The Knowledge Engineering Review, Vol. 28:2, 175–193. © Cambridge University Press, 2013
doi:10.1017/S0269888912000410
First published online 4 March 2013

From business process models to hierarchical task network planning domains

ARTURO GONZÁLEZ-FERRER, JUAN FERNÁNDEZ-OLIVARES and LUIS CASTILLO

Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada c/ Periodista Daniel Saucedo s/n, 18071 Granada, Spain;
e-mail: arturogf@decsai.ugr.es, faro@decsai.ugr.es, l.castillo@decsai.ugr.es

Abstract

Hierarchical Task Network (HTN) planning paradigm has been widely used during the last decade to model and solve planning and scheduling (P&S) problems, and it has proved to be very useful in the planning and coordination of human tasks. At the same time, Business Process Management (BPM) tools are being increasingly used in the modeling of organizations' business practices and processes, but their life cycle has shown to have some shortages (as the possibility to obtain context-dependent plan instances). In this paper we present a methodology and software framework to translate Business Process Models into HTN P&S domains, in order to cover some of these deficiencies.

1 Introduction and motivation

Enterprises and organizations are facing today the emerging challenge of integration and automation of their business processes. The complexity of this issue increases when they have to deal with human-centric processes, as they are usually carried out in an informal manner, and the coordination of the different tasks and participants involved in these processes is very difficult to achieve. In this case, new technologies, mostly oriented to support decision making, have to be introduced to help knowledge workers like organization managers and decision makers to successfully achieve this goal.

A new set of tools and standards has been developed in the last decade in order to define these business process models, grouped under the name of Business Process Management (BPM), facilitating enterprises to detail their business practices, understanding these as frequently repeated acts, habit or custom performed to a recognized level of skill (Lock Lee, 2005). BPM standards are able to deal with goals and tasks specification, environmental analysis, design, implementation, enactment, monitoring and evaluation of business processes (Muehlen & Ho, 2006). Even showing these potentials, BPM tools lacks of better support for decision making capabilities. Thus, Artificial Intelligence (AI) techniques like Planning and Scheduling (P&S) could be integrated into the BPM life cycle, in order to support such features.

Although process modeling standards and tools enable organizations to leverage their business practices, these are usually very difficult to plan in advance. In BPM, the process *model* is used to represent the definition, and the process *instance* is used to represent the corresponding processing at a given timeline. A specific process model can have many different corresponding process instances, and the deployment and execution of these instances strongly depends on a given organizational context at the moment of its enactment. An example of such process model may be the management for the collaborative development of courses within a e-learning center (a special case

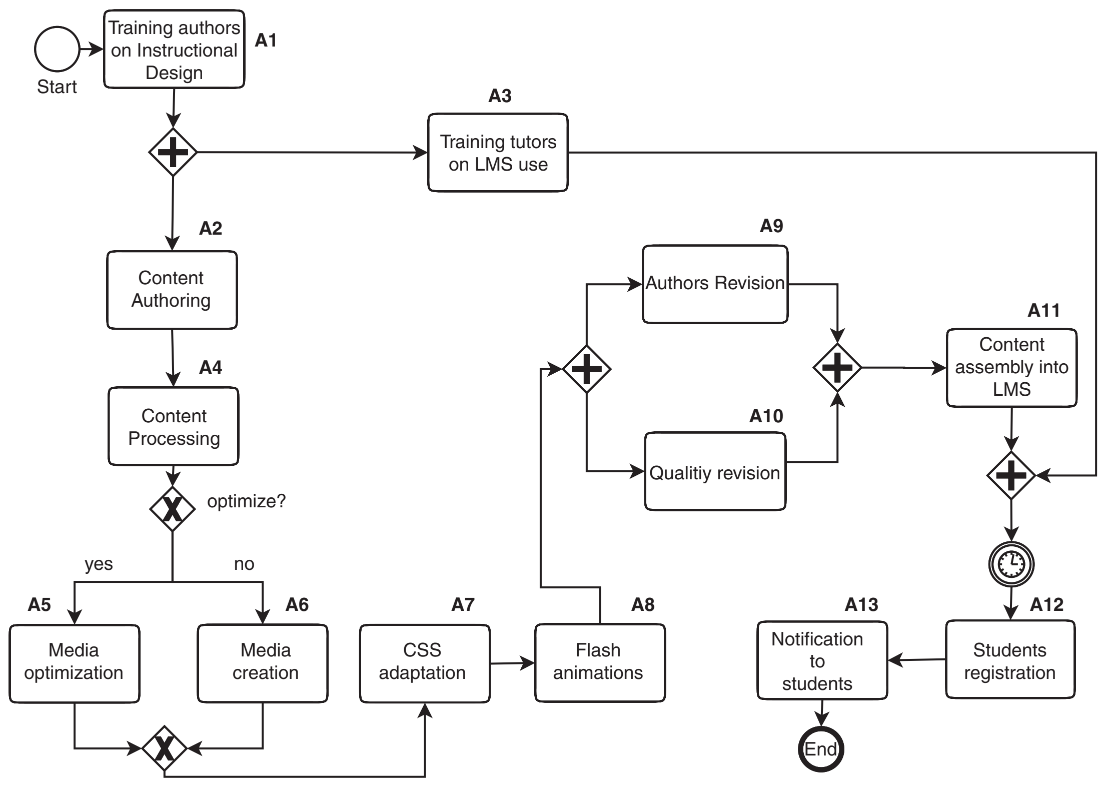


Figure 1 A BPMN (Business Process Management Notation) model example, describing the course development process in a specific e-learning center. There are six different departments (lanes), namely *Training, Authoring, Development, Graphic Design, System Administration* and *Quality Management*, and the activities are grouped as follows: (A1,A3), (A2,A9), (A4,A11), (A5,A6,A7,A8), (A12,A13), (A10). LMS = Learning Management System, CSS = Cascade Style Sheets

of product development processes). Upon a customer request, the manager of the organization needs an estimation of the tasks to be accomplished, the resources to be used in the course production, as well as the time needed to deploy it (see Figure 1). Under these conditions, since the final workflow instance to be carried out cannot be easily devised *a priori*, decision makers rely on either (a) *project management tools* or (b) *business process simulation tools* to support decisions about activity planning (in order to find dependencies between tasks and their time and resources constraints). Option (a) requires to invest much time using it. Option (b) determines several scenarios and simulate them, carrying out a trial-and-error process that may not reflect reality; this happens when a high number of alternatives courses of action makes hard to foresee which tasks should be considered, or when the constraints imposed by the context are significantly hardened. So, it is widely recognized Workflow Management Coalition (WfMC), (2010) that the BPM life cycle presents some weaknesses, and new techniques must be developed at the modeling/generation step, in order to fully cover the needs of knowledge workers for dynamic, adaptable processes.

From the AI P&S point of view, the need to obtain a context-dependent process instance from a given process model can be seen as the problem of obtaining a plan that represents a case for a given situation, and such that its composing tasks and order relations, as well as its temporal and resource constraints, strongly depend on the context for which the plan is intended to be executed. This problem requires at least two strong requirements in order to be solved. First, since the (possibly nested) conditional courses of action that may be found in a process model lead to a vast space of alternative tasks and possible orderings, it is necessary to carry out a search process in order to determine the sequence of actions to be included in the situated plan. Second, the search process necessarily has to be driven by the knowledge of the process model, which in most cases takes a hierarchical structure. Precisely, HTN planning domains are designed in terms of a hierarchy of compositional activities, where every task may be decomposed following different schemas or methods, into different sets of sub-activities. Furthermore, an HTN planning process is a search and deliberative reasoning process guided by knowledge.

The main contribution of this paper is the development of an innovative Knowledge Engineering technique with which, by means of a non-trivial transformation from a preexisting process model, we can automatically generate an AI planning domain. This is something that normally requires great skill and understanding by knowledge engineers. Furthermore, this technique takes advantage of the knowledge and control structures present in the original process model, so that the resulting domain is able to capture those control structures, by means of HTN procedural knowledge. By interpreting the domain generated, and through a search process guided by the knowledge extracted from the process model, an intelligent planner can find situated plans considering different conditions of the process environment, respecting also resource and temporal constraints. Henceforth, this contribution could be the cornerstone for the introduction of a new *planning* stage into BPM tools, improving the support for decision making that they can provide, but also avoiding the traditional difficulty of knowledge-based modeling that AI planning technology entails.

The paper is structured as follows. Section 2 introduces some concepts and technical background about the problem. Section 3 details the Knowledge Engineering procedure developed and its requirements. Section 4 exposes the software framework developed and some results. Section 5 presents relevant related work, and Section 6 describes some conclusions and future work.

2 Technical background

In this section, a description of the subset of Business Process Management Notation (BPMN)/ XML Process Definition Language (XPDL) process modeling elements considered for our approach is introduced first. Then, *process structuredness* is defined in order to delimit some properties that the input process model must fulfill, and *Workflow Patterns* are described, in order to convey why they are used as the main background concept for our transformation. Finally, the Hierarchical Task Network (HTN) planning paradigm is introduced.

2.1 Business Process Management Notation/XML Process Definition Language

The BPMN offer a graphical representation of the process. On the other hand, the aim of XPDL (WfMC, 2008) is to store and exchange a process definition, providing an XML serialization of the former. A description of the XPDL entities considered in our work is exposed:

Activities are logical, self-contained unit of work, carried out by *Participants*. Activities are related to one another via *Transitions*, that can be either *conditional* (involving evaluated expressions that drive the sequence flow path) or *unconditional*, and may result in the sequential or parallel operation of activities. *Gateways* are used to implement decisions affecting the flow path through the process. On a conditional transition exiting a gateway, it can be specified that the transition will be followed only when a specific *Parameter* value match the expression specified in an associated rule. Furthermore, Activities, Gateways and Transitions can be grouped hierarchically into *ActivitySets*, which are embedded subprocesses within a process. *Lanes* denote departments of the organization or process, and activities contained within a specific lane will be done by Participants that belongs to that area (encoded by using *extendedAttributes*, a standard way to augment the semantic of BPMN). Further details about these elements can be explored in González-Ferrer *et al.* (2009).

2.2 Structuredness and workflow patterns

It is important to highlight that the input process model should not be subject to syntactic and semantic errors that could be introduced at the modeling phase, as it is essential that process models not only precisely capture business requirements but also ensure successful workflow execution. Note also that, thinking on a translation process like the one we introduce in the next sections, such errors would be propagated and would result into nonsense planning domains that would be either useless or incorrect. Henceforth, it is important to delimit some properties of the input model and for that reason, the concept of *well-structured* process model is introduced in this subsection.

A correct process model is one without structural flaws, such as deadlocks, dead-end paths, incomplete terminations, etc. (Sadiq & Orlowska, 1997). A *well-structured workflow* is one in which each split control element (e.g. either OR or AND gateways) is matched with a join control element of the same type, and such split-join pairs are also properly nested (a more formal definition can be found in Eder *et al.* (2006)). Structuredness was first introduced on Kiepuszewski *et al.* (2000), and later discussed by Liu and Kumar (2005). Most workflow tools can only support structured workflows, given the fact that unstructured ones are more prone to errors (Liu & Kumar, 2005). Furthermore, some practical approaches have been developed recently in the same direction: some tools provide automated transformations of unstructured models into structured ones (Vanhatalo *et al.*, 2008), while others offer a pattern catalogue to avoid the use of unstructured fragments (Koehler & Vanhatalo, 2007; Koehler *et al.*, 2009).

The approach presented here will be only applicable to well-structured processes. On the one hand, this is not a strong requirement for most commercial tools (SAP R/3 workflow or IBM Filenet impose the structuredness as a requirement for modeling (Kiepuszewski *et al.*, 2000)), and some even do not support the execution of unstructured models (Liu & Kumar, 2005). On the other hand, imposing this requirement at the modeling stage can be beneficial for the end-user, given that the aim of business process models is to be finally executed correctly. In the same manner, if our goal is to find a correct execution plan for the process model, structuredness is relevant as well as a reasonable yet not restrictive condition to be fulfilled by the input process model, as explained later in Section 3.1.1.

At the same time, with the aim of delineating the fundamental requirements that arise during business process modeling on a recurring basis, a set of (typically nested) structures, that capture frequently used relationships between tasks in a process model have been recently defined, known as *Workflow Patterns* (van der Aalst *et al.*, 2003). Although the XPDL language can correctly represent some of these patterns, it lacks of some power for the representation of the most complex (van der Aalst, 2003). Therefore, only the most basic workflow patterns are going to be considered in our approach, those that can be well represented and are expressive enough for the definition of most processes: *serial* (sequence of activities that are executed one after another), *parallel split-join* (activities are executed simultaneously) and *parallel exclusive-OR* (used to capture conditional structures). As shown later, our mapping process will work by detecting these workflow patterns in a process model and translating each of them into its corresponding HTN structure, showing the capacity that the HTN planning paradigm have to capture knowledge expressed through a process model. Next, we introduce this planning paradigm.

2.3 Hierarchical task network planning language

HTN planning domains are designed in terms of a hierarchy of compositional activities. Lowest level activities, named actions or primitive operators, are non-decomposable activities that basically encode changes in the environment of the problem. On the other hand, high level activities, named tasks, are compound actions that may be decomposed into lower level activities. Every task may be decomposed following different schemas, or methods, into different sets of sub-activities. These sub-activities may be either tasks, which could be further decomposed, or just actions. The HTN planning domain language used in this work is a hierarchical extension of PDDL (Planning Domain Definition Language; Long & Fox, 2003) that uses the following notation:

Types, constants, predicates, functions and *durative actions* are used in the same way as in the original PDDL. The *task* element is introduced to express compound tasks, and its definition can include *parameters*, different decomposition *methods* with associated *preconditions* (that must hold

```
(:task BlockPB2
:parameters ()
(:method blpb2
:precondition ()
:tasks ([(AUTHORREVISION ?w1)
(QUALITYREVISION ?w2)]
(ASSEMBLYLMS ?w3))

(:durative-action AUTHORREVISION
:parameters(?w - participant)
:duration (= ?duration 20.0)
:condition(belongs_to_lane ?w Authoring)
:effect (completed authorrevision))

(:durative-action QUALITYREVISION
:parameters(?w - participant)
:duration (= ?duration 10.0)
:condition(belongs_to_lane ?w Quality)
:effect (completed qualityrevision))

(:durative-action ASSEMBLYLMS
:parameters(?w - participant)
:duration (= ?duration 20.0)
:condition(belongs_to_lane ?w Formatting)
:effect (completed assemblylms))
```

Figure 2 Example HTN-PDDL code for a split-join pattern in Figure 1. HTN = Hierarchical Task Network; PDDL = Planning Domain Definition Language

in order to apply the decomposition method) and *tasks* to represent its corresponding lowest level task decomposition. At the problem definition, *objects* are used to define objects present in the problem, *init conditions* are the set of literals that are initially true, and *task-goals* are the set of high level tasks to achieve.

Compound tasks, decomposition methods and primitive actions represented in a planning domain mainly encode the procedures, decisions and actions that are represented in the original BPM model. More concretely, the knowledge representation language, as well as the planner used, are also capable of representing and managing different workflow patterns present in any BPM process model. A knowledge engineer might then represent control structures that define both, the execution order (sequence, parallel, split or join), and the control flow logic of processes (conditional and iterative ones). For this purpose, the planning language allows sub-tasks in a method to be either sequenced, and then they appear between parentheses (T1,T2), or splitted, appearing between brackets [T1,T2] (Figure 2). The IACTIVE^{TM} planner has been chosen, as it is already known how to translate workflow patterns for web services composition (Fdez-Olivares *et al.*, 2007), it manages temporal knowledge (Castillo *et al.*, 2006) and it has been used in several applications (Castillo *et al.*, 2007; Fdez-Olivares *et al.*, 2008).

Section 3 describes the Knowledge Engineering (KE) procedure to extract the HTN domain and problem from a process model.

3 Methodology for the Business Process Management–Hierarchical Task Network translation

The methodology here presented consists on a Knowledge Engineering proposal for capturing knowledge from a BPM model that will finally be represented into an HTN planning domain. The idea behind our translation process is to identify common workflow patterns in a process model (which can be clearly represented as a graph), so that a tree-like structure can be generated, much similar to HTN domains, by carrying out a graph reduction process based on the workflow patterns found, followed by a subsequent process of restructuring into a tree model. So, this KE

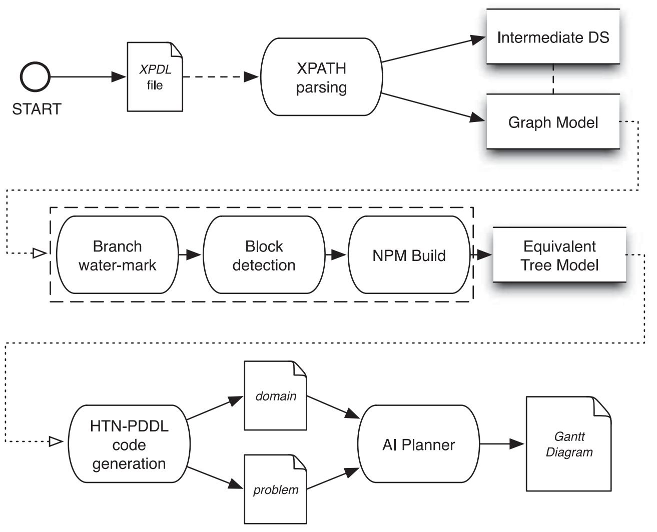


Figure 3 The different stages of the translation process. HTN = Hierarchical Task Network; PDDL = Planning Domain Definition Language; NPM = nested process model. DS = Data Structure

proposal consists of three different stages (an overview of all the steps needed can be explored in Figure 3 and algorithm 1):

(a) First, a input model preprocessing (step 1), storing it into an intermediate data structure and graph model (step 2) that can be easily managed throughout the next stages. Note that the input process model can be designed in some different ways, and can even have different connected subcomponents that represents different subprocesses that are part of a larger one, as explained later.
(b) Next, the detection of different workflow patterns is carried out (step 3), distinguishing their type (serial, split-join, XOR) from the knowledge acquired in the previous preprocessing stage, building up an equivalent tree-like model (step 3.3). This is carried out by arranging those workflow patterns hierarchically, but also keeping the semantic information (about control flow and decisions) present in the process diagram.
(c) Finally, the corresponding code generation takes place (step 4), where the tree model is analyzed, identifying common workflow patterns found in the graph (i.e. serial or parallel split-joins patterns are always coded in the same way), and generating HTN-PDDL code for the corresponding tree fragment.

Next, we proceed to give further insights on the development of these steps.

3.1 Mapping to a graph model

This stage (steps 1 and 2 in algorithm 1) takes as input a standard XPDL file (previously exported from the BPM modeling tool used), reading it by using XPATH (W3C, 1999) parsing technology, which allows searching only the XML entities we are interested in. Then, it produces a graph as result, in which every node represents an activity (or gateway) and every edge represents a transition between two activities (conditional or unconditional, as exposed previously at BPMN/XPDL subsection). Furthermore, the nodes will have associated different attributes (*nodetype*, *name*, *lane*, *duration* for activities, *parameters* for gateways, *activitySetId* for subprocesses, etc.).

Algorithm 1 General overview

```
Input: A process model $I$
Output: HTN-PDDL Planning Domain $D$ and Problem $P$
  1. [Preprocessing] Build structure $\Theta = \{P, R, T, A, Z, L\}$, consisting of different sets of objects found from
     parsing $I$ (Participants, Parameters, Transsition, Activities, ActivitySets and Lanes).
```

2. [PopulateGraph (G, A, T)**]**. Let $A \in \Theta$, $T \in \Theta$, build up a weighted directed graph $G = (V, E)$ where vertex set $V \equiv A$, and edges set $E \equiv T$.

2.1. When found a subprocess node $p \in V$, let $Y_p \in Z$ be the *ActivitySet* of subprocess *p*, and let activities $A' \in Y_p$ and transitions $T' \in Y_p$, build the corresponding subgraph *Up* by calling recursively to PopulateGraph (U_p, A', T'), and store it into *p*.

3. [BlockDetection]. For all $J \subseteq G$, *J* being a *m.c.c* fulfilling properties I, II, III (see Section 3.1.1):

3.1. [BranchWater]. Let $S \in J$ be the start node and initial weight *h* (default is 1.0), $weight(S) = h$, simulate a pipe of water from start to end node, to weight the rest of nodes (being the graph well-structured, opening gateways only have 1 predecessor and closing gateways only have 1 successor):

3.1.a) For all opening SPLIT or XOR gateways *W* such that $(i = 1 \wedge o > 1)$, being $i = |pred(W)|, o = |succ(W)|$, if $p \in pred(W)$, $\forall v / \ v \in succ(W)$, $weight(v) = weight(p)/o$.

3.1.b) For all closing JOIN or XOR gateways *W* such that $(i > 1 \wedge o = 1)$, being $v \in succ(W)$, and $p_j \in pred(W)/1 \leqslant j \leqslant i$, then $weight(v) = \sum_{k=1}^{j} weight(p_j)$

3.2. [Workflow Patterns Detection]. Search alternatively for serial (a sequence of nodes with the same weight) and parallel blocks (the same weight at both starting and closing *gateways*) in subgraph *J*. Then, do a reduction of *J* by replacing the blocks found with special PB (parallel) and SB (serial) nodes, storing internally the set of nodes that have been replaced $\{X_1, \ldots, X_m\}$. Repeat this operation until $size(J) = 1$. Completed this process, the unique node $r \in J$ is either a PB or a SB.

3.3. [RebuildAsTreeModel]. Let $r \in J$ be the root node, expand the tree by adding edges $\{(r, X_1), (r, X_2) \ldots (r, X_m)\}$ using the set of nodes replaced $\{X_1, \ldots, X_m\}_r$ which has been stored previously, and repeat this process recursively for every X_i such that X_i is a PB or SB node.

3.3.a) While doing the rebuild process, for every subprocess node *p*, call to BlockDetection over the subprocess graph U_p just created, in order to obtain its corresponding tree model U_p', and replace subprocess node *p* with subtree U_p' in the tree model J' being built.

When step 3 is completed, a tree model J' is obtained, derived from original subgraph *J*.

4. [Translation] To generate the planning domain *D*, see details on the paper, specifically on algorithms 2, 3, 4 and 5. Obtaining the planning problem *P* is also specified.

3.1.1 Input process model requirements

For the sake of the framework usability and the correctness of the translation process, we need to establish some requirements on the input process model, owing to the fact that not always the diagrams designed have the desired properties for later processing (Koehler & Vanhatalo, 2007). Basically, we introduce some requirements here to guarantee the correct operation of our proposal.

DEFINITION 1. Let $G = (V, E)$ be the graph corresponding to the input process model. Then a connected subgraph $J = (V', E')$, $J \subset G$ is a *maximally connected component (m.c.c.)* of *G* if $\forall u / \{u \in V \text{ and } u \notin V'\}$ there is no vertex $v \in V'$ for which $(u,v) \in E$. It can be defined as a connected subgraph of a graph to which no vertex can be added and it still be connected, or informally (one of) the biggest connected subgraph(s) of a graph[1].

The next three properties have been considered on the input process model:

1. All the m.c.c.'s, $J_i \subset G$ must include an unique start node *s* and an unique end node *e*. According to graph theory, J_i must be *two-terminal.*
2. The input graph model must be *well structured.*
3. The input process model m.c.c.'s must be connected between elements from start to end nodes, so that for every node, at least a path from *s* to *e* exists that contains that node (i.e. the corresponding graph for that m.c.c. is *directed* and *connected*).

A discussion about why some requirements are needed for the input models can be useful for the reader at this point. Section 2.2 gives insight about process models structuredness, usually demanded when a computational analysis of a process model has to be carried out. There are two implicit conditions derived from well structuredness: (a) it is assured that opening gateways only have

[1] `http://www.itl.nist.gov/div897/sqg/dads/HTML/maximallyConnectedComponent.html`

one predecessor and closing gateways only have one successor (the fragment delimited by both gateways is known as a single-entry single-exit fragment, or SESE region) and, perhaps more important, (b) every path that starts in an opening gateway *g* must pass throughout the matching closing gateway for *g* in its way to the end node (e.g. the fragment constituted by A9, A10 in Figure 1).

See Appendix A for a demonstration of the fact that these conditions are, at least, sufficient to guarantee that the block detection procedure, explained in the next subsection, is carried out correctly. Also, it gives more insight on why the m.c.c.'s of the process model are used. For the sake of simplicity, we continue our explanation assuming that the input model only contains one m.c.c.

3.2 Block detection: building a tree model

At the previous step, a graph model of the original process diagram has been built up. The goal of this stage is to build a equivalent tree model from the graph obtained previously (equivalent in the sense that all the knowledge about control flow is kept exactly the same in the new structure, allowing us to represent it into the HTN domain we are building up). This level of the mapping process is based on previous research done in Bae *et al.* (2004), where an algorithm was developed to generate a tree representation of a workflow process, which was used to derive ECA (event-condition-action) rules, helpful for controlling the workflow execution.

The tree representation obtained is called a nested process model (NPM; Bae *et al.*, 2004). It describes how to build up a process model in a top-down manner, representing a root node that is decomposed into a set of subprocesses and tasks. It adopts and generalizes a hierarchical model, allowing to express a parent–child relationship between subprocesses. This tree-like model is adapted to our problem, the representation of P&S domains, taking into account the control-flow information included in *gateways* and *transitions*, adding additional information about the process and data model as well.

Thus, the algorithm for block detection described has the next three steps (see step 3 in algorithm 1):

1. The first step is to mark every node of the graph with a weight, based on a *branch-water* procedure (see Figure 4). It simulates a pipeline network carrying water, being 1.0 the quantity of water poured at the start node, and branching the quantity through the pipe. If the water-level at a specific node is *l,* and the flow is branched into *k* alternatives, then l/k quantity of water is propagated through every alternative node. The water-level measure, that is, the weight of the nodes, is the method used to iteratively identify the most inner blocks in the graph, which allows to build a NPM in a bottom-up approach, as exposed next.
2. The second step is to identify serial and parallel workflow patterns (named *blocks* here) consecutively, using the weight to identify the most inner block. Every time a serial block (SB) or parallel block (PB) is identified, all the nodes that constitutes that block are replaced with a special SB node or PB node, obviously linking the new node with the preceding and successor nodes. As long as the workflow graph fulfills the established requirements, it is easy to see that this process ends having an unique SB or PB block node that constitutes the root node for the NPM.
3. Finally, if the root node is expanded using the nodes it grouped originally, placing them as children, repeating this operation recursively with every SB or PB block node, the new tree-like structure we were looking for is obtained (this is done as a typical breadth-first search algorithm). The result of the procedure constitutes what is called the NPM of the original BPM diagram, using a bottom-up approach (see Figure 5(a)). Observe that nodes with minimum weight lay at the lower levels, going up consequently as their weight increase (this is the reason to look first for the most-inner blocks).

Given that our approach follows a knowledge-driven process, we needed to adapt the original algorithm for Block Detection (Bae *et al.*, 2004), in order to (a) keep all the knowledge acquired in the preprocessing stage, (b) keep the original nodes that gave rise to the new special block nodes and (c) transfer the knowledge present in gateways into the new PB nodes (i.e. the type of gateway, the parameters/rules that drives the flow, etc.), since those gateway nodes are not going to be

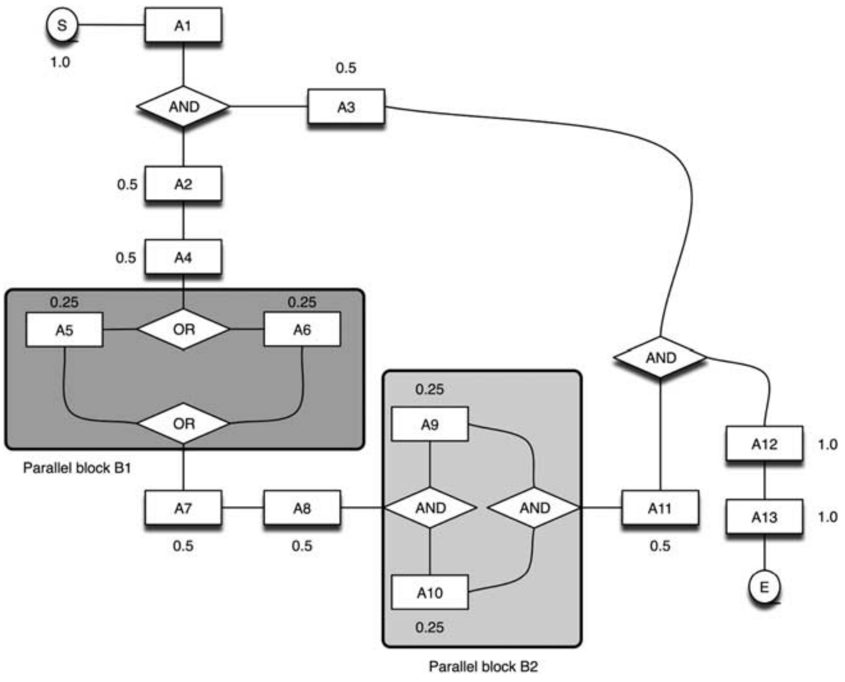


Figure 4 Part of the block detection algorithm applied to graph of Figure 1. The branch-water mark procedure as well as the workflow pattern detection are visible in the picture

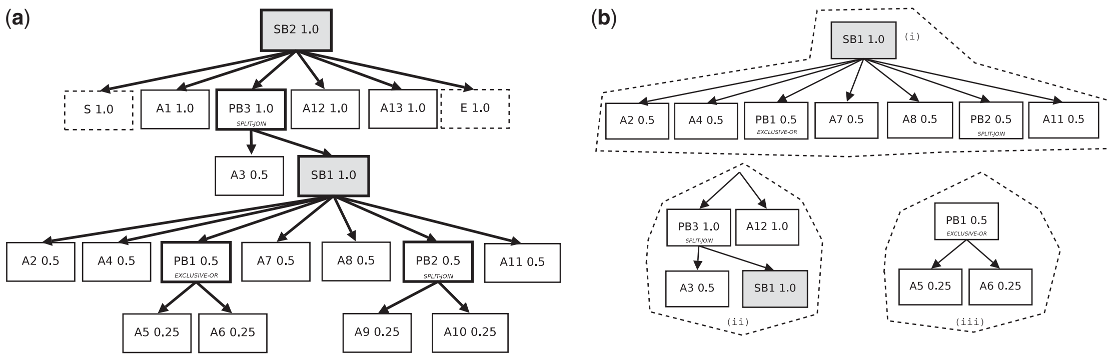


Figure 5 NPM and the workflow patterns identified on it. (a) An NPM generated from the previous BPMN process model. Note that leaf nodes correspond to activities and non-leaf nodes correspond to serial and parallel blocks. (b) Different patterns identified in the NPM representation that are mapped as HTN compound tasks. (i) a serial block, (ii) a split-join block, (iii) a XOR block NPM = nested process model; BPMN = Business Process Management Notation; HTN = Hierarchical Task Network

present on the newly built NPM (but their semantic is maintained, including the relevant information mentioned into the newly created PB nodes). The algorithm complexity is O(n^2), being n the number of edges of the workflow graph.

Note that at this point, being G the input model, we have transformed all the m.c.c.'s (see Section 3.1.1) $J_1, \ldots, J_n \in G$ as tree models $J'_1, .., J'_n$, so that we have one or more NPMs stored in G' that are easily translatable into the corresponding HTNs, as shown next.

3.3 Hierarchical Task Network–Planning Domain Definition Language code generation

In this subsection, specific details are given about the generation of the HTN planning domain and problem files, taking as basis both the tree-like structure (the NPM, Figure 5(a)) and intermediate data structures, already developed in the previous phases. Opposite to the bottom-up approach followed to create the NPM, the generation of HTN-PDDL code is going to follow a top-down approach. As we already have a tree-like model, all we need to do is a breadth-first search over the NPM, considering the information relevant to every node (described along this section), and considering also some patterns related with some kind of nodes (see Figure 5(b)). Next, it is shown how to express the different elements of an HTN-PDDL domain and problem definitions. We also expose the underlying conceptual mapping from XPDL source elements, reflecting both the *process* and *data* models.

Domain name and requirements. These HTN-PDDL blocks are encoded as const strings (the requirements section is considered always the same).

Types. The basic types considered are those useful in any planning domain: *activity*, *participant* and *lane*. Of course, parameter data types must be also generated (see the corresponding item below).

Constants. XPDL *activities* and *lanes* are mapped as HTN-PDDL constants, which are going to be used at the domain and problem files. This is automatically extracted from set Θ (see algorithm 1), and they will be coded in lowercase characters (i.e. activities will be coded as a_x, being x the activity id).

Predicates. At least, two default predicates must be included, useful in almost any process model mapping: (1) *(belongs_to_lane ?p—participant ?l—lane)*. This predicate is used to express which lanes the participant belongs to. It will be used to encode both initial conditions of the problem (one predicate instance for every capacity a specific participant has) and preconditions for the durative actions (a precondition for every activity carried out in a specific lane). (2) *(completed ?a—activity)*. This predicate will encode initial conditions of the problem as well as preconditions and effects for durative actions.

There are also some predicates that should be added dynamically, those that are related to parameters/rules matching pairs (described later at *parameters* item).

Durative actions. Every activity of the process diagram corresponds to a leaf node in the NPM and it is mapped as a *primitive durative action* on the planning domain, as a fragment following the pattern example next to algorithm 2.

```
(:durative-action Ax
   :parameters(?w - participant)
   :duration (= ?duration D)
   :condition(belongs ?w L)
   :effect (completed ax))
```

Algorithm 2 Generate Durative Actions from Activities

```
Require: $G \neq null$ {G is the Nested Process Model}
Output: PDDL definition of found durative actions
 1: $output \Leftarrow''$
 2: Let $F \subset G$ be the set of leaf activity nodes
 3: for all $v$ in $F$ do
 4:     Let $w$ the worker that will be assigned $v$
 5:     Let $L$ be the lane $w$ belongs to
 6:     Let $c$ be a predicate expressing this membership
 7:     Let $d$ be the duration of $v$
 8:     Let $e$ be an effect expressing the completion of $v$
 9:     add to $output$ a durative action $a$ with parameter $w$,
        precondition $c$, duration $d$ and effect $e$
10: return $output$
```

Realise that order constraints among activities, in non-hierarchical planning paradigms, are coded through the use of preconditions in durative actions, being necessary an extra cause-effect analysis. However, in HTN planning paradigm, order constraints are directly mapped into the corresponding syntactic structures developed to that end. So, our approach does not need to abuse of precondition definition, simplifying the process, as exposed next in the definition of compound tasks.

Compound tasks. The HTN-PDDL compound tasks are mapped from those intermediate nodes (non-leaf nodes) of the NPM. These nodes always correspond to workflow pattern blocks (see Figure 5(b)) that are actually specifications of different tasks with control flow mechanisms that are coded as order constraints (sequential/parallel) or as alternatives (if-then).

1. *Serial blocks*. One activity must be executed after other, following a sequence in time. This can be expressed in HTN-PDDL as a sequence of primitive actions and/or tasks surrounded by parentheses. Example next to algorithm 3 represents the fragment of Figures 5(b)(i) and 6(a). Note that, on the one hand, durative actions A_x must be generated with the corresponding parameter ?w$_i$, which express a resource that has to be allocated at planning-time (the participant i is assigned the activity x). On the other hand, compound tasks that are also part of the decomposition

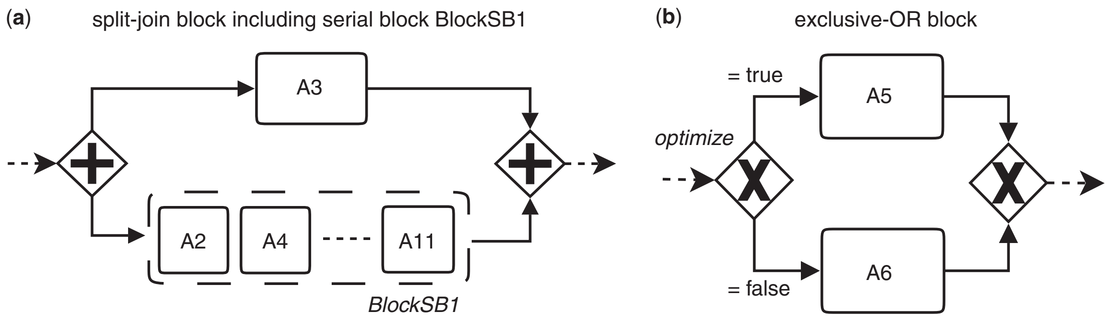


Figure 6 Parallel workflow patterns

can be generated with or without parameter, representing the formal *parameter* that drives the flow in the original XPDL diagram (i.e. 'optimize').

```
(:task SB1
   :parameters ()
   (:method blsb1
   :precondition ()
   :tasks ((A2 ?w1) (A4 ?w2)
   (PB1 ?optimize) (A7 ?w3)
   (A8 ?w4) (PB2)
   (A11 ?w5))))
```

Algorithm 3 Generate Tasks from Serial Blocks

```
Require: $G \neq null$ {G is the Nested Process Model}
Output: HTN-PDDL definition of serial blocks
 1: mark all nodes of G as translatable
 2: for all $node\ v \in G$ do
 3:     if $v$ is a $SerialBlock$ node then
 4:         add to $out$ a compound task named as $v$
 5:         add to $out$ a method definition named as $v$
 6:         $i \Leftarrow 1$
 7:         for all node $j$ being a child node of $v$ do
 8:             if $j$ is an Activity then
 9:                 if $j$ is translatable then
10:                     add to $out$ an action with name $j$ and parameter ?w$i$
11:                     $i \Leftarrow i + 1$
12:             else if $j$ is an Split-Join node then
13:                 add to $out$ a task named as $j$ without parameter
14:                 Let $r$ be right brother node of $j$ in the tree
15:                 mark $r$ as not translatable
16:                 if $j$ is an exclusive-OR node then
17:                     add to $out$ a task named as $j$ with
18:                     the parameter represented in BPMN diagram
19: return $out$
```

2. *Parallel split-join blocks*. They represent a branch of the process flow into two or more flows (*split*) that are carried out simultaneously (without specifying which of them should be executed first), and that finally converge into the same flow again (*join*). These parallel split-join blocks are represented in HTN-PDDL enclosed by square brackets, as the following case next to algorithm 4, that represents the fragment of Figures 5(b)(ii) and 6(a),. Note that A12, the *right brother node* of PB3 in Figure 5(b)(ii), is the activity executed after the *join* gateway. This schema repeats for every split-join block detected.

```
(:task PB3
   :parameters ()
   (:method blpb3
   :precondition ()
   :tasks ([(A3 ?w1)(SB1)]
   (A12 ?w2))))
```

Algorithm 4 Generation of Tasks for Split-Join Blocks

```
Require: $G \neq null$ {G is the Nested Process Model}
Output: HTN-PDDL definition of split-join blocks
 1: $out \Leftarrow ''$
 2: for all $node\ v \in G$ do
 3:     $i \Leftarrow 1$ {worker number}
 4:     if $v$ is a Split-Join node then
 5:         add to $out$ a compound task named as $v$
 6:         add to $out$ a method definition named as $v$
 7:         add a parallel block definition with child nodes of $v$
            {by enclosing with [ ] the actions produced on lines 8-16}
 8:         for all node $j$ child of node $v$ do
 9:             if $j$ is an activity node then
10:                 add to $out$ an action with name $j$ and parameter ?w$i$
11:                 $i \Leftarrow i + 1$
12:             else
13:                 if $j$ has associated a parameter $p$ then
14:                     add to $out$ an task with name $j$ and parameter $p$
15:                 else
16:                     add to $out$ an task with name $j$
17:         if $v$ has a right brother activity node $r$ then
18:             add to $out$ an action with name $r$ and parameter ?w$i$
19: return $out$
```

3. *Exclusive-OR blocks*. In these blocks a gateway node that has a parameter and a corresponding logical expression determines which alternative path to follow. A method is generated for every possible alternative to follow, using the expression as method precondition. The example next to algorithm 5 represents the fragment of Figures 5(b)(iii) and 6(b).

```
:task PB1
:parameters (?x - parameter)
(:method ifA5
   :precondition (value ?x true)
   :tasks (A5 ?w1))

(:method elseA6
   :precondition (value ?x false)
   :tasks (A6 ?w1))
```

Algorithm 5 Generation of Tasks for XOR blocks

```
Require: $G \neq null$ {G is the Nested Process Model}
Output: HTN-PDDL definition of exclusive-OR blocks
 1: $out \Leftarrow ''$
 2: for all $node\ v \in G$ do
 3:     if $v$ is a XOR Parallel node with parameter $x$ then
 4:         add to $out$ a compound task named as $v$
 5:         for all node $j$ child of node $v$ do
 6:             if $x$ value for $j$ is NULL or FALSE then
 7:                 add to $out$ a method named "$if\_j$" with
                    precondition $(value\ ?x\ false)$
 8:                 the method tasks are the actions/tasks $j$
 9:             else
10:                 add to $out$ a method named "$if\_j$" with
                    precondition $(value\ ?x\ true)$
11:                 the method tasks are the actions/tasks $j$
12: return $out$
```

Parameters. Parameters are usually associated to Exclusive-OR PBs, and they can be initially expressed as follows, as long as they have been modeled as *boolean* parameters:

a. Add an HTN-PDDL type ‘parameter’.
b. Add a HTN-PDDL constant for every parameter (i.e. the parameter named *optimize*).
c. Add a predicate (i.e. named *value*) to check boolean values (true, false). It is clear that the parameters and expressions should also be mapped in such a way that different data types besides boolean can be added to the framework, but this is one of the issues considered for future work.
d. Pass the corresponding parameter to the Exclusive-OR block wherever it is used, as done in previous example with parameter *optimize*. This is very easy, as the parameters have been already stored in the intermediate data structure.
e. In the problem file, define the parameter as an initial condition of the problem. Note that parameter values should be passed to the AI planner somehow before interpreting the domain and problem files generated (i.e. it can be given by the user outside the framework).

Besides this mapping, we also tried referring to an external organizational data model stored in Unified Modelling Language (UML), using some of the capabilities of the BPM modeler, as the XPDL standard supposedly supports it, but this feature was somehow experimental in the modeler and we could not complete it. Using UML for storing the data model would be ideal, as there are already authors (Vaquero *et al.*, 2007) who worked out a methodology to express this model in PDDL.

Objects. Every *participant* is going to be defined at the problem file as an object (of ‘participant’ data type). *Init conditions*. Besides parameter values mentioned above, we must include the abilities that every participant (previously defined as object) possesses, in other words, what lane the participant belongs to (using the predicate *belongs_to_lane*). *Goals*. The goal of the problem definition file will be the root node of the NPM, which is always a compound task, that can be iteratively decomposed in order to generate all the process plan. In case that several *m.c.c.* (see Section 3.1.1) are found, the task goal can be considered as the parallel execution of the corresponding NPM for each *m.c.c.* found.

We have described in previous sections the whole KE process followed to map a BPM model to its corresponding P&S domain and problem definitions.

4 The JABBAH software framework

Taking into account the methodology defined for translating BPM models into HTN planning domains, an extensible framework called JABBAH has been developed, able to carry out this translation. It has been developed in Java, and it is based on *jgrapht* library[2], very useful for creating graph data structures, with fully customized nodes and edges implementation. Details about source code, screenshots and even a screencast for JABBAH operation can be found in its website[3].

4.1 Experiments

Some experiments have been carried out by using JABBAH, with a twofold aim in mind. First, to show that the well-structured process models chosen have a corresponding HTN representation able to capture the knowledge present in the process (even within embedded subprocesses) and second, that it can be used to carry out a planning process in order to obtain a context-dependent plan that considers the task ordering and the organization resources, under different environment conditions. Specifically, the experiments have been designed in order to show the P&S capabilities that our approach introduces into the BPM life cycle and, furthermore, to show that this planning stage is carried out efficiently and correctly, guided by the knowledge present in a changing environment, and respecting all the constraints that were introduced in the original process model.

Concretely, the next ones are some expected outcomes of these experiments: (1) check that a corresponding HTN representation exists for process models that include a combination of (possibly nested) workflow patterns; (2) find a plan instance that keeps the temporal semantic associated to those workflow patterns; (3) check that the interpretation of the same HTN domain can find different plan instances for different combinations of input parameters (i.e. decision gateways values); and (4) show that both planning (find the activities that form part of the plan) and scheduling (determine a task ordering that respect time constraints, assigning also resources to activities) are needed in order to find a situated plan.

We have used JABBAH over two real processes, in the field of e-learning and e-health. The first model (Figure 1) represents how to develop and deploy a specific course within the e-learning center at the University of Granada. Having an incoming course request, as well as some available workers with different capabilities each, a plan instance can be obtained providing the managers information about the workers allocation and the make-span of the whole course development, helping to do decision making upon the course request. The second one (available in the 'domain repository' section of the website) represents a general care process starting from a patient entry into a hospital and finishing when the health insurance billing for this patient takes place.

Table 1 may be used to clarify how the hierarchical structure of HTNs is being used and why it is helpful in order to achieve the aim we pursue. Although the plans obtained are merely a sequence of actions, it is important to realize that they are obtained from a domain that has been automatically generated, which includes the procedural knowledge already existing in the initial process model. For example, the e-learning domain contains 13 activities (it has no subprocesses), and two serial, one alternative and two PBs were obtained. These blocks include procedural knowledge (e.g. for PBs, a set of actions A1 must be executed concurrently with respect to another set of actions A2) that is really complex to describe by using non-HTN planning. However, by using the HTN paradigm, a natural and direct parallelism exists between SBs and decomposition methods. At the same time, an alternative block of *n* tasks can be represented with *n* alternative HTN methods. Note that this implicitly represents a knowledge-based search heuristic, that is, knowledge-based rules used to guide the search, something that cannot be done with classical PDDL-based planners. Some concrete results are commented next.

In the first process, using a specific workers assignment at the e-learning center, as well as estimated activities duration, we have checked the viability of the output plans, and also that task

[2] `http://jgrapht.sourceforge.net/`

[3] JABBAH homepage, `http://sites.google.com/site/bpm2hth/home`

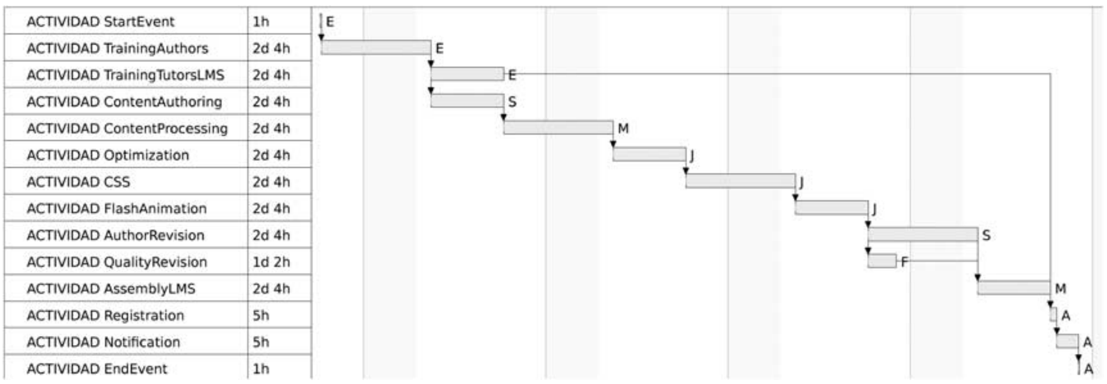
Table 1 BPMN elements and the corresponding HTN elements generated for both models

Process	Lanes	Activities	Subprocesses	starting XOR	closing XOR	starting AND	closing AND	Serial block	Alternative block	Parallel block	Durative actions	Init conditions	Objects	Parameters
e-learning	8	13	0	1	1	2	2	2	1	2	13	7	6	1
e-health	10	16	1	4	4	1	1	5	4	1	16	15	10	4

BPMN = Business Process Management Notation; HTN = Hierarchical Task Network.

Table 2 Different plan instances and allocation for e-health process according to different input values

Emergency	Special	Urgent	Intensive	Register-patient	Open-Emergency	General-Exam	General-Care	Special-Exam	Perform-Observation	Intensive-Care	Prepare-Anesthetic	FillOT-Sheet	Patient Reception	Perform Surgery	VitalSign Monitor	Surgery Report	Checkout Patient	Bill Services	Process Bill	Actions	Expansions	Gen. nodes	Make-span
0	0	0	0	p3		p5	p7											p2	p1	6	5	11	35
0	0	0	1	p3		p5	p7											p2	p1	6	5	11	35
0	0	1	0	p3							p10	p10	p10	p9	p10	p10	p3	p2	p1	10	6	16	58
0	0	1	1	p3							p10	p10	p10	p9	p10	p10	p3	p2	p1	10	6	16	58
0	1	0	0	p3		p5		p6	p7								p3	p2	p1	7	7	14	41
0	1	0	1	p3		p5		p6		p8							p3	p2	p1	7	7	14	36
0	1	1	0	p3							p10	p10	p10	p9	p10	p10	p3	p2	p1	10	6	16	58
0	1	1	1	p3							p10	p10	p10	p9	p10	p10	p3	p2	p1	10	6	16	58
1	0	0	0		p4	p5	p7										p3	p2	p1	6	5	11	29
1	0	0	1		p4	p5	p7										p3	p2	p1	6	5	11	29
1	0	1	0		p4						p10	p10	p10	p9	p10	p10	p3	p2	p1	10	6	16	52
1	0	1	1		p4						p10	p10	p10	p9	p10	p10	p3	p2	p1	10	6	16	52
1	1	0	0		p4	p5		p6	p7								p3	p2	p1	6	5	11	35
1	1	0	1		p4	p5		P6		p8							p3	p2	p1	7	7	14	30
1	1	1	0		p4						p10	p10	p10	p9	p10	p10	p3	p2	p1	10	6	16	52
1	1	1	1		p4						p10	p10	p10	p9	p10	p10	p3	p2	p1	10	6	16	52



ACTIVIDAD StartEvent	1h
ACTIVIDAD TrainingAuthors	2d 4h
ACTIVIDAD TrainingTutorsLMS	2d 4h
ACTIVIDAD ContentAuthoring	2d 4h
ACTIVIDAD ContentProcessing	2d 4h
ACTIVIDAD Optimization	2d 4h
ACTIVIDAD CSS	2d 4h
ACTIVIDAD FlashAnimation	2d 4h
ACTIVIDAD AuthorRevision	2d 4h
ACTIVIDAD QualityRevision	1d 2h
ACTIVIDAD AssemblyLMS	2d 4h
ACTIVIDAD Registration	5h
ACTIVIDAD Notification	5h
ACTIVIDAD EndEvent	1h

Figure 7 Output plan as a Gantt diagram

ordering is respected, that is, an assignment as the following: Emilio (*training*), Storre (*authoring*), Miguel (*html*), Jose (*graphic*), Arturo (*admin*) and FMoreno (*quality*), result on the plan shown as a Gantt diagram in Figure 7.

Concerning the e-health process, we checked how our tool was able to generate different process plan instances using the control knowledge extracted from the process model and respecting some different nested workflow patterns found on it. It is also shown the recursion capabilities of our proposal, by interpreting and translating correctly the embedded sub-processes included. Since this process provides several decision gateways, we better observe how process planning is carried out, given different input parameters that can vary in real situations (e.g. Is it an emergency? Does it need an urgent operation?). Table 2 show the different instances found for these decision points (values 0/1 represent false/true values for the input parameters), showing workers allocation (p1–p10), and the instance total make-span. Moreover, decisions taken by the planner in the search process (planner expansions) and the number of generated nodes are displayed. Note that each of the plan instances were obtained in less than 1 second.

Henceforth, our experiments make clear that a search process is necessary among the possibly different alternative course of actions expressed in a process model, also taking advantage of the knowledge expressed on it (note that the process model itself already discard some alternatives by using control structures). Thus, we opted to use HTN planning, which precisely describes methods to obtain valid plans, incorporating procedural knowledge to avoid exploring nonsense or wrong alternatives. It is important to highlight that non-hierarchical planning is not expressive enough to represent these control structures, and it is based on an exhaustive search process not guided by knowledge, so its use would be costly, searching over alternatives that are not considered initially in the process model. What's more, we have shown that it is not necessary for a process model to include hierarchical structures in order to obtain an HTN representation. However, HTN domains are able to represent the control structures present in BPM models by means of hierarchical structures, something that cannot be done with a classical planner.

Therefore, we have checked the proposed goals with non-trivial scenarios. Nonetheless, future work will be done in order to cover a bigger subset of process models (e.g. cooperative process models), and also to represent complex temporal constraints that could be expressed on them (see Section 6).

5 Related work

The interest of using AI within workflow technology is not new. The report by Myers and Berry (1998) already described how techniques from the AI community could be leveraged to provide several of the advanced process management capabilities envisioned by the workflow community. In addition, much research in the BPM area has lately been directed to achieve transformations from business process models into IT-related implementations. Stein *et al.* (2008) show a deep review of those centering in control flow-centered approaches. Most of them are thought as translations to BPEL (Business Process Execution Language). It is interesting to highlight the work done by Koehler *et al.* (2008), trying to leverage a transformation for execution of business process models, setting out ten different aspects that must be investigated in order to achieve this transformation, using also a tree representation of the process model (Vanhatalo *et al.*, 2009) to achieve their goal.

Even though there are already some approaches (Bouillet *et al.*, 2007; Simpson *et al.*, 2007; Vaquero *et al.*, 2007) devoted to the field of Knowledge Engineering for P&S, they are rather directed to be helpful for planning experts (dealing with the modeling of world objects and actions). Our approach is more aligned with Barták *et al.* (2008), since it deals with the automatic generation of planning domains from expert knowledge introduced previously by using standard tools and languages that are close to IT architects and organization stakeholders. It also shares similarities with Muñoz-Avila *et al.* (2002), which describes how project planning representations are similar to plan representations employed by HTN planners.

6 Conclusions and future work

This paper has made some innovative contributions in order to overcome the traditional drawbacks of acquiring knowledge for later P&S modeling, specifically for the HTN paradigm. Mainly, a sound KE procedure has been developed in order to express well-structured process models as HTN P&S domains, building up an intermediate data structure that organizes the source process diagram as a NPM, simplifying the subsequent transformation into HTN-PDDL code. Furthermore, the JABBAH software framework provides a neat tool for analysts that need to perform activity planning and resource allocation analysis on business workflows, embedding a non-trivial transformation of BPMN-expressed workflows in terms of HTNs. By providing a fully automated support of the analysis, allowing engineers to exploit the vastly diffused BPMN standard for workflow specification, and neatly presenting the results, it may appeal a very wide and relevant audience. It could be useful at project-based, customer service-based processes, or in general, human-centric processes. What is more, it can be helpful for Adaptive Case Management as well, in the sense that these plans could also be leveraged in highly dynamic scenarios, where exogenous events can modify the environment conditions and some kind of adaptive capacity can be demanded, while respecting the original process model.

Concerning future work, in order to cover a bigger subset of process models, the translation of *Associations* and *Message flows* will be implemented, allowing to use JABBAH for the analysis of cooperative process models. Also, an extension called Time-BPMN has been created recently (Gagné & Trudel, 2009) in order to allow the specification of temporal constructs not available in the original BPMN. Since HTN-PDDL is able to correctly represent these temporal constructs (Castillo *et al.*, 2006), we want to extend our framework in order to cover their translation.

Further improvements can be done, like (1) include support for BPMN 2.0 specification, (2) introduce a new block detection algorithm, like the RPST (Vanhatalo *et al.*, 2009), in order to improve the efficiency ($O(n)$), and check also its behavior for P&S domain generation.

Acknowledgments

This work was partially supported by projects P08-TIC-3572 and TIN2008-06701-C03-02.

Appendix A

This appendix is mainly directed to demonstrate that the conditions established for the correct translation are, at least, sufficient.

Demonstration of sufficient conditions. Next, we pass to demonstrate that the conditions imposed for the input model are at least sufficient for the correct operation of our proposal.

Let G be the input process model, let F be a connected subgraph of G ($F \subset G$).

DEFINITION 1. We define F as a *fragment* of G, if F is delimited by an opening gateway o (start node) and a closing gateway c (end node), and there is at least one activity node in every branch found inside the fragment. A fragment can include an arbitrary number of fragments.

DEFINITION 2. A *well-structured fragment* can be defined as a fragment that (a) has only one predecessor for opening gateways and only one successor for closing gateways, and such that (b) every path starting in an opening gateway g, must pass, by imperative, throughout the matching closing gateway for g on its way to the end node of the workflow graph.

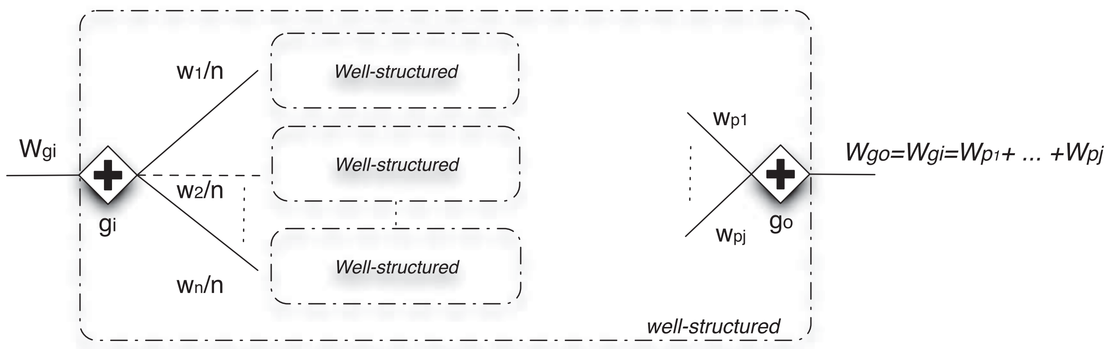
Let o_n be the number of opening gateways (either AND or XOR) of a *well-structured fragment* F, and let c_n be the number of corresponding closing gateways. It is always true that $o_n = c_n$, given that F is *well structured.*

DEFINITION 3. A *minimal well-structured fragment* is a well structured one that fulfills that $o_n = c_n = 1$.

THEOREM 1. A *well-structured fragment* is always reduced to an unique node by the algorithm presented.

Let p be the number of paths an opening gateway g_i divide the flow into (or the number of successors nodes for g_i). Let w_i be the weight that comes into the opening gateway g_i and w_c the weight that comes out of the closing gateway g_o. Then, according to the block detection algorithm, the weight assigned to every successor node j will be $w_{succ_j(g_i)} = w_i/p$.

.



Given the conditions derived from well structuredness, the path for all the branches starting at the opening gateway, $g_i \in F$, must arrive to the matching closing gateway, $g_o \in F$, so it is easy to devise that, as the weights of predecessors of g_o are added to find w_{g_o}, the weight of the opening gateway g_i is preserved at the exit of the closing gateway g_o, $w_{g_o} = w_{g_i}$. If this is not true, it would mean that either (a) exists a path starting on g_i that does not pass throughout g_o in its way to the end node of G, which clearly contradicts the definition of *well-structured fragment*, or (b) F is an unconnected fragment, which is contradictory to our property III (directed and connected graph).

Now, we can devise that, once we can assure that we can weight *well-structured fragments* as explained, and the weight is preserved equal at the start and end of single-entry single-exit regions, it is easy to see that we can reduce this fragment into an unique node, as explained next.

Given that the fragment F is *well structured*, every branch can contain:

a) a sequence of n activities ($n \geqslant 1$);
b) a fragment that is also *well-structured*;
c) an arbitrary (and possibly nested) combination of sequences and *well-structured* fragments.

It is easy to see that, as soon as F is a *well-structured fragment*, it will contain, at least, a *minimal well-structured fragment* F_a (or it is minimal itself, i.e. $F \equiv F_a$). Thus, starting from the most inner *minimal well-structured fragment*, where the lower weights are found (after some divisions of the starting weight into some nested branches), we can reduce this fragment into a special node, and recursively do this operation, taking into account that a similar reduction over a sequence of activities will be done first when the weight found for these sequence activities is lower or equal than it is for the most-inner SESE region (i.e. a PB). To make it more clear, if there exists a sequence within a *minimal well-structured fragment*, the sequence will be always reduced first, as their activities weight is equal to the one for other branches of this fragment (given that is minimal and no more inner gateways can be found).

It is also easy to see that the new special nodes created by the reduction step will then form part of either a previous existing sequence or a previous existing *well-structured fragment*, or in the last step, it is the unique node we are looking for. So, the Theorem 1is clearly demonstrated.

THEOREM 2. If the input workflow graph is not *two-terminal*, we cannot assure that it is *well structured*.

1) This is clearly shown if there is one start node and n end nodes $e_1, \ldots, e_n$ ($n \geqslant 2$). In this case, at least an opening gateway g_i will exist that divides the flow into the two (or more) branches to finish into the different n end nodes, and such that it does not have a corresponding closing gateway g_o. This clearly contradicts the definition of *well-structured* process model. Note that, if some different end nodes want to be created in a BPM model, it is usually solved by creating different two-terminal workflow graphs, connected by using associations or message flows (in order to synchronize their cooperative operation). This is why we use the m.c.c.'s of the input process models, in order to manage also these cases (subject of future work).
2) In the case that n start nodes $s_1, \ldots, s_n$ ($n \geqslant 2$) and one end node e exists, it is also clear that an extra closing gateway (not matching with an opening gateway) will be needed in order to join the flow to the end node, and this clearly contradicts the definition of *well-structured* process model. (What's more, the algorithm should simulate at least n pipes of water, and this doesn't make sense. Note that such a graph could be easily transformed into another with one start node s and an opening gateway that split the flow into the branches corresponding to those paths starting originally from $s_1, \ldots, s_n$.)

Hence, it is clear that if the workflow graph is *two-terminal* and *well structured* (and of course, *connected* and *directed*) then the input process model can only contain arbitrary (and possibly nested) combinations of sequences and *well-structured* fragments. Demonstrations of Theorem 1and Theorem 2show that properties I, II and III are sufficient conditions for the correct operation of the translation proposed.

The use of m.c.c.'s. The reason to use the m.c.c.'s of the input model is mainly the fact that a specific use case of BPM is the development of cooperative processes, where independent processes can be synchronized (i.e. by using *association* and *message flow* BPMN elements) in order to complete the whole process. Although the analysis of cooperative processes is subject for future work, the algorithm is already prepared to cover the translation of these m.c.c.'s into multiple HTNs.

References

Bae, J., Bae, H., Kang, S. & Kim, Y. 2004. Automatic control of workflow processes using ECA rules. *IEEE Transactions on Knowlegde and Data Engineering* **16**(8), 1010–1023.

Barták, R., Little, J., Manzano, O. & Sheahan, C. 2008. From enterprise models to scheduling models: bridging the gap. *Journal of Intelligent Manufacturing* **21**(1), 121–132.

Bouillet, E., Feblowitz, M., Liu, Z., Ranganathan, A. & Riabov, A. 2007. A knowledge engineering and planning framework based on OWL ontologies. In *Proceedings of ICKEPS 2007*. Providence, Rhode Island, USA.

Castillo, L., Fdez-Olivares, J., Garcia-Pérez, O. & Palao, F. 2006. Efficiently handling temporal knowledge in an HTN planner. In *Proceedings of the 16th ICAPS*, Long, D., Smith, S. F., Borrajo, D. & McCluskey, L. (eds). Cumbria, UK, AAAI Press, 63–72.

Castillo, L., Fdez-Olivares, J., García-Pérez, O., Palao, F. & González, A. 2007. Reducing the impact of AI Planning on end users. In *Workshop on Moving P&S Systems into the Real World (Keynote talk)*. Providence, Rhode Island, USA.

Eder, J., Gruber, W. & Pichler, H. 2006. Transforming workflow graphs. In *Interoperability of Enterprise Software and Applications*, Konstantas, D., Bourrières, J.-P., Léonard, M., Boudjlida, N. (eds.). Springer, 203–214.

Fdez-Olivares, J., Castillo, L., Cózar, J. A. & García-Pérez, O. 2008. Supporting clinical processes and decisions by hierarchical planning and scheduling. In *Proceedings of SPARK 08*. Sydney, Australia.

Fdez-Olivares, J., Garzón, T., Castillo, L., García-Pérez, O. & Palao, F. 2007. A Middleware for the automated composition and invocation of semantic web services based on HTN planning techniques. In *Proceedings of CAEPIA 2007*, Lecture Notes in Artificial Intelligence **4788**, 70–79. Springer.

Gagné, D. & Trudel, A. 2009. Time-BPMN. In *Proceedings of CEC 2009*, Hofreiter, B. & Werthner, H. (eds)., IEEE Computer Society, Washington, DC, USA, 361–367.

González-Ferrer, A., Fdez-Olivares, J. & Castillo, L. 2009. JABBAH: a Java application framework for the translation between BPM and HTN-PDDL. In *Proceedings of the 3rd ICKEPS Competition*, Bartak, R., Fratini, S. & McCluskey, L. (eds). Thessaloniki, Greece, 28–37.

Kiepuszewski, B., Ter Hofstede, A. & Bussler, C. J. 2000. On structured workflow modeling. In *Proceedings of CAiSE 2000*, Lecture Notes in Computer Science **1789**, 431–445. Springer.

Koehler, J. & Vanhatalo, J. 2007. *Process ainti-patterns: How to Avoid the Common Traps of Business Process Modeling*. IBM Report RZ3678, 1–40.

Koehler, J., Gschwind, T., Kuster, J., Volzer, H. & Zimmermann, O. 2008. *Towards a Compiler for Business-IT Systems—a Vision Statement Complemented with a Research Agenda*. IBM Report RZ3705.

Koehler, J., Gschwind, T., Wong, J., Favre, C., Kleinoeder, W., Maystrenko, A. & Muhidini, K. 2009. *IBM Pattern-based Process Model Accelerators for WebSphere Business Modeler: Patterns, Transformations and Refactoring*. IBM Report RZ 3738, 1–118.

Liu, R. & Kumar, A. 2005. An analysis and taxonomy of unstructured workflows. In *Proceedings of BPM 2005*, Lecture Notes in Computer Science **3649**, 268–284. Springer.

Lock Lee, L. 2005. Balancing business process with business practice for organizational advantage. *Journal of Knowledge Management* **9**(1), 29–41.

Long, D. & Fox, M. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* **20**, 61–124.

Muehlen, M. Z. & Ting-Yi Ho, D. 2006. Risk management in the BPM lifecycle. In *Business Process Management Workshops*, Lecture Notes in Computer Science **3812**, 454–466. Springer.

Muñoz-Avila, H., Gupta, K., Aha, D. W. & Nau, D. S. 2002. Knowledge based project planning. In *Knowledge Management and Organizational Memories'*, Dieng-Kuntz, R & Matta, N. (eds.). Kluwer Academic Publishers, 125–134.

Myers, K. & Berry, P. 1998. *Workflow Management Systems: An AI Perspective*. AIC-SRI report, 1–34.

Sadiq, W. & Orlowska, M. E. 1997. On correctness issues in conceptual modeling of workflows. In *Proceedings of the 5th European Conference on Information Systems*. Cork, Ireland, 943–964.

Simpson, R. M., Kitchin, D. E. & McCluskey, T. L. 2007. Planning domain definition using GIPO. *The Knowledge Engineering Review* **22**, 117–134.

Stein, S., Kühne, S. & Ivanov, K. 2008. Business to IT transformations revisited. In *Proceedings of the 1st International Workshop on Model-Driven Engineering for BPM*, Lecture Notes in Business Information Processing **17**, 176–187. Springer.

van der Aalst, W. M. P. 2003. *Patterns and XPDL: A critical Evaluation of the XML Process Definition Language*. QUT technical report FIT-TR-2003-06, 1–30.

van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B. & Barros, A. P. 2003. Workflow patterns. *Distributed and Parallel Databases* **14**(1), 5–51.

Vanhatalo, J., Volzer, H. & Koehler, J. 2009. The Refined process structure tree. *Data & Knowledge Engineering* **9**(68), 793–818.

Vanhatalo, J., Volzer, H., Leymann, F. & Moser, S. 2008. Automatic workflow graph refactoring and completion. In *Proceedings of the 6th International Conference. on Service-Oriented Computing*, Lecture Notes in Computer Science **5364**, 100–115. Springer.

Vaquero, T. S., Romero, V. M. C., Tonidandel, F. & Silva, J. R. 2007. itSIMPLE 2.0: an integrated tool for designing planning domains. In *Proceedings of the 17th ICAPS*, Boddy, M., Fox, M. & Thiébaux, S. (eds). Providence, Rhode Island, USA. AAAI Press, 336–343.

W3C 1999. *XML Path Language, v1.0*. Retrieved October 29, 2010, from `http://www.w3.org/TR/xpath`.

Workflow Management Coalition (WfMC) 2008. *XML Process Definition Language Specification, v2.1*. WFMC-TC-1025, 1–216.

Workflow Management Coalition (WfMC) 2010. *Adaptive Case Management*. Retrieved October 29, 2010, from `http://www.xpdl.org/nugen/p/adaptive-case-management/public.htm`.