University of Augsburg
Prof. Dr. Hans Ulrich Buhl
Research Center
Finance & Information Management
Department of Information Systems
Engineering & Financial Management

Discussion Paper WI-256

Automated Planning of Process Models: The Construction of Exclusive Choices

by

Bernd Heinrich, Manuel Bolsinger, Marc Bewernik

in: Proceedings of the 30th International Conference on Information Systems,
ICIS, Phoenix, Arizona, December 2009

Universität Augsburg, D-86135 Augsburg
Visitors: Universitätsstr. 12, 86159 Augsburg
Phone: +49 821 598-4801 (Fax: -4899)
www.fim-online.eu

AUTOMATED PLANNING OF PROCESS MODELS: THE CONSTRUCTION OF EXCLUSIVE CHOICES

Completed Research Paper

Bernd Heinrich
Department of Information Systems,
Production and Logistics Management,
University of Innsbruck
Universitaetsstr. 15, Innsbruck, Austria
bernd.heinrich@uibk.ac.at

Manuel Bolsinger
FIM Research Center,
Department of Information Systems,
Engineering & Financial Management,
University of Augsburg
Universitaetsstr. 12, Augsburg, Germany
manuel.bolsinger@wiwi.uni-augsburg.de

Marc Bewernik
FIM Research Center,
Department of Information Systems,
Engineering & Financial Management,
University of Augsburg
Universitaetsstr. 12, Augsburg, Germany
marc.bewernik@wiwi.uni-augsburg.de

Abstract

In our competitive world, companies need to adapt their processes quickly in order to react, for instance, to changing customer demands. Process models, as means to support process management, are nowadays often created and adapted in a time-consuming, widely used manual manner. Semantic Business Process Management in combination with planning approaches can alleviate this drawback by enabling an automated planning of process models. This paper describes the drawbacks that existing planning algorithms have related to the creation of process models. Therefore, we introduce - based on the design science paradigm - an innovative algorithm (method) that is suitable for the planning of process models focusing on the construction of the control flow pattern exclusive choice. Demonstrating the feasibility and the effectiveness of our method, we implemented our approach as a prototype. Finally, we evaluate the algorithm in terms of different properties like termination and its applicability within a real-use situation.

Keywords: Process modeling, Design Science Research, Automated planning

Introduction

In order to describe the increasingly complex processes within and across enterprises as well as for communication and training purposes, process modeling has proven to be an important instrument. A number of process modeling techniques have been developed in the past decades, including modeling languages like Event-driven Process Chains (EPC) or UML activity diagrams. However, process modeling and optimization are still time-consuming in practice, if new process models need to be designed or existing ones need to be adapted to changing requirements (Becker and Kahn 2003; Borges et al. 2005; Ma and Leymann 2008). Hornung et al. (2007) wrote, for instance, that "Manual process modeling is a time-consuming task and thus increases the total amount of modeling time.". Nevertheless, changing customer needs etc. make it necessary to maintain and adjust process models frequently. Even if reference process models are used, they have to be changed due to the new requirements as well, which again is time-consuming and costly. In addition, in many domains there are no reference process models at all that can be used. With this in mind, a fast and under economic considerations reasonable construction or adaption of process models is often difficult. For that reason, many process management departments in companies also need to deal with the criticism to cause too high costs compared to their benefit (e.g. Recker et al. 2005). A semantic annotation of process models, as envisioned in the research area Semantic Business Process Management (SBPM) can alleviate this drawback (Betz et al. 2006; Brockmans et al. 2006; Hepp et al. 2005; Hepp and Dumitri 2007; Thomas and Fellmann 2007) in combination with existing AI planning techniques (Bertoli et al. 2006; Hoffmann and Brafman 2005) by enabling an automated planning of process models (Heinrich et al. 2008; Henneberger et al. 2008).

In order to plan process models, not only a sequence of actions – the atomic elements of a process – but also other control structures (see Van der Aalst et al. 2003), which are provided by modeling languages and describe the control flow of a process, have to be constructed automatically. Thus, a fundamental challenge for the planning of process models is to consider such control structures, which can be reduced to a few fundamental control flow patterns. As the control flow pattern *exclusive choice* (besides e.g. the *simple merge*, and the *parallel split*) is one of the basic patterns for designing process models, this paper will examine its *automated construction*. The task of an automated construction of process models can be understood as a planning problem (Ghallab et al. 2004) with the objective to arrange the single process components, i.e. the actions, in an appropriate order.

To this end, we introduce – based on the design science paradigm – a technical definition of our planning domain and a novel algorithm (method) for the automated construction of exclusive choices within process models (cp. Figure 1). Therefore, we need special definitions provided by an abstract representation language. The main contributions are as follows[1]:

❶ The construction of exclusive choices is based on the preconditions and effects of actions[2]. In addition, planning has to consider that various, maybe large data types e.g. double (Biron and Malhotra 2004) can be assigned to them and that some actions accept only certain ranges of values of a data type (so called restrictions). We tackle these challenges by means of the representation of possibly infinite *sets of world states*, so called *belief states*.

❷ These belief states are presented in an *abstract representation language* which makes it possible to construct exclusive choices independently from well-known representation languages.

❸ This enables the definition of a planning domain (*conditional deterministic belief-state-transition system*) and a planning problem which allow us to design the necessary conditions (for instance, in UML they are called guards) – needed for the planning of the control flow pattern exclusive choice – automatically by an *algorithm*.

Considering the guidelines for conducting design science research by Hevner et al. (2004), we organize the paper as follows: The second section specifies both the problem context for which the new approach is relevant and the requirements that must be met in order to plan the exclusive choice in process models whereupon the related work is discussed. The third section introduces an abstract representation language and shows how belief states are represented. This is followed by a description of the planning model, our planning domain. Section five answers the key research question of how the exclusive choice control flow pattern can be constructed automatically by

[1] The semantic annotation of actions is analyzed in a step before the planning of control structures and is therefore not the focus of this paper (for a more detailed description of how the semantic annotation is used, we refer to Heinrich et al. 2008).

[2] The term *preconditions* denotes everything an action needs in order to be performed, including input parameters; the term *effects* denotes everything an action provides after it was performed, including output parameters, as it is used in AI planning.

providing the necessary algorithm. The penultimate section is dedicated to an evaluation of our approach. Here, we demonstrate the feasibility and the implementability of our approach by means of a prototype (instantiation) and illustrate its application within a real-use situation (here, our instantiation has the character of a working example (see also Gregor and Jones 2007, p. 323)). Furthermore, we mathematically evaluated the presented approach in terms of different properties like termination. Finally, the last section summarizes our considerations and provides an outlook on future steps.

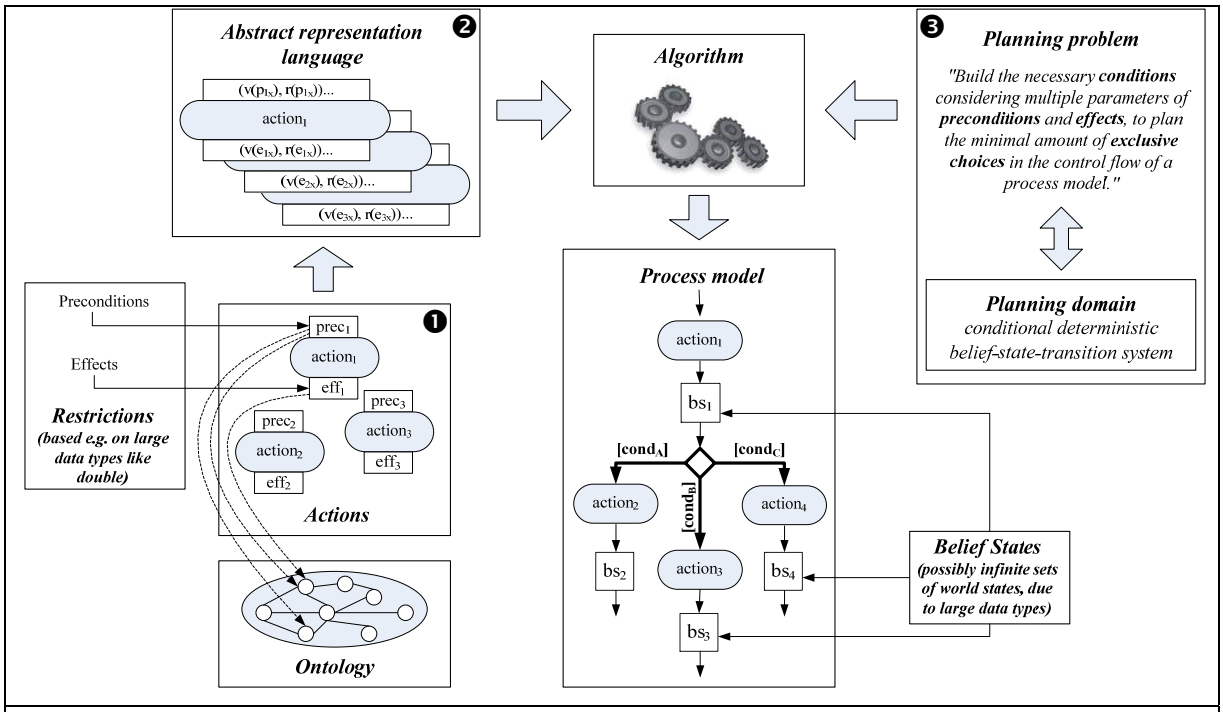

Figure 1. The Planning Approach to Construct the Control Flow Pattern Exclusive Choice

Problem Context, Requirements and Related Work

At first, we present the problem context of this paper with regard to the research strand of SBPM. After this, we discuss the types of processes, which seem to be appropriate for the use of automated process planning in practice. That is followed by aligning the problem context to the planning of exclusive choices, before we formulate the requirements for the planning domain and the algorithm, which are based on literature. At the same time, these requirements are the source for the subsequent analysis of the related work, to define the need for research.

Problem Context

Based on the awareness that process modeling is time-consuming, the emerging research strand of SBPM tries to reach a higher level of automation in the creation and adaptation of process models by means of their semantic annotation as well as planning algorithms. For a more detailed description of SBPM see e.g. Hepp et al. (2005), Hepp and Dumitri (2007) and Thomas and Fellmann (2007). This is the field of study in which we envision the automated planning of process models (which is understood as a plan in the following sections). We propose that if actions, which can be – in accordance to Hepp and Dumitri (2007) – stored in a process library, are semantically annotated, it becomes possible to create process models automatically for a given problem description (see also Henneberger et al. 2008). The annotation of an action includes a semantic annotation of the *preconditions* needed for it to be performed and the *effects* provided after it has been performed. A description for a planning problem comprises, besides the actions, of an initial state representing the overall process input and a set of goals representing the desired process output. Given such a problem description, a planner is expected to build feasible process models (e.g. Heinrich et al. 2008; Henneberger et al. 2008). During the overall planning, the semantic annotation of the corresponding elements of the problem is analyzed by an inference mechanism to identify the

existing dependencies between initial state, actions and goal states. Since this semantic reasoning can be done prior to the planning of the control flow patterns (see Heinrich et al. 2008), we do not address it further in this paper.

As discussed in the introduction, automated planning is thought to contribute to process modeling to design and adapt process models faster. However, the question of which processes are adequate to apply a process planning algorithm to and which are not, needs to be clarified. Such a classification is required to specify the business problem context as well as the boundaries within which the approach is expected to be applied. Processes are classified in literature related to different criteria (e.g. Marjanovic 2005; Weske et al. 2004). Here, criteria like degree of process repetition, frequency of process redesign and process value seem promising. According to the papers mentioned before, a professional *manual* process design is suggested (in contrast to not doing such a process design) especially for repetitive processes that need to be (re)designed repeatedly and which are of high value for a firm. This is justified primarily by economic reasons, i.e. any high initial costs of analyzing and documenting the problem domain (for which the process is constructed) as well as the costs of a manual adaption of an existing process are worthwhile, if the analysis and documentation, once made, can be used again during another redesign. The classification by these authors can be transferred in a sense to automated planning of process models. Here, too, high initial costs occur to analyze and annotate the process actions (preconditions and effects) and to implement the planning algorithm. Another similarity to *manual* process design is that the annotations and the implemented algorithm can later be reused during further (re)design projects, which reduces the costs and time of the design. Therefore, we will focus first on repetitive processes that need to be (re)designed repeatedly and which are of high value. Such processes seem to be the first choice for applying automated planning approaches. This boundary of the problem context is expected to be relevant, since usually there are many processes in companies that belong to this process class. Thus, we have to evaluate later, if under the defined conditions – especially repetitive processes that need to be (re)designed repeatedly – an automated planning of process models is useful.

In our context, we focus on the, so far, *unsolved* issue of planning the control flow pattern exclusive choice in process models. As suggested by Hevner et al. (2004), we decompose the problem of planning a whole process model into subproblems such as the planning of exclusive choices in order to address this subproblem in depth. The planning domain and algorithm have to cope with a number of requirements, which are presented in the following subsection of this paper before related work is discussed.

Requirements

The planning domain must meet general requirements (see also Bertoli et al. 2006; Constantinescu et al. 2004; Meyer and Kuropka 2005; Meyer and Weske 2006; Pathak et al. 2006). These requirements are also the basis for the analysis of existing approaches and AI planning techniques in the next subsection. The following requirements need to be considered:

(R1) *Preconditions and effects and the data types of their parameters*: It must be possible to assign various data types (e.g. defined in Biron and Malhotra 2004) to parameters of preconditions and effects of actions and consider them within the planning model. Some actions may require restricted ranges of values for a data type. Therefore, restrictions have to be considered as well. In other words, it should be possible to specify the range of values an action accepts for a precondition and the range of values it produces for an effect. Since the restrictions of consecutive actions may not match completely, the planning algorithm needs to consider all possible ranges in the process model. In a given world state according to an individual process execution, every parameter can be represented by a unique value, i.e., allowing ranges of values is equivalent to allowing a possibly infinite set of world states.

(R2) *Planning independently of a concrete representation language*: In order to use the planning approach independently of a concrete representation language, a general and formal framework (planning domain and algorithm) has to be built by the use of an abstract representation language.

(R3) *Planning of exclusive choices*: An algorithm should be able to plan control structures and especially exclusive choices in process models automatically. Van der Aalst et al. (2003) provides a comprehensive overview of the various control structures that may be part of a process model. Moreover, he analyzes how these structures are represented in different process modeling languages such as UML activity diagrams. Thus, following these findings, a planning domain needs to consider conditions for constructing control structures, such as *exclusive choice*.

Related Work

The planning problem described in this paper can be characterized as a nondeterministic planning problem with initial state uncertainty. It is nondeterministic because we abstract from an individual process execution. Therefore the realizations of parameter values (and thus the world state) are not determined at the moment of planning (Ghallab et al. 2004). Likewise parameters are not fully determined in the initial state either, which is frequently called initial state uncertainty (R1) (Bonet and Geffner 2001). Although there are already algorithms that can cope with nondeterminism and initial state uncertainty (e.g. Bertoli et al. 2006; Bonet and Geffner 2001), these approaches do not reach out for the planning of process models because of their limited capabilities concerning the construction of control structures (R3). Other algorithms, such as the one presented in Bertoli et al. (2006), progress from an initial belief state to one of the goal belief states. It builds a search tree, to find all possible paths beginning with the initial belief state, branching on conditions. The approaches of such conditional planning (e.g. Bertoli et al. 2006; Hoffmann and Brafman 2005) do not fit our problem. They encode so called observations, which are points in the plan, where it is necessary to determine, if some logical expression is valid or not in order to proceed further in the plan. They encode the observations separately in the form of observation variables and observation actions, respectively, making them part of their planning domain. In practice this means, that there are observations in the domain description, which makes such an observation the only point in the plan, where it might branch (e.g. in order to construct an exclusive choice). Thus, exclusive choices are possible (see also the conditional planners such as Bonet and Geffner 2001; Hoffmann and Brafman 2005) but are "hard-coded" in the domain (e.g. in terms of sensing actions) and additionally restricted to Boolean variables. In process modeling, which is our problem context, such observations are not given as this would simplify planning process models to a great extent. In practice this means, that a planning approach must determine the conditions by itself without having the observations given in advance (R3). Finally, there needs to be a way of representing belief states, in order to map one belief state into another, which is a key problem in belief space according to Geffner (2002) (R1). One promising approach seems to be the use of Binary Decision Diagrams as proposed in Bertoli et al. (2006). Another way is the representation of a belief state implicitly by an initial belief state together with a sequence of actions that leads to this belief state as done in Hoffmann and Brafman (2006). Other planners, for example in Bonet and Geffner (2000), enumerate all states of the world explicitly that may occur after applying an action. Because of large data types (e.g. double), which are essential for the problem context of process modeling, these approaches are not suitable.

As the composition of (semantic) web services forms a similar issue as the automated planning of process models we additionally want to briefly mention some planning and rule-based approaches that have already been adapted for the composition of (semantic) web services (e.g. Liang and Su 2005, Pistore et al. 2005, Weigand et al. 2008, Wu et al. 2003). However, none of these approaches meet the above-mentioned requirements for process planning, especially nondeterministic planning, the construction of exclusive choices, and the ability to handle different data types. Yet, these issues are necessary for the planning of process models.

Abstract Representation Language

In this section we present a formal definition of belief states, making it possible to explicitly represent an infinite state space independently of a concrete representation language. This is the foundation for both the description of our planning model and the development of our algorithm in order to meet the requirements (R1) to (R3). With the abstract representation language it is possible to represent possibly infinite sets of world states quite easily. Furthermore, it is a rather intuitive way – from a process modeling perspective – to express certain preconditions and effects[3]. We do not use other representation languages like set-theoretic representation or state-variable representation (see Ghallab et al. 2004), since we do not have a classical planning problem and we want to build a planning model that is independent of a concrete representation language[4]. In this way, it is a kind of specialized language to describe our problem considered in a new way (March and Smith 1995). When talking about process

[3] As mentioned above, these preconditions and effects need to be semantically annotated. This is realized by linking their parameters already during the specification of the actions with classes of an OWL 2 ontology (Motik et al. 2008). These semantics are analyzed by an inference step prior to the planning of control flows and its patterns (see Heinrich et al. 2008), so that for the further paper it is sufficient to demonstrate the approach on a syntactic basis.

[4] However, if we restrict all of the atoms and belief state variables to be ground, then from a theoretical point of view our abstract representation language has equivalent expressiveness than the languages mentioned.

models, the annotation of their actions includes a specification of the preconditions and the effects. We define a parameter of the preconditions and effects as *belief state tuple* that consists of the parameter name and a set of values, all of which can be assigned to the parameter in a specific world state (according to an individual process execution). Thus the name of a parameter is also understood as a variable that can take on all the values in the set of values. The data type of a parameter is the predefined domain of a belief state tuple.

Definition 1 *(Belief state tuple)*. A *belief state tuple p* is a tuple of a *belief state variable v(p)* and a subset *r(p)* of its predefined domain *dom(p)*, which we will write as $p := (v(p), r(p))$. It is $v(p) \in r(p)$ in a specific world state. When talking about belief states, *v(p)* is the symbol of the belief state variable. The set $r(p) \subseteq dom(p)$ is called the *belief-state-variable restriction* (abbr.: restriction) of *v(p)*, which contains the values that can be assigned to *v(p)* in a possible world state. If $r(p)=\varnothing$ then, the belief state variable does not exist (anymore), allowing the deletion of a belief state variable.

According to this definition, each belief state variable *v(p)* has a predefined data type (e.g. double) specifying the predefined domain *dom(p)*. Additionally, restrictions *r(p)* can be defined for each belief state variable *v(p)*. A restriction can either be described by logical expressions (e.g. $u \geq 4 \wedge u \leq 5$) defining a set of values or an explicit enumeration of values (e.g. $u \in \{4,5\}$) for a specific belief state variable (e.g. *u*).

Example 2. The set $bs_1 = \{ (u,[4,5]), (v,\{T\}), (w,\{T\}), (y,\{F\}), (z,[5,6]) \}$ represents a set of belief state tuples. The restriction of *u* is an interval of the double data type, whereas four and five are part of the interval. The domains of the belief state variables might be predefined as *dom(u)=dom(z)=double, dom(v)=dom(w)=dom(y)=boolean.*

With the definition of a belief state tuple we have one cornerstone of the planning model, presented in the next section. By the use of these belief state tuples we can explicitly represent belief states and explain in what sense we understand them as possibly infinite sets of world states by the help of definition 4 in combination with definition 3.

Definition 3 ($\sqsubseteq$). Let $A = \{u_1, \ldots, u_k\}$ and $B = \{w_1, \ldots, w_m\}$ be two finite sets of belief state tuples, then:

$$A \sqsubseteq B :\Leftrightarrow \forall u \in A\ \exists w \in B: v(u)=v(w) \wedge r(u) \subseteq r(w) \wedge |r(u)|=1.$$

Definition 4 (*Belief state and world state*). Let $BST = \{p_1, \ldots, p_n\}$ be a finite set of *belief state tuples*. A *belief state bs* is a subset of *BST*, containing every belief state variable one time at the most. A world state *s* is a member of the belief state *bs*, in the context that it is $s \sqsubseteq bs$.

The set *bs* is constituted by the restrictions that currently apply to a set of belief state variables. Similar to Petrick and Bacchus (2002), the set *bs* thus could be interpreted as a kind of knowledge base capturing the knowledge about available belief state variables. The set *bs* therefore describes different conceivable world states that may occur during plan execution. It needs to be distinguished from the world state, which generally refers to an individual situation at process execution time. According to literature (e.g. Bertoli et al. 2006; Bonet and Geffner 2000; Hoffmann and Brafman 2005; Hoffmann and Brafman 2006), a set of world states is called belief state. Since the set *bs* is a set of world states in the context of definition 4, we will follow this wording and refer to the set *bs* as a *belief state*.

This way of representing a set of world states is one starting point to define and solve our planning problem. Furthermore, we can explicitly represent belief states in a rather intuitive way. Additionally, with this abstract representation language (R2), a belief state can be a possibly infinite set of world states (R1). Hence, a world state *s* is an instance of a belief state *bs*. In the following section this allows us to describe a concrete transition function, which is needed for the planning model.

Planning Model

As stated above, our approach can be seen as a nondeterministic planning problem with initial state uncertainty. Our approach is inspired by the framework given in Bertoli et al. (2006), but we will describe our domain in the abstract representation language specified before. Furthermore, we will handle nondeterminism in a different way. This is necessary because former approaches include observations in the domain description. In the context of process modeling these observations are not known in advance. To solve this problem, our idea is to automatically create sets of conditions, where a plan can branch. This is based on the presented requirements for the automated planning of the exclusive choice control structure (R3), which leads to conditional plans.

In this section we describe our search process (related to the guidelines for conducting design science research) and start with a nondeterministic state-transition system and its definition. Then we modify it by using the introduced

abstract representation language of the previous section. Instead of states we use belief states in the transition system, which makes it possible to change the transition function in a first step to be deterministic regarding belief states. As a result of that change, we define a *deterministic belief-state-transition system*, making it possible to cope with (R1) and (R2). In a second step we extend the transition function by what we call *conditions*, in order to build branches in a plan. As a consequence, we describe a *conditional deterministic belief-state-transition system* to handle our planning problem at the end of this section.

Nondeterministic State-Transition System – The Starting Point

When being confronted with a nondeterministic planning problem, it is common to use a nondeterministic planning domain. In general, "a nondeterministic state-transition system is defined in terms of its states, its actions, and of a transition function that describes how (the execution of) an action leads from one state to possibly many states" (Bertoli et al. 2006). We use this as a working definition of a nondeterministic state-transition system. More formally a state-transition system and (non-)determinism in state space are defined in Bertoli et al. (2006) as follows.

Definition 5 (*Nondeterministic state-transition system*). "A nondeterministic state-transition system is a tuple $\Sigma = (S, A, R)$, where

- S is a finite set of *states*,
- A is a finite set of *actions*,
- and $R : S \times A \rightarrow 2^S$ is the transition function. The transition function associates to each state $s \in S$ and to each action $a \in A$ the set $R(s, a) \subseteq S$ of next states."

Definition 6 ((*Non-)determinism in state space*). "An action a is *applicable* in a state s ([…]) iff $|R(s, a)| > 0$; it is *deterministic* (*nondeterministic*) in s iff $|R(s, a)| = 1$ ($|R(s, a)| > 1$). If a is applicable in s, then $R(s, a)$ is the set of states that can be reached from s by performing a."

As mentioned, we take this nondeterministic state-transition system as a starting point, and simply define it in a different way by using the introduced abstract representation language of the previous section.

Deterministic Belief-State-Transition System – The First Step

So far, we defined a nondeterministic state-transition system and what we understand as a belief state. With this, we define a deterministic belief-state-transition system and (non-)determinism in belief space. Similar to the working definition of the nondeterministic state-transition system, we formulate a working definition of a so called *deterministic belief-state-transition system* which is defined in terms of its belief states (sets of states), its actions, and of a transition function that describes how (the execution of) an action leads from one belief state to one and only one belief state. More formally:

Definition 7 *(Deterministic belief-state-transition system)*. Let $BST = \{p_1, \ldots, p_n\}$ be a finite set of belief state tuples. A deterministic belief-state-transition system is a tuple $\Sigma_d = (BS, A, \gamma_d)$, such that:

- $BS \subseteq 2^{BST}$ is a finite set of belief states, i.e., each belief state $bs \in BS$ is a subset of BST.
- A is a finite set of actions. Each action $a \in A$ is a triple consisting of the action name and two subsets of BST, which we will write as $a:=(name(a), precond(a), effects(a))$. The **set** $precond(a) \subseteq BST$ are the preconditions of a and the **set** $effects(a) \subseteq BST$ are the effects of a.
- An action a is *applicable* in a belief state bs (denoted with *applicable(a, bs)*) iff $bs \sqsubseteq precond(a)$ ($\sqsubseteq$ as defined in definition 9). This phrases a sufficient condition that needs to be met so that an action a can actually be performed in a belief state bs, since a can be performed in all the possible world states of bs. All belief state variables in *precond(a)* are available in bs. At the same time, the restriction of each belief state variable in bs is a subset of the restriction required by a belief state variable in *precond(a)*. In other words, an action is applicable iff the action can be performed in each world state $s \sqsubseteq bs$.
- The transition function is $\gamma_d : BS \times A \rightarrow 2^{BS}$ with $\gamma_d(bs,a) = \{ (bs \setminus \{ (v_{bs}, r_{bs}) \in bs \mid v_{bs} = v_{effects}, (v_{effects}, r_{effects}) \in effects(a) \}) \cup effects(a) \}$ if $a \in A$ is applicable in $bs \in BS$, and undefined otherwise.
- 2^{BS} is closed under γ_d, i.e., if $bs \in BS$, then for every action a that is applicable in bs, $\gamma_d(bs, a) \in 2^{BS}$.

Definition 8 ((*Non-)determinism in belief space*). An action a is *deterministic* (*nondeterministic*) in bs iff $|\gamma_d(bs, a)| = 1$ ($|\gamma_d(bs, a)| > 1$). If a is applicable in bs, then $\gamma_d(bs, a)$ is the set of belief states that can be reached from bs by performing a.

Definition 9 ($\sqsubseteq$). Let $A = \{u_1, \ldots, u_k\}$ and $B = \{w_1, \ldots, w_m\}$ be two finite sets of belief state tuples, then:

$$A \sqsubseteq B :\Leftrightarrow \forall w \in B\ \exists u \in A: v(w)=v(u) \wedge r(u) \subseteq r(w).$$

Example 10. Let $(a_1,\ precond(a_1):=\{ (u,[3,6]), (v, \{T,F\}), (w,\{T\}), (y,\{F\}) \}, effects(a_1):= \{ (u,[2,7]), (v,\{F\}), (x,[1,4]) \})$ be an action and let $bs_1 = \{ (u,[4,5]), (v,\{T\}), (w,\{T\}), (y,\{F\}), (z,[5,6]) \}$ be a belief state. Here a_1 is applicable in the belief state bs_1, since $bs_1 \sqsubseteq precond(a_1)$, because for each belief state variable among the preconditions of a_1 there is a belief state variable in bs_1 and it is $[4,5]_{u\_bs1} \subseteq [3,6]_{u\_a1}$, $\{T\}_{v\_bs1} \subseteq \{T,F\}_{v\_a1}$, $\{T\}_{w\_bs1} \subseteq \{T\}_{w\_a1}$ and $\{F\}_{y\_bs1} \subseteq \{F\}_{y\_a1}$[5]. As a result it is $\gamma_d(bs_1,a_1)=\{ \{ (u,[2,7]), (v,\{F\}), (w,\{T\}), (x,[1,4]), (y,\{F\}), (z,[5,6]) \} \}$.

We compare definition 5 and definition 7 to show, that definition 7 just extends definition 5, but is basically another way to define a nondeterministic state-transition system. Our transition system is called a belief-state-transition system, since it is not based on states, but on belief states. A nondeterministic state-transition system can be written as a deterministic belief-state-transition system, because:

- Both transition systems have a finite set of sets of world states, $|S|<\infty$ and $|BS|<\infty$. The nondeterministic state-transition system has a finite set of world states S. A single world state $s \in S$ can be understood as a set of world states, having only one element. Due to that, the nondeterministic state-transition system has a finite set of sets of world states. The deterministic belief-state-transition system is based on a finite set of belief states BS, and therefore has a finite set of sets of world states, too.
- Both transition systems have a finite set of actions.
- Both transition systems have a transition function that associates to each set of world states and to each action, a set of next world states.

According to definition 8, the belief-state-transition system is called deterministic, since it is $|\gamma_d(bs, a)| = 1$. This deterministic belief-state-transition system allows transitions from a set of world states, with more than one element, to a set of next world states, which is made possible through the use of belief state tuples. Here, the nondeterministic state-transition system is extended, since it only allows transitions from one world state to a set of next world states. Due to the definition of the belief states tuples, another extension is that a belief state is a possibly infinite set of world states. These two extensions tackle the problem of representing and updating belief states in large state spaces. With this novel model definition, it is now – in contrast to former works – possible to cope with (R1) and (R2).

As the next example demonstrates, the transition function γ_d might leave out transitions that are possible in the process modeling context, due to the fact that a transition takes place only when an action is applicable in a belief state.

Example 11. Let $(a_1,\ \{ (u,[3,6]), (v,\{T, F\}), (w,\{T\}), (y,\{F\}) \},\ \{ (u,[2,7]), (v,\{F\}), (x,[1,4]) \})$ be an action and let $bs_2 = \{ (u,[1,5]), (v,\{T\}), (w,\{T\}), (y,\{T,F\}), (z,[5,6]) \}$ be a belief state. Here a_1 is not applicable in the belief state bs_2, since $bs_2 \not\sqsubseteq precond(a_1)$, because it is $[1,5]_{u\_bs2} \not\subseteq [3,6]_{u\_a1}$ and $\{T,F\}_{y\_bs2} \not\subseteq \{F\}_{y\_a1}$. The transition function $\gamma_d(bs_2,a_1)$ would be considered as not defined, although it would be defined if, for example, $(u,\{4\})$ and $(y,\{T\})$ hold in an individual process execution (a certain world state of bs_2). Thus, it is necessary to consider branches with conditions in the constructed process model (see (R3)).

Conditions – Enabling the Second Step

As a result of example 11, we realize the need to generalize the transition function to allow the performing of what we call *partly applicable* actions in a belief state. In a next step, we therefore extend the transition function γ_d of definition 7 by so called *conditions*, which are comparable to the routing constraints in Sun et al. (2006).

Definition 12 *(Partly applicable)*. An action a is *partly applicable* in a belief state bs (denoted with *partly_applicable(a, bs)*) iff:

[5] The indices are just given to note, to which belief state variable the restriction belongs to, for example $[4,5]_{u\_bs1}$ indicates that the interval of numbers $[4,5]$ is the restriction of u in bs_1.

$$\forall u \in precond(a)\ \exists w \in bs: v(u)=v(w)\ \wedge\ (r(u) \cap r(w) \neq \emptyset).$$

Definition 12 describes a necessary condition that needs to be met so that an action *a* can actually be performed in a belief state *bs.* All belief state variables in *precond(a)* are available in *bs* and the restriction of each belief state variable (i.e. the set of possible values) does not contradict the restriction required by *a* for that belief state variable. However, there may still be situations (certain world states of *bs*) where performing *a* is not possible due to the restrictions. If we allow partly applicable actions, our transition function needs to be able to handle actions that are partly applicable in a belief state.

Example 13. Let $(a_1,\ \{ (u,[3,6]), (v,\{T,F\}), (w,\{T\}), (y,\{F\}) \},\ \{ (u,[2,7]), (v,\{F\}), (x,[1,4]) \})$ be an action and let $bs_2 = \{ (u,[1,5]), (v,\{T\}), (w,\{T\}), (y,\{T,F\}), (z,[5,6]) \}$ be a belief state. Here a_1 is partly applicable in the belief state bs_2, because it is $[1,5]_{u\_bs2} \cap [3,6]_{u\_a1} = [3,5] \neq \emptyset$, $\{T\}_{v\_bs2} \cap \{T,F\}_{v\_a1} = \{T\} \neq \emptyset$, $\{T\}_{w\_bs2} \cap \{T\}_{w\_a1} = \{T\} \neq \emptyset$ and $\{T,F\}_{y\_bs2} \cap \{F\}_{y\_a1} = \{F\} \neq \emptyset$.

If we have a belief state *bs*, then there might exist a nonempty set of actions A_{p_a} with actions that are partly applicable in *bs*. In an individual process execution, for every belief state variable in *bs* a specific value can be observed for a certain world state $s \sqsubseteq bs$. In this world state *s*, it might be possible to perform all actions in A_{p_a} or just the actions in a subset $A_{perform} \subseteq A_{p\_a}$. This means, for every belief state variable of a belief state we have to discover for which observations (constellation of parameter values) an action is in $A_{perform}$. In other words, we need to detect, when it is possible to perform an action and consider it in a process model and when this is not possible. Therefore, we need to find a *set of conditions* under which an action can always be performed, that is, under which set of conditions an action is applicable in *bs* (R3). With these definitions, it is possible to plan exclusive choices, since we now know the conditions and the corresponding actions to construct branches in a process model. Former works (e.g. Bertoli et al. 2006) do not consider sets of conditions (sets of sets of observations). This makes our notation more expressive.

Definition 14 (*Condition*). A *condition q* is a tuple of a *condition variable v(q)* and a subset $r(q) \neq \emptyset$ of its predefined domain *dom(q)*, which we will write as $q := (v(q), r(q))$. It is $v(q) \in r(q)$ in a certain world state. When talking about belief states, then *v(q)* is the symbol of the condition variable. The set $r(q) \subseteq dom(q)$ is called the *condition restriction* of *v(q)*, which is the set of values that might be assigned to *v(q)* in a possible world state.

The restriction *r(q)* of a condition *q* is a set of possible values that might be observed in a world state for one condition variable *v(q)*. A set of conditions *c* is built for every action that is partly applicable in a belief state *bs*, which are the actions in A_{p_a}. This set of conditions *c* might be different for every action in A_{p_a}, and it is – in contrast to other planning problems – not provided prior to planning a process model. It needs to be determined automatically (R3). In a certain world state, we can then perform these actions, where all conditions are *fulfilled*, i.e., for every condition in *c* there is an observed value in the world state, and the observed value of the condition variable is a member of the restriction of the condition for this condition variable. This way it is known in each state, which actions can be performed, or as we call it, can be *executed*. An action can be *executed* in a certain world state of a belief state, either if the action is applicable in the belief state ($c=\emptyset$) or if the action is both partly applicable in the belief state and all of its conditions are fulfilled.

Example 15. Let $(a_1,\ \{ (u,[3,6]), (v,\{T,F\}), (w,\{T\}), (y,\{F\}) \},\ \{ (u,[2,7]), (v,\{F\}), (x,[1,4]) \})$ be an action and let $bs_2 = \{ (u,[1,5]), (v,\{T\}), (w,\{T\}), (y,\{T,F\}), (z,[5,6]) \}$ be a belief state. As shown in example 13, a_1 is partly applicable in bs_2. Since a_1 is partly applicable, we need to find the conditions that have to be fulfilled so that the action can be executed. We can do that by looking for the reason, why a_1 is not applicable in bs_2. It is not applicable, because it is $[1,5]_{u\_bs2} \not\subseteq [3,6]_{u\_a1}$ and $\{T,F\}_{y\_bs2} \not\subseteq \{F\}_{y\_a1}$, as presented in example 11. If we could restrict these restrictions even more, then a_1 would be applicable in bs_2. This is exactly what we do with the set of conditions. If $(u,[3,5])$ and $(y,\{F\})$ holds then a_1 would be applicable in bs_2, which makes $c_1 = \{ (u,[3,5]), (y,\{F\}) \}$ the set of conditions that need to be fulfilled to execute a_1 in a certain world state of bs_2.

As mentioned, the set of conditions might be different for every action that is partly applicable. In order to assign a set of conditions *c* to a belief state *bs* and an action *a* we define a *condition function*. This function associates to each belief state *bs* and each action *a* a set of possible conditions.

Definition 16 *(Condition function)***.** Let $\Sigma_d = (BS, A, \gamma_d)$ be a deterministic belief-state-transition system. Let *C* be a possibly infinite set of conditions. A condition function over *BS* and *C* is a function $\theta : BS \times A \rightarrow 2^C$ (denoted with CONDITIONFUNCTION(*bs*, *a*)), which associates to each belief state *bs* and each action *a* the set of possible conditions $\theta(bs, a) \subseteq C$.

Therefore, not only an action influences the transition from one belief state to another one, but also the conditions under which this action can be executed. We remark that in practice the conditions are not given additionally to the domain in any way, but need to be created by an algorithm (R3).

Planning Domain and Planning Problem – The Second Step

The previous discussions lead to a *conditional deterministic belief-state-transition system*, which we consider to be our *planning domain*.

Definition 17 *(Planning domain)*. Let $BST = \{p_1, \ldots, p_n\}$ be a finite set of belief state tuples. Our *planning domain* on *BST* is a conditional deterministic belief-state-transition system $\Sigma_{cd} = (BS, A, \theta, \gamma_{cd})$, such that:

- $BS \subseteq 2^{BST}$ is a finite set of belief states.
- A is a finite set of actions.
- $\theta : BS \times A \rightarrow 2^C$ is a condition function over BS and A, with the set of conditions $C = \bigcup_{i=1}^{n} \left\{ (v(p_i), r) \middle| \, r \in 2^{r(p_i)} \setminus \varnothing \right\}$.
- The transition function is $\gamma_{cd} : BS \times 2^C \times A \rightarrow 2^{BS}$ with $\gamma_{cd}(bs,c,a) = \{ ((((bs \setminus \{ (v_{bs},r_{bs}) \in bs \mid v_{bs} = v_c, (v_c,r_c) \in c \}) \cup c) \setminus \{(v_c,r_c) \in c \mid v_c = v_{effects}, (v_{effects},r_{effects}) \in effects(a)\}) \setminus \{(v_{bs},r_{bs}) \in bs \mid v_{bs} = v_{effects}, (v_{effects},r_{effects}) \in effects(a) \}) \cup effects(a) \}$ if $a \in A$ is partly applicable in $bs \in BS$ and $c \in 2^C$ is a set of conditions for a, and undefined otherwise.
- 2^{BS} is closed under γ_{cd}, i.e., if $bs \in BS$, then for every action a that is partly applicable in bs, and for every set of conditions $c \in 2^C$ that need to be considered, $\gamma_{cd}(bs,c,a) \in 2^{BS}$.

In contrast to former approaches like Bertoli et al. (2006), our conditions are not part of the domain, since this is not realistic at all in the context of process modeling. Now, they can be automatically derived from the domain, through the condition function and the set C, satisfying (R3).

Example 18. Let $(a_1, \{ (u,[3,6]), (v,\{T,F\}), (w,\{T\}), (y,\{F\}) \}, \{ (u,[2,7]), (v,\{F\}), (x,[1,4]) \})$ be an action and let $bs_2 = \{ (u,[1,5]), (v,\{T\}), (w,\{T\}), (y,\{T,F\}), (z,[5,6]) \}$ be a belief state. Here a_1 is partly applicable in bs_2, because it is $[1,5]_{u\_bs2} \cap [3,6]_{u\_a1} = [3,5] \neq \varnothing$, $\{T\}_{v\_bs2} \cap \{T,F\}_{v\_a1} = \{T\} \neq \varnothing$, $\{T\}_{w\_bs2} \cap \{T\}_{w\_a1} = \{T\} \neq \varnothing$ and $\{T,F\}_{y\_bs2} \cap \{F\}_{y\_a1} = \{F\} \neq \varnothing$. Let $c_1 = \{ (u,[3,5]), (y,\{F\}) \}$ be the set of conditions, that need to be fulfilled to execute a_1 in bs_2, as discovered in example 15. As a result we have $\gamma_{cd}(bs_2,c_1,a_1) = \{ \{ (u,[2,7]), (v,\{F\}), (w,\{T\}), (x,[1,4]), (y,\{F\}), (z,[5,6]) \} \}$.

In practice, for each belief state all partly applicable actions are determined, then the result of the condition function is calculated for each action, and at the end a new belief state is created as described by γ_{cd}. In summary, our planning problem is defined as follows.

Definition 19 *(Planning problem)*. Our *planning problem* is a triple $P = (\Sigma_{cd}, bs, BS_g)$, where:

- $\Sigma_{cd} = (BS, A, \theta, \gamma_{cd})$ is a planning domain.
- $bs \neq \varnothing$, the belief state prior to the exclusive choice, is a member of BS.
- $A_{p\_a} \subseteq A$ is a set of all actions that are partly applicable in bs.
- $BS_g \subseteq 2^{BS}$ is a set of belief states called *goal belief states* that are required to exist after the exclusive choice. The set of goal belief states is:

$$BS_g = \bigcup_{i=1}^{m} \left\{ \gamma_{cd}(bs, \theta(bs, a_i), a_i) \in 2^{BS} \middle| \, a_i \in A_{p\_a} \right\}, m = \left| A_{p\_a} \right|.$$

The planning problem states, that given the planning domain and the belief state bs, each goal belief state must be constructed in order to solve the problem.

Algorithm

The focus of this section is on the algorithm (method) that constructs the conditions and the branches, which is the realization of the condition function. Therefore, we use another algorithm as a starting point (like in Bertoli et al. 2006) that progresses from an initial belief state to a goal belief state. It builds a search tree, to find all the possible

paths starting in the initial belief state. We enhance such an existing approach with our algorithm and its ability to identify the required conditions. The resulting search tree *ST* is a graph of a set of nodes *Nodes(ST)*, which are the belief states, and a set of labeled arcs *Arcs(ST)*. We label the arcs with both, an action and the corresponding conditions that need to be fulfilled to execute this action in the belief state.

Our focus is on how the condition function θ: $BS \times A \rightarrow 2^C$ can be realized, in other words, how the set of conditions can be built. First, we describe how the EXTENDTREE primitive, presented in Bertoli et al. (2006), can be modified to include the condition function (Figure 3 - appendix). Second, we introduce the CONDITIONFUNCTION primitive, see Figure 4, which builds the set of conditions, thus being a realization of our condition function θ. Finally, we present the PARTITION subroutine in Figure 5. This recursive subroutine creates disjoint partitions of the restrictions of certain belief state tuples in a belief state, which are needed to build the set of conditions.

The EXTENDTREE primitive receives the current search tree *ST* and a node, which is a belief state, where the tree can be extended. For every partly applicable action, including also the applicable actions, a set of conditions and the resulting next node are built. This next node is added to the set of nodes *Nodes(ST)* of the current search tree. A new arc is constructed which includes the action and the set of conditions as labels, and then added to the set of arcs *Arcs(ST)*. We go into detail on line 10 of the EXTENDTREE primitive at the end of this section.

An arc has two labels, an action and the set of conditions that need to be fulfilled in order to execute the action in a node *bs* to reach *bs'*. If an action is partly applicable, then the CONDITIONFUNCTION primitive, see Figure 4, needs to be executed. The primitive gets a node *bs* and an action *a*. If *a* is applicable in *bs* then, there is no need for conditions and the empty set is returned. On the other hand, if *a* is just partly applicable (and not applicable), then a set of conditions c_a is created for *a*. The lines 6-10 create A_{p_a}, the set of all partly applicable actions in *bs*. The set bs_p is the set of all belief state tuples in *bs*, where the belief state variable is also in the preconditions of *a* (line 11). The rest of the primitive is then performed for every belief state tuple in bs_p.

We take one element (v_{bs},r_{bs}) of bs_p. The set *Partition* of nonempty sets is a partition of r_{bs}. The elements of *Partition* are pairwise disjoint and the elements of *Partition* cover r_{bs}. The Partition subroutine creates the set *Partition*. In line 15, the set c_{part} is built, which contains those elements of *Partition* that are a subset of r_w, being the restriction of a belief state tuple v_{bs} in the preconditions of *a*. As a next step, the elements of c_{part} are joined to form c_p. This set c_p is the condition for v_{bs} that, possibly among others, needs to be fulfilled in order to execute *a*. In the end, c_p is joined with c_a to construct the set of conditions, which is returned, after lines 13-21 are carried out for every element of bs_p. The CONDITIONFUNCTION primitive creates the minimal amount of conditions, with the minimal sets of observations (due to lack of space, we omit the proof). This leads to a minimal quantity of exclusive choices in the process model, which is advantageous for its presentation and layout.

In line 14 of the CONDITIONFUNCTION primitive the PARTITION subroutine is carried out, which is defined recursively. This subroutine starts with two sets. The set *r* is a restriction r_{bs} of a belief state tuple (v_{bs},r_{bs}). The second set *R* is a collection of restrictions. It is the set of all restrictions for v_{bs} that are part of the preconditions of each action in A_{p_a}. This is done in order to partition r_{bs} in pairwise disjoint sets so that the set c_p in the CONDITIONFUNCTION primitive can be constructed. It is possible that there are subsets of r_{bs}, which are not covered by any precondition of the partly applicable actions. That is to say, there are subsets of r_{bs}, where the intersection of these subsets and the union of the sets in *R* are empty. To handle this case, we always add an arc in the EXTENDTREE primitive (line 10) that leads to the termination of the plan if the values in the union of these subsets are observed in the world state at execution time.

As mentioned, the focus of this section is on the algorithm that plans the conditions and the branches. The conditions are not given additionally to the domain in any way, but are created by the algorithm, as required by (R3).

Evaluation

The presented algorithm was implemented prototypically as part of the open source process modeling tool AgilPro. The algorithm and the prototypical implementation (instantiation) were evaluated as shown in this section.

a) Analysis of the algorithm properties: We mathematically evaluated the algorithm in terms of completeness, minimality, termination and computational complexity regarding time. It could be shown that the approach creates complete results and a minimal number of exclusive choices in a process model for a given problem. Furthermore, the algorithm terminates. Considering the computational complexity regarding time, it can be shown that, for

example, the runtime increases subproportionaly when the initial belief state or the goal belief states are extended. Due to lack of space we only show that the algorithm terminates:

Theorem 20. Given our planning domain, the execution of the algorithm EXTENDTREE terminates.

Proof. Termination is proved by showing that each iteration of every for-loop in the algorithm terminates, and that the number of iterations is finite. If *a* is applicable in *bs*, then the CONDITIONFUNCTION terminates (lines 2-3). The else-case (lines 4-23) is complex.

Before starting with the CONDITIONFUNCTION, we show that the set *Partition* in line 14 is finite. The set *Partition* is built by the subroutine PARTITION, presented in Figure 5. The set R of the PARTITION subroutine equals to the set $\{r_u \mid v_{bs}=v_u,\ (v_u,r_u) \in precond(a_{p\_a}),\ a_{p\_a} \in A_{p\_a}\}$ when it is invoked the first time. The set $precond(a_{p\_a})$ is finite for each $a_{p\_a} \in A_{p\_a}$, due to the fact that it is a subset of *BST*. The set of all actions A is finite, and so is A_{p_a}, because it is a subset of A. As $precond(a_{p\_a})$ and A_{p_a} are finite, the set R is finite, when PARTITION is invoked the first time. Every line of the PARTITION subroutine, besides 8 and 11, terminates, simply because there are just set operations. The subroutine is invoked recursively only a finite number of times. This is due to the following three facts:

- R is finite.
- The number of elements in R decreases by one every time the subroutine is invoked recursively.
- The subroutine is invoked only when there is more than one element in R, representing a lower bound.

In other words, the subroutine is invoked recursively just a finite number of times, because the set R is finite, the number of elements is strictly decreasing and there is a lower bound. Thus, the PARTITION subroutine terminates and the set *solution* is finite. The set *solution* is returned to the CONDITIONFUNCTION, constituting the finite set *Partition*.

With the set *Partition* being finite, it is possible to show that the CONDITIONFUNCTION terminates. Lines 7-9 terminate, because the verification whether a_p is partly applicable in *bs* or not terminates and the procedure of uniting A_{p_a} with a_p terminates. The for-loop of lines 6-10 terminates, since A is finite. We show that lines 14-20 terminate. The set c_{part} is a subset of *Partition*, which makes c_{part} finite. Line 18 terminates, because the procedure of uniting *part* with c_p terminates. Since c_{part} is finite, lines 17-19 terminate. The operation of uniting c_a with c_p terminates. To show, that the number of iterations of lines 14-20 is finite, we show that bs_p is a finite set of belief state tuples. The set *BST* of all belief state tuples is finite. The node *bs* is a subset of *BST*, which makes it finite as well. The set bs_p is a subset of *bs*, and is therefore finite. Thus, the for-loop of lines 13-21 terminates, and so does the CONDITIONFUNCTION. Since the set A is finite the EXTENDTREE primitive terminates. *q.e.d.*

b) Analysis of the implementation: Besides the manual analysis of the source code (structured walk through) by persons other than the programmers, we made a series of tests using the JUnit Framework, including runs with extreme values, JUnit regression tests and unit tests. The implemented algorithms did not show any defects at the end of the test phase.

c) *Formal evaluation of the results:* It can be shown that the constructed exclusive choice is syntactically correct. As stated in Sadiq and Orlowska (2000), an exclusive choice needs to be "exclusive and complete", which means that in each process instance exactly one of the alternative partly applicable actions is executed or the process ends. This is guaranteed due to the facts that the conditions of the partly applicable actions are pairwise disjoint and that these conditions cover the whole domains of the respective belief states. Further forms of evaluation, as for example the proof that no data-flow anomalies exist (Sun et al. 2006) and that the resulting process model fulfills the soundness property as proposed by van der Aalst (2000), are not applicable for an exclusive choice itself (we conducted such evaluations in other papers considering the overall planned process model).

d) Defined Requirements: We presented an abstract representation language in order to explicitly represent a possibly infinite set of world states in the form of belief states, providing an intuitive formalism (from a process modeling perspective) for the planning problem. With this language, we addressed the requirements (R1) and (R2). In our approach, belief states are defined as sets of feasible values of belief state variables and thus implicitly describe sets of conceivable world states. On this basis we constructed a conditional deterministic belief-state-transition system, what we considered to be our planning model. In this planning model we had a concrete representation of our transition function, which also accounts for sets of conditions (cp. (R3)).

e) Operational evaluation of the results: Hevner et al. (2004) stressed that an artifact must be evaluated with respect to the practical utility provided. Since competing artifacts do not exist in our case, a comparison related to efficiency is not possible at all. This in mind, we analyzed the practical applicability in different real-use situations

("proof of construction"). One of those situations – which we consider in the following – is an example taken from the security-order-management of a European financial services provider. Here, processes had to be redesigned in the past due to new products, new regulations or changing organizational requirements (like to outsource parts of a process to external service providers). These processes have to be not only (re)designed repeatedly but are also repetitive processes which are of high value for the firm. We analyzed such previous redesigns and studied two aspects in detail. Firstly, would it be possible at all to apply our algorithm in these situations and to which extent match the results of the automated planning with manually built exclusive choice patterns? And secondly, does the application of the algorithm make sense regarding economical aspects, i.e. do we benefit from the automated planning of process models compared to a manual design and what costs do result from the artifact's application?

Considering the first aspect, it can be said, that the algorithm constructed not only the exclusive choices resulting from the manual redesign, but also additional feasible solutions. For instance, the resulting process was planned for the execution of security-orders where several steps including check routines had to be modeled. For brevity, we only present a small part of the whole process, where the security-order data is entered, the order itself is already validated and we now need to decide which check routine should be used (see Figure 2). The resulting belief state *bs* contains, among others, the tupel *orderAmount* that reaches from *0* to *250,000* Euro and the tuple *orderState* with the values *valid* and *invalid*. We now illustrate the input of the PARTITION subroutine, the result of that subroutine, and how this result is used in the CONDITIONFUNCTION primitive.

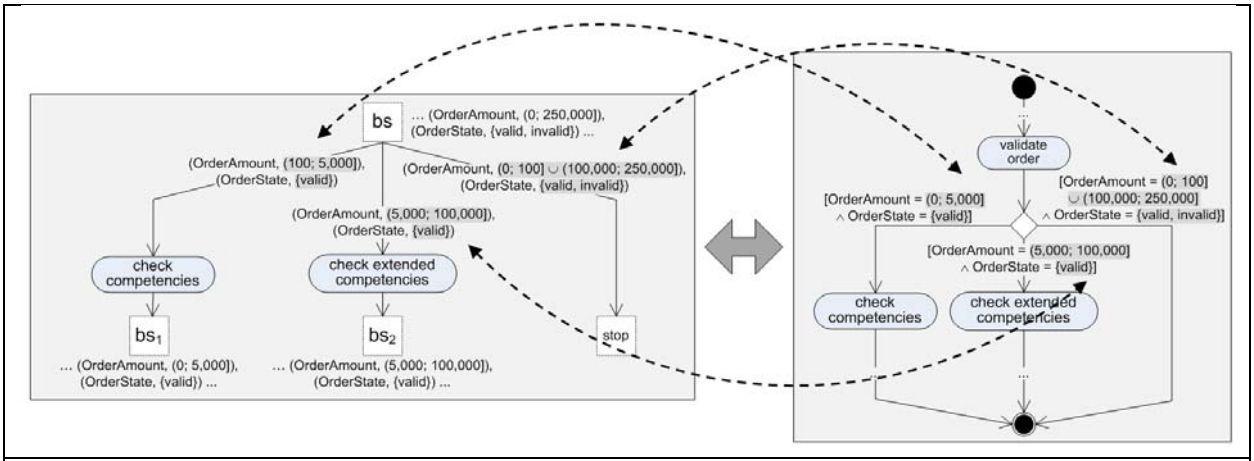

Figure 2. Transformation of a Search Tree ST (Left Side) to an UML Activity Diagram

The partly applicable actions in *bs* are *checkCompetencies* with *(orderAmount, (100; 5,000])*, *(orderState, {valid})* and *checkExtendedCompetencies* with *(orderAmount, (5,000; 100,000])*, *(orderState, {valid})*. For the creation of the respective conditions (only shown for the action *checkCompetencies*), the subroutine PARTITION gets $r = [0; 250{,}000]$ and $R = \{ (0; 5{,}000], (5{,}000; 100{,}000] \}$ in its first execution and provides $solution = \{ (100; 5{,}000], (5{,}000; 100{,}000], (0; 100] \cup (100{,}000; 250{,}000] \}$. The second invocation of PARTITION ($r = \{valid; invalid\}$ and $R = \{valid\}$) returns $solution = \{ \{valid\}; \{invalid\} \}$. With these partitions, the CONDITIONFUNCTION determines for each of the partly applicable actions, the necessary condition that is *(orderAmount, (100; 5,000])*, *(orderState, {valid})* for *checkCompetencies* and *(orderAmount, (5,000; 100,000])*, *(orderState, {valid})* for *checkExtendedCompetencies*. The third condition *(orderAmount, (0; 100] ∪ (100,000; 250,000])*, *(orderState, {valid; invalid})* that needs to be considered does not lead to an executable action but to the end of the process as there is no executable action for it.

The assessment showed the practical applicability of the algorithm in some real-use situations. However, what about the efficiency of such applications? Here, we have to differentiate: To conduct the algorithm, an initial annotation of actions according to their preconditions and effects is necessary. These one-time annotation costs are limited, if a firm already uses a process modeling tool featuring a XML interface. Such an interface can be used in order to export actions to AgilPro. The financial services provider we consider here used, for instance, the ARIS toolset, which allowed us to export a huge number of actions. In the field of the security-order-management about 200 different actions including their preconditions and effects were imported from the ARIS toolset and afterwards checked (semantically). This review comprised for example the check for completeness of the action's preconditions

and effects and on a semantic level if the same parameter names of the preconditions and effects were used for different concepts as well as if different parameter names were used for the same concepts. In some cases, preconditions and effects had to be completed or corrected. Such a review, which is to some extent also necessary during manual process design, ought to evidently be performed more accurately for automated process planning. Annotating ten actions with averagely five preconditions and five effects in about half an hour, the net time for the annotation of actions in our AgilPro tool is comparatively small. For this purpose, the relevant actions have to be defined and labeled before their preconditions and effects are defined and semantically annotated. There, each precondition and effect is specified by choosing the mapping concept from the ontology and determining the appropriate restriction. All in all, the initial costs of automated planning were about 20% higher than the previous costs of manual design.

However, the resulting annotation costs need to be put into perspective, if a redesign of the processes considered is necessary more than just one time (see also the specified problem context in the second section). In future redesigns of the processes the *initial* costs are lower, because the annotations as well as the deployed algorithm can be reused. This even holds for the common case, where annotated actions (e.g. an action "*scan document*") can be reused in further processes (like for example in the loan department). Therefore, it can be seen, that especially under the condition of a higher frequency of process redesigns, an automated planning of processes models leads in sum to slightly higher costs compared to a manual redesign.

Furthermore, in our above mentioned real-use situation considering the security-order-management, a set of feasible process models was generated within one day by means of AgilPro. This is an obvious advantage compared to a manual design, which took more than one week. Also, the planner generated not only the manual designed process model, but other feasible solutions as well. Some of these process solutions need, for instance, less staff capacity of the financial services provider than the manual designed process. This leads to lower process costs of about 2.5% in average for each process run (calculation basis: process *'execution of security-orders'*). Since the security-order-management process is a highly repetitive process, the higher initial costs of automatic planning can be amortized in our case within half a year. And, according to our projects with firms, these circumstances are not unique in practice. But, in any other case, an individual analysis is really necessary to assess if an automated planning is useful. In addition, we have to evaluate another point regarding economical aspects: Several actions – as described above – are not used within only one process, but are reused in other processes, too. Thus, if an action is used several times, its annotation costs can be allocated to all redesigns of those processes using the considered action. However, such economical analysis cannot be done for only a few real-use situations but in medium- or long-term studies. Therefore, the generalizability of our evaluation of the practical utility is limited. Nonetheless, at this time, this limitation is unavoidable since such an iterative study is time-intensive, mitigating the possibility of conducting multiple real-use situations simultaneously. We anticipate that these new cases will support the relevance identified in this paper.

Conclusion

In this paper, we described how control structures can be planned automatically within the research strand of Semantic Business Process Management. Related to the guidelines for conducting design science research by Hevner et al. (2004) we can summarize as follows: Our key *artifact* is a method in terms of an algorithm for planning exclusive choices within a process model. We regard this as an important step to automate and hence to support the task of designing a process model. Both, the algorithm and our planning problem are formally noted and can therefore be well-defined and mathematically evaluated. Based on statements in literature (e.g. Borges et al. 2005; Ma and Leymann 2008) and on our own project expectations that manual process modeling is cost-intensive and very time-consuming, we describe our *problem context*. Here, our artifact is thought to contribute to process modeling to design and adapt process models faster and to be useful regarding economical aspects. Since such a statement cannot hold for every process type, we concentrate on repetitive processes that need to be (re)designed repeatedly. Considering the real-use situations, in which we applied the algorithm, we found that this focus is reasonable. The *evaluation* was done on the one hand by mathematical methods, but not in comparison with competing artifacts, since the artifact solves a heretofore unsolved problem. On the other hand, we studied literature and derived key requirements that a planning algorithm should meet in our problem context. These requirements were not only the guideline when developing our artifact. Moreover the artifact was evaluated against the defined requirements. Additionally, we evaluated the algorithm in real-use situations with respect to its applicability and the practical utility provided. This is appropriate since it analyzes the planning algorithm “in depth in business” (Hevner et al. 2004). Nevertheless, considering such economical analysis, future work is needed and intended to support the

assessment and justification of an automated planning. The paper points out, that existing algorithms have some serious disadvantages, especially regarding the requirements derived from literature. So, they are not appropriate to our context. The *research contribution* of our approach is to avoid these problems and meet the defined requirements. Therefore, our paper fills a gap in science and practice. To support a *rigorous* definition of our artifact, we represented it formally based on an also formally denoted planning domain. Such a technical representation assists a mathematical evaluation of the algorithm, too. The *search process* is on the one hand directed by the requirements. On the other hand, we describe this process beginning with the abstract representation language and its advantages. Furthermore, we show in detail which steps are necessary to develop our artifact (see section planning model). Regarding the *communication* of our results, we choose a more technical, rigorous presentation, because we want to convincingly demonstrate that our artifact can be realized and implemented. However, we also tried to attract a managerial audience by means of the illustrated business problem context as well as the economic aspects of automated planning process models. Further work is proposed on the question of how other control flow patterns, like arbitrary cycles, can be considered. For this, the designed algorithm is a reliable basis.

Appendix

```
1  procedure ExtendTree(bs, ST)
2    forall a ∈ A
3      if partly_applicable(a, bs) then
4        c := ConditionFunction(bs, a)
5        bs' := γcd(bs, c, a)
6                Nodes(ST) := Nodes(ST) ∪ {bs'}
7                Arcs(ST) := Arcs(ST) ∪ ⟨bs, a, c, bs'⟩
8      endif
9    endfor
10   Arcs(ST) := Arcs(ST) ∪ ⟨bs, else, stop⟩
11 end
```

Figure 3. Node Expansion Primitive

```
1  function ConditionFunction(bs, a)
2    if applicable(a, bs) then
3      return ∅
4    else
5      A_p_a := ∅
6      forall a_p ∈ A
7        if partly_applicable(a_p, bs) then
8          A_p_a := A_p_a ∪ a_p
9        endif
10     endfor
11     bs_p := {(v_bs,r_bs) ∈ bs | v_bs = v_pre, (v_pre,r_pre) ∈ precond(a)}
12     c_a := ∅
13     forall (v_bs,r_bs) ∈ bs_p
14       Partition := Partition(r_bs,{r_u|v_bs=v_u,(v_u,r_u)∈ precond(a_p_a),a_p_a∈ A_p_a})
15       c_part := {part ∈ Partition | part ⊆ r_w, v_bs = v_w,(v_w,r_w)∈ precond(a)}
16       c_p := ∅
17       forall part∈ c_part
18         c_p := c_p ∪ part
19       endfor
20       c_a := c_a ∪ c_p
21     endfor
22     return c_a
23   endif
24 end
```

Figure 4. Condition Primitive

```
1  function Partition(r, R)
2    nondeterministically choose rest ∈ R
3    diff := r \ rest
4    intersection := r ∩ rest
5    solution = ∅
6    if |R|>1 then
7      if diff ≠ ∅ then
8        solution := Partition(diff, R \ rest)
9      endif
10     if intersection ≠ ∅ then
11       solution := solution ∪ Partition(intersection, R \ rest)
12     endif
13   else
14     solution := {diff, intersection} \ {∅}
15   endif
16   return solution
17 end
```

Figure 5. Subroutine of the Condition Function

References

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

Bertoli, P., Cimatti, A., Roveri, M., and Traverso, P. 2006. "Strong planning under partial observability," *Artificial Intelligence* (170:4), April, pp. 337-384.

Betz, S., Klink, S., Koschmider, A., and Oberweis, A. 2006. "Automatic user support for business process modeling," in *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, K. Hinkelmann, D. Karagiannis, N. Stojanovic and G. Wagner (eds.), June 2006, pp. 1-12.

Biron, P. V., and Malhotra, A. 2004. "XML Schema Part 2: Datatypes Second Edition," 04/30/2009, (http://www.w3.org/TR/xmlschema-2/).

Bonet, B., and Geffner, H. 2000. "Planning with Incomplete Information as Heuristic Search in Belief Space," in *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*, S. Chien, S. Kambhampati and C. A. Knoblock (eds.), April 2000, pp. 52-61.

Bonet, B., and Geffner, H. 2001. "GPT: A Tool for Planning with Uncertainty and Partial Information," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, B. Nebel (ed), August 2001, pp. 82-87.

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

Brockmans, S., Ehrig, M., Koschmider, A., Oberweis, A., and Studer, R. 2006. "Semantic alignment of business processes," in *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006)*, Y. Manolopoulos, J. Filipe, P. Constantopoulos and J. Cordeiro (eds.), May 2006, pp. 191-196.

Constantinescu, I., Faltings, B., and Binder, W. 2004. "Large scale, type-compatible service composition," in *Proceedings of the IEEE International Conference on Web Services (ICWS 04)*, July 2004, pp. 506-513.

Geffner, H. 2002. "Perspectives on Artificial Intelligence Planning," in *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, July 2002, pp. 1013-1023.

Ghallab, M., Nau, D., and Traverso, P. 2004. *Automated Planning - Theory & Practice*, San Francisco: Morgan Kaufmann.

Gregor, S., and Jones, D. 2007. "The Anatomy of a Design Theory," *Journal of the Association of Information Systems* (8:5), May, pp. 312-335.

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

Henneberger, M., Heinrich, B., Bauer, B., and Lautenbacher, F. 2008. "Semantic-Based Planning of Process Models," in *Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI) 2008*, M. Bichler, T. Hess, H. Krcmar, U. Lechner, F. Matthes, A. Picot, B. Speitkamp and P. Wolf (eds.), February 2008, pp. 1677-1689.

Hepp, M., and Dumitri, R. 2007. "An ontology framework for semantic business process management," in *Proceedings of the 8th International Conference on Business Informatics (WI 2007): eOrganisation: Service-, Prozess-, Market-Engineering*, A. Oberweis, C. Weinhardt, H. Gimpel, A. Koschmider, V. Pankratius and B. Schnizler (eds.), February 2007, pp. 423-440.

Hepp, M., Leymann, F., Domingue, J., Wahler, A., and Fensel, D. 2005. "Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management," in *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE 2005)*, W. Tsai, J. Chung and M. Younas (eds.), October 2005, pp. 535-540.

Hevner, A. R., March, S. T., Park, J., and Ram, S. 2004. "Design Science in Information Systems Research," *MIS Quarterly* (28:1), March, pp. 75-105.

Hoffmann, J., and Brafman, R. I. 2006. "Conformant planning via heuristic forward search: A new approach," *Artificial Intelligence* (170:6), May, pp. 507-541.

Hoffmann, J., and Brafman, R. I. 2005. "Contingent Planning via Heuristic Forward Search with Implicit Belief States," in *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS 05)*, S. Biundo, K. L. Myers and K. Rajan (eds.), June 2005, pp. 71-80.

Hornung, T., Koschmider, A., and Oberweis, A. 2007. "Rule-based Autocompletion of Business Process Models," in *CAiSE Forum 2007, Conference on Advanced Information Systems Engineering*, June 2007.

Liang, Q., and Su, S. 2005. "AND/OR Graph and Search Algorithm for Discovering Composite Web Services," in International Journal of Web Services Research (2:4), October, pp. 48-67.

Ma, Z., and Leymann, F. 2008. "A Lifecycle Model for Using Process Fragment in Business Process Modeling," *Proceedings of the 9th Workshop on Business Process Modeling, Development, and Support (BPDMS 2008) in Conjunction with the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08)*, June 2008, pp. 1-9.

March, S. T., and Smith, G. F. 1995. "Design and natural science research on information technology," *Decision Support Systems* (15:4), December, pp. 251-266.

Marjanovic, O. 2005. "Towards IS supported coordination in emergent business processes," *Business Process Management Journal* (11:5), pp. 476-487.

Meyer, H., and Kuropka, D. 2005. "Requirements for Service Composition," *Technische Berichte Nr. 11 des Hasso-Plattner-Instituts fuer Softwaretechnik an der Universität Potsdam*.

Meyer, H., and Weske, M. 2006. "Automated Service Composition using Heuristic Search," in *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*, S. Dustdar, J. L. Fiadeiro and A. Sheth (eds), September 2006, pp. 81-96.

Motik, B., Patel-Schneider, P. F., and Horrocks, I. 2008. "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax," *W3C working draft, April at http://www.w3.org/TR/owl2-syntax/*, last called: 2009-09-07.

Pathak, J., Basu, S., and Honavar, V. 2006. "Modeling Web Services by Iterative Reformulation of Functional and Non-Functional Requirements," in *Proceedings of the fourth International Conference on Service Oriented Computing (ICSOC 2006)*, A. Dan and W. Lamersdorf (eds.), December 2006, pp. 314-326.

Petrick, R. P. A., and Bacchus, F. 2002. "A Knowledge-Based Approach to Planning with Incomplete Information and Sensing," in *Proceedings of the Sixth International Conference on AI Planning & Scheduling (AIPS 2002)*, M. Ghallab, J. Hertzberg and P. Traverso (eds.), April 2002, pp. 212-221.

Pistore, M., Traverso, P., Bertoli, P., and Marconi, A. 2005. "Automated Synthesis of Composite BPEL4WS Web Services," in Proceedings of the IEEE International Conference on Web Services 2005 (ICWS2005), July, pp. 293-301.

Recker, J., Rosemann, M., and van der Aalst, W. M. P. 2005. "On the User Perception of Configurable Reference Process Models - Initial Insights," in *Proceedings of the 16th Australasian Conference on Information Systems (ACIS'2005)*, B. Campbell, J. Underwood and D. Bunker (eds.), November 2005, pp. 1-10.

Sadiq, W., and Orlowska, E. 2000. "Analyzing Process Models Using Graph Reduction Techniques," in Information Systems (25:2), April, pp. 117-134.

Sun, S. X., Zhao, J. L., Nunamaker, J. F., and Sheng, O. R. L. 2006. "Formulating the Data-Flow Perspective for Business Process Management," *Information Systems Research* (17:4), December, pp. 374-391.

Thomas, O., and Fellmann, M. 2007. "Semantic business process management: ontology-based process modeling using event-driven process chains," *International Journal of Interoperability in Business Information Systems* (2:1), pp. 29-43.

van der Aalst, W. M. P. 2000. "Workflow Verification: Finding Control-Flow Errors Using Petri-Net-based Techniques," in *Business Process Management: Models, Techniques, and Empirical Studies,* W. M. P. van der Aalst, J. Desel and A. Oberweis (eds.), Berlin: Springer, pp. 161-183.

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

Weigand, H., van den Heuvel, W.-J., and Hiel, M. 2008. "Rule-based service composition and service-oriented business rule management," in *Proceedings of the International Workshop on Regulations Modelling and Deployment (ReMoD'08)*, J. Vanthienen and S. Hoppenbrouwers (eds.), June 2008, pp. 1-12.

Weske, M., van der Aalst, W. M. P., and Verbeek, H. M. W. 2004. "Advances in business process management," *Data & Knowledge Engineering* (50:1), July 2004, pp. 1-8.

Wu, D., Sirin, E., Hendler, J., Nau, D., and Parsia, B. 2003. "Automatic Web Services Composition Using SHOP2," in Proceedings of Planning for Web Services Workshop in ICAPS 2003, June 2003.