# Temporal Planning for Business Process Optimisation

**Daniele Magazzeni**
Department of Informatics
King's College London, UK

**Fabio Mercorio**
CRISP Research Centre
University of Milan Bicocca, Italy

**Balbir Barn, Tony Clark, Franco Raimondi**
Department of Computer Science
Middlesex University, London, UK

**Vinay Kulkarni**
Tata Consultancy Services
Pune, India

**Abstract**

In this paper we consider the problem of designing and optimising a business process. Given a set of activities and a fixed budget, the objective is to determine duration and resource allocation for each activity such that the time-to-market is minimised while budget and dependencies constraints are met. We give a formal description of the problem and we show how it can be cast as a temporal planning problem, resulting in a challenging benchmark planning problem involving concurrency and duration-dependent costs.

The user has to define only dependencies among activities, costs of resources and the available budget, and then use a planner to design an efficient process, which is then generated as a Gantt chart. As a case study, we consider a concrete scenario provided by an industrial partner, and we use a temporal planner to design an effective business process.

## Introduction

Business organisations that provide or build products (e.g., software, mixed software-hardware solutions, and even actual goods) are likely to employ abstract modelling languages to describe and analyse their business processes. A number of formal languages are available for modelling business processes, and various tools exist to automate the analysis of the modelled workflows and get advice on how to better invest resources. In a number of instances, including large organisations, the design of business processes is still a manual process that relies on the experience of top-level managers and domain experts to produce sequences of steps that achieve a desired business goal, subject to the minimisation/maximisation of various metrics. As a result, there is no guarantee that the business processes obtained using this process are indeed the most efficient solution. Additionally, the exploration of different options is a very time-consuming task: each new process has to be developed and analysed separately, and alternative solutions need to be compared manually.

In this paper we argue that the design and the optimisation of business processes can be automated using AI planning techniques, thus providing an effective tool to search for "efficient" solutions for resource allocation and task scheduling, or to quickly explore alternative business processes.

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

More in detail, our contributions can be summarised as follows: we present a novel application domain for temporal planning that is derived form a concrete instance provided by an industrial partner; we describe a benchmark domain that results in a challenging multi-objective temporal planning problem; as a practical example, we consider a model-driven software development process and we show how it can be encoded as a temporal planning problem. Finally, we use a temporal planner to generate solutions that minimise time-to-market for a given project budget and we provide a visual representation of the automatically generated non-trivial resource allocation using Gantt charts.

## A Formal Model for Business Processes

In this section we first provide a formal semantics for business processes that abstracts the existing approaches described above. Then, we describe an actual process currently in use at *Tata Consultancy Services*. Then a mapping between the formal model and a temporal planning model is presented, using the concrete business process as a running example.

**Definition 1 (Business Process)** *A* Business Process (BP) $\mathcal{B}$ *is a 10-tuple* $(\mathcal{S},s_i,s_e,\mathcal{P},\mathcal{D},\mathcal{R},\mathcal{A},\mathcal{T},\mathcal{C},b)$*, where:* $\mathcal{S}$ *is a finite set of* states, $s_i \in \mathcal{S}$ *is the* initial state, $s_e \in \mathcal{S}$ *is the* end state, $\mathcal{P}$ *is a finite set of* parameters, $\mathcal{D} : \mathcal{S} \to 2^{\mathcal{S}}$ *is the* dependency function, $\mathcal{R} : \mathcal{S} \to 2^{\mathcal{P}}$ *is the* requirements function, $\mathcal{A} : \mathcal{S} \to 2^{\mathcal{P}}$ *is the* allocation function, $\mathcal{T} : \mathcal{S} \to \mathbb{R}^+$ *is the* temporal function, $\mathcal{C} : \mathcal{S} \times 2^{\mathcal{P}} \times \mathbb{R}^+ \to \mathbb{R}^+$ *is the* cost function *and* $b \in \mathbb{R}^+$ *is the* budget.

The set of states $S$ corresponds to the set of typical business steps, such as "Requirements analysis" or "Code testing". Each state may depend on other states: for instance, "Code testing" depends on "Code generation". The set $\mathcal{P}$ includes parameters such as number of developers, number of testers, number of domain experts, etc. For each state $s \in \mathcal{S}$, $\mathcal{D}(s)$ defines the set of states $s$ depends on, $\mathcal{R}(s)$ defines the requirements for the phase represented by $s$, $\mathcal{A}(s)$ defines the resources allocated for it, $\mathcal{T}(s)$ defines its duration and $\mathcal{C}(s, \mathcal{A}(s), \mathcal{T}(s))$ defines its cost. Finally we have an initial state $s_i \in \mathcal{S}$ (such that $\mathcal{D}(s_i) = \emptyset$), and an end state $s_e \in \mathcal{S}$.

**Definition 2 (Business Process Execution)** *A* Business Process Execution *for the* BP $\mathcal{B}$=$(\mathcal{S},s_i,s_e,\mathcal{P},\mathcal{D},\mathcal{A},\mathcal{T},\mathcal{C},b)$ *is a sequence* $\pi = (s_0a_0t_0)(s_1a_1t_1)(s_2a_2t_2)\dots s_n$*, where,*

$\forall j \geq 0$, $s_j \in \mathcal{S}$, $a_j = \mathcal{A}(s_j)$, $t_j = \mathcal{T}(s_j)$. *A Business Process Execution is* admissible *iff:* (i) $s_0 = s_i$, (ii) $s_n = s_e$, (iii) $\forall j \geq 1, \forall s_k \in \{D(s_j)\} : s_k \in \pi \wedge k < j$, (iv) $\forall s_l \in \mathcal{S}, \mathcal{R}(s_l) \subseteq \mathcal{A}(s_l)$, (v) $\sum_{j=0\ldots n-1} \mathcal{C}(s_j, a_j, t_j) \leq b$.

In particular, conditions (iii), (iv), and (v) require all the dependencies among phases to be satisfied, all phases requirements to be met and the total cost to be within the given budget, respectively.

### The Model-Driven Software Development Process

Table 1 describes the states and the associated functions $\mathcal{R}(s), \mathcal{T}(s)$, and $\mathcal{C}(s, a, t)$ for a model-driven software development (MDSD) process that has been used to deliver several large business applications for past 16 years at Tata Consultancy Services.

A row of the table depicts a specific phase of the MDSD process, the time the phase takes to complete as a percentage of total time taken for completion of the MDSD process, and the various actors participating in the phase along with their relative contribution. For instance, the High Level Design phase requires 5% of the time taken by the overall MDSD process, and requires the participation of a Solution Architect (SA), a Domain Expert (DE) and a Technology Architect (TA). If the overall time required by a project is 100 man-days, this line encodes the fact that the HLD phase requires $0.3 \cdot 5$ TA days, $0.1 \cdot 5$ DE days, and $0.6 \cdot 5$ SA days. The remaining actors are Test Engineer (TE), MDE Expert (ME), Modeller (M), Developer (D), and Tester (T).

The initial state is $s_i$ = `RE`, and the end state is $s_e$ = `END`. The dependency function $\mathcal{D}$ is graphically shown in Figure 1. The function $\mathcal{C}(s, a, t)$ can be derived from the times described in Table 1 and the costs in Table 2 (where costs are normalised to the cost of a tester).

## Business Process Optimisation as a Temporal Planning Problem

The proposal of this paper is a translation of a business process (as defined in Definition 1) into a temporal planning problem, so that planners can be used to find admissible business process executions (as defined in Definition 2) while minimising the time-to-market.

It is worth noting that the use of *temporal* planning is key in this context as it allows modelling of concurrent activities and time-dependent resource allocations, and to compute duration-dependent costs. Furthermore, although the business process design could be seen as a scheduling problem, it represents an interesting domain where planning plays an important role. To this aim, we follow the same approach proposed in (Fox, Long, and Magazzeni 2011; 2012), where the battery scheduling problem is cast as a temporal planning problem and solved using the temporal planner UPMurphi (Della Penna et al. 2009). In particular, in the battery scheduling problem the number of switching actions cannot be identified in advance as well as in the BP domain the number of resources that can be allocated to each phase is not known in advance. Furthermore, the order in which phases are executed, their duration and how concurrency can be exploited are not know, either. In the following we present the main components of the PDDL domain and problem.

**The Planning Domain.** The business process domain presents a number of challenging features to be modelled. First, the business process consists of different phases each of which, in turn, requires a number of tasks to be accomplished. The order of execution of the phases is not fixed, but there is a set of dependencies among phases that must be satisfied, which, however, allow for a set of phases to be executed in parallel. Second, each task is associated with a skill and people of different skills have to be allocated to each phase to accomplish their corresponding tasks. The number of people to allocate is not know in advance, and only an upper bound is provided. Third, the project cost, that needs to be maintained within the given budget, depends on how many days each resource is allocated and thus is modelled as a time-dependent cost. Finally, as we want to optimise the time-to-market, the duration of the plan has to be minimised.

We begin the description of the domain with the `start_project` and `end_project` actions, shown in Figure 2a. The start (end) project action is used to enable (disable) the recruitment of resources and the execution of the project phases. Then, in order to model resource allocation, we distinguish between *employing* a resource (which defines the total number of resources for each skill that will be used) and *allocating* a resource (which defines how resources are used throughout the process). We make this distinction as both recruitment and daily costs of resources must be considered.

*Employing Resources.* Figure 2b shows the `employ` and `dismiss` actions for domain experts (similar actions are defined for other skills). These actions are used for managing the amount of resources recruited over the project. In particular the `employ` action increments the project cost by the cost of recruiting that particular resource (costs of resources of different skills are shown in Table 2) making that available to be allocated. On the other hand, the `dismiss` action, which is applied when the project is completed, is used to dismiss a resource.

*Allocating resources.* The planner can use allocation (deallocation) actions to assign (release) resources of different skills to each phase of the business process before (after) performing that phase through the corresponding execution action. As an example, Figure 2c shows the actions for allocating and deallocating a domain expert. Note that the `deallocate` action for skill `Y` does not require the whole phase to be finished, but only the task for skill `Y` to be completed. This allows a flexible allocation of the same resource to different phases.

*Executing Phases.* Modelling the execution of a phase presents an interesting issue, as a phase consists of one or more tasks to be completed. Furthermore, the duration of each task is defined in terms of man-days for the skill required to perform the task (as shown in Table 1). Let us assume that phase `p` requires skills `A`, `B`, `C`, and for each of them the amount of work is `pAdays`, `pBdays` and `pCdays`.

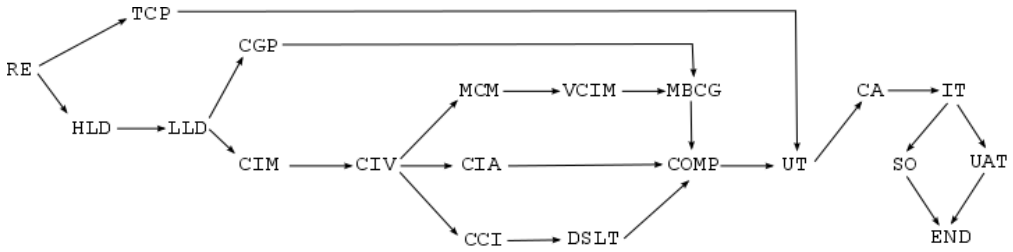


Figure 1: Dependency graph for the MDSD process

Table 1: Phases of the MDSD process

| $s \in \mathcal{S}$ | $\mathcal{T}(s)$ | $\mathcal{R}(s)$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | % Time | DE | SA | TA | TE | ME | M | D | T |
| Requirements Elicitation (RE) | 15 | 0.9 | 0.1 | | | | | | |
| High Level Design (HLD) | 5 | 0.1 | 0.6 | 0.3 | | | | | |
| Test Case Preparation (TCP) | 5 | 0.2 | 0.1 | | 0.7 | | | | |
| Low Level Design (LLD) | 10 | | 0.3 | | 0.7 | | | | |
| Code Generator Procurement (CGP) | 10 | | 0.2 | 0.2 | | 0.6 | | | |
| Component Interface Modelling (CIM) | 2 | | 0.1 | | | 0.1 | 0.8 | | |
| Component Interface Validation (CIV) | 2 | | 0.1 | | | 0.1 | 0.8 | | |
| Component Interface Assembly (CIA) | 1 | | 0.2 | | | 0.1 | 0.7 | | |
| Modelling Component Implementation (MCM) | 5 | | 0.1 | | | 0.1 | 0.8 | | |
| Validation of Component Implementation Model (VCIM) | 5 | | 0.1 | | | 0.1 | 0.8 | | |
| Coding of Component Implementation (CCI) | 7 | | 0.8 | | | | | 0.2 | |
| Model-Based Code Generation (MBCG) | 3 | | 0.1 | | | | 0.1 | 0.8 | |
| DSL translation (DSLT) | 4 | | 0.1 | | | | | 0.9 | |
| Compilation (COMP) | 5 | | | | | | | 1 | |
| Unit Testing (UT) | 5 | | | | | | | 1 | |
| Component Assembly (CA) | 5 | | 0.2 | | | | 0.4 | 0.4 | |
| Integration Testing (IT) | 5 | | 0.1 | | 0.1 | | | | 0.8 |
| User Acceptance Testing (UAT) | 5 | 0.1 | 0.1 | | 0.1 | | | | 0.7 |
| Sign Off (SO) | 1 | 0.4 | 0.3 | | 0.3 | | | | |
| END | | | | | | | | | |

```
(:action  start_project
 :parameters (?p - phase)
 :precondition (and
  (todo_project)
  (is_first_phase ?p))
 :effect (and
  (running_project)
  (not (todo_project))))

(:action  end_project
 :parameters (?p - phase)
 :precondition (and
  (completed ?p)
  (is_last_phase ?p))
 :effect (and
  (project_completed)))
```

(a)

```
(:action  employ_DE
:parameters ()
:precondition (and
 (< (employed_DE) (max_DE))
 (running_project))
:effect (and
 (increase (available_DE) 1)
 (increase (employed_DE) 1)
 (increase (total_project_cost)
  (employment_cost_DE))))

(:action  dismiss_DE
 :parameters ()
 :precondition (and (project_completed)
  (> (employed_DE) 0) (> (available_DE) 0))
 :effect (and
  (decrease (employed_DE) 1)
  (decrease (available_DE) 1)))
```

(b)

```
(:action allocate_DE
 :parameters (?p - phase)
 :precondition (and
  (doing ?p)
  (> (available_DE) 0))
 :effect (and
  (increase (allocated_DE ?p) 1)
  (decrease (available_DE) 1)))

(:action deallocate_DE
 :parameters (?p - phase)
 :precondition (and
  (completed_DE ?p)
  (>= (available_DE) 0)
  (> (allocated_DE ?p) 0))
 :effect (and
  (decrease (allocated_DE ?p) 1)
  (increase (available_DE) 1)))
```

(c)

Figure 2: (a) The `start_project` and `end_project` actions. (b) The `employ` and `dismiss` actions for a Design Expert. (c) The `allocate` and `deallocate` actions for a Design Expert

If the planner has allocated `pAres`, `pBres`, `pCres` resources to phase `p`, then the duration of the phase is

$$\max_{i \in \{\mathtt{A},\mathtt{B},\mathtt{C}\}} \left( \frac{\mathtt{p}_i\mathtt{days}}{\mathtt{p}_i\mathtt{res}} \right)$$

Therefore, the effects of the action become effective only when all the tasks have been completed. On the other hand, the resources of skill $j$ allocated for the phase become available as soon as the task requiring skill $j$ terminates, even if the other tasks of the phase are still executing.

Modelling such a scenario is not trivial, and the proposed solution is illustrated in Figure 4. For each phase of the business process, an *envelope* action is used, whose duration is left to the planner, which encapsulates $k$ durative actions (where $k$ is the number of different tasks required to complete the phase), whose duration depends on the resources previously allocated by the planner.

As an example, Figure 3a shows the envelope action to perform a phase, while Figure 3b shows the action for the task requiring domain experts. Note that the number of resources to be allocated to the task is a significant value that the planner will identify carefully. Indeed, this value affects

Table 2: Normalised costs of resources of different skills

| Skill | per-day-cost | employment cost |
|---|---|---|
| DE | 5 | 30 |
| SA | 5 | 30 |
| TA | 4 | 25 |
| TE | 2 | 20 |
| ME | 5 | 30 |
| M | 2 | 20 |
| D | 1.25 | 12.5 |
| T | 1 | 10 |

both the duration of a task and, in turn, the cost for executing it, since a resource is paid according to its daily cost and the task duration.

**The Planning Problem.** The goal is to complete the whole project satisfying all the dependencies. Furthermore, the total project cost must be within the given budget. To this end, the goal has the condition that `total_project_cost` must be no greater than `budget`, where `total_project_cost` depends on the number of resources with different skills allocated to a task and the per-day cost of each resource (as shown in Table 2). As we said before, we are interested in minimising *Time-to-Market*. This is mapped into the planning metric `(:metric minimize (total-time))`.

A fragment of the PDDL problem is shown in Figure 3c, where we show key elements for phase Compilation (COMP) and resource Developer (D). The budget is fixed to $1,500$ (normalised to the cost of a tester). The `per_day_cost_D`, the `employment_cost_D` and the maximum number of Developers that can be employed are defined according to the normalised costs of resources shown in Tab. 2.

The predicate `(todotask_D COMP)` is used to specify which kind of skills are needed to complete the phase (here a Developer is needed to perform the Compilation phase). The dependency graph for the MDSD process is defined through the predicate `(depends COMP CIA MBCG DSLT)` which constrains the execution of the COMP phase to the completion of three distinct phases, that are the Component Interface Assembly, Model-Based Code Generation, and DSL Translation.

## Experimental Results

To solve the business process domain, we used the forward chaining temporal planner POPF (Coles et al. 2010). We considered the MDSD of Table 1, with a budget of $1,500$ and a maximum number of employees for each skill to be employed equal to 5. We used a x64 Linux machine equipped with 6 GB of RAM and we considered the best solution found by POPF within 30 minutes[1]. We found a solution for the MDSD process ($P$ in the following) that requires about 15 days, with a total cost of $1,202$.

A complete Gantt chart for the solution $P$ that gives an overview of the execution of phases and tasks has been generated[2]. Here we focus on a fragment of it (shown in Figure 5) which allows us to highlight the key element of the plan, that is the optimised parallel execution of several phases. In particular, the model allows the switching of a resource between phases even when they are still on-going. Specifically, Figure 5 focuses on phases *Modelling Component Implementation* (MCM), *Component Interface Assembly* (CIA), and *Coding of Component Implementation* (CCI). Note that all these three phases need to complete a task which involves Software Architects (boxes with a green vertical texture in Figure 5). Furthermore, the three phases need to be completed in order to start the *Compilation* phase (according to the dependency graph shown in Figure 1). In order to speed up the execution of these parallel phases the planner decides to assign 4 out of 5 Software Architects to complete the task of phase *Coding of Component Implementation*. At the same time, it first assigns the remaining Software Architect to phase *Component Interface Assembly*, and then to phase *Modelling Component Implementation*, while the phase *Coding of Component Implementation* is still running. Through a non-trivial allocation of resources between tasks, the planner is able to reduce the project duration and optimise the budget usage.

### Evaluation

The generated solution has been validated by our industrial partner, in comparison with plans used in the company for similar projects. From a practical point of view, our industrial partner confirmed that the added value of the plan-based solution comes from having a plan that suggests efficient durations for phases where resources are switched between ongoing phases, which is key for reducing the time-to-market and that would be hard to be planned manually.

As a further evaluation, we compare the solution $P$ with two other plan-based solutions $P_A$ and $P_B$, as described in the following.

**Solution $P_A$.** First we want to show how planning can provide different high quality solutions according to the amount of resources available, by modifying (if needed) the process itself, and not only the resource allocation. Therefore we modify the planning problem by allowing the planner to recruit up to 6 workers (instead of 5) for each skill. The planner is then able to find the plan $P_A$, which is more expensive (although still within the assigned budget) but shorter then $P$. Figure 6 shows a comparison between the two solutions. As can be noticed, the planner produces a different process, and $P_A$ differs from $P$ not only in the resource allocations and phase durations, but also in the order in which the phases are executed.

**Solution $P_B$.** Second we want to show how the use of planning can effectively help the business process design, comparing to what one could achieve without using a planner. To this aim, based on the industrial partner's experience, we modify the planning domain to reflect as much as possible how the business process is currently designed in industry. As noticed before, a typical approach followed by man-

[1] POPF is an any-time planner, which improves the current solution as time is given.

[2] Due to the space limitation the complete Gantt chart has been made available at `http://goo.gl/Ki7RvX`

```
(:durative-action execute_phase
:parameters (?p ?dp1 ?dp2 ?dp3 -
phase)
:duration (and (<= ?duration
 (upper_bound ?p)))
:condition (and
  (at start (todo ?p))
  (at start (running_project))
  (at start (depends ?p ?dp1 ?dp2
   ?dp3))
  (at start (completed ?dp1))
  (at start (completed ?dp2))
  (at start (completed ?dp3))
  (at end (completed_DE ?p))
  (at end (completed_SA ?p))
  (at end (completed_TA ?p)))
:effect (and
  (at start (doing ?p))
  (at end (completed ?p))
  (at end (not (doing ?p)))
  (at end (not (todo ?p)))))
```

(a)

```
(:durative-action task-execution_DE
 :parameters (?p - phase
 ?nT - somenumber)
 :duration (= ?duration
  (/ (duration_task_DE ?p)
  (value_of ?nT)))
 :condition (and
  (at start (todotask_DE ?p))
  (over all (doing ?p))
  (over all (= (allocated_DE ?p)
   (value_of ?nT)))
  (at end (>= (employed_DE)
   (value_of ?nT))))
 :effect (and
  (at start (not (todotask_DE ?p)))
  (at end (completed_DE ?p))
  (at end
   (increase
    (total_project_cost)
    (* (* (value_of ?nT) ?duration)
    (per_day_cost_DE))))))
```

(b)

```
(is_first_phase RE) (is_last_phase END)
(= (budget) 1500) (= (total_project_cost) 0)
;; Developer
(= (employed_D) 0) (= (max_D) 5)
(= (available_D) 0) (= (employment_cost_D) 12.5)
(= (per_day_cost_D) 1.25)

;; columns R(s) of Tab. 1 for a Developer
(todotask_D CCI) (todotask_D MBCG)
(todotask_D DSLT)  (todotask_D COMP)
(todotask_D UT)  (todotask_D CA)
(depends COMP CIA MBCG DSLT) ;; graph of Fig.1

;; COMP PHASE
(completed_DE COMP) (completed_TA COMP)
(completed_TE COMP) (completed_ME COMP)
(completed_M COMP) (completed_T COMP)
(completed_SA COMP) (= (duration_task_D COMP) 5)

(:goal (and (project_completed)
  (<= (total_project_cost) (budget)))
(:metric minimize (total-time)))
```

(c)

Figure 3: (a) An example of `execute_phase` action. (b) An example of `task-execution` action. (c) An extraction of PDDL problem for the `COMP` phase

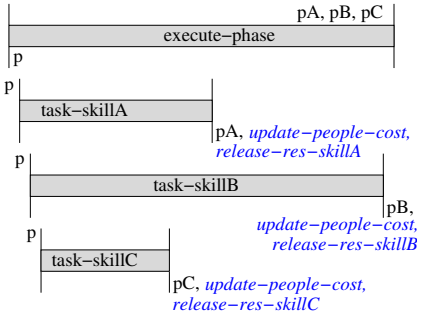


Figure 4: Envelope action for task execution

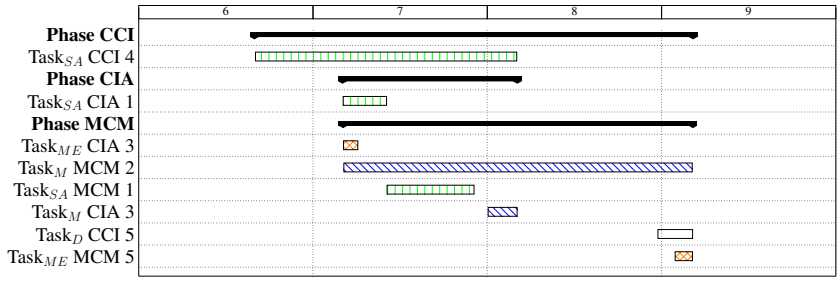


Figure 5: A fragment of the Gannt Chart of Solution $P$ . Relevant tasks are highlighted with a green vertical texture, an orange crosshatched texture or a blue slanted texture, to refer to Software Architects (SA), MDE Experts (ME) or Modellers (M), respectively.

agers is to move a resource to a different phase only when the current phase is finished, as the switching of resources between ongoing phases is hard to be planned manually. To model such a scenario, we modify the `deallocate (?p - phase)` action requiring the whole phase to be finished to release a resource. The best solution found by the planner is plan $P_B$ shown in Figure 7 in comparison with $P$. As expected, each phase duration is much longer than in plan $P$, and the whole project takes 41 days longer than in $P$, when faced with the same budget.

Note that dependencies between phases do not limit the planner in deciding the order in which phases are executed. Indeed, both $P_A$ and $P_B$ solutions present a different execution flow, while continuing to satisfy the dependency graph[3].

## Discussion and Related Work

A number of P&S techniques as well as CP-based approaches have been applied to the domain of BPM, aiming to help workflow designers. In particular, FlowOpt (Barták et al. 2011; 2012) is a tool for workflow optimisation, based on constraint satisfaction techniques. The user can visually model a workflow, and the tool automatically generates a production schedule represented as a Gantt chart. The user can then vary the quantities of items to be produced and the tool will modify the schedule accordingly also suggesting some improvements such as buying new resources. On the other hand, JABBAH (Gonzalez-Ferrer, Fernandez-Olivares, and Castillo 2013) provides a mapping between BPM and the HTN planning paradigm. The tool takes as input a workflow graph (described in the BPMN notation) and translates it into HTN-PDDL code. Then the planner IACTIVE is used to find a valid plan, i.e., a valid Gantt chart for the given BPM. In the work by Senkul and Toroslu (Senkul and Toroslu 2005), workflows described in the WSL language are translated into constraint programs in Oz, and then CP techniques are used to find valid resource allocations. Other related approaches include (Lombardi and Milano 2009; Valls, Pérez, and Quintanilla 2009; Wang et al. 2011; Wang and Smith 2005).

Although these papers are all relevant, a key issue that differentiates our work is the *temporal optimisation*. In par-

[3]The complete set of PDDL domain/problems/plans have been made publicly available at `http://goo.gl/OlUDWS`

ticular, we consider tasks with flexible durations whose values are optimised by the planner. Furthermore, the order of tasks is not fixed, but is chosen by the planner, too. This motivates the exploration of using a planner to minimise time-to-market while respecting budget constraints. Furthermore, users have to define only dependencies among phases, leaving to the planner the decision on how to better schedule the activities for improving efficiency while respecting the dependencies.

BPM has also been modelled as a resource-constrained project scheduling problem (RCPSP), where a set of tasks (activities) with fixed start times and durations, have to be executed without interruption using a given set of resources. But again, existing works consider activities with fixed durations (Hartmann and Briskorn 2010). Recently the multi-mode RCPSP framework has been proposed, where an activity can be associated with several modes, each modelling a feasible pair (resource allocation/duration) for the activity. This approach, however, requires the set of possible modes to be enumerated by the user, which represents a significant issue for BPM designers when faced with large projects. Instead, in the planning based approach, this issue is left to the planner which can look for efficient solutions going beyond the limited set of possibilities provided by the user.

Finally, it is worth mentioning the work (Hoffmann, Weber, and Kraft 2012) developed in collaboration with SAP where the planner FF is used for process composition.

## Conclusion and Future Work

In this paper we described how the problem of designing and optimising a business process can be cast as a temporal planning problem. Our experience at Tata Consultancy Services over nearly two decades has shown that the correct design of business processes can make the difference between successful and unsuccessful projects. However, in spite of the large body of work available for the generation and verification of business processes (see for instance (Bianculli, Ghezzi, and Spoletini 2007) and references therein), the support for the automatic *optimisation* of business processes is still at an early stage. We presented a contribution on this direction, modelling the design and the optimisation of the business process as a temporal planning domain. We then provided an effective solution for an industrial case study using a temporal planner to find plans that minimise time-to-market for a given project budget.

The domain, as such, represents a challenging problem for planning as it requires concurrency and the handling of durative actions with duration-dependent costs. Beyond that, we hope that the problem we consider in this paper, where activities to be scheduled have a flexible duration dependent on resource allocation and time-to-market needs to be minimised, can represent an interesting benchmark for the community and foster the application of P&S and CP-based techniques to the domain of BPM optimisation.

A natural future work for extending the proposed model is to exploit the expressive power of PDDL3 (Gerevini and Long 2006) and use *preferences* to take into account soft constraints. Furthermore, it will be challenging to deal with the multi-objective optimisation involved in this problem (such as time-to-market vs budget tradeoff) and provide richer suggestions to business organisations. Finally, we want to explore the use of mixed approach where planning and scheduling techniques can be interleaved to find efficient solutions.

## References


Barták, R.; Jaska, M.; Novák, L.; Rovensky, V.; Skalicky, T.; Cully, M.; Sheahan, C.; and Thanh-Tung, D. 2011. Workflow optimization with FlowOpt: On modelling, optimizing, visualizing, and analysing production workflows. In *Proc. TAAI'11*, 167–172.

Barták, R.; Jaska, M.; Novák, L.; Rovensky, V.; Skalicky, T.; Cully, M.; Sheahan, C.; and Thanh-Tung, D. 2012. FlowOpt: Bridging the gap between optimization technology and manufacturing planners. In *Proc. ECAI'12*, 1003–1004.

Bianculli, D.; Ghezzi, C.; and Spoletini, P. 2007. A model checking approach to verify BPEL4WS workflows. In *Proc. SOCA'07*, 13–20.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proc. ICAPS*.

Della Penna, G.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2009. UPMurphi: a tool for universal planning on PDDL+ problems. In *Proc. ICAPS'09*, 106–113. AAAI Press.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic construction of efficient multiple battery usage policies. In *Proc. ICAPS'11*, 74–81.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based policies for efficient multiple battery load management. *J. Artif. Intell. Res. (JAIR)* 44:335–382.

Gerevini, A., and Long, D. 2006. Preferences and soft constraints in PDDL3. In *Proceedings of ICAPS Workshop on Planning with Preferences and Soft Constraints*.

Gonzalez-Ferrer, A.; Fernandez-Olivares, J.; and Castillo, L. 2013. From business process models to hierarchical task network planning domains. *The Knowledge Engineering Review* 28(2):175–193.

Hartmann, S., and Briskorn, D. 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 207(1):1–14.

Hoffmann, J.; Weber, I.; and Kraft, F. 2012. SAP speaks PDDL: Exploiting a software-engineering model for planning in business process management. *J. Artif. Intell. Res. (JAIR)* 44:587–632.

Lombardi, M., and Milano, M. 2009. A precedence constraint posting approach for the rcpsp with time lags and variable durations. In *Principles and Practice of Constraint Programming-CP 2009*. Springer. 569–583.

Senkul, P., and Toroslu, I. H. 2005. An architecture for workflow scheduling under resource allocation constraints. *Information Systems* 30(5):399–422.

Valls, V.; Pérez, Á.; and Quintanilla, S. 2009. Skilled workforce scheduling in service centres. *European Journal of Operational Research* 193(3):791–804.

Wang, X., and Smith, S. F. 2005. Retaining flexibility to maximize quality when the scheduler has the right to decide activity durations. In *ICAPS*, 212–221.

Wang, X.; Policella, N.; Smith, S. F.; and Oddi, A. 2011. Constraint-based methods for scheduling discretionary services. *Ai Communications* 24(1):51–73.

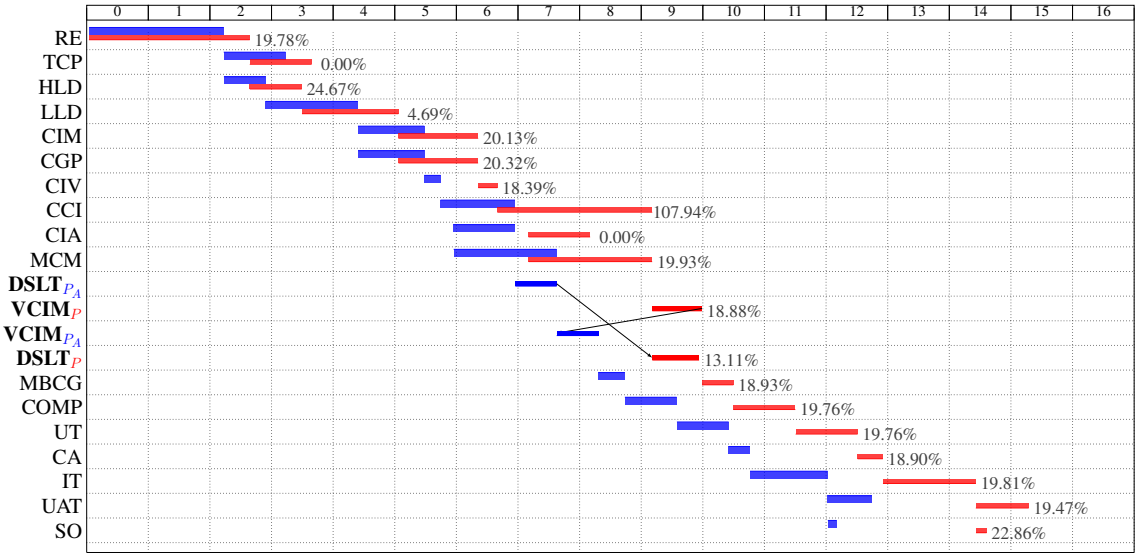


Figure 6: Solution $P_A$. Plan $P$ (red) using at most 5 workers costs 1,202.405 with time-to-market 15.289. Plan $P_A$ (blue) using at most 6 workers costs 1,338.672 with time-to-market 12.755. The duration of each phase for solution $P_A$ is compared with the corresponding phase of solution $P$. The difference is shown in percentage with respect to $P$.

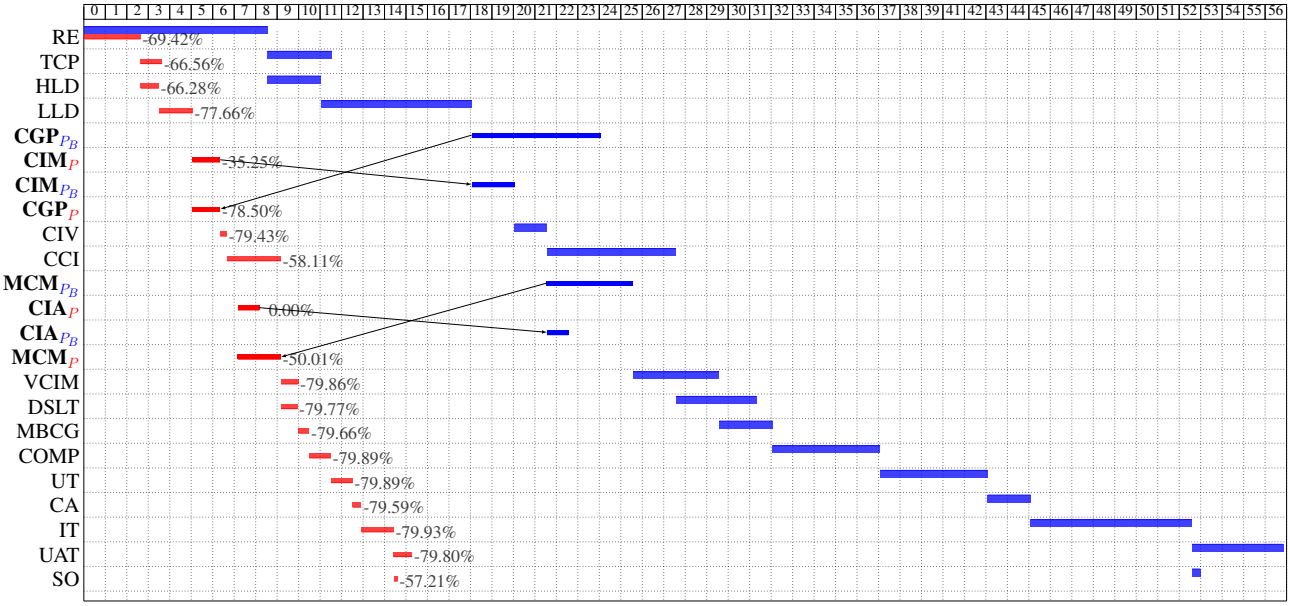


Figure 7: Solution $P_B$. Plan $P$ (red) using at most 5 workers costs 1,202.405 with time-to-market 15.289. Plan $P_B$ (blue) with no resource exchange between phases costs 1,205.979 with time-to-market 55.851. The duration of each phase for solution $P_B$ is compared with the corresponding phase of solution $P$. The difference is shown in percentage with respect to $P$.