

Process model adaptation using semantic technologies

Florian Lautenbacher, Thomas Eisenbarth, Bernhard Bauer

Angaben zur Veröffentlichung / Publication details:

Lautenbacher, Florian, Thomas Eisenbarth, and Bernhard Bauer. 2009. "Process model adaptation using semantic technologies." In 2009 13th Enterprise Distributed Object Computing Conference Workshops, 1-4 Sept. 2009, Auckland, New Zealand, edited by Vladimir Tasic, 301-9. Los Alamitos, Calif.: IEEE. <https://doi.org/10.1109/edocw.2009.5331985>.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under the following conditions:

Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publizieren>



Process Model Adaptation using Semantic Technologies

Florian Lautenbacher, Thomas Eisenbarth and Bernhard Bauer
Programming Distributed Systems Lab
University of Augsburg
Augsburg, Germany
{*lautenbacher, eisenbarth, bauer*}@ds-lab.org

Abstract—Modeling and executing business processes with the help of software requires so much human work that the software life-cycles can not keep up with the fast changing demands of today’s global markets. Therefore, a mechanism is required to adapt these process models automatically. In this paper we introduce a novel approach for the adaptation of existing process models using semantic technologies, thereby building on semantic annotation of process models as well as on automatic planning approaches.

Index Terms—Process; Modeling; SBPM; Token; Semantic

I. 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 now. Formally defined process models establish the basis for automatic execution of processes in enterprises. This becomes increasingly important to compete in markets because of the possibility to gain shorter time to market, increased customer satisfaction 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.

These demands require that processes in a company as well as between several companies need to be adapted frequently with the result that often the process models are not actualized at the same time and therefore get outdated soon. Since these process models are adapted by hand, which is a time-consuming job, a mechanism to automatically adapt these process models is required when the underlying process 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 Semantic Business Process Management (SBPM) [1] 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 in 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 [2].

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. The adaptation process first searches in 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 result of planning is integrated into the process model and validated afterwards.

The remainder of this paper is organized as follows. In Section II we summarize some basics about (semantic) business process modeling and the automated planning of process models using our planner *SEMPA*. Section III describes the tasks that are necessary in order to adapt an existing process model which is exemplified in a proof of concept in Section IV. We show some related work in Section V, before we conclude with directions for further research.

II. 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 [3]. Other details that are normally part of a process model (roles, applications, etc.) are neglected in this paper, but could easily be integrated.

We view a process (or workflow) model formally as a directed graph \mathcal{G} which is denoted by $(\mathcal{N}, \mathcal{E})$, where \mathcal{N} is the set of nodes and \mathcal{E} the set of edges. \mathcal{N} consists of the disjoint subsets: \mathcal{N}_{start} , \mathcal{N}_{stop} , \mathcal{N}_{action} , \mathcal{N}_{fork} , \mathcal{N}_{join} , $\mathcal{N}_{decision}$ and \mathcal{N}_{merge} . The terms *action*, *decision*, etc. are thereby used in analogy to UML activity diagrams [4].

Each node $n \in \mathcal{N}$ has a set of incoming and outgoing edges denoted as $\mathcal{E}_{in}(n)$ and $\mathcal{E}_{out}(n)$ respectively. Furthermore, the

graph \mathcal{G} has to satisfy the following conditions:

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

As pointed out in [5], 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, behavioral, 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, $SemAn : \mathcal{N} \cup \mathcal{E} \rightarrow C_{Ont_s}$. $SemAn$ 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* $\mathcal{G}_{sem} = (\mathcal{N}_{sem}, \mathcal{E}_{sem}, Ont_s)$ with $\mathcal{N}_{sem} = \{(n, SemAn(n)) | n \in \mathcal{N}\}$ and $\mathcal{E}_{sem} = \{(n_{sem}, n'_{sem}) | n_{sem} = (n, SemAn(n)) \wedge n'_{sem} = (n', SemAn(n')) \wedge (n, n') \in \mathcal{E}\}$. C_{Ont_s} is a set of concepts of (possibly different) ontologies of the set of ontologies Ont_s ($C_{Ont_s} \subseteq Ont_s$).

An *ontology* $Ont \in Ont_s$ is a “(formal) explicit specification of a (shared) conceptualization” [6] 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 [6] actually speaks of “classes, relations, functions and other objects”, whereby current languages of the semantic web such as OWL [7] 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 semi-automated 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 $n \in \mathcal{N}$ is at least annotated with some kind of semantic input and output data. $SemIn(n) : \mathcal{N}_{sem} \rightarrow C_{Ont_s}$ ($SemOut(n)$ 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 checking for equivalence of concepts, inference techniques and so forth.

Therefore, we use parameters that are defined as (*label, domain, restriction*) for each input and output and 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 can similar to [8] 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 failure has happened 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 [4], is returned. An overview about these planning steps can also be seen in Figure 1.

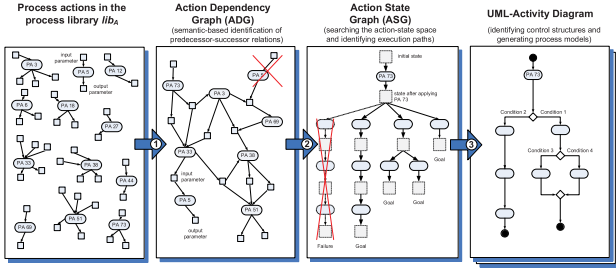


Fig. 1. Three Steps of SEMFA

III. 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. [9] or [10]): 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 [10].

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.

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. The process of identifying those process actions is often a manual task, but process mining techniques or monitoring tools could be used, too. 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.

The tasks for process model adaptation are the following:

- 1) Computation of the fragments that need to be adapted (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)

- 2) Identification of the initial state of each process fragment
- 3) Calculation of the goal state of each process fragment
- 4) Re-planning the process fragments considering the changed process actions
- 5) Integration of the planning result into the process model
- 6) Validation of the adapted process model

We will now elaborate each task in further detail.

A. Computation of the fragments that need to be adapted

All process actions that have been changed in the process library lib_A are searched for in all existing process models and are marked for 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 \mathcal{G} with $\mathcal{N}' \subseteq \mathcal{N}$ and $\mathcal{E}' = \mathcal{E} \cap (\mathcal{N}' \times \mathcal{N}')$ such that there exist edges $e, e' \in \mathcal{E}$ with $\mathcal{E} \cap ((\mathcal{N} \setminus \mathcal{N}') \times \mathcal{N}') = \{e\}$ and $\mathcal{E} \cap (\mathcal{N}' \times (\mathcal{N} \setminus \mathcal{N}')) = \{e'\}$ (cp. [11]). 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 \mathcal{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 [12]. We shortly summarize this algorithm here.

This algorithm builds on a token propagation mechanism. Tokens and token algorithms have already been described elsewhere, e.g. in [13]. The token-flow is calculated in two steps: first, single tokens propagate through the graph and second, tokens from the same origin are re-combined.

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 $\mathcal{N}_{decision}$ or \mathcal{N}_{fork}), all of the gateway's outgoing edges $e \in \mathcal{E}_{out}$ are labeled with the same token: At nodes with out 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 (\mathcal{N}_{merge} or \mathcal{N}_{join}), \mathcal{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

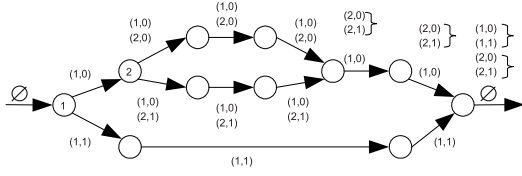


Fig. 2. Example of Converging Token Flow

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 (recombination). Components can be derived by matching pairs of edges with equal token sets.

Different to [13], 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.

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:

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

Each edge in the graph is assigned a subset of \mathbb{T} by the *token labeling function* $t : \mathcal{E} \rightarrow \mathcal{P}(\mathbb{T}) \cup \{\perp\}$. Token creation occurs at nodes with $|\mathcal{E}_{out}(n)| > 1$. Tokens propagate and unite at nodes with $|\mathcal{E}_{in}(n)| > 1$. 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 $e_1, e_2 \in \mathcal{E}, e_1 \neq e_2 : t(e_1) = t(e_2)$ holds, they mark the beginning and the end of a component \mathcal{C} , i.e. $\{e_1, e_2\} = \{\text{source}(\mathcal{C}), \text{sink}(\mathcal{C})\}$.

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 \mathcal{N}_{start} , call the *tokenFlow* function with an initial labeling $t(\mathcal{N}_{start}) := \emptyset$. The function *pick* ($e \in \mathcal{E} \mid \text{prop} : e \rightarrow \mathbb{B}$) nondeterministically selects an element e from \mathcal{E} , that meets a required property, i.e. $\text{prop}(e) = \text{true}$. 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

Algorithm 1 Calculate Token-flow

Require: $t : \mathcal{E} \rightarrow \mathcal{P}(\mathbb{T}) \cup \{\perp\}$ // initial marking
 $O \subseteq \mathcal{N}$ // to-do set of nodes
Ensure: $t : \mathcal{E} \rightarrow \mathcal{P}(\mathbb{T})$

```

1: processNode( $n \in \mathcal{N}$ ) {
2:    $\mathcal{P}(\mathbb{T}) M := \bigcup_{u \in t(\mathcal{E}_{in}(n))} u$ ;
3:   if  $|\mathcal{E}_{out}(n)| = 1$  then
4:      $t(\text{elt}(\mathcal{E}_{out}(n))) := M$ ;
5:   else
6:     int  $i := 0$ ;
7:     for all  $e \in \mathcal{E}_{out}(n)$  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( $n \in O \mid t(\mathcal{E}_{in}(n)) \neq \perp$ );
16:    processNode( $n$ );
17:     $O = O \setminus n$ ;
18:    timestamp( $n$ );
19:  end while
20:  for all  $e \in \mathcal{E}$  do
21:    for all  $n \in \text{first}(t(e))$  do
22:      if  $t(e) \supseteq \{(n, i) \mid i < |\mathcal{E}_{out}(n)|\}$  then
23:         $t(e) = t(e) \setminus \{(n, i) \mid i < |\mathcal{E}_{out}(n)|\}$ 
24:      end if
25:    end for
26:  end for
27: }
```

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 [12].

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 $\mathcal{O}(|\mathcal{N}| + |\mathcal{E}|)$.

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 re-planning in the further steps.

B. Identification of the initial state of each process fragment

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

Algorithm 2 Compute Initial State

Require: Strongly connected component C

```
1: computeInitialState( $C$ ) {
2:  $n := \text{getFirstNode}(C)$ 
3:  $init := SemIn(n)$  // initial state
4:  $removeList := SemOut(n)$ 
5: for all  $n := \text{getFollowingNodeInComponent}(n, C)$  do
6:   for all Parameter  $P \in SemIn(n)$  do
7:     if  $\neg(P \in init) \wedge \neg(P \in removeList)$  then
8:        $init = init \cup P$ 
9:     end if
10:   end for
11:    $removeList := removeList \cup SemIn(n)$ 
12: end for
13: return  $init$ 
14: }
```

big process model, probably 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 $init := SemIn(n_1) \cup \bigcup_{i=2..|\mathcal{N}|} (SemIn(n_i) \setminus SemOut(n_{i-1}))$ (compare Alg. 2). The complexity of this algorithm is $\mathcal{O}(|\mathcal{N}| + |P|)$.

C. Calculation of the goal state of each process fragment

Goal states $= G_1, G_2, \dots, G_k$ specify the set of $k \in \mathbb{N}$ different (sets of) parameters $G_x \subseteq P$ (with $x = 1, \dots, k$) which shall be reachable in a feasible solution. The calculation of the goal state works 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 ($\bigcup_{i=1..|\mathcal{N}|} SemOut(n_i)$).

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

D. Re-planning the process fragments considering the changed process actions

Now that we have the initial state and the goal state we can re-plan the process fragment. The automatic planning of business processes has already been described in [2], but did not specify how the adaptation of process models can be realized. Shortly summarized the planning steps consider all existing process actions $n \in \mathcal{N}_{action}$ that have been stored

in the process library lib_A (where the changed process actions have been stored, too). SEMPA first computes the dependencies between the process actions n considering their semantic input ($SemIn(n)$) and output descriptions ($SemOut(n)$). Afterwards, the planning computes an action-state-graph which has two partitions: the process actions \mathcal{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 non-deterministic planning [14] with initial state uncertainty [15]. This action-state-graph is the basis to build the process model in the last step and to identify control structures such as e.g. $\mathcal{N}_{decision}$ or \mathcal{N}_{join} . There is no estimation regarding complexity of the aforementioned step as it is not examined in further detail here. We refer to [2] for further details.

E. Integration of the planning result into the process model

The result of the planning is first put into an own embedded subprocess (available e.g. as StructuredActivityNode in UML) and this subprocess is connected with edges that before were the entry and exit edges of the process fragment. In a second step this embedded subprocess can be removed as well as \mathcal{N}_{start} and \mathcal{N}_{stop} of the subprocess and the rest of the content can directly be connected with e_{entry} and e_{exit} of the subprocess. 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 basically can be achieved by depth-first search that results in complexity of $\mathcal{O}(|\mathcal{N}| + |\mathcal{E}|)$.

F. Validation of 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 [12] again that is using the already computed components as well as the re-planned fragments.

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

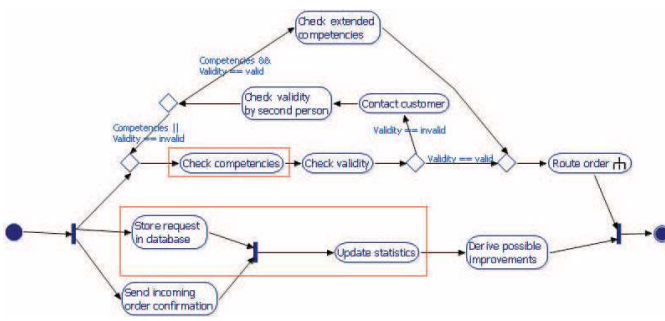


Fig. 3. Proof of Concept from the Financial Area with Marked Process Actions

this paper. That is why we cannot give the exact costs for the entire process.

IV. PROOF OF CONCEPT

For a proof of concept 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 (see e.g. [16]). 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 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):

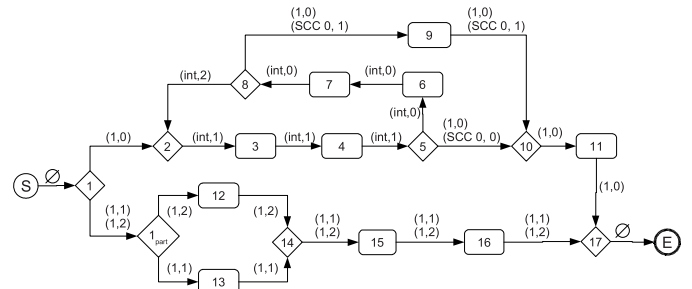


Fig. 4. Tokens in the Graph

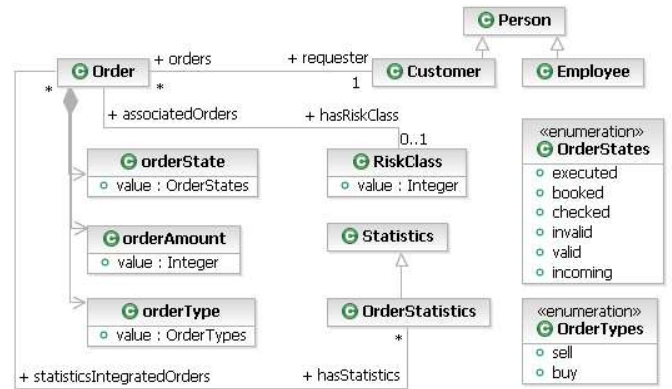


Fig. 5. Ontology in the Proof of Concept

- *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.

As first task 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

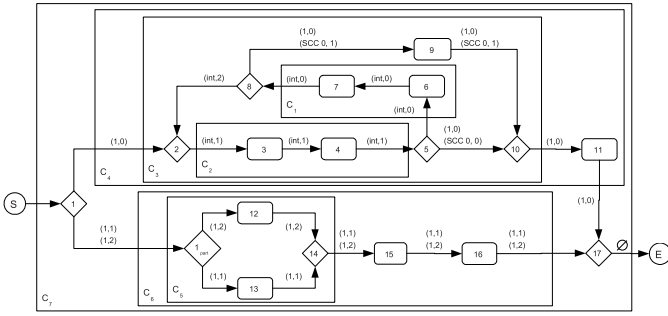


Fig. 6. Discovered Components

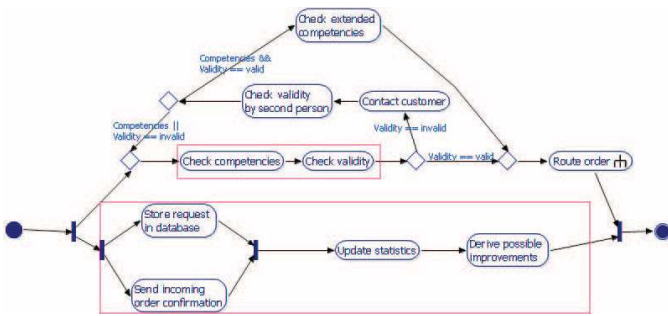


Fig. 7. Determined Process Fragments for Planning

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.

As second 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 $SemIn(Check\ competencies) \cup SemIn(Check\ validity) \setminus SemOut(Check\ competencies)$. 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 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 third 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 C_2 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*.

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 to calculate loops. 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 [17]. For the calculation of the components using the introduced token-flow algorithm we integrated the Workflow-Codegeneration framework that has been described in [18].

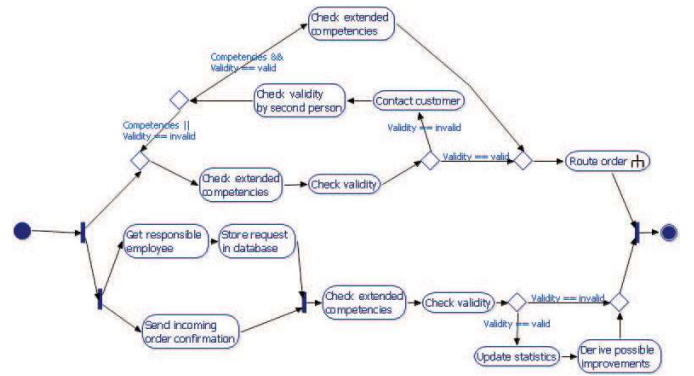


Fig. 8. Final Planning Result

V. 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 (e.g. [19]). In [20] 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 $\mathcal{N}_{decision}$ and \mathcal{N}_{fork} .

Koschmider shows in her dissertation [21] 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 can not be adapted.

In [22] 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, [23] shows how business rules and processes can be combined during modeling.

[24] describes an adaptation of process models when the underlying web services have changed. Thereby, business processes need to be captured in OWL-S [25] 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 [26], [27]. 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 (see e.g. [28], [29], [30], [31]). 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 [32]) may be alleviated.

VI. 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. Thereby, we use semantic technologies and perform a re-planning of identified process fragments.

In the future we work on the identification of duplicates in the re-planned 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 as well as considering business rules in addition to process actions).

REFERENCES

- [1] M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel, "Semantic business process management: A vision towards using semantic web services for business process management," in *Proceedings of IEEE ICEBE, Beijing, China*, October 2005, pp. 535–540.
- [2] M. Henneberger, B. Heinrich, F. Lautenbacher, and B. Bauer, "Semantic-based planning of process models," in *Proceedings of Multikonferenz Wirtschaftsinformatik (MKWI)*, February 2008.
- [3] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases 14*, vol. 3, pp. 5 – 51, 2003.
- [4] OMG, "Unified modeling language (UML) superstructure, version 2.1.2." OMG Specification, November 2007. [Online]. Available: <http://www.omg.org/docs/formal/05-07-04.pdf>
- [5] O. Thomas and M. Fellmann, "Semantic business process management: Ontology-based process modeling using event-driven process chains," *International Journal of Interoperability in Business Information Systems*, vol. 2 (1), 2007.
- [6] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5 (2), pp. 199–220, 1993.
- [7] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, "OWL2: The next step for OWL," *Journal on Web Semantics*, vol. 6, no. 4, pp. 309 – 322, November 2008. [Online]. Available: <http://www.w3.org/TR/owl2-syntax/>, asof2008-04-16
- [8] S. Degwekar, H. Lam, and S. Y. Su, "Constraint-based brokering (CBB) for publishing and discovery of web services," *International Journal on Electronic Commerce Research*, vol. 7, no. 1, pp. 45 – 67, 2007.
- [9] O. Hanseth, E. Monteiro, and M. Hatling, "Developing information infrastructure: The tension between standardization and flexibility," *Science, Technology and Human Values*, vol. 11, no. 4, pp. 407–426, 1996.
- [10] J. Gebauer and F. Schober, "Information system flexibility and the cost efficiency of business processes," *Journal of the Association for Information Systems*, vol. 3, pp. 122–147, 2006.
- [11] J. Vanhatalo, H. Völzer, and F. Leymann, "Faster and more focused control-flow analysis for business process models through SESE decomposition," in *ICSOC*, 2007.
- [12] M. Götz, S. Roser, F. Lautenbacher, and B. Bauer, "Token analysis of graph-oriented process models," in *Proceedings of DDBP 2009, Auckland, New Zealand*, September 2009.
- [13] OMG, "Business Process Modeling Notation Specification, Version 1.2." formal/09-01-03, January 2009.
- [14] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning*. Elsevier, San Francisco, 2004.
- [15] B. Bonet and H. Geffner, "GPT: A tool for planning with uncertainty and partial information," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001, pp. 82–87.
- [16] R. Malhotra and D. Malhotra, "The impact of internet and e-commerce on the evolving business models in the financial services industry," *International Journal of Electronic Business*, vol. 4, no. 1, pp. 56 – 82, 2006.
- [17] F. Lautenbacher, "Execute your processes with Eclipse JWT," in *Proceedings of JAX / SOACon / Eclipse Forum Europe, Mainz, Germany*, April, 20 - 24 2009.
- [18] S. Roser, F. Lautenbacher, and B. Bauer, "Generation of workflow code from DSMs," in *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling, Montreal, Canada*, October 2007. [Online]. Available: http://www.informatik.uni-augsburg.de/lehrstuehle/swt/vs/publikationen/veroeffentlichungen/2007_DSM/
- [19] M. Hepp and D. Roman, "An ontology framework for semantic business process management," in *Proceedings of Wirtschaftsinformatik, Karlsruhe, Germany*, February – March 2007.
- [20] G. Koliadis and A. Ghose, "Verifying semantic business process models in inter-operation," in *Proceedings of IEEE SCC 2007, Salt Lake City, Utah, USA; July, 9-13, 2007*.

- [21] A. Koschmider, "ähnlichkeitsbasierte modellierungsunterstützung für geschäftsprozesse," Ph.D. dissertation, Universität Fridericiana zu Karlsruhe, Germany, 2007.
- [22] T. Graml, R. Bracht, and M. Spies, "Patterns of business rules to enable agile business processes," *Enterprise Information Systems*, vol. 2, no. 4, pp. 385–402, November 2008.
- [23] M. Milanovic, D. Gasevic, and G. Wagner, "Combining rules and activities for modeling service-based business processes," in *EDOC, Munich, Germany*, 2008.
- [24] U. K. Tripathi, K. Hinkelmann, and D. Feldkamp, "Life cycle for change management in business processes using semantic technologies," *Journal of Computers*, vol. 3, no. 1, pp. 24–31, January 2008.
- [25] D. Martin, M. Paolucci, and M. Wagner, "Bringing semantic annotations to web services: OWL-S from the SAWSDL perspective," in *Proceedings of ISWC/ASWC, Busan, Korea*, November 2007.
- [26] L. T. Ly, S. Rinderle, and P. Dadam, "Semantic correctness in adaptive process management systems," in *BPM 2006, Vienna, Austria*. Springer-Verlag, September, 5–7 2006, pp. 193–208.
- [27] L. Ly, S. Rinderle, and P. Dadam, "Integration and verification of semantic constraints in adaptive process management systems," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 3–23, 2008.
- [28] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for web service composition using SHOP2," *Journal of Web Semantics 1*, vol. 4, pp. 377 – 396, 2004.
- [29] Q. Lang and S. Su, "AND/OR graph and search algorithm for discovering composite web services," *International Journal of Web Services Research 2*, vol. 4, pp. 46 – 64, 2005.
- [30] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi, "Automated synthesis of composite BPEL4WS web services," in *3rd International Conference on Web Services*, 2005, pp. 293 – 301. [Online]. Available: <http://astroproject.org/downloads/dissemination/ICWS05.pdf>
- [31] H. Meyer and M. Weske, "Automated service composition using heuristic search," in *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*, Vienna, Austria, 2006, pp. 81 – 96.
- [32] K. Haniewicz, M. Kaczmarek, and D. Zyskowski, "Semantic web services applications - a reality check," *Journal Wirtschaftsinformatik*, vol. 50, no. 1, pp. 39–45, 2008.