Automated Planning of Process Models- Towards a Semantic-based Approach

Bernd Heinrich
Department of Information Systems, University of Innsbruck, Innsbruck, Austria,
E-mail: bernd.heinrich@uibk.ac.at

Mathias Klier
Department of Information Systems, University of Innsbruck, Innsbruck, Austria,
E-mail: mathias.klier@uibk.ac.at

Steffen Zimmermann
Department of Information Systems, University of Innsbruck, Innsbruck, Austria,
E-mail: steffen.zimmermann@uibk.ac.at

ABSTRACT
Companies need to adapt their processes quickly in order to react to changing customer demands or new regulations, for example. Process models are an appropriate means to support process setup but currently the (re)design of process models is a time-consuming manual task. Semantic Business Process Management in combination with planning approaches can alleviate this drawback. This means, that the workload of (manual) process modeling could be reduced by constructing models in an automated way. Since existing, traditional planning algorithms show drawbacks for the application in Semantic Business Process Management, we introduce a novel approach that is suitable especially for the Semantic-based Planning of process models. In this chapter, we focus on the semantic reasoning, which is necessary in order to construct control structures, such as decision nodes, which are vital elements of process models. We illustrate our approach by a running example taken from the financial services domain. Moreover, we demonstrate its applicability by a prototype and provide some insights into the evaluation of our approach.

INTRODUCTION
In times of dynamically shifting markets, companies, especially those integrated in electronic supply chains, have to adapt or even restructure their processes frequently. First, regarding the sales market of a company, changing customer needs and new offers of emerging competitors need to be considered and demand for quick reactions in the form of enhanced services and innovative products. Second, a national and international network of business partners gets more and more important in order to be able to offer best-of-breed-products and customized solutions instead of commodities. Such approaches, along with an efficient design of the supply chain, constitute distinguishing factors between competitors. Third, the market of suppliers for a company, especially of IT suppliers, is expanding. Only ten years ago, the share of proprietary and individual software has been considerably higher than today. In the future, it should be possible to design and modify company specific applications by composing Web Services provided by external software suppliers according to predefined business processes. In any of these three "market views" processes constitute the starting point for dynamic modifications along the value chain.
In order to counter the above mentioned requirements of a dynamic and flexible (re)design, traditional techniques and tools for process modeling and optimization seem to be insufficient or inadequate to some extent. Reasons are that traditional modeling techniques for process (re)design imply a significant degree

of manual work (e.g. Becker & Kahn, 2003; Borges et al., 2005; Ma & Leymann, 2008) or result repeatedly in a high demand for communication and clarification because of different terminologies (Becker et al., 2000; Thomas & Fellmann, 2006).

According to the ongoing research in the area of Semantic Business Process Management (SBPM), a higher degree of automation concerning the use of process models can contribute to a solution (cf. Hepp et al., 2005; Thomas & Fellmann, 2006). More precisely, we envision the automated design of process models. As this task can be regarded as a kind of planning problem (cf. Ghallab et al., 2004; Henneberger et al., 2008; Heinrich et al., 2009), we speak of an automated planning of process models. One basis for the automated planning of process models constitute Semantic Web standards like the Web Ontology Language (OWL) that enables a semantically enriched description of process models and their elements (e.g. Betz et al., 2006; Drumm et al., 2006). These standards have already been used for Semantic Web Service Composition. Yet, in contrast to Semantic Web Service Composition approaches, the planning of process models is conducted on a conceptual level independent from the underlying technology. The composition of Web Services is accomplished for a specific problem, that means a number of Web Services is arranged together to deliver one distinct and previously defined output. For the planning of process models, however, we abstract from one individual process execution and its implementation. Thus, process models are initially technology independent and may partially be realized by different combinations of available Web Services (from different providers) that may be chosen afterwards by means of economic aspects like cost and risk. This two-step approach is advantageous as it increases flexibility and bears optimization potential. It is moreover reasonable to assume that the step from descriptions of process models and process actions to concrete implementations using Web Services is relatively small (e.g. Drumm et al., 2006).

Yet, the automated planning of process models – in the sense of an automated design of entire new process models – is hardly discussed in the scope of SBPM, if at all. Doubtlessly the conceptual and technological basis for an automated planning of process models is to a certain extent already present in the areas of Artificial Intelligence (AI) planning and Semantic Web Service Composition. Several approaches in both domains indeed exist, but the planning problem and particularly the planning of process models is far from being solved (cf. Heinrich et al., 2009). For instance, Semantic Web Service Composition approaches are often restricted to a manageable number of sequentially executed Web Services and do normally not focus complex compositions, including vital control structures in process models like parallel split.

Therefore, the aim of our paper is to introduce an approach, called SEMPA (SEMantic-based Planning Approach) that supports and enables the automated planning of process models using semantically described process actions. Figure 1 clarifies the problem setting and specifies some terms and concepts used. Based on a semantic description of the domain (in terms of the OWL ontology) and the description of actions which are available in a library, the planning algorithm is supposed to find process models for a given problem definition. The problem definition includes, besides the library and the ontology, an initial state that characterizes the overall inputs of the process and one to many goals. Goals are representing process outputs.

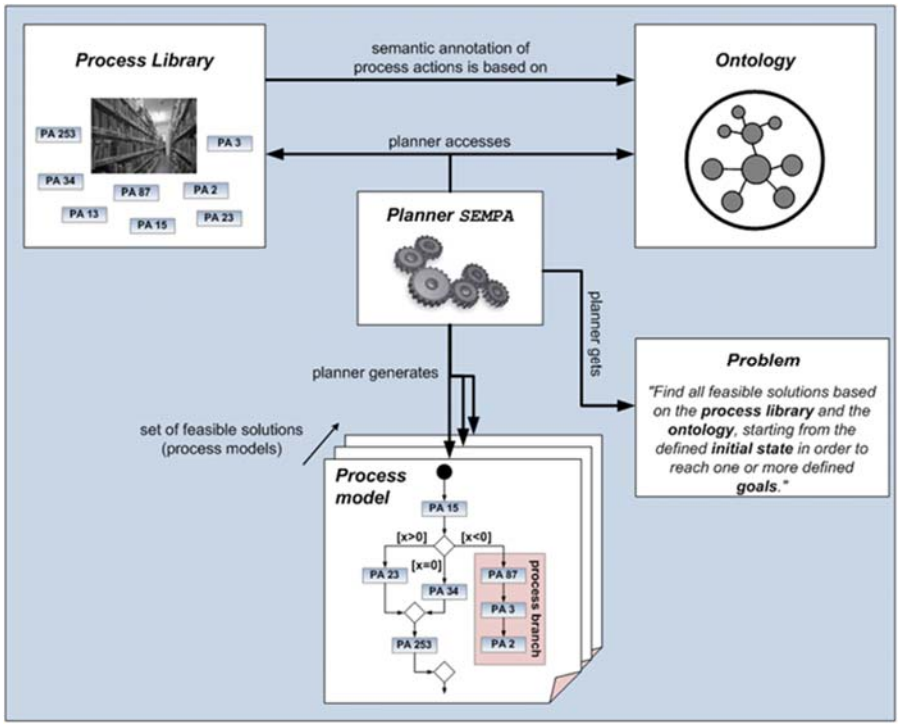

Figure 1. The basic idea of automated planning of process models

We organized the chapter as follows: In the next section we introduce a running example. Afterwards, we elaborate the fundamental requirements for the planning of process models and give a short review of the relevant literature. Based on that, the planning algorithm SEMPA which is conceptually divided into three steps is elaborated. We then briefly discuss the evaluation of our approach before we finally summarize the main results and define further research.

RUNNING EXAMPLE

The basic idea will be illustrated by a running example originating from the financial services industry. The problem definition of this use case is inspired by a project with a financial services provider. We consider a situation where a new process model for the execution of an order (e.g. stock order) is needed. In the initial state there is the order that has been entered by a customer. The goal is to execute the order (executed and entered are two feasible values of the order status). At the same time the guidelines of the financial services provider require a risk assessment for each order to be executed. Based on this, a process modeler would try to develop a new process model reusing existing actions that are already part of similar models. This complies with the frequently appearing case for financial products that new process models require established actions (at the same time new actions can be considered as well).

The manual task of modeling the new process should now be supported by an automated planner, i.e. feasible process models should be planned in an automated way. In a first step, the originally high number of available actions was reduced for the purpose of this problem definition to some coarse grained actions: *validate order*, *assess risks*, *check competencies*, *check extended competencies*, *book order* and *execute order* (each action is described later in detail). Therefore, a semantic annotation of the actions is mandatory, which ensures that all actions share a common meaning (e.g. of an *order* as the subject of the process model) and ambiguities and misinterpretations are avoided. Moreover, restrictions specified for the actions need to be considered. For instance the action *check competencies* is only applicable to orders with an *order amount* less than 5,000[i]. A human modeler would consider these restrictions in the course of modeling although they are not part of the goal description. We will describe in the next section how this translates into concrete requirements for an automated planning taking the running example into account.

REQUIREMENTS FOR PROCESS PLANNING AND RELATED WORK

In a first step, we elaborate the basic requirements for the planning of process models. Then, we give a review of selected existing approaches with respect to these requirements.

Requirements for planning process models in the context of SBPM

Firstly, we give an overview of the requirements a process planner needs to meet in detail. They constitute criteria to evaluate existing approaches as well as the planner we suggest. The following requirements derived from literature (cf. e.g. Constantinescu et al., 2004; Meyer & Kuropka, 2006) need to be considered:

(R1) *Input and output parameters of actions*[ii]: In order to plan a feasible process model, the dependencies between actions need to be considered. Therefore, the planner has to identify dependencies between actions based on their input and output parameters. If, for instance, the action *validate order* requires the input parameter *order,* then the *order* either needs to exist in the initial state or must have been generated as an output parameter of a previously executed action. Actions and input and output parameters respectively are described independently of an individual planning problem (i.e. independently of the specific process goals), as actions are supposed to be used in different process models.

(R2) *Multiple processing of actions*: Each available action can be used 0 to n times within one particular process model. For instance, there could be an action *check signature* that is necessary both when entering and changing an order. This essentially implies that at first the relevant actions to a given planning problem need to be discovered in a potentially large library.

(R3) *Composite input and output parameters*: In other approaches oftentimes rather simple input and output parameters are assumed (see below). However, this is not sufficient for the planning of realistic process models. Instead, the planner needs to handle composite parameters, i.e. actions may require or generate composite parameters which consist of at least two atomic parameters. An *order* (composite parameter) in our example for instance consists of the atomic parameters *orderState*, *orderAmount* and *orderType*.

(R4) *Numerical and alphanumerical data types*: It must be possible to assign numerical and alphanumerical data types (cf. Biron & Malhotra, 2004) to each atomic parameter in the ontology (in contrast to other approaches that for the sake of simplicity assume only Boolean variables). For example the *orderAmount* may be of the data type *positiveInteger*. This definition applies for any action in the library that has the parameter *orderAmount*. However, some actions may require certain ranges of values of an input parameter for instance. As already mentioned the action *check competencies* is only reasonable for *orders* with an *orderAmount* less than 5,000. Such restrictions have to be considered, i.e. it should be possible to specify the range of values an action accepts for an input parameter and the range of values it produces for an output parameter.

(R5) *Non-deterministic planning*: Due to restrictions the input and output parameters of two consecutive actions may not match completely. This happens for instance if the action *check competencies* succeeds an action that produces an output parameter *order* and the *orderAmount* can also adopt a value higher than 5,000. As the concrete realizations of values of input and output parameters (of a single process run) are not determined at the moment of planning, the planner needs to consider all possible realizations. This requirement is referred to as non-deterministic planning (cf. Henneberger et al., 2008).

(R6) *Semantic-based Reasoning:* The planner has to use a given ontology that contains all input and output parameters and their data types and describes relations between them (such as the subclassof relation). Input and output parameters of actions have to be compared based on the ontology in order to identify dependencies between actions. For the analysis of the dependencies, inference mechanisms have to be applied by the planner to detect identities, equivalences and specialization relations etc. If, for instance, an action requires an input parameter *order* and it can be deduced from the ontology that a *stock order* is a specialization of *order*, then a *stock order* can be processed as an input parameter as well.

(R7) *Control structures in process models:* A planner must be able to plan control structures in process models in an automated way. Van der Aalst et al. (2003) provide a comprehensive description of the various control structures (workflow patterns) that may be part of a process model and workflow respectively. They moreover analyze how these structures are represented in different process modeling languages (like for instance UML activity diagrams (Russel et al., 2006)). A process planner needs to consider for example the following basic and advanced structures (cf. van der Aalst et al., 2003): exclusive choice, parallel split, synchronization, simple merge, and arbitrary cycles.

(R8) *Set of feasible solutions:* Besides control structures there are often many feasible solutions (alternative process models) for one given problem definition. A feasible solution is any process model, where all goal states can be reached starting in the initial state. For example, different feasible solutions occur when two or more process actions do not have any input/output dependencies and therefore a sequential as well as a parallel composition may be possible. This applies for instance to the actions *assess risks* and *check extended competencies* in our example, that can be executed sequentially (starting with either of the two) or in parallel, which results into three feasible solutions. The planner has to construct such feasible process models in an automated way.

After specifying the requirements to a process planner in general, we will give a review of existing approaches related to this issue.

Related Work

The automated planning of process models – in the sense of an automated design of entire new process models – is hardly discussed in the research strand of SBPM. Related work can be mainly found in the fields of AI and Web Service Composition. In the following, we will discuss approaches related to these research areas.

Automated Planning

The task of finding a number of actions that transform an initial state into a goal state is called planning (Ghallab et al., 2004). In this respect, a process model can be regarded as an abstract plan. Each action, when executed, results into a new state (thereby a state is constituted by the output parameters produced by the actions). However, an important characteristic is that for the planning of process models we abstract from an individual process execution and therefore the individual realizations of parameter values (and thus the current state in a single process run) are not determined at the moment of planning. In fact a process model should be more general and hold for different possible process runs. This conforms to so-called *non-deterministic* planning problems (Ghallab et al., 2004). Non-deterministic planning approaches are oftentimes further differentiated depending on the extent to which concrete values can be observed at execution time, i.e. when the plan (or in our case the process depicted by the model) is executed. Assuming limited knowledge at execution time seems to be odd – although for example the *order amount* of a particular order is not determined at the moment of planning, there is no reason, why it should be unknown at process

execution time, too – but it makes perfectly sense for traditional automated planning domains such as robotics. For instance, a robot may have limited sensors and thus may be not able to capture its current position in a room exactly. This has to be accounted for when actions are planned for the robot. Thus, non-deterministic planning approaches are further differentiated into non-deterministic planning approaches under full observability, under partial observability (also called planning with sensing), and under no observability (no sensors at all). Existing approaches to the mentioned problem domains will be analyzed in the following with respect to our problem setting.

Considering our requirements and in particular requirement (R4), *non-deterministic planning approaches under no observability* can be omitted here because they generally build sequential plans (Bonet & Geffner, 2000). Process models should, however, contain control structures (e.g. exclusive choices).

Plans in *non-deterministic planning domains under full observability* may contain branches. Several approaches have been introduced in this area (cf. Cimatti et al., 1998; Kabanza et al., 1997). However, these approaches explicitly enumerate all states that may occur after applying an action. This is not practicable in our case, since parameters and their data types may have a nearly infinite domain (e.g. real number). Therefore, approaches for planning under full observability do not contribute to a solution, as well.

Non-deterministic planning approaches under partial observability are for instance Bertoli et al. (2001), Bonet & Geffner (2001), and Hoffmann & Brafman (2005). They build so-called conditional plans that branch at the conditions of certain variables which are tested by pre-defined sensing actions. Most of these approaches accomplish a search in the space of belief states, where a belief state encapsulates the set of states that are conceivable at a certain point in planning. Petrick & Bacchus (2002), in contrast, introduce a knowledge based approach to planning, where knowledge bases provide means to describe possible states of the world without enumerating them explicitly (which is advantageous for the reasons outlined above). Again, however, an important problem with most of these approaches is that they do not support numerical data types (requirement (R4)). Or if numbers are integrated then they are regarded as resources, which is not appropriate for planning process models. A prominent example of a resource is time; another example maybe fuel (e.g. in a car driving scenario). Actions are consuming (or producing) proportions of a resource, i.e. they are increasing or decreasing the value of the corresponding numerical variable. Resources are considered as constraints to the planning problem or they are part of a goal specification. For instance, the challenge may be to find a plan that reaches a goal with minimum consumption of time and fuel, which is obviously quite different to our problem setting. An additional drawback of most planning approaches is related to requirement (R7). Although non-deterministic planning approaches under partial observability build conditional plans, the conditions at the branches are in general given in advance (by the planning domain) and are restricted to most often binary variables. In process modeling, which is our problem context, such conditions are not given as this would simplify planning process models to a great extent (detailed in Heinrich et al., 2009).

In summary, we can state that because of these differences none of the existing approaches in AI planning reaches out for the planning of process models.

Web Service Composition

In fact, the approaches with the closest similarity to the specified problem setting are automated Web Service Composition approaches because they employ to some extent planning techniques as well. Additionally, automated Web Service Composition approaches make similar assumptions. For instance, several approaches assume that Web Services are semantically described applying for example SAWSDL or OWL-S. These languages are based on semantic Web standards (RDF, OWL) and Web Service standards respectively (SOAP, WSDL, WS-BPEL) and have in common, that they enable the semantic specification of input parameters, output parameters, preconditions and effects. As already mentioned actions can be described in a similar fashion. Preconditions and effects were not mentioned so far. In a process modeling context they most often refer to the input and output of actions and can therefore be modeled as restrictions (as we will see later on). Henceforth there are analogies between the planning of process models and Web Service Composition, which justifies analyzing them in more detail with regard to the requirements stated above.

Few papers so far try to categorize the nearly unmanageable amount of Web Service Composition approaches (cf. Berardi et al., 2006; ter Beek et al., 2006). In the following paragraphs we briefly discuss the approaches which are – to the best of our knowledge – the ones with the highest similarity to the problem setting described above.

In Lang & Su (2005) the Web Service Composition problem is regarded as a search problem in a so called AND/OR graph. This approach contains several ideas that are interesting in our context. The authors select and compose Web Services based on their input/output dependencies ((R1), (R2)), even considering composite input and output parameters, as demanded by requirement (R3). However, the approach does not account for data types of input and output parameters (R4). Semantic based reasoning capabilities are mentioned, but not elaborated in detail (R6). Moreover, non-deterministic planning (R5) and control structures (R7) are not considered at all. But the approach at least theoretically enables the extraction and subsequent evaluation of all feasible solutions to their composition problem (see (R8)).

The next two approaches we consider are SHOP2 and Golog, which both are employed for Web Service Composition (cf. Sirin et al., 2004; McIlraith & Son, 2002). Given a list of tasks that have to be achieved and a set of Web Services that accomplish these tasks, these planners build a plan representing an ordered sequence of Web Services that needs to be executed. The basis for planning is the description of input and output parameters as well as preconditions and effects, which is in accordance with (R1). However, the tasks to be accomplished are specified in advance, which simplifies the selection of appropriate Web Services to a great extent (R2). Input and output parameters are neither complex, nor do they have data types ((R3), (R4)). Moreover, at the moment of planning, all necessary values are fully determined and a deterministic plan can be constructed. Consequently, both planner do not consider requirements (R5) and (R8), and give therefore no idea or solution. Control structures may be part of the given problem definition but are not constructed in an automated way (R7). Semantic based reasoning capabilities (R6) are not reported in these approaches. However, it seems the authors are still working on this issue.

Pathak et al. (2006) propose a framework for modeling Web Service Composition and Execution based on symbolic transition systems. Compared to the approaches discussed before, they focus additionally on non-functional requirements (e.g. Quality of Service-parameters like cost). The composition algorithm then chooses Web Service Compositions based on these requirements. That means a set of feasible solutions (alternative Web Service Compositions) with the same functionality but different non-functional properties can be constructed. Thus in contrast to Shop2 and Golog, requirement (R8) is fulfilled.

The algorithms analyzed so far lack the capability to construct more complex plans that is considered by the next approaches. A planner that has already been employed for Web Service Composition is MBP, as described in Pistore et al. (2005). When we face this approach with our requirements, apparently (R1) is met, since the input and output parameters are considered for planning. As in the current solution only two Web Services are orchestrated, it is difficult to assess how the selection and multiple use of component Web Service works (R2). Composite input and output parameters and numerical data types are not supported ((R3), (R4)). However, the MBP support non-determinism (R5). Accordingly, plans can include the control structure exclusive choice (R7), but no other control structure like parallel split. Semantic based reasoning is not employed as the authors concentrate on a syntactical composition (R6). Moreover it is not the intention to generate all feasible solutions to their planning problem or even to evaluate them (R8).

In Meyer & Weske (2006) a heuristic search algorithm for automated Web Service Composition is presented. Considering the requirements specified above, we observe that some, but still not all requirements are fulfilled. Planning is accomplished using the input and output parameters as well as preconditions and effects (R1). The paper does not go into detail whether the chosen Web Services are the only available ones and whether a Web Service might be used more than once (R2). The authors mention that the composition will be extended in future versions by numerical state properties, but currently the requirements (R3) and (R4) are not met. Moreover, the composition does not consider semantic descriptions so far (R6). The requirements (R5) and to a certain extent also (R7) are fulfilled, since the planning graph supports simply alternative and parallel process flows. As only one plan is returned, requirement (R8) is not met.

Constantinescu et al. (2004a, b) propose a type-based approach for Web Service Composition. Their approach meets only parts of the claimed requirements. The approach fulfills (R1), as input and output parameters are used as a planning basis. Requirement (R2) is also met, because the same Web Service may occur in several paths of the presented solution. The approach supports primitive data types including numerical values (R4) but no composite parameters (R3). Requirement (R5) is met, since partially matching Web Services lead to different branches in the composite solution. Semantic descriptions and appropriate reasoning mechanisms (R6) are considered only elementary as matching parameters need to be semantically equal. Furthermore, only exclusive choices and the corresponding simple merges can be deduced from the result (R7). The extraction of different feasible solutions (R8) is not supported.

Authors	Approach	(R1) Input and output parameters of actions	(R2) Multiple processing of actions	(R3) Composite input and output parameters	(R4) Numerical and alpha-numerical data types	(R5) Non-deterministic planning	(R6) Semantic-based Reasoning	(R7) Control structures in process models	(R8) Set of feasible solutions
Lang and Su 2005	AND/OR-graph	✓	✓	✓	-	-	(✓)	-	(✓)
Sirin et al. 2004	SHOP2 (HTN Planning)	✓	(✓)	-	-	-	(✓)	-	-
McIlraith and Son 2002	Golog language (situation calculus)	✓	(✓)	-	-	-	(✓)	-	-
Pathak et al. 2006a, b	Symbolic Transition Systems	✓	✓	-	-	(✓)	-	(✓)	✓
Pistore et al. 2005	State Transition Systems	✓	(✓)	-	-	✓	-	(✓)	-
Meyer and Weske 2006; Kuropka and Weske 2008	Heuristic Search via Hill Climbing	✓	✓	-	-	✓	-	(✓)	-
Constantinescu et al. 2004	Type-based Composition	✓	(✓)	-	✓	✓	(✓)	(✓)	-

Table 1. Selected Approaches for Web Service Composition

After all, we summarize that none of the discussed approaches meets all of the requirements for process planning (see Table 1). Non-deterministic planning, the construction of control structures, the ability to handle composite input and output parameters and numerical data types as well as the usage of appropriate inference mechanisms seem to be most challenging. Yet, especially these issues provide interesting questions for the planning of process models: How can we derive dependencies between actions by means of a semantic analysis considering numerical data types as well as restrictions on these data types? How can we account for composite parameters in this context? How can the identified dependencies be used for non-deterministic planning? These questions are addressed in the following section.

SEMANTIC-BASED PLANNING OF PROCESS MODELS

In the following, the SEMPA algorithm for the planning of process models will be introduced. Therefore the defined requirements constitute the starting point for the design. To be able to address these requirements, the algorithm is conceptually divided into three steps:

1. To identify dependencies between actions originating from their input and output parameters, we create an Action Dependency Graph (ADG). The ADG includes actions and corresponding parameters (as nodes). We use semantic reasoning in this step of SEMPA, i.e. we analyze the classes of input and output parameters and their relations defined in the ontology.

2. The ADG describes no direct sequences of actions, yet. Hence, in the next step of SEMPA we employ a forward search algorithm to determine all sequences of actions leading from the initial state to the goals. As result, we obtain an Action State Graph (ASG) that comprises all feasible solutions to the corresponding planning problem.

3. In the third step, we design the control structures in the ASG and build syntactically correct process models. These process models are finally presented to the user.

Figure 2 illustrates the three steps of the algorithm. They are described below in detail.

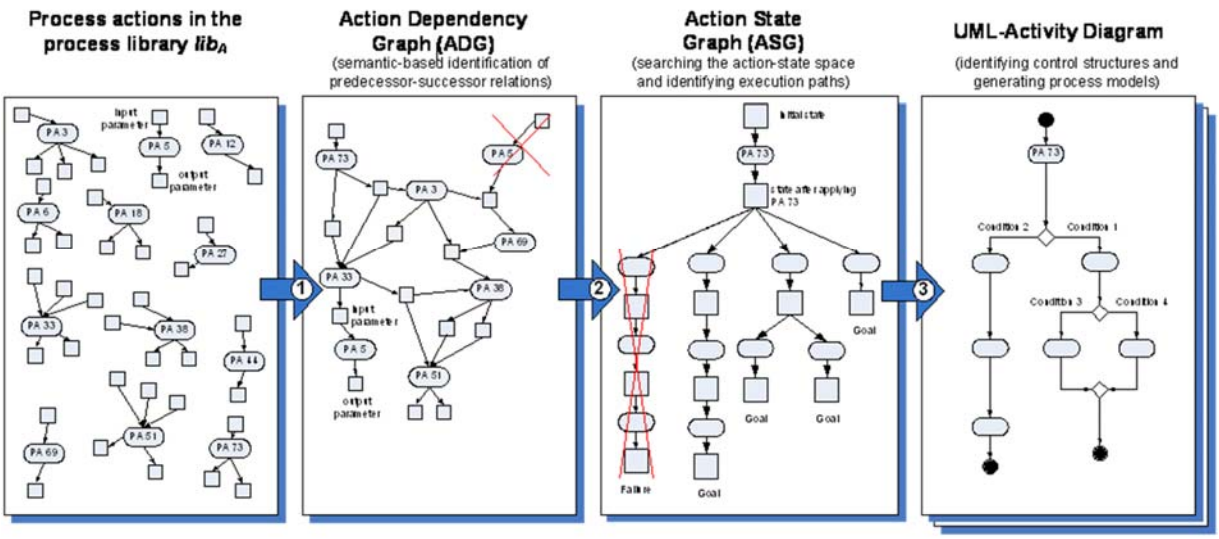

Figure 2. The three steps of the SEMPA approach.

The division of the algorithm into three steps has a couple of advantages:

- As already mentioned in requirement (R2) the number of available actions may be significantly large. The division allows us to reduce complexity while efficiency can be increased. Actions that can never be part of a feasible solution of a specific problem definition are separated out in the ADG (step one) and are not processed in further steps. Accordingly, by means of the ADG the search space for the following forward search (step two) can be significantly reduced. Moreover, dependencies analyzed once in the ADG can be reused for further planning problems.

- Semantic reasoning – demanded by requirement (R6) – is a complex and thus time-consuming task, especially in the presence of composite parameters with different data types and restrictions (cf. requirements (R3) and (R4)). Therefore, semantic reasoning is accomplished mainly once (in the first step). This avoids redundant analysis during the following steps of the algorithm.

- The ASG forms a (language independent) basis for the extraction of process models. Characteristics of process modeling languages concerning for instance control structures are only considered in the last step (cf. requirement (R7)). This gives us the flexibility to consider different process modeling languages at a time without adapting the fundamental search algorithm. On the other hand it provides the possibility to develop different search algorithms generating the ASG without affecting the extraction of process models.

Assumptions and Definitions

Before we introduce the three steps of the algorithm with a focus on the first step, we will provide essential assumptions and definitions.

(A1) Let P be the set of all input and output parameters. A parameter p (with $p \in P$) is either an atomic or a composite parameter. An atomic parameter is a triple (l_p, dom_p, r_p) which consists of an identifier l_p, its domain $dom_p \in Dom$, and the restriction r_p on the domain. A composite parameter $(l_p, \{(l_{p_1}, dom_{p_1}, r_{p_1}), ..., (l_{p_n}, dom_{p_n}, r_{p_n})\})$ is a tuple containing an identifier l_p and a set of $n \geq 2$ atomic parameters, with $n \in \mathbb{N}$.

Parameters establish the basis for planning. For reasons of simplicity, we focus on atomic parameters in the formal descriptions. They have a (global, unchangeable) domain dom_p that is defined in an ontology (cf. 0). This domain is either a primitive data type (cf. data types in XML-Schema (Biron et al., 2004)) or an ontological class. Additionally, each action can define an individual restriction r_p for each of its parameters p (with $r_p \subseteq dom_p$ if dom_p is a primitive data type respectively $r_p \sqsubseteq dom_p$ if dom_p is an ontological class). If for instance an action is only executable for a proper subset of the domain dom_p (in the case of primitive data types) the input parameter p is restricted to $r_p \subseteq dom_p$. To comply with requirement (R3), also composite parameters are defined. Figure 3 illustrates the composite parameter *order*.

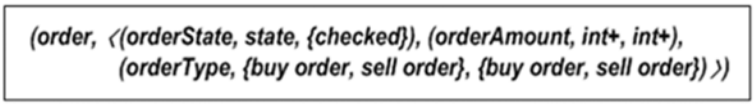

Figure 3. The parameter order.

The parameter *order* is defined by its atomic elements *orderState*, *orderAmount* and *orderType*. The *orderAmount* for example is defined as a positive integer value etc.

(A2) Let A be the set of all actions. An action $a \in A$ is completely characterized by a set of input parameters $In_a \subseteq P$, which is required for the execution of a and a set of output parameters $Out_a \subseteq P$, which is provided by a after execution. All actions a are stored as a triple (a, In_a, Out_a) in a library lib_A.

An action a is characterized by its input In_a and its output Out_a. In contrast to proposed standards like OWL-S, WSDL-S or WSMO, we do not distinguish between the information space used by the actions (often referenced to as inputs and outputs of a Web Service) and the state (of the world) that might be changed due to the execution of an action (often referenced to as preconditions and effects). Here, we assume that the (world-)state is modeled – together with the information-space – by a set of parameters and especially their restrictions.

Figure 4 illustrates the specification of process actions with an example of our use case. The action *validate order* requires an arbitrary *order* as input parameter and returns the same *order* with an altered *orderState*.

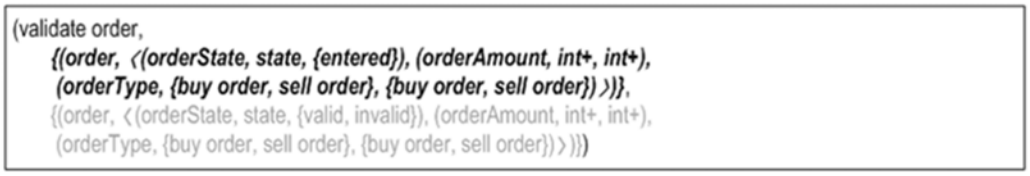

Figure 4. The action validate order.

(A3) Let *Ont* be a single consistent ontology, in which both the input and output parameters with their domains in the form of classes[iii] as well as the mutual binary relations of these classes are described.

Our assumption here is that there exists a common ontology. As there are already many approaches dealing with the alignment and matching of several ontologies, we neglect the fact that actions could be defined using multiple ontologies (for ontology mapping and alignment cf. Giunchiglia et al., 2007). The ontology *Ont* defines for instance equality and specialization relations between parameters (represented by relations between ontological classes). These relations and their usage in the course of the semantic analysis will be described below in detail.
An ontology for the introduced running example is illustrated in Figure 5.

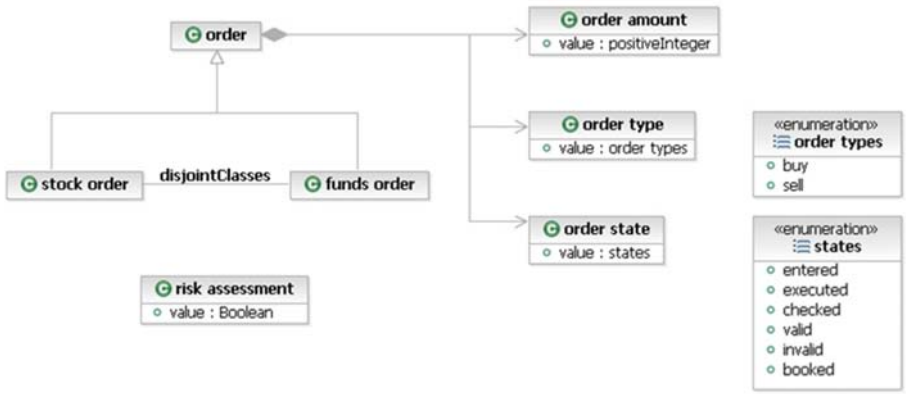

Figure 5. Ontology used for the running example.

(A4) The objective of the algorithm is the planning of feasible solutions (in the sense of process models) for the problem defined as $Prob = (Ont, Init, Goals, lib_A)$. *Ont* is the reference ontology and $Init \subseteq P$ the initial set of parameters (initial state). In addition, $Goals = \{G_1, G_2, ..., G_k\}$ defines the set of $k \in \mathbb{N}$ different (sets of) parameters $G_x \subseteq P$ (with $x = 1, ..., k$), which shall all be met in each feasible solution. lib_A is the given library containing all existing actions.

According to requirement (R8) the algorithm should provide the set of feasible solutions so that in a further step the optimal solution can be chosen. For each feasible solution and for each $G_x \in Goals$, there exists a certain path in the resulting process model so that G_x can be reached.
In the following, we will describe our approach in detail starting with the first step of SEMPA.

Generation of the Action Dependency Graph (ADG)

First, we determine the dependencies among actions. It is crucial to mention that dependencies do not determine a concrete sequence of actions. The following example illustrates this statement: As an input parameter can possibly be provided by alternative actions, dependencies do not necessarily translate into direct predecessor-successor-relationships. The dependencies between actions are stored in the ADG and are provided as basis for further steps.

To determine the dependency between two actions $a, b \in A$, we have to analyze if there is an output parameter $p \in Out_a$ of a that can be used as an input parameter $q \in In_b$ of b or vice versa, i.e. if input and output parameters match. In the simplest case, all input parameters of one action are identical to the output parameters of another action. But not every dependency is the result of a simple comparison of all input and output parameters of exactly two actions. First, input parameters may be provided by the output parameters of more than one action. Additionally, it might be necessary to compare the parameters not only syntactically, but also use their semantics to identify dependencies, as the input parameters might be deduced from the available output parameters (of one or several actions) through semantic-based reasoning. For instance, an input parameter q of b could be non-identical to an output parameter p of a, but nevertheless being associated by an equivalence or a specialization relation. This causes a dependency of b from a, which is not directly obvious from the syntax of the individual parameters. Such dependencies can be deduced from the ontology using semantic-based reasoning (cf. (R4) and (A3)). We therefore consider the relations *identity* ($=$), *equivalence* ($\equiv$) and *specialization* ($\sqsubseteq$) between parameters. Two parameters are *identical* ($= \subseteq P \times P$), if identifier and domain are identical (with no consideration of the associated restrictions). The *equivalence* ($\equiv \subseteq P \times P$) associates two parameters analogously to the *specialization* ($\sqsubseteq \subseteq P \times P$), if and only if a reasoner can deduce an *equivalentClasses* relation respectively a *subClassOf* relation between the classes from the ontology that represent the parameter.

The matching between an output parameter p of a and an input parameter q of b proceeds in two steps and uses the semantic relations to identify dependencies (cf. Table 2).

I. Analyze parameter relations	**II.** Compare restrictions r_p with r_q	**Matching**
(1.) $p = q \vee p \equiv q \vee p \sqsubseteq q$	(1.1) $r_p \subseteq r_q$	***complete matching***
	(1.2) $r_p \cap r_q \neq \varnothing \wedge r_p \setminus r_q \neq \varnothing$	***partial matching***
	(1.3) $r_p \cap r_q = \varnothing$	***no matching***
(2.) $p \sqsupset q$	(2.1) $r_p \cap r_q \neq \varnothing$	***partial matching***
	(2.2) $r_p \cap r_q = \varnothing$	***no matching***
(3.) $p \not\equiv q \wedge p \not\sqsubseteq q \wedge p \not\sqsupset q \wedge p \neq q$		***no matching***

Table 2: Conditions for dependencies between an output parameter p and an input parameter q

I. Analyze parameter relations

First, we analyze, if there exists a relation between the two parameters. Besides the simple relations *identity* ($p = q$) and *equivalence* ($p \equiv q$), the *specialization* relation needs closer consideration. If a returns a parameter p that is a *specialization* of the parameter q ($p \sqsubseteq q$), then b can be executed, as each occurrence of p is also an occurrence of q. Therefore, the constellations $p = q$, $p \equiv q$ or $p \sqsubseteq q$ constitute the first case (1.) of Table 2. In the second case (2.), if $p \sqsupset q$ holds, there might arise the situation that a returns p which at the same time is not an occurrence of the *specialized* parameter q and thus b is not executable. Consequently, a complete dependency cannot be asserted. Only a partial dependency may exist. For example, if a produces an *order* as output parameter p and b is able to process only *stock orders* as input parameter q (with *stock order* $\sqsubseteq$ *order, fund order* $\sqsubseteq$ *order* and *stock order* is different from *fund order* according to at least one

atomic parameter) then *a* can return *stock orders* as well as *fund orders* although the latter one cannot be handled by *b*. So a partial dependency will be determined, if the set of possible instances of an output is a superset of the processible set of input instances. The last case (3.) with $p \neq q \wedge p \equiv q \wedge p \not\sqsubseteq q \wedge p \not\sqsupseteq q$ results into no dependency, because *b* can never handle the output of *a* (detailed in Heinrich et al., 2008).

II. Compare restrictions

As actions may restrict the domain of input and output parameters, we additionally need to consider restrictions. Even if the input parameter *q* matches with the output parameter *p*, restrictions still can prevent action *b* to be executed and a partial respectively no dependency is asserted. No dependency occurs if the restrictions do not match at all, i.e. the intersection of both restrictions is empty (1.3 and 2.2). A partial dependency is possible, if the restrictions match partially. Thus, restriction r_p contains both, values that are also values of restriction r_q and values that exist in r_p but not in r_q (1.2 and 2.1). Finally, both restrictions may also match completely respectively all values of r_q are also values in r_p (1.1). So a complete dependency is not avoided by them and depends on the result of I.

As an outcome of this matching procedure, we obtain either no dependency or a dependency that might be complete or partial. These identified dependencies are represented in the ADG. Therefore, each action is connected with the parameters it provides as output and with the parameters it uses as input. To enable semantic relations, we have to distinguish between an available output parameter and the parameter used as input in the following steps of SEMPA. Therefore, we store the input parameter as label of an input edge which connects the available output parameter (as node) with the action. Based on this, the ADG is defined as a directed, bipartite graph $G_{ADG} = (V_{ADG}, E_{ADG})$ with the set of nodes V_{ADG} and the set of edges E_{ADG}.

(D1) The set of nodes V_{ADG} consists of the two partitions $Part_a^{ADG} \subseteq lib_A$ containing the actions identified by the planning algorithm and $Part_p^{ADG} \subseteq P$ containing the input and output parameters corresponding to the identified actions ($V_{ADG} := Part_a^{ADG} \cup Part_p^{ADG}$).

(D2) The set of edges E_{ADG} is the union of input edges E_{in} and output edges E_{out} ($E_{ADG} := E_{in} \cup E_{out}$).

(D3) E_{in} is defined as a set of input edges $((p, a), p_{in})$ consisting of a directed edge (p, a) and an edge label p_{in}. $p \in Part_p^{ADG}$ denotes a parameter that is required as an input parameter $p_{in} \in In_a$ by the action $a \in Part_a^{ADG}$. Here, p_{in} does not necessarily have to be identical to p, but needs to be associated by a semantic relation implicating a dependency. If such a relation exists between the involved process actions, we write $p \sim p_{in}$.
$E_{in} := \{((p, a), p_{in}) \mid p \in Part_p^{ADG}, p_{in} \in In_a, a \in Part_a^{ADG}, p_{in} \sim p\}$

(D4) The set of output edges E_{out} consists of all directed edges (a, p_{out}), where $p_{out} \in Part_p^{ADG}$ is an output parameter of the action $a \in Part_a^{ADG}$.
$E_{out} := \{(a, p_{out}) \mid a \in Part_a^{ADG}, p_{out} \in Part_p^{ADG}, p_{out} \in Out_a\}$

To briefly illustrate the ADG, we take an action *check competencies* that provides the parameters *exchangeOrder* (an order that is routed directly to a stock exchange) and *riskAssessment* as output. Another action *execute order* is able to process arbitrary *orders* and a *riskAssessment* as input parameter. Besides the obviously complete dependency based on the *riskAssessment* parameter, we detect a further dependency of *execute order* from *check competencies*. Due to the semantic reasoning, we can determine that *exchangeOrder* is a *specialization* of *order* (step I.1.) and that the relevant restrictions are either identical

(step II.1.1.) or that the restriction of the provided parameter is a non-empty, strict subset of the restriction of the required parameter (step II.1.2.). An excerpt of the ADG is shown in Figure 6.

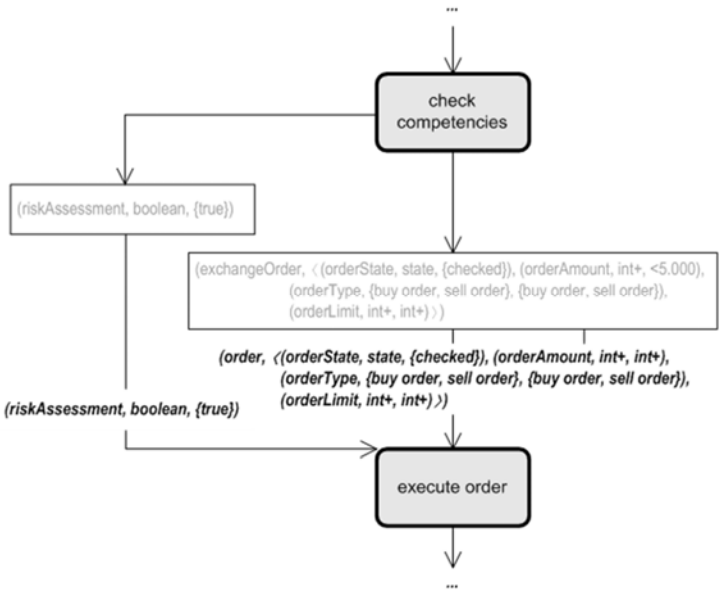

Figure 6. Excerpt of the ADG illustrating the dependency, based on a semantic matching.

The algorithm constructs the ADG starting with the goals G_x. It identifies at first all actions in lib_A delivering at least one of the parameters, the goal parameters can be deduced from, and adds them to the graph. Afterwards, the algorithm searches iteratively actions, on which already added actions depend. Those actions are added to the graph as well. This procedure is continued until either all input parameters of the graph (that are not given in the initial state) are provided by at least one action or until there are no more actions in the library that could provide the required input parameters. Here, the ADG contains only those actions that might be required to reach the goals. Actions, whose inputs are never needed by any of the actions being part of the ADG, are not considered in the graph. Also actions that are required to receive only *Init* will be ignored and not added to the ADG.
Subsequently, a post-processing takes place. Some of the actions in the current ADG may have input parameters that cannot be delivered by other actions. These actions (and their input parameters as well as the output parameters that are only provided by these actions) are iteratively removed from the graph. Afterwards, each remaining output parameter is removed if it is not needed as input inside the ADG. If all output parameters of an action are deleted, we also delete the providing action, because such actions do not contribute to reach one of the goals. Both steps – removing input parameters and output parameters (with their requiring/providing actions) – are executed recursively until no more parameter can be removed and we obtain a stable ADG. Thus, we increase efficiency in the subsequent steps of the algorithm. Note that the situation may occur that some goals could not be achieved completely. This happens, if there is a parameter as element of the goals, which is not provided by any action or the initial state. Those goals are reported to the user as they cannot be fulfilled and a further consideration during the planning process is not reasonable.

Generation of the Action State Graph (ASG)

The subsequent order of actions is far from being determined by the ADG. In the second step of SEMPA, we therefore apply an algorithm that searches the action-state space in order to bring actions in a feasible order. We define a state in this context as the current set of parameters that are available at a certain point in the plan:

(D5) A state s is a subset of the set of parameters P, $s \subseteq P$.

After applying an action a to a state s, a new state is determined by a so called state-transition function (cf. Heinrich et al., 2009). To put it in a nutshell: Output parameters that already existed in s are updated considering the new restrictions, output parameters that did not exist before are added to the new state, and parameters that existed before are deleted to the new state.
In this context it has to be pointed out, that our notion of a state differs slightly from the definitions usually applied for planning problems. Whereas classical planning approaches in general rely on the definition of a state as a set of propositions or first-order literals and logical connectives (cf. Ghallab et al., 2004), we consider a state as a kind of knowledge base containing the (available) output parameters of the previously executed actions. Defining a state as a set of parameters conforms to a state-variable representation of a planning problem, where parameters are considered as variables adopting ranges of values within their specified domain. This is advantageous, as it constitutes a more intuitive representation in comparison to classical planning approaches and simplifies the usage of numerical data types. At the same time this notion of a state conveys the non-deterministic character of the planning problem as a state is not constituted by the individual values of parameters but by the restrictions that currently hold for the parameters (cf. requirement (R5)).
An appropriate (non-deterministic) planning algorithm must be able to handle composite input and output parameters with various data types (cf. requirements (R3) and (R4)). Moreover the result of planning should be a suitable basis to extract different feasible solutions (R8) and to design control structures (R7) in the third step of SEMPA. For the further understanding of our algorithm, we need to contemplate on the relationship between actions and states first, taking into account these requirements. This essentially comprises two issues: First, it has to be defined under which circumstances an action is regarded to be applicable in a state, and second, the transformation of states through actions has to be examined.
An action a is called to be applicable in a state s, if for each input parameter of a there is at least one parameter in s matching this input parameter. In this respect, complete as well as partial matches are considered, i.e. for each input parameter of a there must be at least one completely or partially matching parameter in s. Input and output parameters of actions have already been matched in the first step of the algorithm. The corresponding information – represented by the ADG – is now used to identify applicable actions (for a technical definition cf. Heinrich et al., 2009). After considering an action in the plan, a new state is determined. The new parameters generated by the action are added and changes regarding the restrictions of already existing parameters are updated.
With these prerequisites we can now turn our attention to the search algorithm itself. The algorithm implements a forward search in the action-state space. Starting with the initial state *Init*, it identifies applicable actions by means of the ADG. After each action a new state is determined. As a state in general can be followed by several executable actions, this procedure results into a branched structure we call ASG. More precisely, the ASG is a directed graph, which consists of two kinds of nodes, action nodes and state nodes that appear in alternating order, starting with the initial state *Init:*

(D6) An ASG is an acyclic, bipartite, directed graph $G_{ASG} = (V_{ASG}, E_{ASG})$, with the set of nodes V_{ASG} and the set of edges E_{ASG}.

(D7) The set of nodes V_{ASG} consists of two partitions, the set of action nodes $Part_a^{ASG}$ and the set of state nodes $Part_s^{ASG}$. Each state node $s \in Part_s^{ASG}$ is representing one distinct state in the ASG. An action node $n \in Part_a^{ASG}$ is defined as $n: = (a, j)$, with an action $a \in Part_a^{ADG}$ and $j \in \mathbb{N}$ (index).

We use the index *j* to be able to differentiate action nodes containing identical actions (an action may be planned more than once according to assumption (A2)).
Figure 7 shows a part of the ASG in our example. In the initial state only the action *validate order* is applicable. This leads to a new state, where the element *orderState* being part of the composite parameter *order* changes its value either to *valid* or *invalid*. In this state three different actions can be applied.

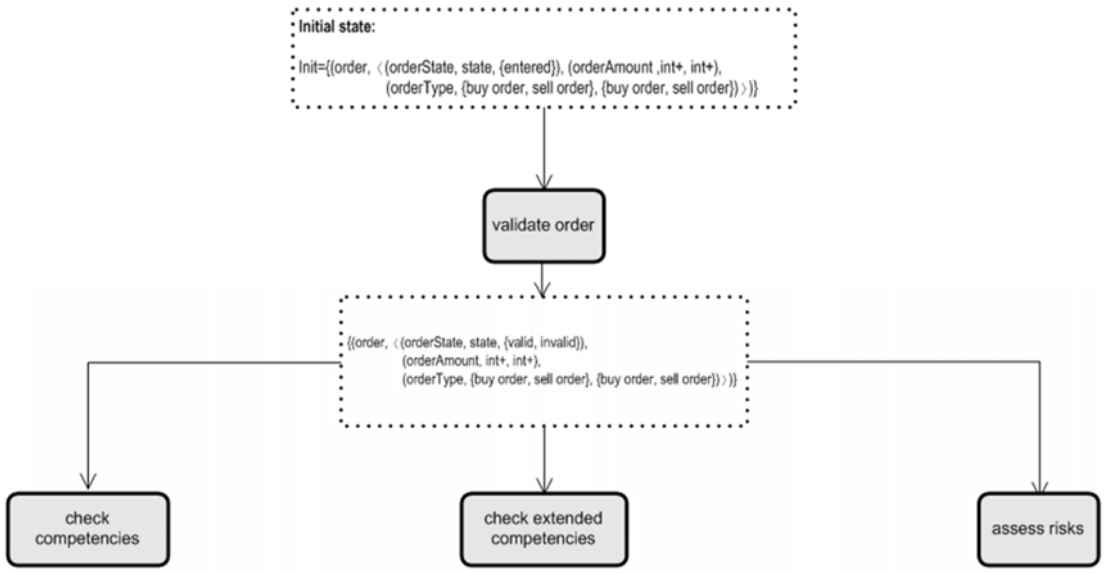

Figure 7. Extract of the ASG in the example.

As in each state a potentially high number of actions may be applicable, a large action-state space needs to be explored. Thus, there are measures included in the algorithm to avoid unnecessary searching (orientated at the ideas of Bertoli et al., 2001). We will give a short overview of these measures in the following paragraphs (cf. Henneberger et al., 2008).

1. Planning is conducted by a depth-first search. Backtracking takes place, if a state *s* is encountered that meets at least one of the specified goals, i.e. $s \subseteq G_x \in Goals$. The depth-first search guarantees fast results, if only a limited number of alternative feasible solutions is needed (if for instance the user only wants to choose from a limited number of solutions) and the solutions comprise only a few branches. At the same time the algorithm ensures that all paths are explored and all feasible solutions are found.

2. The algorithm includes a simple loop checking schema: It stops exploring a current state, if an identical state has already occurred. Consider for instance the case that a state *t'* after applying an action *a* is identical to a state *t* before *a*. Theoretically the action *a* could be applied in *t'* again, which eventually would result in an infinite loop. The algorithm detects that *t'* is identical to *t* and initiates a backtracking.

3. States leading to at least one of the specified goals are saved. New states are compared to these states. If there is an identical state in an already explored path leading to one of the specified goals, then a link is established to this state. Further analysis of the current path is not necessary anymore and a backtracking takes place. This case occurs for instance, when there are two actions *a* and *b*, that can be put in either order *a* before *b* or *b* before *a* after a state *s* and both alternatives result into an identical state *s'*. If there is at least one path from *s'* reaching a goal, then this path is explored just

once by the algorithm. In this respect it is an advantage of the depth-first search that a path is completely explored before the next path is analyzed.

4. Likewise, the algorithm tries to determine (as far as possible in advance), whether it is impossible to reach a goal from the current state. In this case the algorithm does not have to explore this state any further and again backtracking takes place. To this end, the information about failures during the search (i.e. situations where no action can be applied anymore or where the search resulted into an infinite loop) is saved in a failure log. Whenever a newly created state meets the conditions specified in the failure log, processing of the current state is stopped and again a backtracking is initiated.

Generation of Process models

In the third step of SEMPA control structures are constructed within the ASG and process models (feasible solutions) are created. In the following, we illustrate the fundamental ideas and techniques of this step of SEMPA omitting the underlying technical definitions (cf. Heinrich et al., 2009 for a technical description). The construction of both control structures and process models have not yet been regarded by traditional planning approaches so far (cf. Meyer & Kuropka, 2006). Therefore, it is a widely novel challenge. Up to now we are able to consider the control structures *sequence*, *parallel split*, *synchronization*, *exclusive choice,* and *simple merge*. The control structure *exclusive choice* indicates different branches of a process model that are merged by the corresponding control structure *simple merge*. *Parallel split* and *synchronization* are used to allow concurrent execution of actions that are independent in terms of input and output parameters for example. These control structures in fact constitute a subset of the control structures defined in (R7). Future versions of the algorithm are supposed to handle additional control structures.
In order to construct control structures, the third algorithm within SEMPA analyzes the alternating state and action nodes in the ASG as well as the partial and complete dependencies between actions defined in the ADG. The design of *sequences* is straight forward as all actions in the (acyclic) ASG are already in a feasible order. If there is a state node in the ASG with one preceding action node *a* and exactly one succeeding action node *b*, then we can build a *sequence b* following *a*.
Generally a state node may have more than one succeeding action node in the ASG. Yet, not every branching indicates an *exclusive choice* as the following considerations will show: An ASG comprises all possible paths from the initial state to the goals. Let there be two actions that can alternatively be applied as they deliver identical output parameters for instance. In the ASG this entails a branching with two different paths ultimately leading to the same goal. Obviously the two paths belong to two different feasible solutions, i.e. two different process models. In this case one feasible solution is only a sub graph of the ASG. In order to represent single feasible solutions in the form of process models we need to separate them in the ASG. Accordingly, we need to clearly distinguish branches in the ASG resulting into *different branches within* a process model from branches that are ultimately leading to *alternative process models* representing different feasible solutions. But how can we construct *exclusive choices* under these circumstances? Consider a state *s* with a number of succeeding actions $a_1 \ldots a_n$. The algorithm analyzes the sets of input parameters of these actions (considering as well the set of parameters provided by state *s*):

- If restrictions are mutually exclusive for at least one input parameter, then obviously no feasible combination of values of the parameters in *s* exists, so that more than one action can be executed.

- If restrictions are identical for all input parameters, then all of the actions $a_1 \ldots a_n$ can be executed at a time. They can be regarded as alternatives that may belong to different feasible solutions. We mark the state node representing *s* with a delimiter symbol in order to separate out different feasible solutions. For scalability reasons we keep on working with one overall ASG and extract different feasible solutions later on.

- If neither of these cases applies, the algorithm determines disjoint (concerning parameter restrictions) sets of actions. We will describe the basic principle with an example in our example. Consider the state after *validate order* in the ASG in Figure 7. In this state the ASG branches into three different process actions: *check competencies*, *check extended competencies* and *assess risks*. Restrictions are neither identical nor mutual exclusive. However, in this case, the restrictions for all input parameters of the actions *check extended competencies* and *assess risks* are identical (both requiring *orderAmount > 5,000*). This indicates they are alternatives and a delimiter symbol is needed. Additionally, regarding the input parameter *orderAmount* there is no overlap with the remaining action *check competencies* (requiring *orderAmount* ≤ *5,000*). Thus, the algorithm adds a decision node distinguishing between the cases ≤ *5,000* and > *5,000*. By this, we demonstrated the idea considering exactly one input and output parameter at a time (here: *orderAmount*). The idea remains the same, even when there are several input and output parameters. The main difference is that analysis and decomposition get more complex (cf. Heinrich et al., 2009).

- The algorithm also tests if there are feasible combinations of values of the output parameters in a state *s*, so that no action can be executed at all. In order to get a syntactically correct activity diagram, we then add a decision node branching directly *t*o a UML final node. In our example this applies in the state after the action *validate order*. All of the preceding actions require an input parameter *orderState = valid*. In the case *orderState = invalid*, which is also a reasonable *result of validate order,* no action can be executed. Thus, the process *needs to terminat*e (in a final *node).*

*After construc*ting *exclusive choices* the algorithm t*ries to find simple merges*. Here, the information about prior *exclusive choices* is needed. Therefore, this information is passed to all subsequent state nodes along the ASG, i.e. for each state node n_i we save a cumulative, ordered set M_i of (distinguishable) control structures that occurred along the way from the initial state to this node. We need the sets M_i to realize branches that can be merged (in contrast to branches belonging to alternative solutions). Starting with the state nodes in the ASG representing goals we now identify state nodes where we can merge different branches of the ASG.

Finally, we test for cases where we can parallelize actions using the control structures *parallel split* and *synchronization*. For that reason, we analyze all state nodes n_k in the ASG marked with a delimiter symbol before. For instance, when a number of l actions following n_k in each branch are identical but in another order and at the same time the sequence of l actions in all branches is completed with a state node representing identical states, then we can place a parallelization. Again we demonstrate our approach with our example. As determined before, in the state after *validate order*, we have to differentiate two feasible solutions. As we can see, the actions *check extended competencies* and *assess risks* are independent from each other. We could either execute *check extended competencies* before *assess risks* or the other way around and furthermore we could parallelize both actions.

With the considerations above we can finally construct syntactically correct UML activity diagrams which are graphical representations of our feasible solutions. State nodes in the ASG representing a goal are replaced by final nodes. The initial state is replaced by a start node.

How does the process model – planned by SEMPA – now finally look like in our running example? A conceivable planning result is depicted in Figure 8. The financial services provider receives a *stock order* which is validated at first. If the *stock order* is "*invalid*" (e.g. because of lack of data), the process is aborted. If we have a "*valid*" *stock order,* it is differentiated, whether the *order amount* is greater than or equal *5,000* or less than *5,000*. In the latter case, competencies are checked. Otherwise (≥ *5,000*), an extended check of competencies is required and a separate risk assessment takes place. Having finished these actions, the *stock order* can finally be executed.

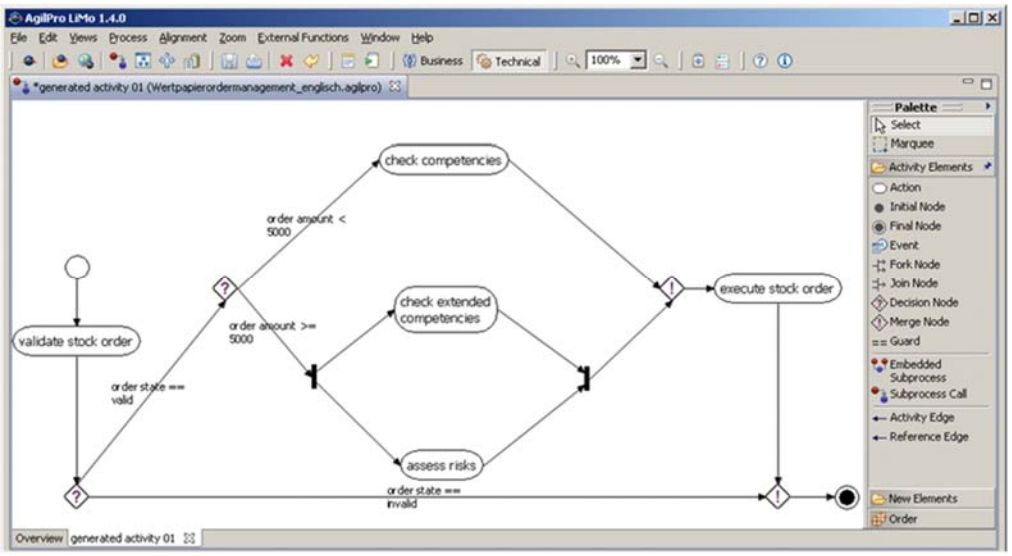

Figure 8. UML activity diagram in the process modeling tool AgilPro[iv].

EVALUATION

SEMPA has been evaluated in several ways. First, it has been prototypically implemented. Second, SEMPA and its prototypical implementation have been evaluated with regard to the requirements and have been applied to various process modeling problems in practice. Third, the SEMPA algorithm and the outcome of planning, namely the process models, have been analyzed theoretically including formal criteria (e.g. termination of the planning algorithm). In the following, we will describe selected aspects of the evaluation starting with the prototypical implementation.

SEMPA has been realized as a plug-in for the open-source process modeling tool AgilPro. Using AgilPro, processes can be modeled, displayed in different views or simulated. We use OWL 2 to define ontologies, because of the higher expressive power in comparison to OWL 1.0. For the processing of ontologies, the OWL API (OWL Application Programming Interface) is used. The OWL API is a Java interface and implementation for the handling of ontologies in OWL. It supports OWL Lite and OWL DL and offers an interface for inference engines such as Pellet. The latter is employed for the semantic-based reasoning in the prototypical implementation. Since Pellet as well as the OWL API implement the DIG-interface, other Semantic Web frameworks (such as KAON2) could be used as well. The prototypical implementation has been subject to an evaluation. It has been tested with jUnit tests and the algorithm has been applied to several examples in different application domains with diverse actions, parameters, etc.

Facing the approach with the requirements outlined before reveals the following insights: Obviously, the input and output parameters of actions constitute the basis for planning; they are matched in the first step of SEMPA. Accordingly, requirement (R1) is fulfilled. Considering the definition of input and output parameters above, requirements (R2), (R3), and (R4) are fulfilled as well. Furthermore, semantic-based reasoning capabilities (requirement (R6)) are supported. SEMPA is capable of determining identity, equality and specialization relationships between parameters in the ontology and consider them for the planning of the process models. In the third step, control structures are constructed in the ASG and finally process models are generated. In requirement (R7) different control structures have been outlined that should be recognized: *exclusive choice*, *parallel split*, *synchronization*, *simple merge*, etc. The previous sections described how those can be recognized, focusing particularly on *exclusive choices*. Since feasible solutions are represented as valid UML activity diagrams, moreover, requirement (R8) is fulfilled.

From a theoretical perspective, we can show that the planned process models are *valid* with respect to certain criteria like completeness, minimality, termination and computational complexity regarding time. In this respect, we have to refer to Heinrich et al. (2009) due to length restrictions.

RESULTS AND FURTHER RESEARCH

In this chapter we introduced the SEMPA approach for the automated planning of process models. The idea of SEMPA was motivated by a real world problem, which means the high effort for the manual modeling of processes. Our search for an algorithm for the automated planning of process models was guided by a set of requirements. The three steps of SEMPA have been prototypically implemented. From different test scenarios an order management process of a financial services provider has been chosen to illustrate the application of SEMPA. Moreover, the approach can also be applied to plan processes of other domains and industries like e.g. industrial production processes. Therefore, it is necessary that the domain specific actions with their inputs and outputs including resource requirements or capacity restrictions are semantically described and available in a process library. After all, the contributions to the field of SBPM in general and planning of process models in particular are:

1. SEMPA constructs the control structures *sequence*, *parallel split*, *synchronization*, *exclusive choice* and *simple merge* in an automated way. In comparison to existing planners there is a particular improvement e.g. regarding the structures *exclusive choice* or *parallel split*.

2. Currently manually designed process models are suffering from inconsistently used terms which restricts their economically reasonable application, i.e. before using or adapting process models terms have to be clarified, coordinated and adjusted (especially if modelers are altering). The specification of an ontology together with semantic reasoning can sustainable contribute to a reduction of costs for clarification and adjustment. In contrast to many existing planners, SEMPA is creating added value here. A faster and more flexible construction of process models seems possible. The added value of a semantic description applies also to the implementation of processes with Web Services. It enables an automated search (cf. Drumm et al., 2006) for (semantic) Web Services based on the semantic description of the actions. This is of particular interest, as the number of new Web Services that are offered, will probably increase significantly. Thus a continuous and cost-effective valuation of such new offers regarding their application in process models is gaining importance.

3. Existing planning approaches mainly focus on the generation of a single feasible solution for a problem definition with a start point and exactly one single end point (one goal). With SEMPA process models with multiple end points (set of goals) can be generated. This is of particular importance, because today and in the near future it seems to be unrealistic that a planner generates a model for a business-critical process (e.g. an order process) isolated for exactly one goal and that this process can be realized in an automated way afterwards without (further) discussions of other processes.

4. Another important aspect is the generation of a set of feasible solutions for a given goal specification. It is the starting point to derive the set of efficient solutions (those that are not dominated by other feasible solutions) and finally an optimal solution for a specific company considering economic aspects. For this purpose not only a mapping of available Web Services realizing the feasible solutions, but also economic evaluation parameters and the Quality of Service properties of a Web Service have to be available.

Alongside also critical points have to be mentioned which define our further research. Currently not all control structures that could be part of process models are supported by the algorithm. Furthermore, information such as resources, execution time and cost are not considered for planning in the current version of SEMPA. We intend to advance especially the planning algorithm in the second step. To this end established planning approaches with time and resources provide an appropriate starting point. The SEMPA approach forms an appropriate fundament for this as well as for the aforementioned enhancements and thus serves as a suitable basis for further research.

REFERENCES

Becker, J., & Kahn, D. (2003). The process in focus. In J. Becker, M. Kugeler & M. Rosemann (Eds.), *Process Management. A Guide for the Design of Business Processes* (pp. 1-12). Berlin: Springer.

Becker, J., Rosemann, M., & von Uthmann, C. (2000). Guidelines of Business Process Modeling. In W. M. P. van der Aalst, J. Desel & A. Oberweis (Eds.), *Business Process Management, Models, Techniques and Empirical Studies* (pp. 30-49). Berlin: Springer.

Berardi, D., De Giacomo, G., Mecella, M., & Calvanese, D. (2006). Automatic Web Service Composition: Service-Tailored vs. Client-Tailored Approaches. In *Proceedings of the 4th International Workshop on AI for Service Composition (AISC) 2006*. Retrieved February 15, 2010, from http://ftp.dis.uniroma1.it/~degiacom/papers/2006/bera-calv-degi-mece-AISC-2006.pdf.

Bertoli, P., Cimatti, A., Roveri, M., & Traverso, P. (2001). Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In B. Nebel (Ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI) 2001* (pp. 473-478). San Francisco: Morgan Kaufmann.

Betz, S., Kling, S., Koschmider, A., & Oberweis, A. (2006). Automatic User Support for Business Process Modeling. In K. Hinkelmann, D. Karagiannis, N. Stojanovic & G. Wagner (Eds.), *Proceedings of the Workshop on Semantics for Business Process Management at the 3rd European Semantic Web Conference (ESWC) 2006* (pp. 1-12). Berlin: Springer.

Biron, P. V., & Malhotra, A. (2004). *XML Schema Part 2: Datatypes Second Edition*. Retrieved February 15, 2010, from http://www.w3.org/TR/xmlschema-2.

Bonet B., & Geffner H. (2000). Planning with Incomplete Information as Heuristic Search in Belief Space. In S. Chien, S. Kambhampati & C. A. Knoblock (Eds.), *Proceedings of the 5th International Conference on AI Planning and Scheduling (AIPS) 2000* (pp. 52-61). Madison: AAAI Press.

Bonet, B., & Geffner, H. (2001). GPT: A Tool for Planning with Uncertainty and Partial Information. In B. Nebel (Ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI) 2001* (pp. 82-87). San Francisco: Morgan Kaufmann.

Borges, M. R. S., Pino, J. A., & Valle, C. (2005). Support for decision implementation and follow-up. *European Journal of Operational Research, 160*(2), 336-352.

Cimatti, A., Roveri, M., & Traverso, P. (1998). Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI) 1998* (pp. 875-881). Madison: AAAI Press.

Constantinescu, I., Faltings, B., & Binder, W. (2004a). Large scale, type-compatible service composition. In: *Proceedings of the IEEE International Conference on Web Services (ICWS) 2004* (pp. 506-513). Retrieved February 15, 2010, from http://www.dip.deri.org/documents/Constantinescu-et-al-Large-Scale-Type-Compatible-Service-Composition.pdf

Constantinescu, I., Faltings, B., & Binder, W. (2004b). Type-based composition of information services in large scale environments. In N. Zhong, H. Tirri, Y. Yao & L. Zhou (Eds.), *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI) 2004 (pp. 306-312)*. Retrieved February 15, 2010, from http://ieeexplore.ieee.org/servlet/opac?punumber=9689

Drumm, C., Lemcke, J., & Namiri, K. (2006). Integrating Semantic Web Services and Business Process Management: A Real Use Case. In K. Hinkelmann, D. Karagiannis, N. Stojanovic & G. Wagner (Eds.),

Proceedings of the Workshop on Semantics for Business Process Management at the 3rd European Semantic Web Conference (ESWC) 2006 (pp. 1-12). Berlin: Springer.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning*. San Francisco: Elsevier.

Giunchiglia, F., Yatskevich, M., & Shvaiko, P. (2007). Semantic Matching: Algorithms and Implementation. *Journal on Data Semantics, 9,* 1-38.

Heinrich, B., Bewernik, M., Henneberger, M., Krammer, A., & Lautenbacher, F. (2008). SEMPA - A Semantic Business Process Management Approach for the Planning of Process Models. *Business & Information Systems Engineering (formerly WIRTSCHAFTSINFORMATIK), 50*(6), 445-460 (in German).

Heinrich, B., Bolsinger, M., & Bewernik, M.-A. (2009). Automated Planning of Process Models: The Construction of Exclusive Choices. In H. Chen & S. A. Slaughter (Eds.), *Proceedings of the 30th International Conference on Information Systems (ICIS) 2009* (Paper 184). Berlin: Springer.

Henneberger, M., Heinrich, B., Bauer, B., & Lautenbacher, F. (2008). Semantic-Based Planning of Process Models. In M. Bichler, T. Hess, H. Krcmar, U. Lechner, F. Matthes, A. Picot, B. Speitkamp & P. Wolf (Eds.), *Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI) 2008* (pp. 1677-1689). Berlin: Gito.

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 W. Tsai, J. Chung & Y. Muhammad (Eds.), *IEEE International Conference on e-Business Engineering (ICEBE) 2005* (pp. 535-540). Retrieved February 15, 2010, from http://ieeexplore.ieee.org/servlet/opac?punumber=10403.

Hoffmann, J., & Brafman, R. I. (2005). Contingent Planning via Heuristic Forward Search with Implicit Belief States. In S. Biundo, K. Myers & K. Rajan (Eds.), *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS) 2005* (pp. 71-80). Menlo Park: AAAI Press.

Kabanza, M., Barbeau, M., & St-Denis, R. (1997). Planning Control Rules for Reactive Agents. *Artificial Intelligence, 95*(2), 409-438.

Lang, Q. A., & Su, S. Y. W. (2005). AND/OR Graph and Search Algorithm for Discovering Composite Web Services. *International Journal of Web Services Research, 2*(4), 46-64.

Ma, Z., & Leymann, F. (2008). A Lifecycle Model for Using Process Fragment in Business Process Modeling. In S. Nurcan, R. Schmidt, P. Soffer, E. Hunt, X. Franch & R. Coletta (Eds.), *Proceedings of the 9th Workshop on Business Process Modeling, Development, and Support held in conjunction with the CAiSE'08 Conference* (pp. 1-9). Aachen: Sun SITE Central Europe.

McIlraith, S., & Son, T. C. (2002). Adapting Golog for Composition of Semantic Web Services. Retrieved February 15, 2010, from www.cs.nmsu.edu/~tson/papers/kr02gl.pdf.

Meyer, H., & Kuropka, D. (2006). Requirements for Automated Service Composition. In J. Eder & S. Dustdar (Eds.), *Business Process Management Workshops, Lecture Notes in Computer Science Vol. 4103* (pp. 439-450). Vienna: Springer.

Meyer, H., & Weske, M. (2006). Automated Service Composition using Heuristic Search. In: S. Dustdar, J. L. Fiadeiro & A. Sheth (Eds.), *Proceedings of the 4th International Conference on Business Process Management (BPM) 2006* (pp. 81-96). Berlin: Springer.

Pathak, J., Basu, S., & Honavar, V. (2006). Modeling Web Service Composition using Symbolic Transition Systems. In P. Doshi, R. Goodwin & A. Sheth (Eds.), *Papers from the AAAI Workshop on AI-Driven Technologies for Service-Oriented Computing 2006*. Menlo Park: AAAI Press.

Petrick, R. P. A., & Bacchus, F. (2002). A Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In M. Ghallab, J. Hertzberg & P. Traverso (Eds.), *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS) 2002* (pp. 212-221). Menlo Park: AAAI Press.

Pistore, M., Traverso, P., Bertoli, P., & Marconi, A. (2005). Automated Synthesis of Composite BPEL4WS Web Services. In *Proceedings of the 3rd International Conference on Web Services* (pp. 293-301). Retrieved February 15, 2010, from http://klase.itc.it/paper_firb_astro_klase/additional/AM_ICWS05.pdf.

Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., & Wohed, P. (2006). On the Suitability of UML 2.0 Activity Diagrams for Business Process Modeling. In J. F. Roddick & A. Hinze (Eds.), *Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modeling (APCCM) 2006* (pp. 95-104). Hobart: Australian Computer Society.

Sirin, E., Parsia, B., Wu, D., Hendler, J., & Nau, D. (2004). HTN Planning for Web Service Composition Using SHOP2. *Journal of Web Semantics, 1*(4), 377-396.

Smith, M. K., Welty, C., McGuinness, D. L. (2004). *OWL Web Ontology Language Guide*. Retrieved February 15, 2010, from http://www.w3.org/TR/owl-guide/.

Ter Beek, M. H., Bucchiarone, A., & Gnesi, S. (2006). *A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods, Technical Report 2006-TR-15, ISTI, Consiglio Nazionale delle Ricerche*. Retrieved February 15, 2010, from http://fmt.isti.cnr.it/WEBPAPER/TRWS-FM06.pdf.

Thomas, O., & Fellmann, M. (2006). Semantische Integration von Ontologien und Ereignisgesteuerten Prozessketten. In M. Nüttgens, F. Rump & J. Mendling (Eds.), *Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten EPK 2006* (pp.7-23). Bonn: GI.

Van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., & Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases*, *14*(3), 5-51.

---

[i] Units may be Dollars or Euros respectively. For the sake of simplicity we omit units in the following text.

[ii] In this chapter, we use the terms input and output parameters instead of the terms *preconditions* and *effects*, which are well-established in the field of AI planning. Following this, we use the term *input parameter* for everything an action needs in order to be performed and the term *output parameter* for everything an action provides after it was performed. This also includes e.g. also resource requirements or capacity restrictions.

[iii] Besides simple classes, an OWL ontology can also define complex classes by constructors like union, intersection and complement or by enumerating its individuals as class extension (cf. Smith et al., 2004).

[iv] The notation of the tool is based on UML activity diagram although there are minor differences concerning the form of the symbols (e.g. decision node).