Semantic-Based Planning of Process Models

Matthias Henneberger, Bernd Heinrich Florian Lautenbacher, Bernhard Bauer

Abstract: Process modelling has proved to be a powerful instrument to describe and manage the increasingly complex processes within and across enterprises. Yet, it requires a significant amount of manual work to create, adapt and maintain process models. This workload could be reduced if the process models are automatically generated and adapted. Semantic Business Process Management in combination with planning approaches can contribute to the solution. In this paper we describe that existing planning algorithms show drawbacks for this application and therefore introduce an innovative algorithm that is suitable for the semantic-based planning of process models.

1 Introduction

Process modeling has proved to be a powerful instrument to describe the increasingly complex processes within and across enterprises in order to realize them by means of application systems as well as for communication and training purposes. In the past twenty years a number of process modeling approaches have been developed. For instance processes can be described using Event-driven Process Chains (EPC) or UML activity diagrams and can be stored in enterprise-wide process libraries. However, process modeling and optimization approaches still have one major drawback. Setting up process models is a time consuming job. Nevertheless, changing customer requirements, new jurisdiction, etc. make it necessary to adapt and maintain business processes frequently. A semantic annotation of process models as envisioned in the research area Semantic Business Process Management in combination with existing planning approaches can solve this drawback and enable a (semi-)automatic, semantic-based planning of process models.

In this paper we introduce SEMPA, an approach including an algorithm suited for the automatic planning of process models. We specifically address the question, how established AI planning approaches can be used as a basis to develop an innovative planner for the semantic-based planning of process models. To this end we will provide a technical definition of the planning problem and we will describe how the specific characteristics of the planning problem are considered within the SEMPA approach.

The rest of the paper is organized as follows: In section two we show how the planning of process models can be related to Semantic Business Process Management and define our prerequisites. In section three we introduce the SEMPA approach, before we go into more details about the automatic planning of process models. Our approach consists of three steps: we describe in further detail the actual planning part (second step) and its

result as well as the extraction of process models from this result (third step). Section four is dedicated to related work in AI and web service composition, before section five concludes with our considerations and provides an outlook for future research.

2 Semantic Business Process Management and Basic Problem Setting

Recently, new research activities in the area of Semantic Business Process Management (SBPM) based on the vision of [HL+05] have emerged (see e.g. [BK+06, BE+06, HD07]). One major goal of SBPM is to reach a higher level of automation in the creation and adaptation of process models and execution of processes by means of their semantic annotation. This requires that terms used in the models are technically described with concepts of an ontology (e.g. using OWL). As pointed out by [TF07], the semantics of meta-model elements for process modeling and their relations are already defined by well established approaches for process and enterprise modeling like ARIS. However, the terms used to specify individual model elements (e.g. the name of a particular function or an input parameter) and their semantics are still left to the modeler. Problems in comprehension and ambiguities are the consequence of inconsistently used terms – either the same term describes different elements of the problem context or different terms point to the same elements of the problem context. Especially in practice this leads to discussions or manual enhancement of the process models requiring a huge amount of work. 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.

Most approaches in SBPM (compare [TF07]) concentrate on the annotation of process models. In contrast we focus on the (semi-)automated planning of process models enabled by the semantic annotation. Based on a given problem description feasible process models should be automatically generated based on a set of semantically annotated process actions (constituting one major element of process models) or sub-processes. Thereby the (re-)design of processes through process models respectively should be accelerated. This conforms to [WM+07] describing the lifecycle of a semantic business process and pointing out that autocompletion of process models is an important issue in SBPM. However, we do not only provide the user with possibilities for an autocompletion of fragments of a process model (as in [BK+06]), but also with a (semi-)automatic planning of process models which only requires human interaction when the planned process models need to be evaluated by the modeler regarding business aspects that have not been considered for planning (e.g. cost or resources).

Whereas some approaches in SBPM suggest a comprehensive conceptualization of all model- and meta-model elements of a process model [TF07], we choose a less restrictive approach for the annotation of process actions and sub-processes, which is similar to semantic web service standards (e.g. SAWSDL). We assume that each process action is described by a unique name, a set of input parameters it needs to be executed and a set of output parameters it provides after execution. Input and output parameter names are defined as classes in an ontology. Moreover, for each atomic input and output parameter a domain is specified in the ontology, i.e. the set of feasible values the parameter can

adopt. The domain is either given by a primitive data type (e.g. Boolean or integer) or an ontological class; or it is an enumeration of predefined instances. In reality, process actions may require input parameters of a particular value or within a certain range. Likewise a process action may generate an output parameter having a value that is restricted to a certain range. Thus the specification of the input and output parameters of a process action also comprises certain restrictions concerning their values.

In our prototypical application process actions and their input and output parameters are directly described in a process modeling tool with regard to the ontology which is implemented in OWL. We will not elaborate on implementation details in this paper (for OWL we refer to [DG04]), but introduce a more abstract notation that is appropriate for planning later on.

3 Automatic Planning of Process Models

In this section we firstly outline our approach at a whole, before we classify the planning part into existing planning approaches and describe it in further detail.

3.1 The SEMPA-Approach

Within the project SEMPRO[1] an approach called SEMPA (SEMantic-based Planning Approach) has been developed that proceeds in three steps:

- In the **first step**, a semantic matching between input and output parameters of process actions takes place by evaluating the semantic relations in the ontology[2]. Besides the trivial case of an identity relation (e.g. an output parameter of one process action can be used as an input parameter of another process action because parameters are identical) also other relations are considered. For instance parameters connected via a sub-class-of relation in the ontology may be as well compatible under certain circumstances.[3] The semantic information about input and output parameters is stored in an action dependency graph (ADG) that is employed in the following steps. This ADG only includes process actions that lead to one of several given goals. Thus semantic reasoning is mainly done in the first step and redundant and time-consuming analysis can be avoided.

- In the **second step**, planning is accomplished by a search in the action-state space to achieve the given goals. The algorithm currently implements a forward-search starting in the initial state. In the course of planning applicable process actions are

---

[1] The SEMPRO project is supported by the German Research Foundation (DFG).
[2] Here, we assume that there exists only one ontology. For several ontologies, ontology mapping techniques (compare [KS03]) could be applied.
[3] Consider the case that one process action requires an *order* as an input parameter and another process action produces an output parameter *stock order* (*stock order* being a sub-class of *order* in the ontology), then both parameters match.

derived from the ADG. We call the resulting plan an action state graph (ASG). It forms a (language independent) basis for deriving process models in the next step.

- Finally, in the **third step** process models are derived from the ASG by identifying control structures and considering the syntactical elements of a concrete process modeling language. Thereby, different process modeling languages can be supported at a time without adapting the fundamental search algorithm in the second step. Currently, we are focusing on UML activity diagrams but the intention is to extend the approach to other languages, e.g. Petri-Nets.

As the focus in this paper is on the connection of SBPM and planning, we will now provide a technical definition of the planning problem that has to be solved in the second step of the SEMPA approach.

3.2 Definition and Classification of the Planning Problem

As already mentioned, process actions are semantically described by specifying their input and output parameters with respect to an ontology. We will use the term parameter in the following to denote the state variables for our planning problem (for state-variable representations of planning problems and differences in comparison to a STRIPS language or derivatives see e.g. [BN93], [GNT04]). For the planning of process models we abstract from an individual process execution and therefore the realizations of parameter values (and thus the current state) are not determined at the moment of planning. This conforms to *non-deterministic* planning problems [GNT04]. Likewise in the initial state parameters are not fully determined either, which can be regarded as a form of *initial state uncertainty* [BG01]. In order to account for non-determinism and initial state uncertainty, parameters are not assigned individual values but so called restrictions, i.e. sets of values that are currently conceivable for these parameters. This leads us to the following definitions:

Definition 1: Let P be the set of all parameters. A parameter $p \in P$ can be either an atomic parameter or a composite parameter. An atomic parameter is defined as a tuple $p:=(l_p, dom_p, r_p)$, where l_p is the parameter name, dom_p is the domain, i.e. the set of feasible values according to the data type of the parameter, and $r_p \subseteq dom_p$ is the restriction. A composite parameter is a tuple $p:=(l_p, \{p_1, p_2, ..., p_m\})$, where $\{p_1, p_2, ..., p_m\}$ is a set of parameters (m>1).

Definition 2: A process action (or sub-process) a is defined as $a:=(name_a, In_a, Out_a)$, where $name_a$ is the name of the process action, $In_a \subseteq P$ is a set of input parameters and $Out_a \subseteq P$ is a set of output parameters.

We define that each process action is described by a unique name, a set of input parameters it needs to be executed and a set of output parameters it provides after execution. Restrictions thereby form a rather intuitive way (from a process modeling perspective) to express certain preconditions and effects. This is different to semantic web services where preconditions and effects are used instead of restrictions on inputs and outputs. Note, however, that in contrast to preconditions and effects, restrictions are

always bound to single parameters. We can now specify the planning domain and the planning problem:

Definition 3: The planning domain is defined as $D=(P, S, lib_A, \gamma)$, where P is the set of parameters, S is a set of process states, lib_A is a set of available process actions or sub-processes and γ is a state-transition function. A process state $s \in S$ is defined as a set of parameters, i.e. $s \subseteq P$ and $S \subseteq 2^P$. The state transition function $\gamma : S \times lib_A \rightarrow S$ maps pairs of process states and process actions into the set of process states.

It is important to mention that a process state is constituted by the restrictions that currently hold for the parameters rather than by individual values of the parameters. Similar to [PB02] a process state could thus be interpreted as a kind of knowledge base capturing the knowledge about the currently available parameters. In this respect a *process state* needs to be clearly distinguished from the *state of the world* that refers to an individual situation at execution time. A process state describes different conceivable states of the world. A process action deterministically maps one process state into another process state.

Definition 4: The planning problem is defined as $Prob=(D, Init, Goals)$, where $\emptyset \neq Init \in S$ is the initial state and $Goals=\{G_1, G_2, ..., G_n\}$ are representing the goals of the planning problem. Each goal G_j $(j=1, ..., n)$ is again defined as a set of parameters, i.e. $G_j \subseteq P$.

The intention of the second step within the SEMPA approach is to plan an action state graph that is suited to derive different feasible process models (that can afterwards be evaluated). In a feasible process model each goal should be reachable, i.e. it is possible to specify several (conflicting) goals for one process model (which are then achieved in different branches of the process model). Consequently, planning is accomplished until *all* paths from the initial state to *any* of the specified goals are explored.

Case study: In the following we will illustrate our approach by a small case study of the financial services industry. We consider a situation where a new process model for the execution of orders (e.g. stock order) is needed. The overall process input is an order that has been entered by the customer. The overall process output is again the order that has to be executed now. Figure 1 depicts the corresponding initial state and the set of goals in the notation introduced above (for the sake of simplicity, we only consider one goal, thus there is only one element in the set of goals). The *order* is a composite parameter having a positive *order amount,* an *order type* which is either *buy order* or *sell order,* and an *order state*, which needs to be *entered* in the initial state and *executed* in the goal (*state* thereby represents a set of conceivable order states that are not enumerated explicitly here but nevertheless are defined in the ontology).

Init={(order, {(order state, state, {entered}), (order amount ,int+, int+), (order type, {buy order, sell order}, {buy order, sell order})})}

Goals={{(order, {(order state, state, {executed}), (order amount ,int+, int+), (order type, {buy order, sell order}, {buy order, sell order})})}}

Figure 1: Initial state and Goals in the case study

Several (semantic annotated) process actions are available in a process library and can be used for automatic planning. The specification of these process actions provides Figure 2.[4] Input parameters have been marked in bold and are italicized; output parameters are printed in grey. The process action *validate order* for instance has exactly one (composite) input parameter *order* which is at the same time the only output parameter. Input parameter restrictions demand an *order state* that is *entered*; the process action on the other hand provides an *order* whose *order state* is restricted to *valid* or *invalid*.

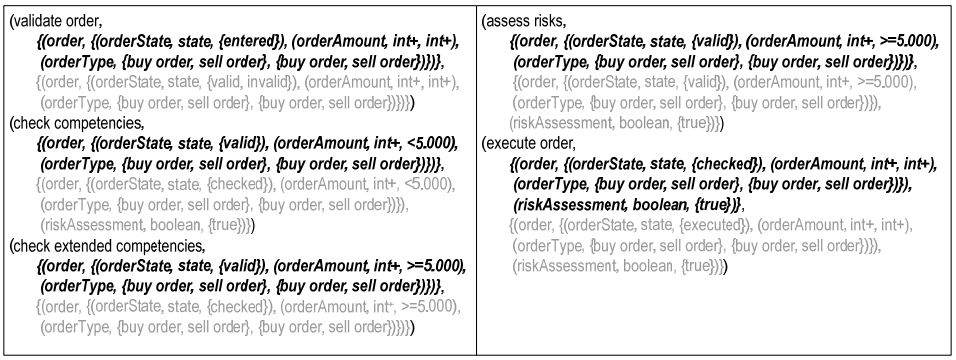

Figure 2: Specification of process actions

3.3 The Planning Algorithm within the SEMPA Approach

The algorithm conducts a forward search in the space of process states. Beginning in the initial state applicable process actions are identified. The ADG as result of the first step is used to derive process actions that are applicable for a certain set of available parameters in a process state (considering as well semantic relations). Thereby, we differentiate between applicable and strongly applicable process actions[5]:

Definition 6: A process action $a:=(name_a, In_a, Out_a)$ is *applicable* to a process state s if: $\forall\ (l_{in}, dom_{in}, r_{in}) \in In_a\ \exists\ (l_p, dom_p, r_p) \in s \wedge l_{in} \cong l_p \wedge dom_{in} = dom_p\ \wedge r_p \cap r_{in} \neq \emptyset$.

Definition 7: A process action $a:=(name_a, In_a, Out_a)$ is *strongly applicable* to a process state s if: $\forall\ (l_{in}, dom_{in}, r_{in}) \in In_a\ \exists\ (l_p, dom_p, r_p) \in s \wedge l_{in} \cong l_p \wedge dom_{in} = dom_p\ \wedge r_p \subseteq r_{in}$.

Definition 6 describes a necessary condition that needs to be met so that a process action a can actually be executed in a process state s. All input parameters are available in s and the restriction of each parameter (i.e. the set of possible values) does not contradict the restriction required by a for that parameter. The parameters of the process state s and the input parameters of a are semantically compared ($\cong$) whether they are

[4] Figure 2 only contains the process actions relevant for the following text. Process actions that are separated out in the first step and are therefore not part of the final solution are not depicted here.
[5] These definitions (as well as the following ones) assume only atomic parameters in order to avoid writing overhead. Obviously the definition can be easily adapted to composite parameters.

equivalent, sub-class-of or one part of the other. However, there may still be situations where executing *a* is not possible (due to the restrictions). Definition 7 in contrast phrases a sufficient condition. All input parameters are available in *s* and at the same time, the restriction of each parameter is a subset of the restriction required by *a* for that parameter. To put it in other words, a process action *a* is *applicable* in a process state *s*, if it can be executed in at least one of the different states of the world represented by *s,* and it is *strongly applicable* if it can be executed in all states of the world represented by *s*. Within the planning algorithm, we are first solving the "relaxed planning problem" conveyed by definition 6, which results into an ASG. A detailed analysis of restrictions (implied by definition 7) is deferred to the third step, when process models are derived from the ASG. As we will see later on, this approach is advantageous as it avoids unnecessary branches during planning.

After applying a process action *a* to a process state *s*, a new process state is determined with the state-transition function. Output parameters that already existed in *s* are "updated" considering the new restrictions and output parameters that did not exist before are added to the process state, which is conveyed by the following definition:

Definition 8: The state-transition function is defined as: $\gamma(s,a):=\{p_1, p_2, ..., p_o | p_j \in Out_a \vee p_j \in s\}$ with $j=1,...,o$, meaning that p_j is an output parameter of *a* or, if *a* does not have such an output parameter, then $p_j \in s$.

When a new process state is determined, it is evaluated whether it satisfies one of the specified goals. Again, we differentiate:

Definition 9: A process state *s satisfies* $Goal_j$ if $\forall g:=(l_g, dom_g, r_g) \in Goal_j \; \exists p:=(l_p, dom_p, r_p) \in s \wedge l_g \cong l_p \wedge dom_g = dom_p \wedge r_g \cap r_p \neq \emptyset$.

Definition 10: A process state *s strongly satisfies* $Goal_j$ if $\forall g:=(l_g, dom_g, r_g) \in Goal_j \; \exists p:=(l_p, dom_p, r_p) \in s \wedge l_g \cong l_p \wedge dom_g = dom_p \wedge r_p \subseteq r_g$.

Obviously, definition 9 again conveys a necessary but not sufficient condition as there may still be states of the world represented by process state *s*, that do not meet the goal. In this case *s* needs to be decomposed into the states of the world that do meet the goal and those that do not meet the goal. Planning then proceeds with the latter ones.

On this basis we can now describe the planning algorithm. Planning is accomplished by a depth-first search. Figure 3 illustrates how the graph is iteratively built; the symbols marked dark grey denote the current planning state in each case. Process states are only described in the left graph. Parts of a process state that are marked in grey indicate changes in comparison to the previous process state. Because of the depth first search the first path is explored until a goal is reached (more precisely until a process state strongly satisfies a goal) or until a failure occurs (graph 3a). Then a backtracking takes place and the second path is explored (graph 3b) until we ultimately get a complete graph (3c).

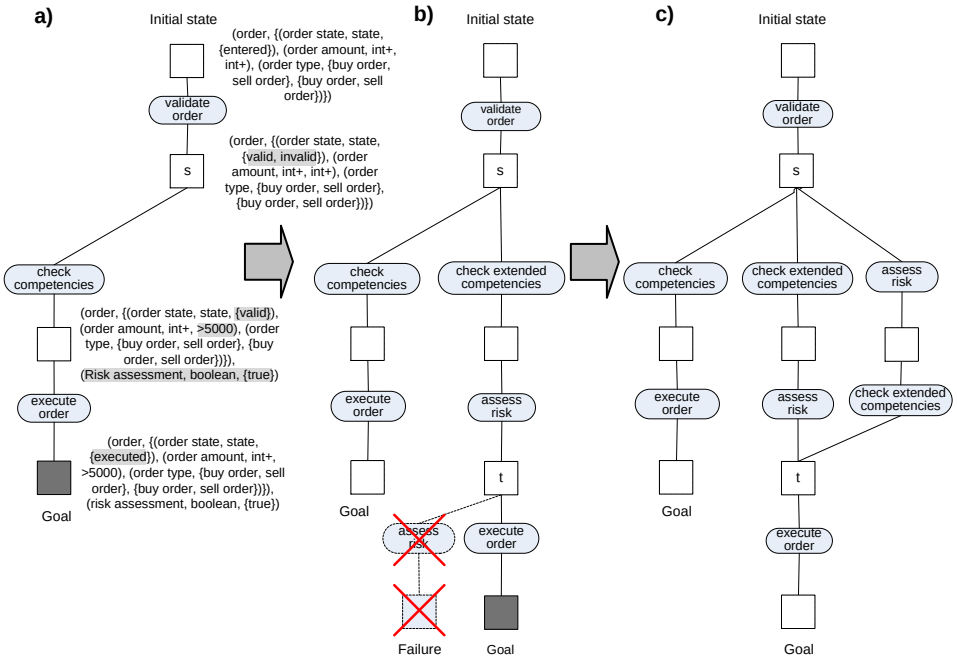

Figure 3: Building the planning graph

A specific characteristic of the algorithm is that it does not stop planning, if one of the given goals is reached. Instead a complete graph is build, comprising all possible paths from the initial state to the goals. This is necessary, as elaborated in the previous section. Nevertheless, the planning algorithm terminates. As we do not have conditional effects, a finite number of process actions can only produce a finite number of process states (see [He02]). As we moreover separate out infinite loops in the ASG (see below), it is guaranteed that the algorithm terminates. Additionally, an explosion of the graph is avoided. On the one hand the search space is already limited to the process actions in the ADG. On the other hand, we included measures in the algorithm that are suited to further reduce the search space. We will only describe the basic ideas:

- Firstly, process states on the currently explored path are saved. Whenever a new process state is reached (in the same path), it is tested whether there is already an identical process state. This indicates a cycle and the algorithm initiates a backtracking (see [GNT04]). The graph b) in Figure 3 provides an example. In the process state *t*, the process action *assess risk* can be applied again which leads to an identical process state. This is recognized by the algorithm as a failure.

- Secondly, process states satisfying a goal are saved. Whenever an identical process state is derived (in another path), then the algorithm does not need to explore this part of the search space again because the same state will always lead to the same sub-graph. Therefore a link is established to the already explored process state and

again a backtracking is initiated. This is illustrated in graph c) in Figure 3, where the two paths in the middle lead to an identical process state (*t*).

- Thirdly, if a failure occurs (e.g. a cycle is detected or no process actions are applicable), the process state is saved and a failure entry is stored in a failure log. For a new process state, it is tested, whether any of the failures in the failure log apply to this process state (cp. [BC+01] for a detailed description of this feature). In this case again a backtracking is initiated.

3.4 Extracting the Process Models from the Plan

Based on the ASG feasible process models are derived in the third step. Currently, we focus on UML activity diagrams. They provide a vivid and plain representation of processes and at the same time are based on a sufficiently technical definition (at least since version 2.0). The idea here is to identify control structures in the ASG that are essential elements of process models. So far we are able to consider the control structures *sequence*, *parallel split*, *synchronization*, *exclusive choice* and *simple merge* (compare [VT+03] for these and other control structures).

In the ASG exclusive choices and parallel splits cannot be identified directly, because the semantic of the branches is still unclear. Consider Figure 3c: There are three process actions applicable in *s*: *check competencies*, *check extended competencies*, and *assess risk*. Some process actions can be executed in parallel (e.g. *check extended competencies* and *assess risk*), some are mutual exclusive (e.g. *check competencies* and *assess risk*). In order to solve this ambiguity, process states are iteratively decomposed, so that in the resulting process states each process action is either strongly applicable or not applicable at all. This is illustrated in Figure 4. The process state *s* is divided into mutual exclusive (regarding the restrictions) sets of parameters determining different process states. We used UML decision node symbols to indicate a decomposition step and again marked changing parts of a process state in grey. The first decision node separates cases where the *order state* is *invalid* (in this case no process action is strongly applicable) from cases where the *order state* is *valid*. The second decision node differs cases where the *order amount* is less than 5000 (*check competencies* is strongly applicable) from cases where the *order amount* is greater or equal 5000 (*check extended competencies* and *assess risk* are strongly applicable).

The decomposition has similarities with the application of sensing actions in planning ([BG00] and [WA+98]). Sensing actions in general are applied to further investigate the current state of the world. Yet, the problem setting in our case is different. Firstly, most AI planning approaches assume that sensing is accomplished only with regard to Boolean variables (e.g. using binary decision trees). Secondly, we already know the set of applicable process actions and thus can decompose goal-oriented with respect to these process actions. The latter one in fact is a major advantage of doing this analysis in a subsequent step and not in the course of planning.

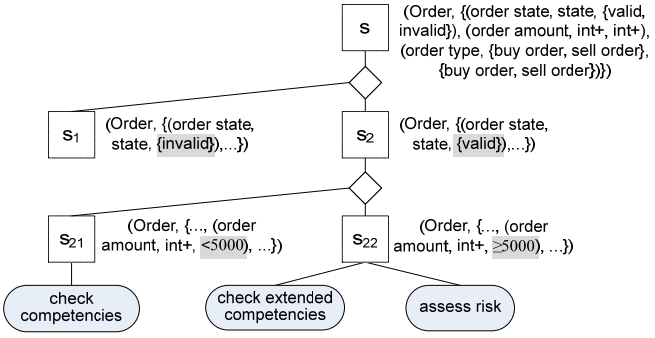

Figure 4: Decomposition of process states

After decomposing process states in the ASG, UML activity diagrams can be derived. Due to lack of space, we can only give a rough idea of the following steps (cp. *Figure 5*): The already identified exclusive choices (and the corresponding UML decision nodes) are completed by extracting the conditions (the guards) from the succeeding process states. Parallel splits as well as simple merges and synchronization merges are identified and the corresponding UML symbols are added. The initial state is replaced with an initial node and each process state representing a goal is replaced by a final node. To avoid "loose ends" process states having no outgoing edge are directly connected to a final node (for instance process state s_1). Finally, all remaining process states are deleted and previous process actions are directly connected to succeeding process actions.

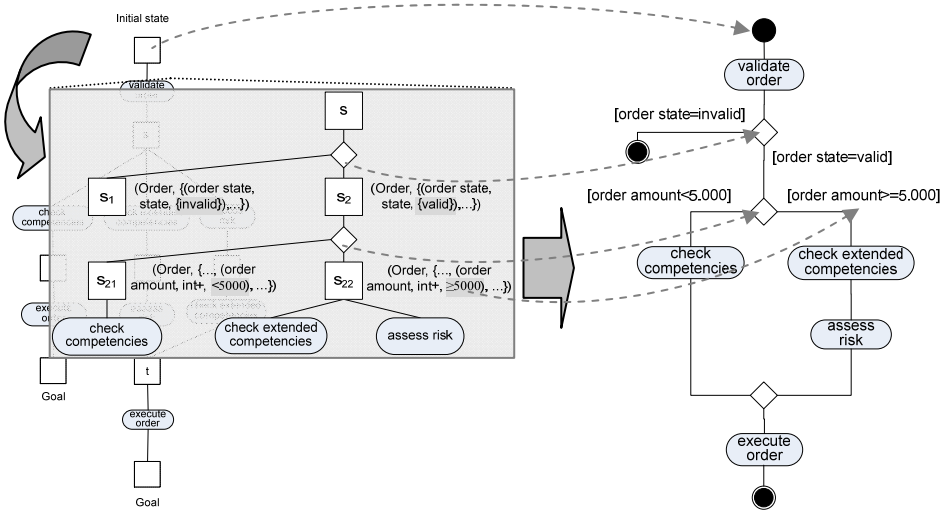

Figure 5: Transformation to an UML activity diagram

Figure 5 shows a resulting UML activity diagram on the right side. Obviously, not all paths of the ASG need to be transferred to the UML activity diagram. The choice of paths leads to different feasible process models. In the UML activity diagram in *Figure 5*

for instance *check extended competencies* is applied before *assess risk.* Another feasible process model can be derived if the process actions are applied the other way around. Moreover also a parallelization of these process actions is possible, because they have no predecessor successor-relation. A critical point is that the number of feasible process models explodes quite quickly. Therefore in the current version only a limited number is presented to the user, although theoretically all feasible process models could be derived.

4 Related Work

We already elaborated that SBPM forms the foundation for the automatic planning of process models. The existing approaches in this area (e.g. [BE+06], [HD07]) however, do not provide a solution to the automatic planning problem. In this section we will therefore evaluate whether existing approaches in AI planning and web service composition are suited in this context.

We characterized our problem as a planning problem in non-deterministic domains. Several approaches have been introduced to solve non-deterministic planning problems, e.g. [CRT98], [KBS97], and [Sc87]. In general they explicitly enumerate all states of the world that may occur after applying an action. This is not practicable in our case. Because of infinite and continuous data types (e.g. real number) it is not even possible. Therefore in our formalism, process states are defined as sets of feasible values of parameters and thus *implicitly* describe sets of conceivable states of the world. A similar problem occurs when planning under partial observability (also called planning with sensing) is applied. Planning approaches in this area are for instance [BC+01], [BG01], [WA+98]. They accomplish a search in the space of belief states, where a belief state encapsulates (explicitly) the set of states that are conceivable at a certain point in planning. These sets can become extremely large and therefore are as well impracticable in our context. The only exception to the best of our knowledge constitutes [PB02]. They introduce a knowledge based approach to planning. Similar to our approach, knowledge bases provide means to describe possible states of the world without enumerating them explicitly. Yet, there are other problems inhibiting the direct application of existing planning approaches for the planning of process models. In order to build practical process models, a planer must handle state variables with different data types. Especially numerical state variables are challenging in this context. In general numerical variables are considered as resources, e.g. [BG01, PB02], and thus different to the way we have to employ them. Moreover, planning for several (conflicting) goals is not considered so far.

Some planning approaches have already been adapted for the composition of (semantic) web services (e.g. [SW+03], [LS05], [PT+05]). [BD+06] and [TB+06] try to categorize the nearly unmanageable amount of web service composition approaches. However, none of these approaches meets the requirements for process planning. Especially, non-deterministic planning, the identification of control structures, the ability to handle composite input and output parameters and numerical data types seem to be most challenging. Yet, these issues are necessary for the planning of process models.

5 Conclusion and Outlook

In this paper, we introduced the semantic-based, automatic planning of process models as a new application for planning approaches and SBPM. We provided an intuitive (from a process modeling perspective) formalism for the planning problem. Based on existing work in AI planning, we moreover described an innovative planning algorithm. Specific contributions of the SEMPA planning algorithm in comparison to other planning approaches are the following: The algorithm is capable of handling parameters with numerical data types and can cope with different (conflicting) goals. Several measures enable an efficient search in the action-state space. For instance differentiating between applicable and strongly applicable actions exploits the specific structure of the planning problem. Moreover the search space is restricted. For instance already analyzed process states do not have to be explored again. Finally, the ASG (as a result of planning) forms a suitable basis to derive process models, as we demonstrated for the example of UML activity diagrams.

The SEMPA algorithm has been prototypically implemented as a plug-in in the process modeling tool AgilPro. One critical point in the current implementation relates to resulting UML activity diagrams. The form of the diagrams depends e.g. on the order in which parameters are considered during the decomposition of process states (in the case study for instance we could first examine the *order amount* and then the *order state*). The resulting diagrams are in fact semantically equivalent but may differ in their comprehensibleness for a user. We are currently working on appropriate heuristics to address these issues. Additionally, future work is intended on the question how other advanced control structures like arbitrary cycles can be recognized during planning. Moreover, we want to enhance our algorithm to support also non-functional properties (like cost or time constraints) during the planning.

References

[BC+01] Bertoli, P.; Cimatti, A.; Roveri, M.; Traverso, P. (2001b): Planning in Nondeterministic Domains under Partial Observability via Symbolic Model checking. In: IJCAI01. pp. 473-478.

[BD+06] Berardi, D.; De Giacomo, G.; Mecella, M.; Calvanese, D.(2006): Automatic Web Service Composition: Service-Tailored vs. Client-Tailored Approaches. In: AISC2006.

[BE+06] Brockmans, S.; Ehrig, M.; Koschmider, A.; Oberweis, A.; Studer, R. (2006): Semantic Alignment of Business Processes. In: ICEIS 2006. pp. 191-196.

[BG00] Bonet, B.; Geffner, H.(2000): Planning with Incomplete Information as Heuristic Search in Belief Space. In: AIPS00. pp. 52-61.

[BG01] Bonet, B.; Geffner, H. (2001): GPT: A Tool for Planning with Uncertainty and Partial Information. In: IJCAI01. pp. 82-87.

[BK+06] Betz, S.; Klink, S.; Koschmider, A.; Oberweis, A.(2006): Automatic User Support for Business Process Modelling. In: Workshop on SBPM 2006. pp. 1-12.

[BN93] Bäckström, C.; Nebel, B. (1993): Complexity results for SAS+ planning. In: IJCAI-93. pp. 1430-1435.

[CFB04] Constantinescu, I.; Faltings, B.; Binder, W.(2004): Large scale, type-compatible service composition. In: ICWS04. pp. 506-513.

[CRT98] Cimatti, A.; Roveri, M.; Traverso, P. (1998): Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In: AAAI/IAAI. pp. 875-881.
[DG04] Dean, M.; Guus, S. (2004): OWL Web Ontology Language Reference.
[GNT04] Ghallab, M.; Nau, D.; Traverso, P. (2004): Automated Planning. Elsevier, San Francisco.
[HD07] Hepp, M.; Dumitri, R. (2007): An Ontology Framework for Semantic Business Process Management. In: Wirtschaftsinformatik07. pp. 423-440.
[He02] Helmert, M. (2002): Decidability and Undecidability Results for Planning with Numerical State Variables. In: AIPS02.
[HL+05] Hepp, M.; Leymann, F.; Domingue, J.; Wahler, A.; Fensel, D.(2005): Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In: IEEE ICEBE 2005. pp. 535-540.
[KBS97] Kabanza, M.; Barbeau, M.; St-Denis, R. (1997): Planning Control Rules for Reactive Agents. In: Artificial Intelligence 95 (2), pp. 409-438.
[KS03] Kalfoglou, Y.; Schorlemmer (2003): Ontology Mapping: The State of the Art. In: Knowledge Engineering Review Journal 18 (1), pp. 1-31.
[LS05] Lang, Q.; Su, S.. (2005): AND/OR Graph and Search Algorithm for Discovering Composite Web Services. In: International Journal of Web Services Research 2 (4), pp. 46-64.
[Ma04] Martin, D.; et al. (2004): OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004. last called: 2007-04-16.
[PB02] Petrick, R..; Bacchus, F. (2002): A Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In: AIPS02. pp. 212-221.
[PT+05] Pistore, M.; Traverso, P.; Bertoli, P.; Marconi, A. (2005): Automated Synthesis of Composite BPEL4WS Web Services. In: ICWS05. pp. 293–301.
[Sc87] Schoppers, M. J. (1987): Universal Plans for Reactive Robots in Unpredictable Environments. In: IJCAI87. pp. 1039-1046.
[SW+03] Sirin, E.; Wu, D.; Hendler, J.; Nau, D.; Parsia, B. (2003): Automatic Web Services Composition Using SHOP2. In: ICAPS03.
[TB+06] ter Beek, M.; Bucchiarone, A.; Gnesi, S. (2006): A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods. Technical report, 2006-15. last called: 2007-01-16.
[TF07] Thomas, O.; Fellmann, M. (2007): Semantic Business Process Management: Ontology-based Process Modeling Using Event-Driven Process Chains. In: Journal IBIS, 2 (1), 2007
[VT+03] van der Aalst,W.; ter Hofstede, A.; Kiepuszewski, B.; Barros, A. (2003): Workflow patterns. In: Distributed and Parallel Databases 14 (3), pp. 5-51.
[WA+98] Weld, D..; Anderson, C.; Smith, D. (1998): Extending graphplan to handle uncertainty and sensing actions. In: AAAI98 and IAAI98. pp. 26-30.
[WM+07] Wetzstein, B.; Ma, Z.; Filipowska, A.; Kaczmarek, M.; Bhiri, S.; Losada, S.; Lopez-Cobo, J.; Cicurel, L. (2007): Semantic Business Process Management: A Lifecycle Based Requirements Analysis. In: SBPM07, Innsbruck, Austria, June 7, pp. 1-11.