JABBAH: A Java Application Framework for the Translation Between Business Process Models and HTN

Arturo González-Ferrer
Centro de Enseñanzas Virtuales
University of Granada
c/ Real de Cartuja 36-38, Spain 18071
arturogf@ugr.es

Juan Fernández-Olivares and **Luis Castillo**
Departamento de Ciencias de la Computación e IA
University of Granada
c/ Periodista Daniel Saucedo s/n, Spain 18071
{faro, l.castillo}@decsai.ugr.es

Abstract

HTN planning paradigm has been widely used during the last decade to model and solve planning and scheduling problems. Even so, little research have been oriented to represent and generate these planning domains automatically with the help of software tools. In this paper we present an extensible software framework directed to cover this goal, proposing an innovative knowledge engineering method that transform a workflow graph into an equivalent nested process model, which simplifies the subsequent mapping to HTN-PDDL. Some results in the field of e-learning management are also exposed.

1. Introduction

The difficulty of writing Planning and Scheduling (P&S) domains is well known by the AI community, and usually a lot of human effort is necessary to explore the real problems that are likely to be modeled, capturing the acquired knowledge with accuracy into a planning domain, that is usually coded using non-intuitive languages as PDDL (Long and Fox 2003) or any of its flavours. Despite being a difficult task, still little work has focused in helping to do it in a convenient way. However, this kind of problem is specially suitable for the Knowledge Engineering (KE) discipline.

Even though there are already some approaches (Simpson, Kitchin, and McCluskey 2007; Vaquero et al. 2007; Bouillet et al. 2007) devoted to the field of KE for P&S, they are rather directed to be helpful for planning experts (dealing with the modeling of world objects and actions). The approach here presented is more aligned with (Barták et al. 2008), and deals with the automatic generation of planning domains from expert knowledge introduced by using existing tools and standard languages that are close to IT architects and organization stakeholders.

Concretely, we propose in this paper the development of a software framework that is able to automatically map this acquired knowledge into P&S domain and problem definitions. Our work is focused on the reuse of Business Process Modeling (BPM) tools (Havey 2005). They are able to deal with goals and tasks specification, environmental analysis, design, implementation, enactment, monitoring and evaluation of business processes (Muehlen and Ho 2006).

Exploiting the common field between BPM and P&S is interesting, not only because we could use robust, formal and mature software tools to capture the knowledge we want to represent into the corresponding P&S domain (data, activities, rules, performers, etc), but also because we could reuse existing process models that have already been designed by software architects for a specific problem, offering P&S as a possible solution for a very wide range of application fields, taking as input a pre-existing process model. Moreover, introducing an automated P&S system into the BPM life cycle (Muehlen and Ho 2006) of a company, capable of both interpreting and reasoning about an initial workflow model representation, can provide support for decision making on key issues like tasks organization, resources allocation, or even requirement and use cases analysis.

The work here presented is based on the hypothesis that the process structure, the ordering constraints and the control flow structures of a BPM model, can be captured by an HTN knowledge representation language. Hence, we could use an state-of-art HTN planner that takes this domain representation as input and use its output in order to obtain action plans helpful for management tasks. This existing equivalence between both BPM and HTN made us consider the development of a software tool to carry on the transformation of one model into the other.

So, the contribution of this paper is that, having a BPM design of any organizational process, modeled under some previous requirements, we can extract the corresponding HTN planning domain and problem files directly from the original process diagram without the interaction from any planning expert. Moreover, this is done keeping the process control-flow restrictions as well as the data model reflected on it. Furthermore, some experiments have been carried on to support the organization and management of e-learning course development requests, allowing to check the usefulness of our approach.

The paper is structured as follows. Section 2 introduces some concepts and technical background about the problem. Section 3 details the Knowlegde Engineering procedure developed. Section 4 exposes some requirements on the input process diagram. Section 5 exposes some results and Section 6 describes some conclusions and lessons learned.

2. Technical Background

In this section we introduce the BPMN/XPDL business modeling languages, which has been chosen for our work. We also introduce *workflow patterns*, in order to convey why we decide to use them as the main background concept for our transformation, and which are also the basis for choosing the *Hierarchical Task Network* (HTN) paradigm to model the resulting planning domain.

2.1 BPMN/XPDL

XPDL stands for *XML Process Definition Language*. The goal of XPDL (WfMC 2008) is to store and exchange a process definition, offering an XML serialization of the *Business Process Management Notation* (BPMN) graphical representation of the process diagram. The main advantage of using XPDL as modeling language is that it is commonly used among business analysts, and it can be used to represent the organization activity easily. There are a lot of modeling tools that already incorporate XPDL natively or as an additional plug-in. Although some used directly BPEL (Business Process Execution Language) (García-Bañuelos 2008) to design the process diagram, ideally this should be done in XPDL, as it was thought for modeling, not for execution (Palmer 2007). Next, an overview of the XPDL entities and attributes considered in our work is exposed:

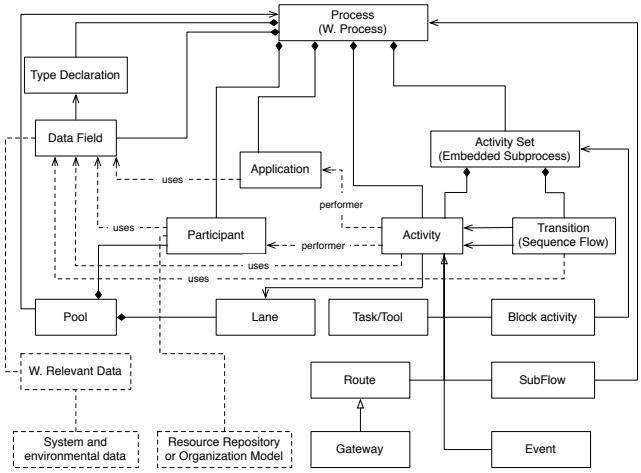

Figure 1: XPDL metamodel

Activities. They comprise a logical, self-contained unit of work, which will be carried out by *participants* and/or computer applications. Activities are related to one another via *transitions*.

Gateways. Special activities used to implement decisions that affect the sequence flow path through the process.

Transitions. Transitions may be either *conditional* (involving expressions which are evaluated, driving the sequence flow path) or *unconditional*, and may result in the sequential or parallel operation of individual activities.

Lanes. They denote areas or departments of the organization or process, and every activity contained within that particular lane will be done within that area. So, we can use them to establish a way to define capabilities that a particular activity requires. So, a participant also needs to belong to a specific lane, in order to be able to complete this activity.

Participants. They can be differentiated by their definition scope: a) Those defined at *process level* can be considered as possible resources that can be allocated to one or more activities. They will have associated precondition definitions, established by the lanes the participant belongs to. This membership can be specified as an *extendedAttribute* tag for every participant at modeling time. b) Those defined at *activity level* will force that specific activity to be done by the participant specified.

Parameters and DataFields. These entities are used to define the process data model. We can control flow from a gateway by creating a parameter (which has an associated data type) and values to be used in an associated rule. On a conditional transition exiting the gateway, we can specify that the transition will be followed only when parameter values match the expression specified in the rule. Information that is internal to the process is represented as *Data Fields* and information required outside the process is represented by *Parameters*.

XPDL modeling tool We have used TIBCO Business Studio, a modeling tool that use the BPMN graphical notation, is offered for free and includes support for latest release of XPDL v2.1.

Taking advantage of our expertise on e-learning management, we have analyzed a specific organisational process to manage the collaborative creation of e-learning courses within the virtual learning center of the University of Granada. This process implies the participation and interaction of different roles (instructional designers, graphic designers, HTML developers, sysadmins, tutors, etc.), and it has an explicit activity time ordering (see figure 2). The transformation of this process into a P&S domain will help to make the most of the e-learning center workload, also offering an estimation (to both managers and customers) about the time needed to deploy the requested course.

2.2 Workflow Patterns

Workflow Patterns (van der Aalst et al. 2003) are those generic structures found in a graph representation that capture frequently-used relationships between tasks in a process, and that are typically nested to form the whole process model. The XPDL language can represent some of them, although it lacks of some power for the correct representation of complex patterns (van der Aalst 2003). Therefore, only the most basic ones are going to be considered throughout the paper, those that can be well represented and are expressive enough for the definition of most processes: *serial*, *parallel split-join*, and *parallel exclusive-OR* (usually 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 to its corresponding HTN structure.

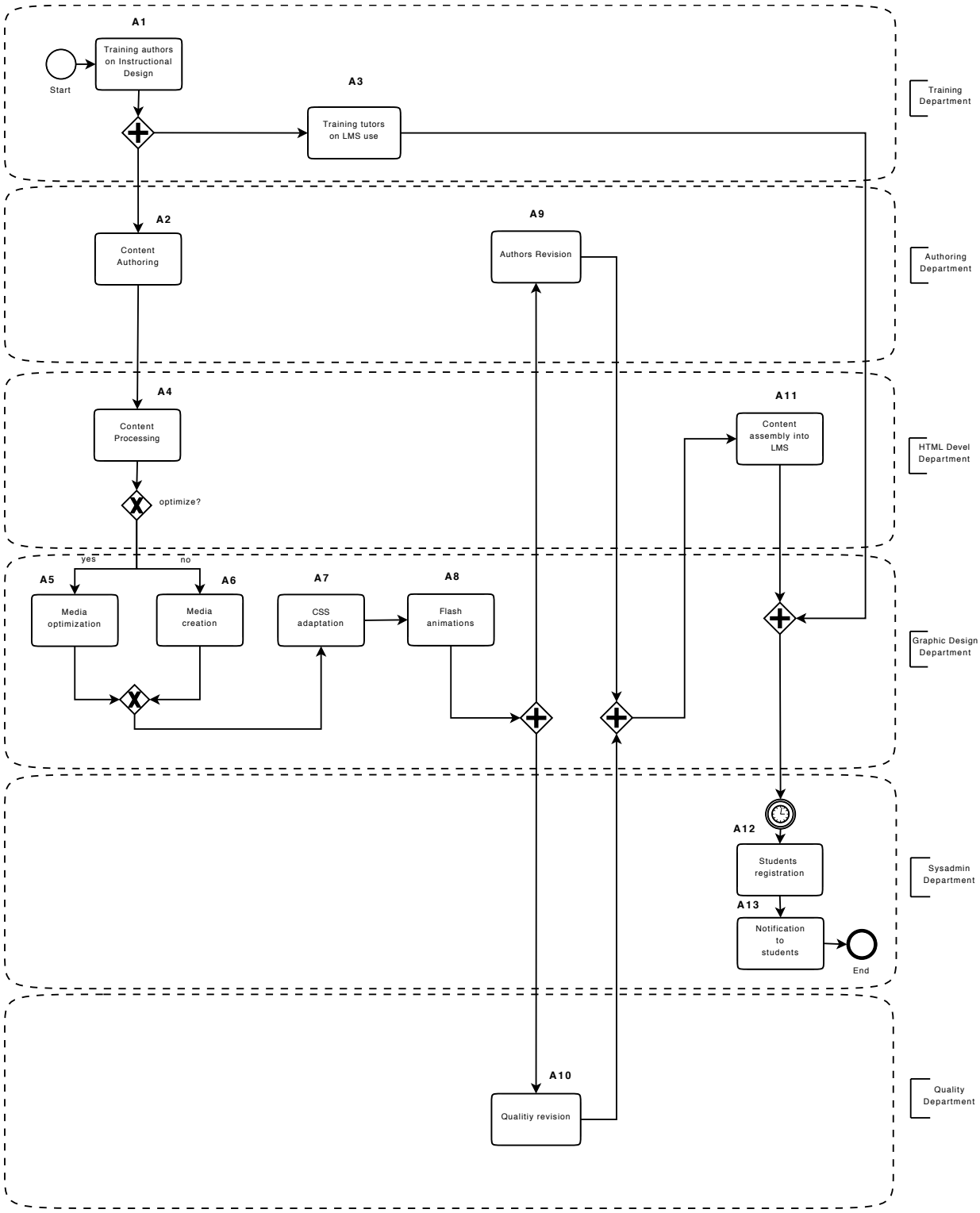

Figure 2: An example organisational process modeled using the BPMN graphical notation, describing the development of courses in a specific e-learning center. On the right side, text annotations boxes show the lane represented with point lines on the left; the boxes, named A1-A13, represent the activities; the arrows represent the transitions; the elements with the symbols 'x' and '+' are the exclusive-OR and parallel gateways; the participants are not represented graphically, and can only be explored within the BPM modelling tool. This model can be serialized as a XPDL stream, and later analyzed by JABBAH.

2.3 Hierarchical Task Network Planning

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 which 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. HTN paradigm is able to represent the hierarchical structure of the domain and it is also expressive enough to capture the expert knowledge in order to drive the planner to a desirable solution.

HTN-PDDL notation The HTN planning domain language used in this work is a hierarchical extension of PDDL (Long and Fox 2003) that uses the following notation.

Types, constants, predicates, *functions*, and *durative-actions* are used in the same way that in original PDDL language. In addition, the *task* element is introduced to express compound tasks. Its definition can include *parameters*, different decomposition *methods* with associated *preconditions* (that must hold in order to apply the decomposition method) as well as *tasks* to represent its corresponding lowest level task decomposition.

At the problem definition, *objects* is used to define objects that are present in the problem, *init conditions* to define the set of literals that are initially true, and *task-goals* to define 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 braces [T1,T2].

We have used the IACTIVE™ planner for this paper, as it is already known how to translate workflow patterns for semantic web services composition (J.Fernandez-Olivares et al. 2007), as well as its adaptation to temporal knowledge (Castillo et al. 2006). In addition, it has already been used in several applications (Castillo et al. 2007; Fdez-Olivares et al. 2008).

Next section describes the KE procedure needed to extract the P&S domain and problem from a process diagram.

3. Translation Overview

Roughly speaking, what we want to do is to identify common patterns in a workflow model (which can be clearly seen as a graph), so that we can generate a tree-like structure, much similar to HTN domains. This entails the resolution of two main problems: a) analyze the workflow model to get a corresponding graph, b) interpret the resulting graph, mapping it to a tree-like structure. To do this, a collateral challenge, out of AI Planning scope but necessary, is the transformation of the graph into a tree-like structure, which has been done using an algorithm described later at section 3.2.

So, our Knowledge Engineering proposal consists of three different stages (see figure 3) which are necessary in order to develop a sound approach for the problem of capturing knowledge from a BPM model that will finally be represented into an HTN planning domain:

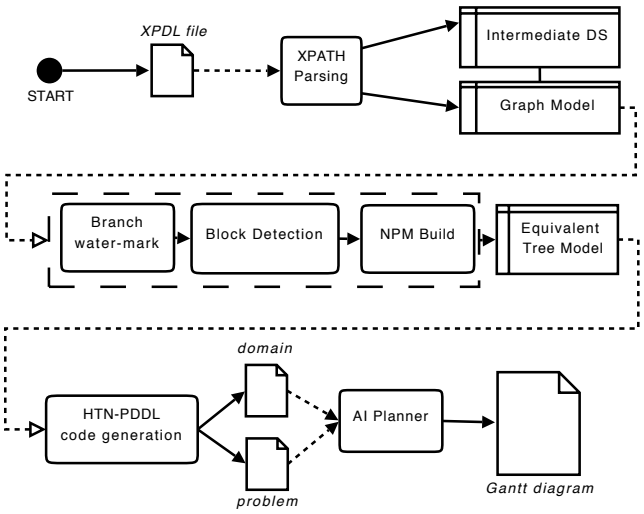

Figure 3: The different stages of the translation process

a) Firstly, we need to parse the source XPDL document, storing it into an intermediate data structure and graph model that can be easily managed throughout the next stages.

b) Then, we need to detect the different blocks of workflow patterns (parallel and serial blocks), distinguishing their kind from the knowledge acquired in the previous parsing stage, and build up an equivalent tree-like model. This is carried on by arranging those workflow patterns hierarchically, but also keeping the semantic information (about control flow and decisions) present in the process diagram.

c) Finally, we need to do a code generation phase, where we analyze the tree model that has been populated previously, trying to generalize common patterns found in the graph (i.e. serial or parallel split-joins patterns are always coded in the same way), and writting the HTN-PDDL code that corresponds to the tree-graph fragment analyzed.

Next, we proceed to give further insights on the development of these 3 steps.

3.1 Mapping to an Intermediate Graph Model

This step 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 obtains as result a graph 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, we will keep all the relevant information about participants, lanes, parameters, etc. by using an associated data structure that will be used throughout the mapping process.

It's important to note that both *gateways* and *transitions* elements are the main elements that drive the control flow in XPDL workflow graphs. They are also the main elements considered for our work and, from a workflow patterns perspective, they will define how to map organizational processes into planning and scheduling domains, so that the future plans developed by our software framework will mainly act according to their definition in the process diagram.

At this point, we have developed a graph model of the original process diagram that can be further worked out in order to achieve our goal.

3.2 Block Detection: Mapping to a Tree Model

The goal of this stage is to build an equivalent tree model from the graph obtained in the previous phase. Our work for 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 later used to derive ECA (event-condition-action) rules, helpful for controling the workflow execution.

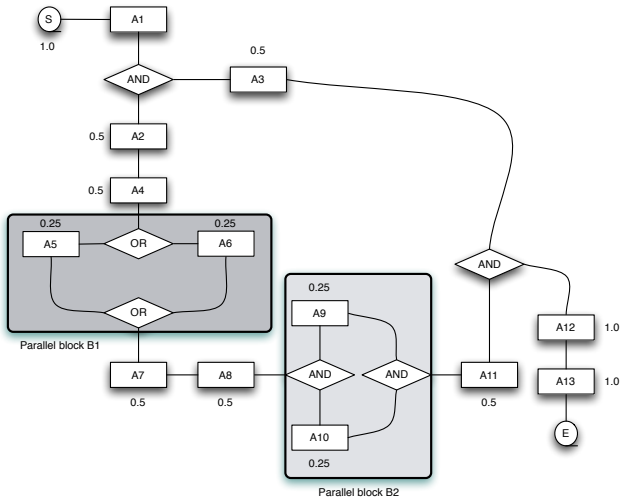

Figure 4: Part of the block detection algorithm applied to graph of figure 2. We can appreciate the branch-water mark procedure as well as the workflow pattern detection.

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 which is decomposed into a set of subprocesses and tasks, and so on. It adopts and generalizes a hierarchical model, allowing to express a parent-child relationship between subprocesses. We adapted this tree-like model to our particular problem, the representation of P&S domains, taking into account the control-flow information included in *gateways* and *transitions*, as expressed before, adding aditional information about the process and data model as well. Thus, the algorithm for block detection described has the next three steps:

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 is the method used to identify the most inner block in the graph, which is important for the next steps. It 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 (we call them *blocks* here) consecutively, using the weight to identify the most inner block. Every time we identify a serial or parallel block, we substitute all the nodes that constitutes that block with a special SERIAL_BLOCK (SB) node or PARALLEL_BLOCK (PB) node, obviously linking the new node with the preceeding and succesors nodes, in order to keep the graph being *connected* and *two-terminal* (it has an unique start and end node). If the workflow graph fulfills the requirements commented before, it is easy to see that this process ends having an unique SB or PB block node that constitutes the root node for the nested process model we want to build up.

3. Finally, if the root node is now expanded using the nodes it grouped originally, placing them as children, and we do this operation recursively with every SB or PB block node, we can draw the new tree-like structure that we were searching for. This is done as a typical breadth-first search algorithm. The result of the procedure constitutes what we called the *nested process model* (NPM) of the original BPM diagram, using a bottom-up approach (see figure 5). Observe that those nodes with mininum weight lay at the lower levels, and go up consecuently as their weight increase (this is the reason to look first for the most-inner blocks).

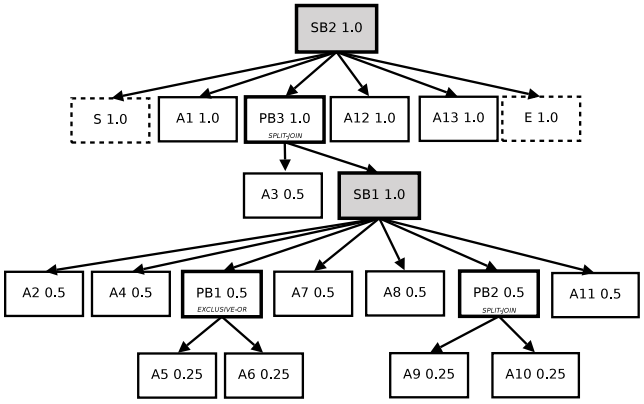

Figure 5: An example of a Nested Process Model generated from the previous BPMN process model. Note that leaf nodes correspond to activities and non-leaf nodes correspond to serial and parallel blocks

There are some considerations we must stress on. Firstly, we need to keep all the knowledge acquired in the parsing stage (lanes, participants, parameters and its association to gateways, etc.), being important to have custom implementations of graph nodes and edges. Second, we must keep the nodes that gave rise to the new special block nodes, introduced in the second step. And last, the knowledge present in gateways must be transferred to the new parallel block nodes (i.e., we must transfer the type of gateway, the parameters/rules that drives the flow, etc...), as soon as those gateways nodes are not going to be present on the new 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 (Bae et al. 2004).

3.3 HTN-PDDL Code Generation

Now we give specific details about how we generate the HTN planning domain and problem files, taking as basis both the tree-like structure (the NPM, figure 5) and intermediate data structures, already developed in the previous phases.

As 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. It is clear to see that, 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 6).

Next, we expose how to express the different elements of an HTN-PDDL domain and file 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 that are going to be useful in any planning domain: *activity*, *participant* and *lane*. Of course, parameters data types must be also generated (see the corresponding item below).

Constants. XPDL *activities* and *lanes* will be mapped as HTN-PDDL constants, which are going to be used later throughout the domain and problem files. This is automatically extracted from the intermediate data structure obtained in section 3.1, and they will be coded in lowercase characters (i.e. activities will be coded as a_x, being x the activity id).

Predicates. We must include, at least, two default predicates, 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, in other words, what abilities correspond to every participant. It will be used to encode both initial conditions of the problem (one predicate instance for every ability a participant posess) and preconditions for the durative actions (a precondition for every activity within a 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 next pattern:

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

For every *k*, being *k* an activity of the NPM, a corresponding durative action *Ax* is generated, being *x* the id number, whose effect is the completion of the activity *ax* (which was coded as a constant previously, and the associated predicate named *completed*). The duration of the activity, D, which is coded in XPDL using an *extendedAttribute* tag, and the lane the activity belongs to, L, are mapped directly from the corresponding XML attributes present on the XPDL activity.

Realize 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.

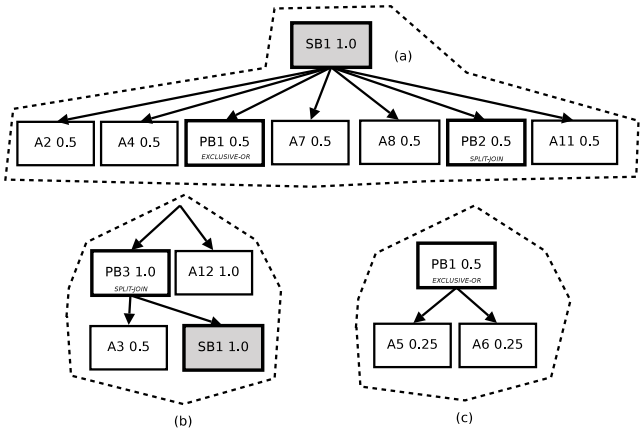

Figure 6: Different patterns identified in the NPM representation that are mapped as HTN compound tasks. (a) a serial block, (b) a split-join block, (c) a exclusive-OR block

Compound Tasks. The HTN-PDDL compound tasks are mapped from those intermediate nodes (non leaf-nodes) of the Nested Process Model. These nodes always correspond to workflow pattern blocks (see figure 6), that are actually specifications of different tasks with control flow mecanisms 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. Next example represents the fragment of figures 6(a) and 7:

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

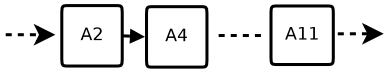

Figure 7: a serial block fragment

Note that, on one hand, durative actions A_x must be generated with the corresponding parameter $?w_y$ which express a resource that has to be allocated at planning-time (the participant *y* is assigned the activity *x*). On the other hand, compound tasks that are also part of the decomposition can be generated with or without parameter, representing the formal *parameter* which drives the flow in the original XPDL process diagram (i.e. the parameter 'optimize' in the example above controls which flow to follow, as exposed next).

2. ***Parallel Split-Join Blocks***. They represent a branch of the process flow into two or more flows (*split*) that are carried on 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, that represents the fragment of figures 6(b) and 8:

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

Note that A12, the *right brother node* of PB3 in figure 6(b), is the activity executed after the *join* gateway. This scheme repeats for every parallel split-join block detected in the nested process model.

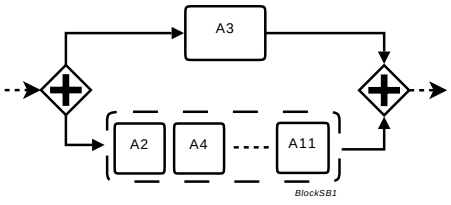

Figure 8: a parallel split-join block fragment

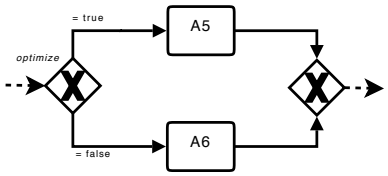

Figure 9: a parallel exclusive-OR block fragment

3. ***Parallel Exclusive-OR Blocks***. They represent blocks which flow is controlled by a gateway node which has associated both a formal parameter and a corresponding logical expression that controls which alternative flow must be followed. We generate a method for every possible alternative to follow, using the expression as precondition of the defined method, as the next example, that represents the fragment of figures 6(c) and 9:

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

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

It's clear that we should also map the parameters and expressions in such a way that different kind of parameters/expressions pairs and its associated data types can be added to the framework in a future. We have already done it for boolean data type, as described next.

Parameters. Parameters are usually associated to Exclusive-OR parallel blocks, and they can be initially expressed as follows, as soon 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).

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).

Other data types could be included using a similar methodology, but adding more powerful rule expressions (step c) is still one of the features to be improved in the JABBAH framework. Besides this mapping, we also tried referring to an external organizational data model stored in UML, using some of the capabilities of the BPM modeller, as the XPDL standard supposedly supports it, but this feature was somehow experimental in the modeller 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) that 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* described before). **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.

So 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. In the next section, we introduce some restrictions on the input process model, necessary to guarantee the correctness of our solution.

4. Input process model requirements

For the sake of the framework usability, we need to establish some requirements on the input process model, owing to the fact that not always the designed diagrams have the desired properties for later processing (any BPM expert knows about this circumstance). Thus, we have considered the next three conditions on the input process model:

a) The input process model must include an unique start node *s* and an unique end node *e*. Extrapolating this property to the equivalent graph model, it must be *two-terminal*.

b) All the gateway nodes that split the flow in the input process model, must have a corresponding gateway node that joins the flow again. Extrapolating this property to the equivalent graph model, it must be *well-structured*.

c) The input process model must be connected between elements from start to end nodes, so that for every node, always exists a path from *s* to *e* that goes through that node. Extrapolating this property to the equivalent graph model, it must be *directed* and *connected*.

The proposed requirements are demanded in order to guarantee that the workflow pattern detection stage is carried on correctly (b), as well as the branch water-mark procedure included in that stage (a, c). Some inspiring works for the establishment of these requirements have been the SESE (single-entry single-exit) regions of a graph (García-Bañuelos 2008), and the *process structure tree* (Vanhatalo, Volzer, and Koehler 2009). It seems natural to delimit the sort of process models that can be worked out, as usually done in other research related to BPM (van der Aalst 1999).

Maybe, the most demanding requirement for the user are (a, b), but fortunately some transformations have been already developed in order to obtain *well-structured* graphs from unstructured ones (Vanhatalo et al. 2008) (which also eliminates the need for an unique end node).These transformations could be introduced either into BPM modeling tools or into the JABBAH framework itself, in order to increase the number of models it can analyze, being unnecessary to modify the diagram manually. Although no validation checking about the input has been developed yet in our software tool, we would like to include it in a near future, so that it can be helpful to the IT architects during the design process by giving tips at design-time, similarly to any other simulation engine included within traditional BPM tools.

5. Experiments

The JABBAH framework described above has been developed following an object-oriented methodology. It is based on the Java graph library *jgrapht*, which provides a complete and customizable library, covering to a large extent our needs for creating graph data structures, with fully customized nodes and edges implementation. Furthermore, its corresponding visualization libraries, *jgraph* and *jgraphlayout* help us to develop a visual interface, allowing to see the original graph extracted from the source XPDL document as well as its corresponding NPM.

We have done some experiments by using JABBAH over the process shown at figure 2. It represents the whole process 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, we want to assign an activity to every worker, so that we can have a plan over time that tells the e-learning managers information about the workers allocation as well as the end time of the whole course development, which allows to do an anticipated decision-making upon the course request.

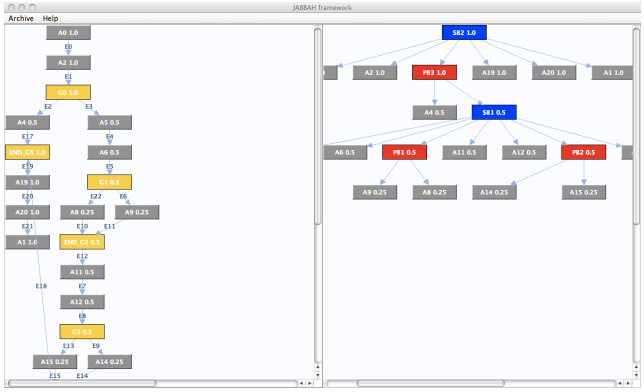

Figure 10: Screenshot of JABBAH framework running over the example process

The first thing to do is to design the process diagram (figure 2) with the help of the TIBCO Business Studio process modeler. The only inputs the managers have to provide once the process model has been designed are a) the estimated duration of every activity and b) the abilities every specific worker posesses. Both requirements are usually known by managers. Then, we export the diagram as an XPDL file, and import that file using JABBAH, which will show both the process diagram (on the left) and the corresponding nested process model (on the right), as shown in figure 10. Figures 4, 5 and 6 described the first two stages of our KE procedure applied to the mentioned process, and how the NPM is built up. At the same time, the HTN-PDDL

translation process is carried on over the NPM, saving the domain and problem files in an output directory. We can then interpret those files by using the IACTIVE™ planner, and get the corresponding plan.

Using different abilities assignment for real workers at the e-learning center, as well as estimated activities duration, we have checked the viability of the output plans. An example assignment as the following: Emilio (*training*), Storre (*authoring*), Miguel (*html*), JoseBa (*graphic*), Arturo (*admin*) and FMoreno (*quality*), would result on the output plan shown as a Gantt diagram at figure 11.

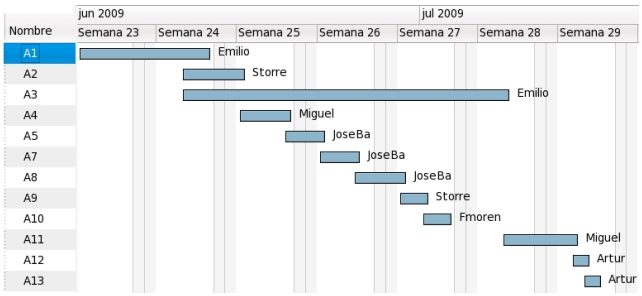

Figure 11: Output plan as a Gantt diagram

So, we end up checking that the KE procedure developed is useful within this particular scenario, and it can be extrapolated to multiple scenarios, as soon as the process diagram is represented in the terms specified in this paper.

6. Conclusions and Lessons Learned

This paper has made some innovative contributions in order to overcome the traditional drawbacks of P&S modeling, specifically for the HTN paradigm. Mainly, a sound KE procedure has been developed in order to express BPM process diagrams as HTN P&S domain, building up an intermediate data structure that organises the source process diagram as a nested process model, simplifying the subsequent transformation into HTN-PDDL code. This is very useful, not only by the number of application areas it can give support to, but also because most of the processes are being modeled with BPM tools, what increases the usability of JABBAH.

What's more, our paper hints at a future direction to follow on the automatic capture and generation of HTN planning problems from organizational processes, and it could give rise to any other profitable approaches, at least in the area of Enterprise Resource Planning (ERP) and Supply Chain Management (SCM), which has already been part of previous research in P&S (Pardoe and Stone 2006). Some results were obtained in the field of e-learning management, which was very useful and interesting for the IT personal at the e-learning center of the University of Granada. What's more, JABBAH can fill some existing gaps in BPM tools, as exposed next.

6.1 Business Prospects

Understanding and estimating the time and cost to complete a product development process is a key business challenge. Typically, managers have relied in project management tools (PMT) for planning purposes, but the interdependencies between time and resource constraints make it very difficult to analyze activity costs and resource requirements using traditional PMTs . The introduction of computed-aided Business Process Simulation (BPS) tools traditionally helped to capture the resource constraints, decisions rules and stochastic behaviour of real situations. But, while the strength of BPS tools relay in their ability to incorporate stochastic situations in the model, their use imply that, to find the best resource allocation scenario, the manager has to determine various scenarios and simulate them (Tumay 1996). First, this is not very realistic, as the simulation relays on subruns for a specific scenario which is usually not repeatable, as the constraints evolve in time. Second, BPS tools are based on trial-and-error mecanisms that don't help the manager to do the correct allocation of resources to activities. Sometimes this circumstance can be a serious problem as the constraints get harder, making difficult to find a correct assignment.

It's important to note that the resource allocation feature is still a requirement to be improved in BPM/BPS tools (Castellanos et al. 2006). The JABBAH framework directly tackle both BPS inconvenients exposed above. On the one hand, JABBAH would be used every time that a new order request was received, being more realistic, as it would evaluate the existing constraints at that specific moment. On the other hand, the traditional trial-and-error mecanism will be replaced by JABBAH, as it helps the manager to decide a good scenario, while keeping all the constraints defined in the process.

Furthermore, the results obtained by the JABBAH framework could be incorporated back into the BPM simulation engine (usually, the simulation scenario is expressed using an XPDL extension), so that the manager could simulate the process with the obtained assignment. Hence, after some executions, workflow mining tools could be used to investigate how much the processes have improved by using the new tecnique (in terms of resources under-utilization or over-utilization, in terms of production and benefits, etc).

JABBAH could be useful at project-based and customer service-based processes, that is, processes where there are customers which ask for a product which, after the corresponding development process carried on by a collaborative teamwork made up of different humans, departments or roles (and why not, software applications or web services), is finally supplied to the customer. This could be expanded to other kind of projects as long as we improve the representation of the data model, which is one of our future challenges.

6.2 Future Work

The expression of temporal dependencies is surprisingly poorly addressed by the different BPM standards, and can be very difficult to introduce not only on the modeling side, but also in the enactment side. A specific extension called Time-BPMN (Gagné and Trudel 2009) has been created recently to, on the one hand, simplify the temporal constructs of the original BPMN, and on the other hand, allow the specification of temporal constructs that were not possible in the original BPMN. Specifically, the temporal constructs

of time points, intervals/durations, temporal constraints and temporal dependencies have been considered in this extension, which is based on Allen's interval algebra. As exposed in (Castillo et al. 2006), the HTN-PDDL extension used in JABBAH is able to correctly represent these temporal constructs. So, it seems reasonable that the next step of our work will be the consideration of this extension, and the automatic translation into HTN-PDDL of the temporal constructs that can be depicted through Time-BPMN, so that our tool acquire a better capability to express other complex scenarios.

Furthermore, we must emphasize that, though we used XPDL in our work, the JABBAH framework has been developed with the idea of *extensibility* in mind, taking into consideration future growth. That means that if we would like to use the next BPMN 2.0 specification, that supposedly will include an XML serialization itself, we could implement a different parsing method, keeping the same intermediate data structure populated correctly, as exposed previously at section 3.1. Similarly, the block detection algorithm used (Bae et al. 2004), could be substituted by other similar approaches. We would like to add the RPST (Vanhatalo, Volzer, and Koehler 2009), in order to improve the efficiency of the block detection method (*O(n)*), checking also its behaviour for P&S domain generation. Last but not least, since we used a language independent tree-like output model, we could introduce a plug-in for any differen planning language, as long as they respect the HTN paradigm.

This would provide us a sandbox environment where we could test different techniques, measuring how they behave and how far will their capacity of expression go, in terms of P&S modeling. The JABBAH framework has just started and hopefully it can be tested over some other different process models, so that we can enrich its design, which still needs stressing on the improvement of the process data model, as soon as the process model and control-flow perspective have been the ones that got an intense dedication at this early stage of development.

References

Bae, J.; Bae, H.; Kang, S.; and Y.Kim. 2004. "Automatic Control of Workflow Processes Using ECA Rules". *IEEE Transactions on Knowlegde and Data Engineering* 16(8).

Barták, R.; Little, J.; Manzano, O.; and Sheahan, C. 2008. "From enterprise models to scheduling models: bridging the gap". *Journal of Intelligent Manufacturing*.

Bouillet, E.; Feblowitz, M.; Liu, Z.; Ranganathan, A.; and Riabov, A. 2007. "A Knowledge Engineering and Planning Framework based on OWL Ontologies". In *ICKEPS 2007*.

Castellanos, M.; Casati, F.; Sayal, M.; and U.Dayal. 2006. *LNCS 3811*. Springer. chapter "Challenges in Business Process Analysis and Optimization", 1–10.

Castillo, L.; Fdez-Olivares, J.; García-Pérez, O.; and Palao, F. 2006. "Efficiently handling temporal knowledge in an HTN planner". In *Proceedings of 16th ICAPS*, 63–72.

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

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

Gagné, D., and Trudel, A. 2009. "Time-BPMN". In *Proceedings of 1st International Workshop on BPMN*.

García-Bañuelos, L. 2008. "Pattern Identification and Classification in the Translation from BPMN to BPEL". In *Proceedings of OTM 2008*, 436–444. Springer.

Havey, M. 2005. *"Essential Business Process Modeling"*. O'Reilly.

J.Fernandez-Olivares; Garzón, T.; Castillo, L.; O.García-Pérez; and Palao, F. 2007. "A Middleware for the automated composition and invocation of semantic web services based on HTN planning techniques". In *LNAI*, volume 4788, 70–79. Springer.

Long, D., and Fox, M. 2003. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". *Journal of Artificial Intelligence Research* 20:61–124.

Muehlen, M., and Ho, D. T.-Y. 2006. *Business Process Management Workshops, LNCS 3812*. Springer. chapter "Risk Management in the BPM Lifecycle", 454–466.

Palmer, N. 2007. *BPM and Workflow Handbook*. Workflow Management Coallition. chapter "Workflow and BPM in 2007: Business Process standards see a new global imperative", 9–14.

Pardoe, D., and Stone, P. 2006. "Predictive Planning for Supply Chain Management". In *Proceedings of ICAPS*.

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

Tumay, K. 1996. "Business Process Simulation". In *Proceedings of Winter Simulation Conference*, 93–98.

van der Aalst, W.; ter Hofstede, A.; Kiepuszewski, B.; and Barros, A. 2003. "Workflow Patterns". *Distributed and Parallel Databases* 14(1):5–51.

van der Aalst, W. 1999. "Formalization and Verification of Event-Driven Process Chains". *Information and Software Technology* 41(3):639–650.

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

Vanhatalo, J.; Volzer, H.; Leymann, F.; and Moser, S. 2008. "Automatic Workflow Graph Refactoring and Completion". In *LNCS*, volume 5364. Springer. 100–115.

Vanhatalo, J.; Volzer, H.; and Koehler, J. 2009. "The Refined Process Structure Tree". *Data & Knowledge Engineering* 9(68):793–818.

Vaquero, T.; Romero, V.; Tonidandel, F.; and Silva, J. 2007. "itSIMPLE 2.0: An Integrated Tool for Designing Planning Domains". In *Proceedings of 17th ICAPS*.

W3C. 1999. "XML Path Language, v1.0". *http://www.w3.org/TR/xpath*.

WfMC. 2008. "XML Process Definition Language Specification, v2.1". *WFMC-TC-1025* 1–216.