemIn*(*Check validity*) \ *SemOut*(*Check competencies*). The input of *Check competencies* has been an incoming order with a positive order amount, more formally defined as (*Order*, {(*OrderState*, OrderStates, {incoming}), ({*OrderAmount*, int, > 0}), (*OrderType*, OrderTypes, {Buy, Sell})}) whereas the output had an *OrderState* that was not {incoming} anymore, but {checked}. *Check validity* requires a checked

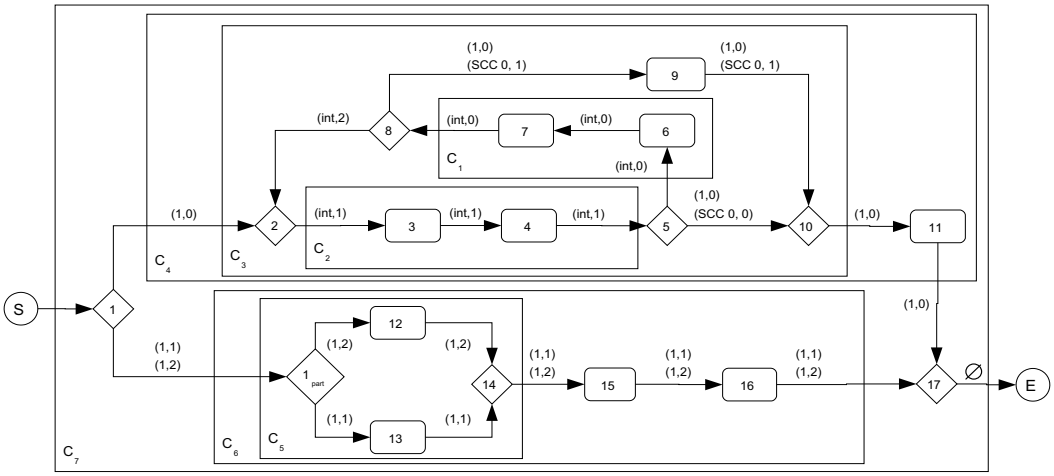

Figure 6: Discovered Components

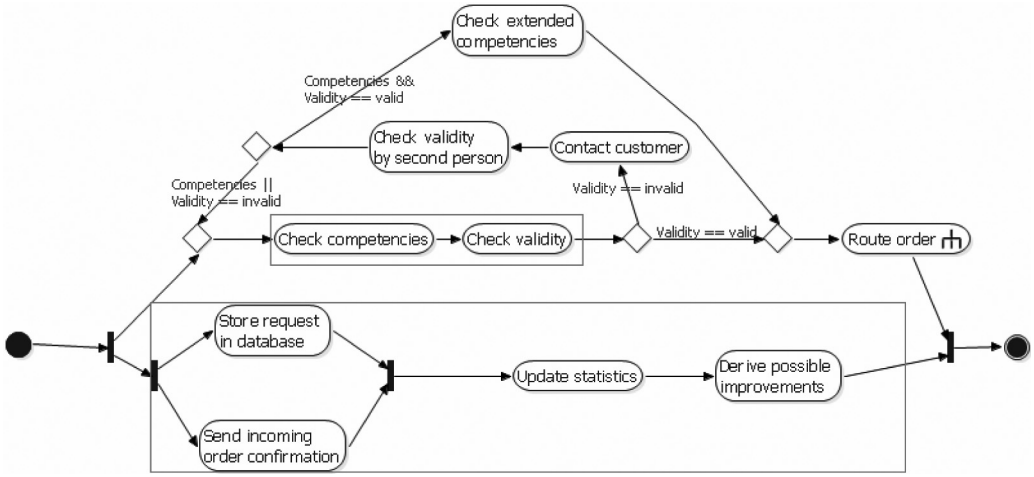

Figure 7: Determined Process Fragments for Planning

order and a customer (*Customer*, Person, Person) and returns an order that is {valid}. Hence, the initial state of C_2 is {(*Order*, {(*OrderState*, OrderStates, {incoming}), ({*OrderAmount*, int, > 0}), (*OrderType*, OrderTypes, {Buy, Sell})}), (*Customer*, Person, Person)}.

As fourth task we compute the goal states of component C_2 and C_6. The goal state of component C_2 can be computed as an order that needs to be valid (*Order*, {(*OrderState*, State, {valid}), ({*OrderAmount*, int, > 0}), (*OrderType*, Type, {Buy, Sell})})

Now the planning for both fragments can start. This results in two (small) independent process models which are integrated as embedded subprocesses into the already existing process model. For *component C2 the new action* Check extended competencies has been found which is now the only process action that returns a checked *order* which is the input of *Check validity* that again returns a valid *order*.

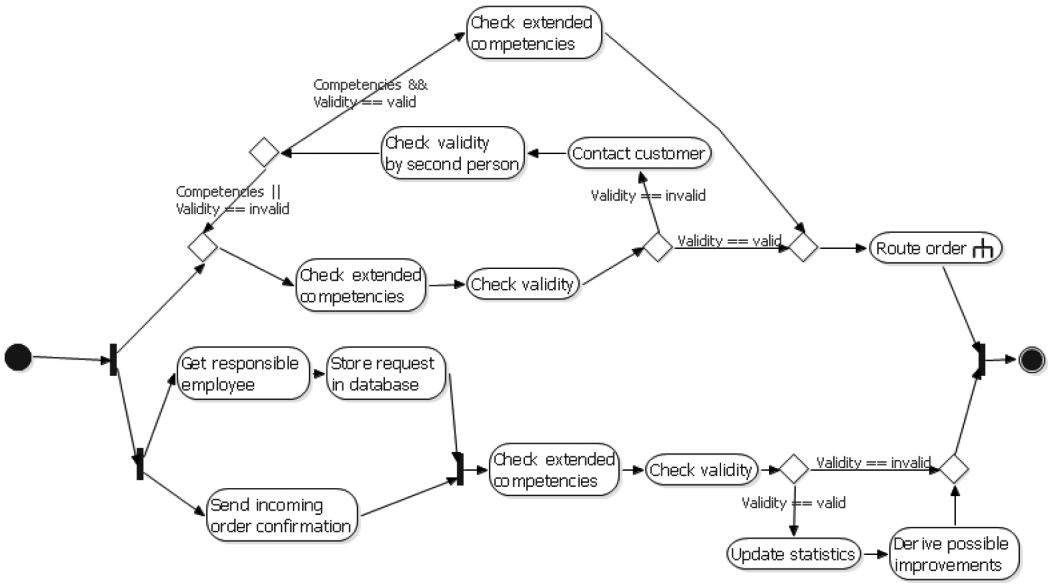

Figure 8: Final Planning Result

Afterwards, the embedded subprocesses can be resolved and directly integrated into the process model (cp. Figure 8). In the end the validation algorithm affirms that all dependencies have been fulfilled and that there are no deadlocks.

Please note that our planning algorithm is currently not capable of calculating cycles. The existing loop in the upper part of the original process model has been created by a human modeler. The business analyst might notice that the process actions in the new planned area are similar to existing process actions. Therefore, a re-combination of the process model might make sense, but this up to the business analyst.

The process model adaptation mechanism uses the implementation of the planner SEMPA on top of Eclipse JWT (Lautenbacher, 2009) utilizing OWL API together with the Pellet reasoner and an ontology based on OWL from which the semantic information is derived from. For the calculation of the components using the introduced token-flow algorithm we integrated the Workflow-Codegeneration framework that has been described in Roser *et al* (2007).

5. RELATED WORK

Already some work exists on using semantic annotations of process models: often semantic annotation is used to identify suitable semantic web services for execution (Hepp and Roman, 2007). In Koliadis and Ghose (2007) the authors apply annotations to verify interoperating processes that are captured in process models and try to find inconsistencies between actions that have been semantically annotated with effects. Thereby, they also consider control structures such as $N_{decision}$ and N_{fork}.

Koschmider (2007) shows in her dissertation how a user can be assisted during the modeling of processes by calculating the similarity between the existing process model and fragments from other models. Thereby, not only semantic dependencies, but also syntactic, structural and linguistic similarities are considered. However, already completed process models cannot be adapted.

In Graml *et al* (2008) the authors use business rules to provide a mechanism to adapt the control-

flow of a business process. The underlying idea here is that business rules often are already implicitly contained in a process model and therefore should be extracted to make it more agile. In difference to our approach no reasoning technologies are used which makes it difficult to resolve ambiguities. Similarly, Milanovic *et al* (2008) shows how business rules and processes can be combined during modeling.

Tripathi *et al* (2008) describes an adaptation of process models when the underlying web services have changed. Thereby, business processes need to be captured in OWLS (Martin *et al*, 2007) and when a service has been changed, the sequence of services can be calculated again. However, this approach does not consider more complex control structures or compute advanced process models to our knowledge.

There is also ongoing work concerning (semantic) correctness of BPM systems: Especially changes and their effects during execution are analyzed (Ly *et al*, 2006; 2008). However, this is not the focus of our research as we investigate the design time of process models and therefore handle a different phase of the business process lifecycle.

Furthermore, the planning of process models has similarities with (semantic) web service composition approaches (Sirin *et al*, 2004; Lang and Su, 2005; Pistore *et al*, 2005; Meyer and Weske, 2006). These approaches aim at composing (executable) workflows, consisting of individual semantic web services which are arranged together to achieve one distinct goal. Most approaches either do not plan complex compositions comprising e.g. alternative or parallel control flows or cannot handle numerical variables and enumerate states explicitly which is necessary in our context.

Additionally, there are important conceptual differences between the planning of process models and web service composition. We want to support process modelers in the task of designing (technology-independent) process models. Thus, the result of planning should be a visual and (for a human being) comprehensible representation of the process, whereas in the context of web service composition the specification of the workflow above all should be machine interpretable. Also, the planning of process models is conducted on a higher level of abstraction. Since actions do not need to be executable in the first place, their semantic annotation can be more "lightweight". Thereby, a major disadvantage of semantic web service technologies (see the discussion in Haniewicz *et al*, 2008) may be alleviated.

6. CONCLUSION AND FUTURE RESEARCH

In this paper we have introduced an adaptation mechanism for existing process models. If the demands of customers change, new jurisdiction and regulations appear or a supplier adapted its process, then only the process actions that already exist in a process library need to be changed and all process models can be adapted automatically. We identified and described the phases of this adaptation mechanism in detail and demonstrated the approach in a case study. Therefore, we use semantic technologies for the re-planning of identified process fragments.

In the future we will work on the identification of duplicates in the re-planned process model as well as decision support and metrics in case there is more than one feasible solution to be integrated in a process model. This will be done as part of the validation step and we aim to automatically simplify the process model. Additionally, we work on improving the planning algorithm (e.g. including arbitrary cycles during planning).

Taking business rules into account is also an important prerequisite to consider economical requirements on a process model: e.g. a process is not allowed to cost more than 1000 Euro or roles are not able to work on several actions in parallel. These requirements will require a refinement of the existing algorithms which we will analyze (together with its impacts) in the automotive domain shortly.

REFERENCES

BONET, B. and GEFFNER, H. (2001): GPT: A tool for planning with uncertainty and partial information. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 82–87.

DEGWEKAR, S., LAM, H. and SU, S.Y. (2007): Constraintbased brokering (CBB) for publishing and discovery of web services. *International Journal on Electronic Commerce Research*, 7(1): 45 – 67.

GEBAUER, J. and SCHOBER, F. (2006): Information system flexibility and the cost effciency of business processes. *Journal of the Association for Information Systems*, 3: 122–147.

GHALLAB, M., NAU, D. and TRAVERSO, P. (2004): *Automated Planning*. Elsevier, San Francisco.

GÖTZ, M., ROSER, S., LAUTENBACHER, F. and BAUER, B. (2009): Token analysis of graph-oriented process models. In *Proceedings of DDBP 2009*, Auckland, New Zealand.

GRAML, T., BRACHT, R. and SPIES, M. (2008): Patterns of business rules to enable agile business processes. *Enterprise Information Systems*, 2(4): 385–402.

GRAU, B.C., HORROCKS, I., MOTIK, B., PARSIA, B., PATELSCHNEIDER, P. and SATTLER, U. (2008): OWL2: The next step for OWL. *Journal on Web Semantics*, 6(4): 309 – 322.

GRUBER, T.R. (1993): A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5 (2): 199–220.

HANIEWICZ, K., KACZMAREK, M. and ZYSKOWSKI, D. (2008): Semantic web services applications – a reality check. *Journal Wirtschaftsinformatik*, 50(1): 39–45.

HANSETH, O., MONTEIRO, E. and HATLING, M. (1996): Developing information infrastructure: The tension between standardization and flexibility. *Science, Technology and Human Values*, 11(4): 407–426.

HEINRICH, B., HENNEBERGER, M., KRAMMER, A., BEWERNIK, M. and LAUTENBACHER, F. (2008): SEMPA – Ein Ansatz des Semantischen Prozessmanagements zur Planung von Prozessmodellen. *Journal Wirtschaftsinformatik*, November/December.

HENNEBERGER, M., HEINRICH, B., LAUTENBACHER, F. and BAUER, B. (2008): Semantic-based planning of process models. In *Proceedings of Multikonferenz Wirtschaftsinformatik (MKWI)*.

HEPP, M., LEYMANN, F., DOMINGUE, J., WAHLER, A. and FENSEL, D. (2005): Semantic business process management: A vision towards using semantic web services for business process management. In *Proceedings of IEEE ICEBE*, Beijing, China, 535–540.

HEPP, M. and ROMAN, D. (2007): An ontology framework for semantic business process management. In *Proceedings of Wirtschaftsinformatik*, Karlsruhe, Germany.

KOLIADIS, G. and GHOSE, A. (2007): Verifying semantic business process models in inter-operation. In *Proceedings of IEEE SCC 2007*, Salt Lake City, Utah, USA; July, 913.

KOSCHMIDER, A. (2007): Ahnlichkeitsbasierte Modellierungsunterstützung für Geschäftsprozesse. Ph.D. thesis, Universität Fridericiana zu Karlsruhe, Germany.

LANG, Q. and SU, S. (2005): AND/OR graph and search algorithm for discovering composite web services. *International Journal of Web Services Research* 2(4): 46–64.

LAUTENBACHER, F. (2009): Execute your processes with Eclipse JWT. In *Proceedings of JAX / SOACon / Eclipse Forum Europe*, Mainz, Germany.

LY, L., RINDERLE, S. and DADAM, P. (2008): Integration and verification of semantic constraints in adaptive process management systems. *Data Knowl. Eng.*, 64(1): 3–23. ISSN 0169-023X.

LY, L.T., RINDERLE, S. and DADAM, P. (2006): Semantic correctness in adaptive process management systems. In BPM 2006, Vienna, Austria, 193–208. SpringerVerlag.

MALHOTRA, R. and MALHOTRA, D. (2006): The impact of internet and e-commerce on the evolving business models in the financial services industry. *International Journal of Electronic Business*, 4(1): 56–82.

MARTIN, D., PAOLUCCI, M. and WAGNER, M. (2007): Bringing semantic annotations to web services: OWLS from the SAWSDL perspective. In *Proceedings of ISWC/ASWC*, Busan, Korea.

MEYER, H. and WESKE, M. (2006): Automated service composition using heuristic search. In *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*, 81–96. Vienna, Austria.

MILANOVIC, M., GASEVIC, D. and WAGNER, G. (2008): Combining rules and activities for modeling service-based business processes. In *Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops*, 11–22. IEEE Computer Society, Washington, DC, USA. ISBN 978-0-7695-3720-7.

OMG (2007): Unified modeling language (UML) superstructure, version 2.1.2. OMG Specification.

OMG (2009a): Business Process Modeling Notation Specification, Version 1.2. formal/09-01-03.

OMG (2009b): Unified modeling language (UML) superstructure, version 2.2. OMG Specification.

PISTORE, M., TRAVERSO, P., BERTOLI, P. and MARCONI, A. (2005): Automated synthesis of composite BPEL4WS web services. In *3rd International Conference on Web Services*, 293–301.

ROSER, S., LAUTENBACHER, F. and BAUER, B. (2007): Generation of workflow code from DSMs. In *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling*, Montreal, Canada.

SIRIN, E., PARSIA, B., WU, D., HENDLER, J. and NAU, D. (2004): HTN planning for web service composition using SHOP2. *Journal of Web Semantics* 1(4): 377–396.

THOMAS, O. and FELLMANN, M. (2007): Semantic business process management: Ontology-based process modeling using event-driven process chains. *International Journal of Interoperability in Business Information Systems*, 2 (1).

TRIPATHI, U.K., HINKELMANN, K. and FELDKAMP, D. (2008): Life cycle for change management in business processes using semantic technologies. *Journal of Computers*, 3(1): 24–31.

VAN DER AALST, W., TER HOFSTEDE, A., KIEPUSZEWSKI, B. and BARROS, A. (2003): Workflow patterns. *Distributed and Parallel Databases* 14(3): 5–51.

VANHATALO, J., VÖLZER, H. and LEYMANN, F. (2007): Faster and more focused control-flow analysis for business process models through SESE decomposition. In *ICSOC*.

BIOGRAPHICAL NOTES

Thomas Eisenbarth holds a diploma in computer science from the University of Augsburg and has been employed as a researcher at the Distributed Systems Lab at the University of Augsburg in Germany since 2009. His research interests include the combination and application of semantic technologies on business process models.

Thomas Eisenbarth

Florian Lautenbacher received his PhD at the University of Augsburg, Germany, and holds a diploma in Computer Science from the same university. His research interests are in applying semantic technologies to model-driven software engineering, in particular in workflow and business process technologies as well as service-oriented architectures. Moreover he is involved in several other national and international projects mostly related to business process management and SOA. He has published more than 20 scientific papers.

Florian Lautenbacher

Bernhard Bauer has been professor and head of the Programming of Distributed Systems Group at the University of Augsburg since 2003. He holds a diploma in Computer Science from the University of Passau, Germany, and a PhD in computer science from the Technische Universitaet Muenchen, Germany. For more than six years he has worked in industry. The focus of his research group at the university is on industrialization of software engineering and software operation. The main research areas are model-driven software development, semantic technologies as well as self-organizing systems to improve the automation of the software product lifecycle as well as the autonomy of software systems. He has published more than 100 scientific papers in the area of agent-based systems and agent-oriented software engineering, compiler construction, (semantic-enabled) model-driven software engineering and autonomous systems.

Bernhard Bauer