Business Model Design as a Temporal Planning Problem: Preliminary Results

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

A number of formalisms and notations are available to design business models. Typically, a top-level model is created manually with one of these formalisms by a team of business experts, and the model is then analysed using simulations and model-based testing to find the most efficient configuration for resource allocation. The aim of this paper is to present a novel application domain for planning, together with a benchmark problem based on an industrial partner's experience: we encode the problem of finding the most efficient resource allocation for a business process as a planning problem. In particular, we consider a list of actions that various divisions in an organization can take, together with their associated costs, and we look for a solution that minimises time-to-market for a given project budget.

1 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. As described below, a number of formal languages are available for modelling business processes, and various tools are available to automate the analysis of the modelled workflows with the aim of minimising time-to-market, production costs, maximising return on investment, etc. To the best of our knowledge, however, the design of business models 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 minimization/maximization of various metrics. As a result, there is no guarantee that the business models obtained using this process are indeed the most efficient solution. Additionally, the exploration of different options is a very time-consuming task: each new model has to be developed and analysed separately, and alternative solutions need to be compared manually.

In this paper we argue that the design of business models 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 models.

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 for our domain.

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 a Gantt chart.

Related work: UML Activity Diagrams (UML 2012) and their associated profiles are among the most commonly used representations for business process modelling. A number of tools support UML modelling, including Eclipse plug-ins (`www.eclipse.org`) and IBM Rational Rose (`www.ibm.com/software/rational/`). Various approaches have been presented to analyse and verify activity diagrams (Eshuis 2006; Forster et al. 2007). The key difference with our work is that the input of all the existing tools (and for the methods mentioned below) is a *business model* that is manually created: using a planner an *optimised* model (in terms of resource allocation) is *automatically* generated.

Other business modelling languages include Petri Nets (van der Aalst 1998; Hinz, Schmidt, and Stahl 2005) and Event-Drive Process Chains (EPC) (van der Aalst 1999). Due to space constraints we refer to (List and Korherr 2006) for the description of additional methods.

Perhaps the closest work to ours is the translation of Status and Action Management (SAM) models to PDDL (Hoffmann, Weber, and Kraft 2012). This work has been developed in collaboration with SAP (`www.sap.com`), and is based on the use of an adaptation of the planner FF (Hoffmann and Nebel 2001). Using this approach, business process modellers can employ a planner to refine high-level actions and automatically generate sequences of steps expressed in BPMN (Business Process Modelling Notation (OMG 2012)). The main difference with our proposal is that we consider *cost* and *duration* of actions, together with planning *metrics*.

2 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*. In section 3 a mapping between the formal model and a temporal planning model is presented, using the concrete business process as a running example.

2.1 The Model

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* $\pi = (s_0 a_0 t_0)(s_1 a_1 t_1)(s_2 a_2 t_2) \dots s_n$*, 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:*

1. $s_0 = s_i$
2. $s_n = s_e$
3. $\forall j \geq 1, \forall s_k \in \{D(s_j)\} : s_k \in \pi \wedge k < j$
4. $\forall s_l \in \mathcal{S}, \mathcal{R}(s_l) \subseteq \mathcal{A}(s_l)$
5. $\sum_{j=0 \dots n-1} \mathcal{C}(s_j, a_j, t_j) \leq b$

In particular, conditions 1, 2, and 3 of Definition 2 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.

2.2 The MDSD 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.

3 Business Process as a 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. Furthermore, although the business process design could be seen as a scheduling problem, the setting we consider makes it a planning problem. In fact, the number of resources that can be allocated to each phase is not known in advance as well as the order in which phases are executed, their duration and how concurrency can be exploited.

In the following we present the main components of the PDDL domain and problem.

3.1 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, while 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, different tasks require people of different skills, and people have to be allocated to each phase/task. The number of people to allocate is not know in advance, and only an upper bound is provided. Third, the project cost needs to be computed, depending on the resource allocation, and maintained within the given budget. 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 `whole_project` action, shown in Figure 2. It is an envelope action, which encloses the whole process execution and whose duration is chosen by the planner. Therefore, in order to minimise the time-to-market, the planner will try to minimise the duration of the `whole_project` action, which, in turn, is used to set dynamically the value of `project_time` that is used to compute the cost of the project, as described in the following.

In order to model resource allocation, we distinguish between *employing* a resource (which defines the total number

$s \in \mathcal{S}$	$\mathcal{T}(s)$	$\mathcal{R}(s)$							
	% 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									

Table 1: Phases of the MDSD process

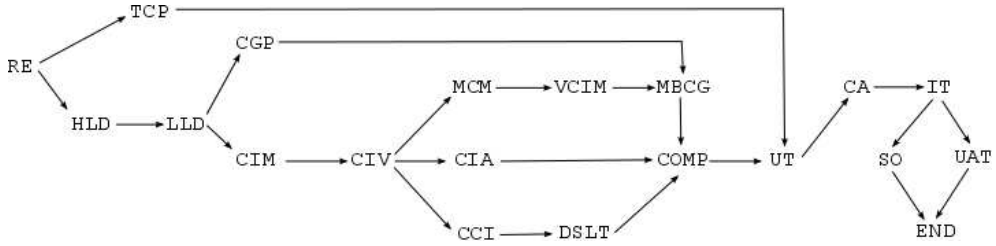

Figure 1: Dependency graph for the MDSD process

```
(:durative-action whole_project
 :parameters (?p - phase)
 :duration (<= ?duration (MAX_PLAN_LENGHT))
 :condition (and (at start (todo_project))
   (at start (is_last_phase ?p))
   (at end (completed ?p)))
 :effect (and (at start (running_project))
   (at start (not (todo_project)))
   (at end (project_completed))
   (at end (assign (project_time) ?duration))))
```

Figure 2: The `whole_project` action

of resources for each skill that will be used) and *allocating* a resource (which defines how resources are used throughout the process).

Employing Resources. Figure 3 shows the `employ` and `payOne` actions for domain experts (similar actions are defined for other skills). These actions are used for managing the amount of resources and for updating the project cost accordingly. 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). On the other hand, the `payOne` action, which is applied when the project is completed, is used to increment the project based on the daily cost of the resource and the duration of the project.

Allocating resources. The planner can use allocation (deallocation) actions to assign (release) resources of different skills to each phase of the business model before (after) performing that phase through the corresponding execution action. As an example, Figure 4 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 subtask 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 each skill required to perform the task (as shown in Table 1). Let us assume that task `p` requires skills `A`, `B`, `C`, and for each of them the amount of work is `pAdays`, `pBdays` and `pCdays`. If

```
(: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  payOne_DE
:parameters ()
:precondition (and (project_completed)
 (> (employed_DE) 0)
 (> (available_DE) 0))
:effect (and (increase (total_project_cost)
        (* (project_time) (per_day_cost_DE)))
             (decrease (employed_DE) 1)))
```

Figure 3: The `employ` and `payOne` actions

```
(: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)
  (> (allocated_DE ?p) 0))
 :effect (and
  (decrease (allocated_DE ?p) 1)
  (increase (available_DE) 1)))
```

Figure 4: An example of *allocation* and *deallocation* actions

the planner has allocated `pAres`, `pBres`, `pCres` resources to task `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 sub-tasks have been completed. On the other hand, the resources of skill j allocated for the task become available as soon as the sub-task requiring skill j terminates, even if the other sub-tasks are still executing.

Modelling such a scenario is not trivial, and the proposed solution is illustrated in Figure 7. For each phase of the business process, an *envelope* action is used, whose duration is left to the planner. Then, the task is split into k durative actions (where k is the number of different subtask required to complete the phase), whose duration depends on the resources previously allocated by the planner.

As an example, Figure 5 shows the envelope action to perform a phase, while Figure 6 shows the action for the subtask requiring domain experts. Furthermore, the model allows the presence of continuous non-linear constraints over resources, as the business model design is expressed in terms of a temporal planning problem.

```
(: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)))))
```

Figure 5: An example of *envelope-execution* action

```
(:durative-action executive_subtask_DE
 :parameters (?p - phase)
 :duration (= ?duration (/
  (duration_subtask_DE ?p) (allocated_DE ?p)))
 :condition (and (at start (todosubtask_DE ?p))
  (over all (doing ?p))
  (at end (>= (employed_DE) (allocated_DE ?p)))
 :effect (and (at start (not (todosubtask_DE ?p)))
  (at end (completed_DE ?p))))
```

Figure 6: An example of *subtask-execution* action

3.2 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 employed 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))`.

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

Table 2: Normalized costs of resources of different skills

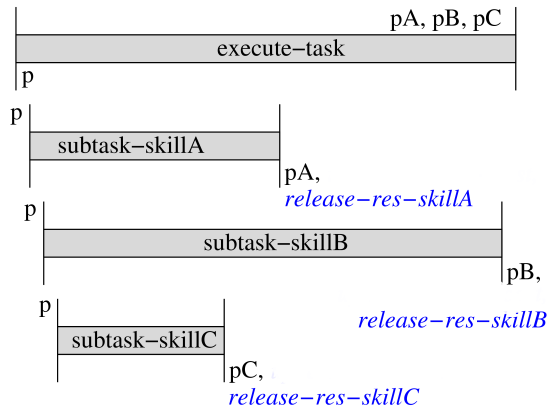

Figure 7: Envelope action for task execution

A fragment of the PDDL problem is shown in Figure 8, where we show key elements for phase Compilation (COMP) and resource Developer (D).

The budget is fixed to $3,000$. The `per_day_cost_D`, the `employment_cost_D` and the maximum number of Developers that can be employed are defined according to the normalized costs of resources shown in Tab. 2.

Then, the predicate `(todosubtask_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. The duration of the phase COMP and its three subtasks is initiated through predicates `upper_bound COMP` and `duration_subtask` respectively, as explained in Sec. 2.2. Finally, as described above, the goal is to find a feasible Business Process Execution according to Def. 2 minimizing the time-to-market.

4 Experimental Results

Given the model detailed above, we used the POPF[1] planner (Coles et al. 2010) to synthesise a solution. Figure 9 provides a Gantt chart of an efficient solution found by POPF in less than 30 minutes, working on a x64 Linux machine equipped with 6 GB of RAM. The MDSD process requires about 15 days, with a total cost of 2.686 for people, against an available budget of 3.000. Notice that the maximum number of employees for each skill has been limited to 5.

An approach typically followed by managers - and confirmed by our industrial partner - 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. Conversely, the key element of the plan is the optimised parallel execution of several phases

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

```
(= (MAX_PLAN_LENGHT) 100) ;;sum of T(s) in Tab.1
(is_last_phase END)
(= (budget) 3000)
(= (total_project_cost) 0)
(= (project_time) 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
(todosubtask_D CCI) (todosubtask_D MBCG)
(todosubtask_D DSLT)  (todosubtask_D COMP)
(todosubtask_D UT)  (todosubtask_D CA)

;; graph of Fig.1
(depends COMP CIA MBCG DSLT)

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

(= (upper_bound HLD) 5)
(= (duration_subtask_D COMP) 5)

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

Figure 8: An extraction of PDDL problem for the `COMP` phase

as the model allows the switching of a resource between phases even when they are still on-going. To give a few examples, Figure 11 focuses on phases Modelling Component Implementation (MCM), Component Interface Assembly (CIA), and Coding of Component Implementation (CCI) of the Gantt chart depicted in Figure 9. These phases start in parallel on the sixth day of the project execution. In particular, the phases Modelling Component Implementation and Component Interface Assembly have the same skill requirements as shown in Table 1, that is Software Architects and MDE Experts.

In order to speed up the execution of these parallel phases (on which the next Compilation phase depends) the planner decides to allocate the available Modellers on the Component Interface Assembly task until it ends, then the planner switches them on the same task of the phase Modelling Component Implementation. The same happens for tasks requiring MDE Experts. On the other hand, all these three phases need to complete a task which involves Software Architects (boxes with a green vertical texture in Figure 11). As a consequence, the planner decides to assign 4 out of 5 Software Architects to complete the task of phase Coding of Component Implementation. Simultaneously, it first assigns

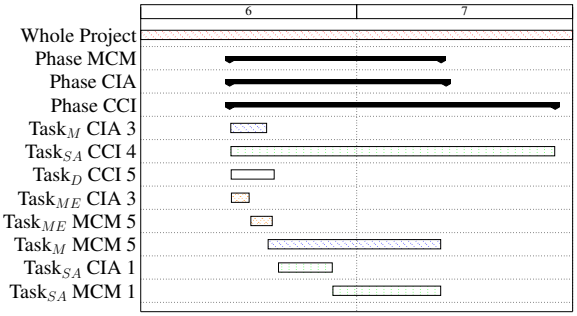

Figure 11: A focus on the Gannt Charts of Figure 9. Relevant tasks are highlighted with a green vertical texture when referring to Software Architects (SA), with an orange crosshatched texture to refer MDE Experts (ME) and with a blue slanted texture in referring to Modellers (M).

the remaining Software Architect to phase Component Interface Assembly, and then to the phase Modelling Component Implementation. This non-trivial allocation of resources between tasks allows the planner to compute an optimised task duration to minimize the project time maximising the budget usage. Finally, a less efficient plan is provided in Figure 10 that requires 57 days to complete the MDSD process. Notice that, in comparison with the optimised solution of Figure 9, this plan recruits less employees but requires more budget to complete the process. This example shows that there is much room for optimisation in this domain using automated planning.

5 Conclusion and Future Work

In this paper we have presented how the problem of designing a business model can be cast as a planning problem. Our experience at Tata Consultancy Services over nearly two decades has shown that the correct design of business models can make the difference between successful and unsuccessful projects. However, in spite of the large body of work available for the verification of *existing* business processes (see for instance (Bianculli, Ghezzi, and Spoletini 2007) and references therein), there is currently no support for the *automatic* generation of business processes. We have presented a first approach on this direction, modelling the design of the business process as a temporal planning domain and finding plans that minimise time-to-market for a given project budget.

The abstract model described in Section 2.1 captures the key elements of most modelling languages described in the Introduction: we are currently working on automatic translators from mainstream notations (OMG 2012; Clark, Barn, and Oussena 2011) to PDDL, and for the future we envisage automatic tool support to enable the communication between modelling tools and an appropriate planner.

A natural future work for extending the proposed model is to exploit the expressive power of PDDL3.0 (Gerevini and Long 2006) and use *preferences* to take into account soft constraints and model further Key Performacen Indicators. Finally, to deal with the multi-objective optimisation involved in this problem and provide richer suggestions to business organisation, the use of *Pareto frontiers* (Sroka and Long 2012) appears promising.

References

Bianculli, D.; Ghezzi, C.; and Spoletini, P. 2007. A model checking approach to verify BPEL4WS workflows. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, SOCA '07, 13–20. Washington, DC, USA: IEEE Computer Society.

Clark, T.; Barn, B. S.; and Oussena, S. 2011. LEAP: a precise lightweight framework for enterprise architecture. In *Proceeding of the 4th Annual India Software Engineering Conference, ISEC 2011*, 85–94. ACM.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Eshuis, R. 2006. Symbolic model checking of UML activity diagrams. *ACM Trans. Softw. Eng. Methodol.* 15(1):1–38.

Forster, A.; Engels, G.; Schattkowsky, T.; and Van Der Straeten, R. 2007. Verification of business process quality constraints based on visual process patterns. In *Theoretical Aspects of Software Engineering, 2007.*, 197 –208.

Gerevini, A., and Long, D. 2006. Preferences and soft constraints in pddl3. In *Proceedings of ICAPS Workshop on Planning with Preferences and Soft Constraints*.

Hinz, S.; Schmidt, K.; and Stahl, C. 2005. Transforming BPEL to Petri nets. *Business Process Management* 220–235.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)* 14:253–302.

Hoffmann, J.; Weber, I.; and Kraft, F. 2012. SAP speaks PDDL: Exploiting a software-engineering model for planning in business process management. *Journal of Artificial Intelligence Research* 44:587–632.

List, B., and Korherr, B. 2006. An evaluation of conceptual business process modelling languages. In *Proceedings of the 2006 ACM symposium on Applied computing*, 1532–1539. ACM.

OMG. 2012. OMG business process model and notation. `www.bpmn.org/`. Last accessed: 13 November 2012.

Sroka, M., and Long, D. 2012. Exploring metric sensitivity of planners for generation of pareto frontiers. In *STAIRS*, volume 241 of *Frontiers in Artificial Intelligence and Applications*, 306–317.

UML. 2012. OMG formal specifications. `http://www.omg.org/spec/`. Last accessed: 12 November 2012.

van der Aalst, W. 1998. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers* 8(01):21–66.

van der Aalst, W. 1999. Formalization and verification of event-driven process chains. *Information and Software technology* 41(10):639–650.

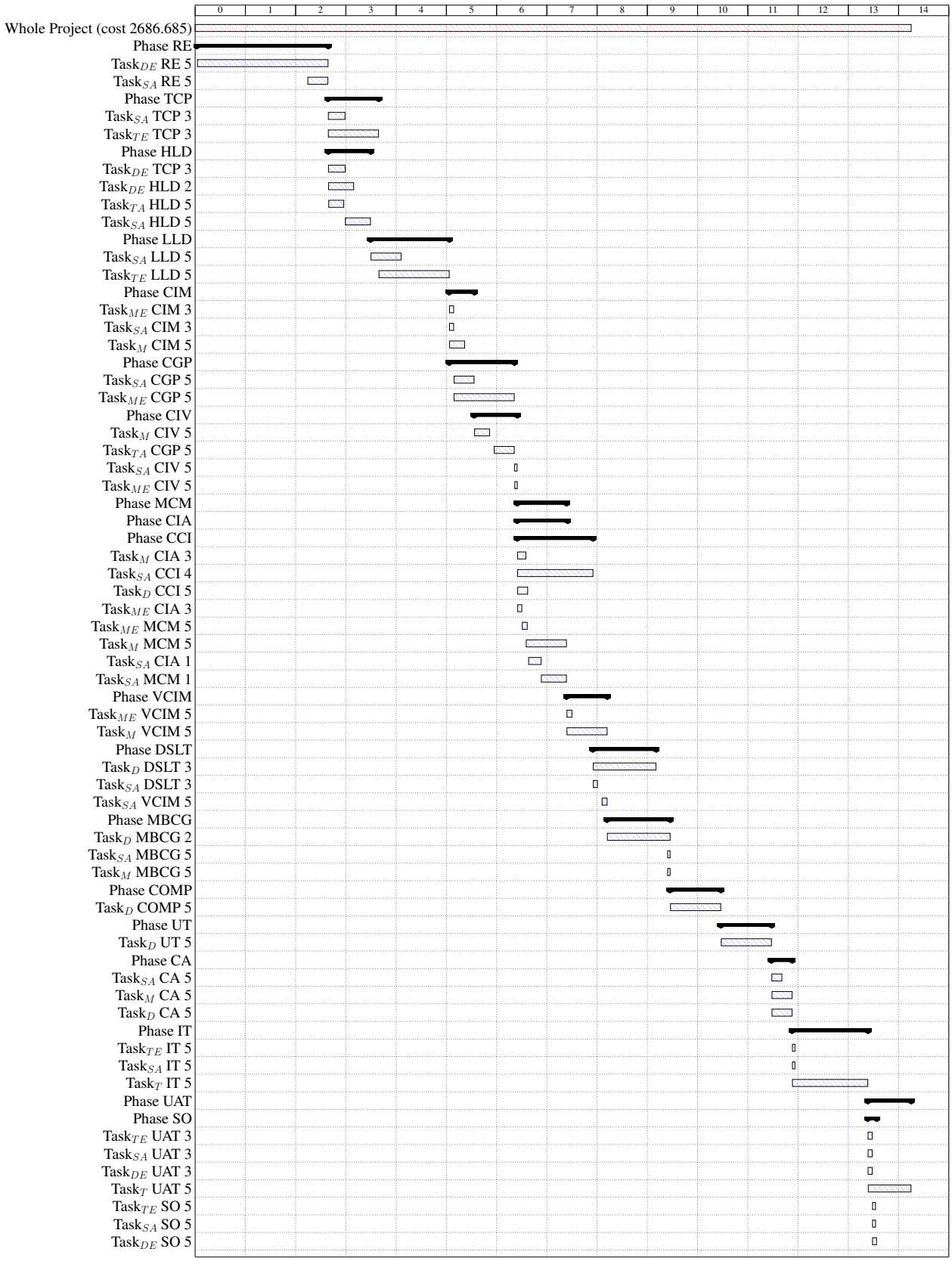

Figure 9: Gannt Charts of a POPF solution. The black bar represents the overall phase duration whereas a blue bar Task$_X Y K$ represents the execution of task X of the phase Y using K allocated employees.

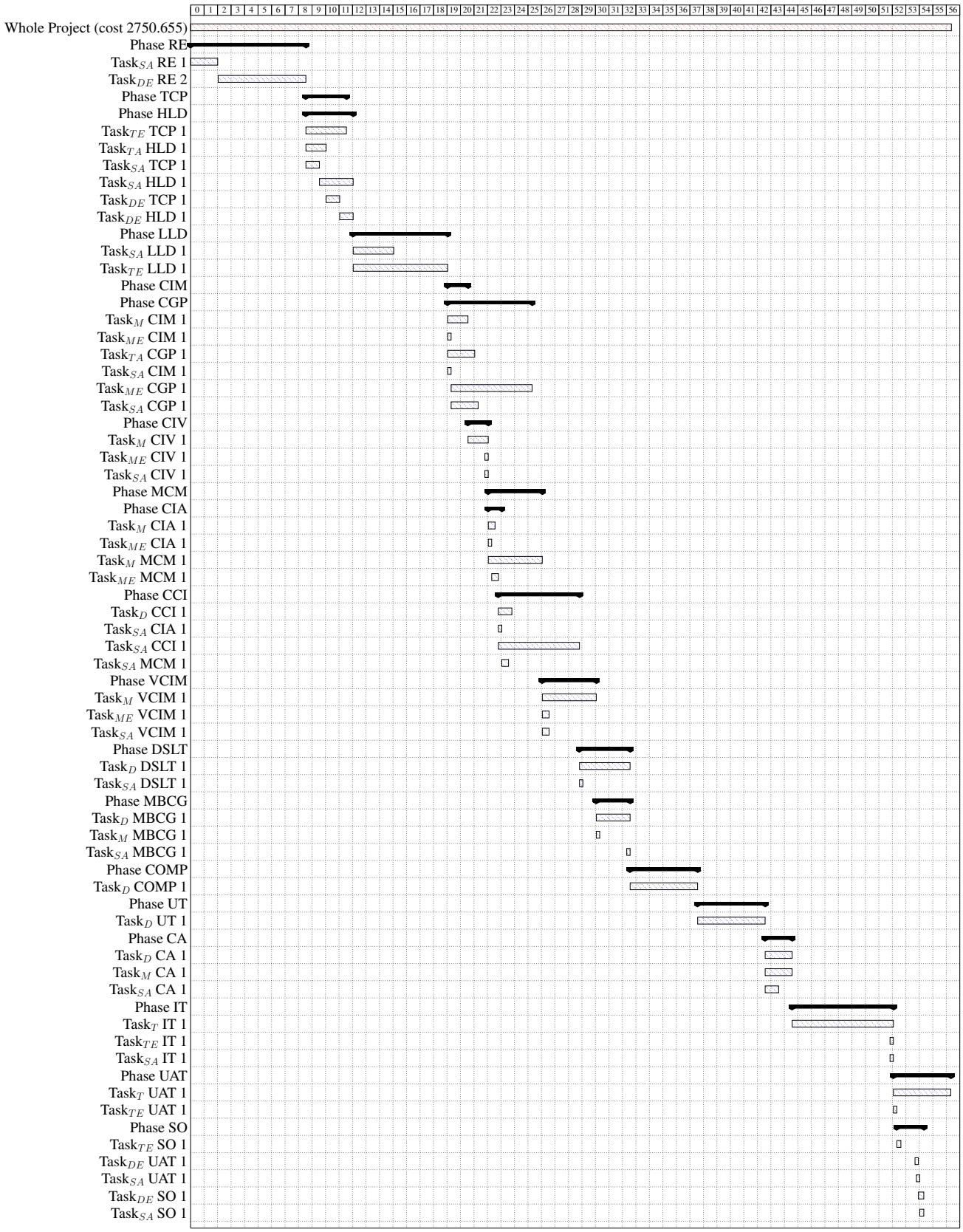

Figure 10: Gannt Charts of a POPF *feasible* solution. The black bar represents the overall phase duration whereas a blue bar Task$_{X}YK$ represents the execution of task X of the phase Y using K allocated employees.