A PLANNING AND SCHEDULING PERSPECTIVE FOR DESIGNING BUSINESS PROCESSES FROM DECLARATIVE SPECIFICATIONS

Irene Barba and Carmelo Del Valle
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla, Avda Reina Mercedes s/n, 41012, Seville, Spain

Keywords: Planning, Scheduling, Business Processes Management.

Abstract: Usually, business process models are manually achieved by business analysts and most of current modelling languages are of imperative nature. As a consequence, non-optimized or faulty models can be obtained. This work proposes a planning based approach to give business analysts assistance for the process models generation. This approach entails the selection and the order of the activities to be executed (planning), and the resources allocation involving temporal reasoning (scheduling), both considering function optimization. The process information is specified in a declarative way, that is translated into the standard planning language PDDL. A friendly graphic language is used (ConDec-R, an extension of ConDec).

1 INTRODUCTION

In the past few years, most of the organizations need to adapt to the new commercial conditions as well as to respond to competitive pressures, so there exists an increasing interest in the effective management of business processes (BP). BP Management (BPM) supports BP using methods, techniques, and software to design, enact, control and analyze processes involving humans, organizations, documents and other sources of information (van der Aalst et al., 2003).

Scheduling problems (Brucker and Knust, 2006) entails the suitable generation of execution plans for a set of tasks related by temporal and resource constraints, optimizing some functions. In a wider perspective, in Artificial Intelligence (AI) planning (Ghallab et al., 2004), the tasks to be executed are not established a priori, so it is necessary to select a suitable set of actions that must be executed in a correct order, generally optimizing some objectives. In the past years, there is an increasing interest in the application of AI P&S techniques to automate the production and execution of BP (Kearney et al., 2003; González-Ferrer et al., 2009; Barba and Del Valle, 2010).

In BPM systems, in general, a user specifies the model through a modelling language, such as BPMN (White and et al., 2004). In order to design a suitable model, the user must deal with several aspects, such as the resource allocation, the tasks properties or the relations between them, possibly optimizing some functions. In most cases, the BP information is provided to the system through imperative modelling languages. In this work, it is proposed a declarative language for the BP information. The work (Fahland et al., 2010) analyzes the differences between imperative and declarative process modelling languages with respect to build-time modifications (maintainability).

Several works concerning BP based on Linear Temporal Logic, LTL (Clarke Jr. et al., 1999), can be found. (Pesic and van der Aalst, 2006; van der Aalst and Pesic, 2006) propose a graphic tool for modelling the processes through some templates that can be translated to LTL formulas. ConDec (Pesic and van der Aalst, 2006) is a declarative language to specify dynamic BP models using a graphical notation which can be mapped to formulas in LTL. In this work, an extension of ConDec, named ConDec-R (Sect. 2), has been defined since ConDec does not allow reasoning about resources directly.

The Planning Domain Definition Language (PDDL) (Ghallab and et al., 1998) is a (standard) language for the specification of planning problems and solutions, so that any generic planner that supports PDDL is capable to solve a wide scope of problems of different nature specified through this language. PDDL 2.2 (Hoffmann and Edelkamp, 2005) also allows handling of numeric values, durative actions, plan objective functions, derived predicates and timed initial literals.

In Fig. 1, a graphical representation of the current

Barba I. and Del Valle C..
A PLANNING AND SCHEDULING PERSPECTIVE FOR DESIGNING BUSINESS PROCESSES FROM DECLARATIVE SPECIFICATIONS.
DOI: 10.5220/0003149005620569
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 562-569
ISBN: 978-989-8425-40-9

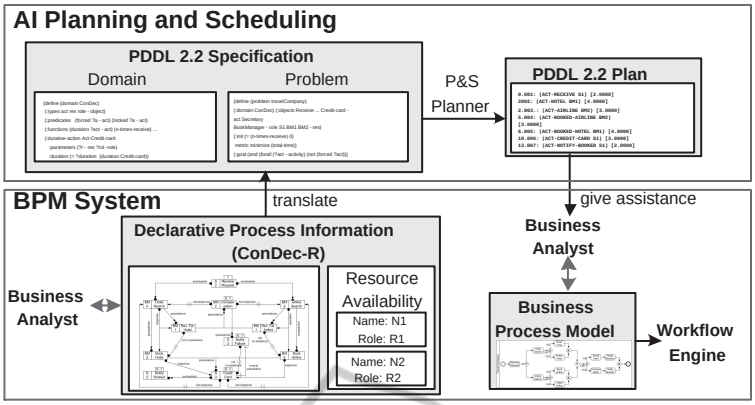

Figure 1: An AI-based approach for the generation of business process models.

approach is shown. First, the business analyst provides the declarative process information through a ConDec-R specification, that is translated to a PDDL 2.2 specification to be used as input of a planner for obtaining a feasible optimized BP execution plan. Lastly, this PDDL 2.2 plan gives the business analyst assistance for the BP model generation. The main contributions of this paper can be summarized as:

- An application of an AI-based approach for solving the planning and scheduling of the tasks involved in the BP model through automatic planners, in order to generate BP execution plans. This approach considers task properties, resources allocation and the optimization of some functions.
- A translation from a formal and widely used language (LTL) to PDDL 2.2 of several templates related to BP definitions.
- An extension of a graphic declarative language for BP, including resources treatment.

The paper is organized as follows: Sect. 2 explains the proposed declarative language, Sect. 3 details the translation from the declarative specification to PDDL 2.2, Sect. 4 shows a case of study, and Sect. 5 presents some conclusions and future work.

2 DECLARATIVE SPECIFICATION OF BP

In this work, the user must provide the process information to the system in a declarative way. In order to do this, an extension of ConDec (ConDec-R) is used, including information about the resources required for the execution of the activities. ConDec (Pesic and van der Aalst, 2006) is a graphical language based on declarative specifications for modelling and enacting dynamic BP. For the definition of relationships between activities, ConDec proposes an open set of constraints based on LTL. One important difference when modelling with ConDec is that a ConDec-activity represents multiple executions of a P&S-activity, so that a ConDec-activity can be executed several times.

The main contribution of ConDec-R regarding to ConDec, is the reasoning about resources. Unlike ConDec, in ConDec-R the activities execution requires resources of a specific role, and there are several resources with the competences defined by a role. This information can be easily added to ConDec. A ConDec-R problem specification must include (an example can be seen in Sect. 4):

- As in ConDec, the tasks that can be executed in the BP enactment. In some cases, constraints about the number of times that one activity must be executed are specified (it appears above the associated activity in the graphical representation).
- The role of the required resource and the estimated duration of each task, which are specified on the left side of the task (extension of ConDec).
- Available resources with the competences of a role (extension of ConDec).
- As in ConDec, the relations between activities through constraint templates (parameterized graphical representations of LTL formulas).

3 TRANSFORMATION FROM CONDEC-R TO PDDL 2.2

PDDL (Ghallab and et al., 1998) is a (standard) language for the specification of planning problems and solutions, so that any generic planner that supports PDDL is capable to solve a wide scope of problems

of different nature specified through this language. PDDL 2.2 (Hoffmann and Edelkamp, 2005) specifications include a domain file and a problem file.

3.1 Domain Description

A PDDL 2.2 domain contains the following items:

Predicates: They represent the properties of objects that can be true or false. Several predicates have been considered for the BP execution plan generation (Table 1). Different predicates for the temporal locks of the activities are used in order to differentiate between the reasons of the lock.

Functions (Fluents): They represent values, that can vary over time, associated to objects, allowing handling of numeric values. We have used two kind of fluents: `(duration ?act)`, that is the constant duration of the activity `act`; and `(n-times-activity)` that represents the number of times that an activity has been executed until the current state.

Actions/Operators (Durative): They allow the evolution of the system by means of state changes. For the BP execution plan generation, there exists one durative action representing the execution of each activity. In the proposed approach, there exists a PDDL 2.2 durative action associated to each ConDec-R activity, since both are durative and, also, can be executed several times. Furthermore, all the actions have a base specification (Fig. 2) that is extended through preconditions and effects depending on the relations in which the concerning activity is involved (Sect. 3.3). For the execution of an activity (Fig. 2), two conditions `(:condition)` must be satisfied: it must not be locked for any reason, and there must exist a free resource with the role required by the activity. The predicates `forced` and `locked` are used in order to ensure the feasibility of the constraints established by the ConDec-R templates. After the activity execution, some effects `(:effect)` are given: the `n-times` function is increased; the activity becomes not forced to execute; and the required resource becomes busy (only) during the activity execution.

3.2 Problem Description

A PDDL 2.2 problem contains the following items:

Objects: They represent the things in the world that are noteworthy for the specified problem. In the current proposal, three kinds of objects can be distinguished: `activity`, `resource` and `role`.

Initial State: Initially, all the resources are considered free and the number of times one activity has been executed is 0 (fluent `(n-times-activity)`). Also, it is necessary to include the predicates `(role Resource Role)` and `(resource Activity Role)` for the related objects that present these relations, together with the corresponding value for the fluent `(duration Activity)`.

```
(:durative-action Activity
    :parameters (?r - res ?rol - role)
    :duration (= ?duration  (duration Activity))
    :condition (and (at start (not (locked Activity)))
                    (at start (and (free ?r)
                                   (resources Activity ?rol)
                                   (role ?r ?rol))))
    :effect     (and (at end (done Activity))
                     (at end (increase (n-times-activity) 1))
                     (at end (not (forced Activity)))
                     (at start (not (free ?r)))
                     (at end (free ?r))))
```

Figure 2: PDDL 2.2 base specification for the `Activity` actions.

Goal: Things that must be true at the end of the plan. There is a base goal specification, that is extended depending on the relations between the activities (Sect. 3.3). The goal is reached when there are no activities to be executed (`(:goal (and (forall (?act - activity) (not (forced ?act)))))`).

Objective Function: Plan quality measures (metrics). In the current proposal, the minimization of the total time of the plan is pursued: `(:metric minimize (total-time))`.

3.3 Transformation from ConDec-R Templates to PDDL 2.2

ConDec-R considers the same templates than ConDec (van der Aalst and Pesic, 2006). For a specific problem, the relations between the activities can extend the base durative action specification (Fig. 2); and the base goal for the problem specification. As follows, the considered relations are described, together with the effect they have in the PDDL 2.2 specification:

I) Existence constraints:

1. EXISTENCE_N(A): `A` must be executed more or equal than `N` times. The goal `(>= (n-times-A) N)` is added.
2. ABSENCE_N(A): `A` must be executed less than `N` times. The goal `(< (n-times-A) N)` is added.
3. EXACTLY_N(A): `A` must be executed `N` times exactly. The goal `(= (n-times-A) N)` is added.

II) Relation constraints:

1. RESPONDED EXISTENCE(A,B): If `A` is executed, then `B` also must be execu-

Table 1: Predicates for the automatic generation of optimized BP execution plans.

Predicate	Description
(forced ?act - activity)	The activity act has to be executed before the end of the plan.
(locked-temp ?act - activity)	act is temporarily locked to be executed due to alternate and responded absence relations (Sect. 3.3).
(locked-chain ?act - activity)	act is temporarily locked to be executed due to chain relations (Sect. 3.3).
(locked-perm ?act - activity)	act can not be executed anymore.
(locked ?act - activity)	Derived predicate that is defined as the disjunction of locked-temp , chain and perm .
(resources ?act - activity ?ro - role)	act requires a resource with the role ro to be executed.
(role ?r - resource ?ro - role)	There exists a resource r that presents the role ro.
(free ?r - resource)	The resource r is free.
(done ?act - activity)	The activity act has been executed.

ted. The goal (or (= (n-times-A) 0) (> (n-times-B) 0)) is added.

2. CO-EXISTENCE(A,B): The execution of A forces the execution of B, and vice versa. The goal (= (> (n-times-A) 0) (> (n-times-B) 0)) is added.
3. RESPONSE(A,B): After the execution of A, B must be executed always. It leads to add an effect to the durative-action associated to A: (at end (forced B)) .
4. PRECEDENCE(A,B): Before B, A must have been executed. It leads to add a condition to the action associated to B: (at start (done A)) .
5. SUCCESSION(A,B): Relations Response(A,B) and Precedence(A,B) must hold.
6. ALTERNATE RESPONSE(A,B): After the execution of A, B must be executed, and between each two executions of A, there must be at least one execution of B. It leads to add two effects to A: (at end (forced B)) and (at start (locked-temp A)) ; and one effect to B: (at end (not (locked-temp A))) .
7. ALTERNATE PRECEDENCE(A,B): Before the execution of B, A must have been executed, and between each two executions of B, A must be executed. It leads to add: a condition to B: (at start (done A)) ; an effect to B: (at start (locked-temp B)) ; and an effect to A: (at end (not (locked-temp B))) .
8. ALTERNATE SUCCESSION(A,B): Relations Alternate Response(A, B) and Alternate Precedence(A,B) must hold.
9. CHAIN RESPONSE(A,B): Straight after A, B must be executed. It leads to add one effect to A for each activity C that does not match with B: (at end (locked-chain C)) ; and one to B for each activity C that does not match with B: (at end (not (locked-chain C))) .
10. CHAIN PRECEDENCE(A,B): Straight before B, A must be executed. It leads to add an effect to A: (at end (not (locked-chain B))) ; and an effect to all the activities different from A (even B): (at start (locked-chain B)) .
11. CHAIN SUCCESSION(A,B): Relations Chain Response(A,B) and Chain Precedence(A,B) must hold.

III) Negation constraints:

12. RESPONDED ABSENCE and NOT CO_EXISTENCE(A,B): If B is executed, then A can not be executed, and vice versa. It leads to add an effect to A: (at start (locked-perm B)) ; and one to B: (at start (locked-perm A)) .
13. NEGATION RESPONSE, NEGATION PRECEDENCE, NEGATION SUCCESSION(A,B): After the execution of A, B can not be executed. It leads to add an effect to A: (at start (locked-perm B)) .
14. NEGATION ALTERNATE RESPONSE(A,B): Between two executions of A, B can not be executed. It leads to add an effect to B: (at start (when (done A) (locked-perm A))) .
15. NEGATION ALTERNATE PRECEDENCE(A,B): Between two executions of B, A can not be executed. It leads to add an effect to A: (at start (when (done B) (locked-perm B))) .
16. NEGATION ALTERNATE SUCCESSION(A,B): Relations Negation

`Alternate Response(A,B)` and `Negation Alternate Precedence(A, B)` must hold. It leads to add one effect to `B`: `(at start (when (done A) (locked-perm A)))` ; and one to `A`: `(at start (when (done B) (locked-perm B)))` .

17. NEGATION CHAIN RESPONSE, NEGATION CHAIN PRECEDEN- CE, NEGATION CHAIN SUCCESSION(A,B): `B` can not be executed straight after the execution of `A`. It leads to add an effect to `A`: `(at start (locked-chain B))` ; and an effect to all the activities but `A`: `(at start (not (locked-chain B)))` .

The set of templates can be extended (van der Aalst and Pesic, 2006). One extension is the relation `mutual substitution` (van der Aalst and Pesic, 2006), establishing that, at least, one of two activities should occur. Another extension corresponds to the *branched constraints* (van der Aalst and Pesic, 2006) from one source activity to several sink ones, so the relation is given between the source and, at least, one of the sinks; or from several source activities to one sink, so the relation is given between, at least, one of the sources and the sink (examples in Sect. 4). When the branched constraint has only one source activity, non-deterministic effects are given for such activity. The inclusion of non-deterministic effects can be treated by stochastic planners (Dean et al., 1995). To the best of our knowledge, at present there is not an available planner able to treat with all the features required: durative actions, simultaneous action execution, non-deterministic effects and optimization. In order to overcome this problem, in this work, the non-deterministic ConDec-R problem is automatically translated to a set of deterministic ones following the algorithm 1, so generic planners can automatically solve the different deterministic problems.

The basic idea of Alg. 1 is explained as follows: each non-deterministic relation $ndr(r,A,B,C)$, where r is the given relation, A is the source and B,C are the sinks, means that, at least, one of relations $r(A,B)$ or $r(A,C)$ must be given. In order to treat both possibilities, two deterministic problems are solved, one considering the deterministic relation $r(A,B)$, and the other one considering $r(A,C)$. It is necessary to consider both possibilities each time the source activity is executed, leading to a limitation in the ConDec-R problems to be treated with the proposed approach: the maximum cardinality of the activities that are source of a non-deterministic relation, must be specified in the ConDec-R problem in order to generate the corresponding deterministic problems in a suitable way. Let n be the number of non-deterministic relations

Algorithm 1: Deterministic ConDec-R problems.

input : a non deterministic problem
$NDP < DR, NDR >$
output: a set of deterministic problems
$DP < DR >$

$Probs \leftarrow \{\}$;
$n \leftarrow$ number of non-deterministic relations considering the source maximum cardinality;
for $i \leftarrow 0$ **to** 2^{n-1} **do**
  $Prob \leftarrow NDP.DR \cup$ `DFromND` $(i, NDP.NDR)$;
  $Probs \leftarrow Probs \cup Prob$;
return $Probs$;

Function: `DFromND(` *int i, set NDR*`)`.

$s \leftarrow \{\}$;
foreach $ndr(r,A,B,C)$ *in* NDR **do**
  $rem = i\%2$;
  if $rem == 0$ **then**
    $s \leftarrow s \cup dr(r,A,B)$;
  else
    $s \leftarrow s \cup dr(r,A,C)$;
  $i = i/2$;
return s;

taking into account the maximum cardinality of the activities that are source of a non-deterministic relation. Then, in general, 2^n deterministic problems can be generated in order to deal with all the possibilities. In Alg. 1, the input is a ConDec-R non-deterministic problem (*NDP*), composed by a set of deterministic relations (*DR*), and a set of non-deterministic ones (*NDR*). As a result, a set of deterministic problems (*DP*), is obtained. *Probs* is a set that contains 2^n deterministic problems at the end of the algorithm. The function (`DFromND`) is in charge of generating different combinations of deterministic relations for each problem from the set *NDR*.

4 AN EXAMPLE

The Acme Travel Company problem is an adaptation of the one presented in (Snell, 2002), that was specified through DecSerFlow language in (van der Aalst and Pesic, 2006). As follows, the considered problem is described:

1. Acme Travel receives an itinerary from Karla, the customer.
2. After checking the itinerary for errors, the process determines which reservations to make, simulta-

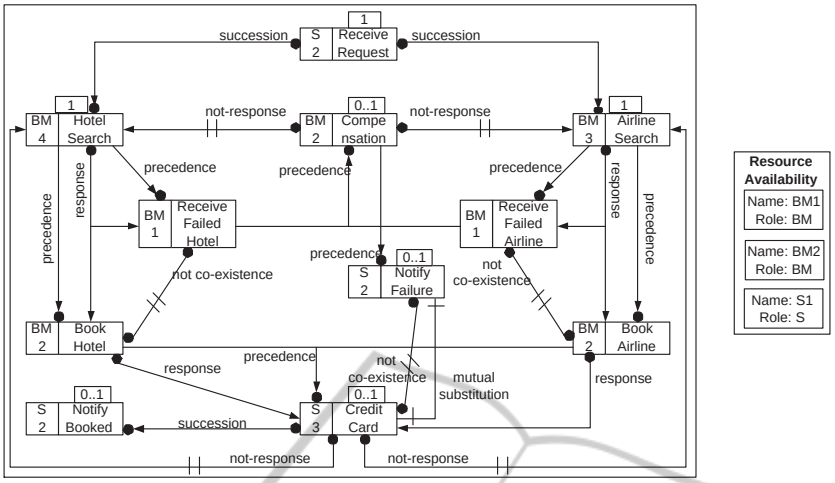

Figure 3: ConDec-R specification for Acme Travel Company.

neously asking for information to the appropriate airline and hotel agencies to make the appropriate reservations.

3. If any of the two reservation tasks fails, the itinerary is cancelled by performing the "compensate" activity and Karla is notified of the problem.
4. Acme Travel waits for confirmation of the two reservation requests.
5. Upon receipt of confirmation, Acme Travel notifies Karla of the successful completion of the process and sends her the itinerary details.
6. Once Karla is notified of either the success or failure of her requested itinerary, she may submit another travel request.

The activities in (van der Aalst and Pesic, 2006) are modelled as web services. Conversely, in this work, the activities are tasks that need to use some shared resources to be executed. Two roles are considered: Book Manager (BM) and Secretary (S). Also, it is estimated that two resources with role Book Manager (BM1 and BM2), and one with role Secretary (S), will be available for the BP enactment. Considering that only one instance is executed at the same time, the total time of the resulting plan must be minimized. In Fig. 3 the ConDec-R model of the problem is shown. As can be seen, eleven activities are presented:

Receive Request: A Secretary must attend the client request. This activity will be done exactly once.

Hotel Search: A BM asks for information to several hotel agencies to make the appropriate reservation. After the execution of *Receive Request*, it must be executed always, and before its execution, *Receive Request* must have been executed also (relation succession).

```
(define (domain
ConDec)
  (:requirements :adl :fluents :durative-actions)
  (:types act res role - object)
  (:predicates   (forced ?a - act) (locked ?a - act)
                 (locked-temp ?a - act)... (done ?a - act)
                 (role ?r - res ?ro - role) (free ?r - res)
                 (resource ?a - act ?ro - role))
  (:functions (duration ?act - act) (n-times-receive) ...
              (n-times-credit-card))
  (:durative-action Act-Credit-card
     :parameters (?r - res ?rol - role)
     :duration (= ?duration  (duration Credit-card))
     :condition (and (at start (not (locked Credit-card)))
          (at start (and (free ?r) (resources Credit-card ?rol)
                         (role ?r ?rol)))
          (at start (and (done Book-hotel) (done Book-airline))))
     :effect (and (at start (and (not (forced Credit-card))
                                 (not (free ?r))))
                  (at end (and (done Credit-card)
                               (increase (n-times-credit-card) 1)
                               (free ?r)))
                  (at end (forced Notify-booked))
                  (at end (and (locked-perm Hotel)
                               (locked-perm Airline)
                               (locked-perm Notify-failure)))))
```

Figure 4: PDDL 2.2 domain specification for the Acme Travel Company.

Book Hotel: A BM must book the appropriate reservation. *Hotel Search* has to be executed before it (precedence).

Receive Failed Hotel: A BM must receive the failure notification in the case it happens. *Hotel Search* has to be executed before it (precedence). On the other hand, after *Hotel Search*, one of the activities *Book Hotel* or *Rec. Failed Hotel* must be executed (branched response). Lastly, only one of *Book Hotel* and *Receive Failed Hotel* can be

```
(define (problem travelCompany)
(:domain ConDec)
(:objects Receive... - act Secretary BookManager - role
          S BM1 BM2 - res)
(:init (= (n-times-receive) 0)...(= (duration Credit-card) 3)
       (free S)...(role S Secretary)
       (resources Receive Secretary)...
       (resources Credit-card Secretary)
(:metric minimize (total-time))
(:goal (and (forall (?act - activity) (not (forced ?act)))
             (= (n-times-receive) 1) (< (n-times-compensation) 2)
             (< (n-times-notify-failure) 2)
             (< (n-times-notify-booked) 2)
             (< (n-times-credit-card) 2)
             (or (> (n-times-credit-card) 0)
                 (> (n-times-notify-failure) 0)))))
```

Figure 5: PDDL 2.2 problem specification for the Acme Travel Company.

executed (not co-existence).

Airline Search: A BM asks for information to several airline agencies to make the appropriate reservation. As *Hotel Search*, it presents a succession relation to *Receive Request*.

Book Airline: A BM must book the appropriate reservation (previously detected in *Airline Search*). *Airline Search* has to be executed before it (precedence).

Receive Failed Airline: A BM must receive the failure notification in the case it happens. It is involved in the same relation that *Receive Failed Hotel*, but regarding the airline activities.

Compensation: A Secretary must study the compensation for the client. This activity has to be preceded for at least one of *Receive Failed Hotel* or *Airline* (branched precedence relation). It can not be executed after neither *Hotel Search* nor *Airline Search* (not response).

Notify Failure: A Secretary must report the failure. It has to be preceded by *Compensation*.

Credit Card: A Secretary proceed to make the payment. One of *Book Hotel* or *Book Airline* has to be executed before it (branched precedence). Also, after *Book Hotel* and *Book Airline*, it must be executed (response relations). One and only one of *Notify Failure* and *Credit Card* must be executed (not co-existence and mutual substitution). Also, after *Credit Card*, neither *Hotel Search* nor *Airline Search* can be executed (not response).

Notify Booked: A Secretary must report the information about the book to the client. After the execution of *Credit Card*, it must be executed always, and before its execution, *Credit Card* must have been executed also (succession).

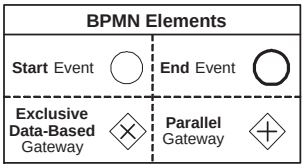

Figure 6: Some BPMN elements.

For this example, a part of the PDDL 2.2 domain is shown in Fig. 4, including all the aspects except that only a representative activity (Credit Card) is specified as an example. Also, the PDDL 2.2 problem specification is shown in Fig. 5. Taking the PDDL 2.2 specification as input, the planner solves the problem generating the optimum execution plan: allocating the available resources and temporarily assigning the start and the end times for the activities execution. This plan can be used to guide the BP model design.

The Business Process Modelling Notation (BPMN) (White and et al., 2004) is a standard for modelling BP flows and web services, and provides a graphical notation for specifying BP in a Business Process Diagram (BPD). The BPD is composed, between others, by events, gateways (Fig. 6), activities and swimlanes. An event represents something that happens during the enactment of a BP and affects its execution flow, specifically the start event initiates the flow of the process, while the end event finishes this flow. Gateways are in charge of controlling how sequence flows interact as they converge or diverge within a process, specifically the *exclusive data-based gateway* can be used as a decision point or as a way to merge several sequence flows into one; while the *parallel gateway* provides a mechanism to fork and synchronize the flows. Swim lanes are graphic ways of organizing and categorizing the BP activities, specifically pools represent the participants in a BP, and lanes are used to organize the activities within a pool according to roles or resources.

Taking into account the PDDL 2.2 solutions, an optimized and feasible BPMN can be designed (Fig. 7). It is composed by a pool named Travel Company that contains three lanes, BM1, BM2 and S. In Fig. 7, RFA is a boolean that represents a fail given during the booking of the airline, while RFH represents the same for the hotel booking.

5 CONCLUSIONS AND FUTURE WORK

This work proposes a PDDL 2.2 model for the optimal BP execution plan generation when specifying the process information in a declarative way, apply-

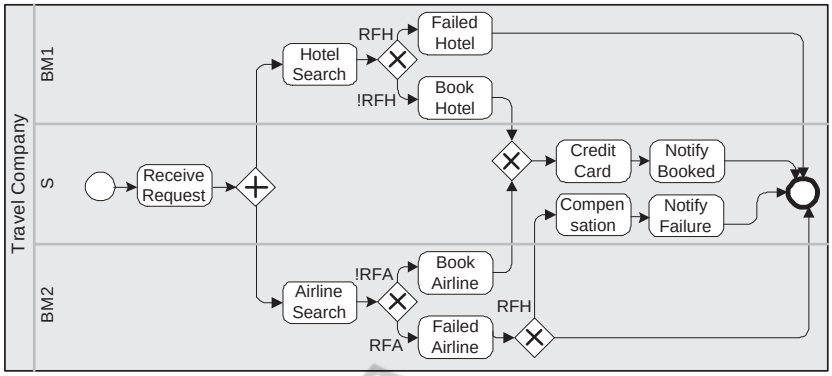

Figure 7: BPMN for the Acme Travel Company.

ing an AI-based planning and scheduling approach to consider resources allocation and the minimization of the plan duration. The BP information is provided through a friendly graphic language (ConDec-R).

As future work, it is intended to develop a tool for the automatic generation of process models from PDDL 2.2 plans. Furthermore, the use of different AI-based approaches to generate process models from declarative specifications will be analyzed.

ACKNOWLEDGEMENTS

This work has been partially funded by the Consejería de Innovación, Ciencia y Empresa of Junta de Andalucía (P08-TIC-04095) and by the Spanish Ministerio de Ciencia e Innovación (TIN2009-13714) and the European Regional Development Fund (ERDF/FEDER).

REFERENCES

Barba, I. and Del Valle, C. (2010). Planning and scheduling of business processes in run-time: A repair planning example. In *Proceedings of the 19th International Conference on Information Systems Development (ISD 2010)*. Springer (in press).

Brucker, P. and Knust, S. (2006). *Complex Scheduling (GOR-Publications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Clarke Jr., E., Grumberg, O., and Peled, D. (1999). *Model Checking*. The MIT Press.

Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. E. (1995). Planning under time constraints in stochastic domains. *Artif. Intell.*, 76(1-2):35–74.

Fahland, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., and Zugal, S. (2010). Declarative versus imperative process modeling languages: The issue of maintainability. *Lecture Notes in Business Information Processing*, 43 LNBIP:477–488.

Ghallab, M. and et al. (1998). Pddl - the planning domain definition language. Technical report, CVC TR-98-003/DCS TR-1165.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann, Amsterdam.

González-Ferrer, A., Fernández-Olivares, J., and Castillo, L. (2009). Jabbah: A java application framework for the translation between business process models and htn. In *International Competition on Knowledge Engineering for Planning ICKEPS*.

Hoffmann, J. and Edelkamp, S. (2005). The deterministic part of ipc-4: an overview. *J. Artif. Int. Res.*, 24(1):519–579.

Kearney, P., Borrajo, D., Cesta, A., Matino, N., and Mehandjiev, N. (2003). Planet workflow management r&d roadmap. http://scalab.uc3m.es/∼dborrajo/planet/wm-tcu/Roadmap-WM-phase-II.pdf.

Pesic, M. and van der Aalst, W. M. P. (2006). A declarative approach for flexible business processes management. In *Business Process Management Workshops*, pages 169–180. Springer.

Snell, J. (2002). Automating business processes and transactions in web services: An introduction to bpelws, ws-coordination, and ws-transaction. http://www.ibm.com/developerworks/webservices/library/ws-autobp/.

van der Aalst, W. M. P. and Pesic, M. (2006). Decserflow: Towards a truly declarative service flow language. In *LNCS 4184*, pages 1–23.

van der Aalst, W. M. P., ter Hofstede, A. H., and Weske, M. (2003). Business process management: A survey. *Int. Conf. BPM 2003, Proceedings*, pages 1–12.

White, S. and et al. (2004). Business Process Modeling Notation (BPMN), Working draft, Version 1.0.