Resource Allocation with Dependencies in Business Process Management Systems*

Giray Havur, Cristina Cabanillas, Jan Mendling, and Axel Polleres

Vienna University of Economics and Business, Austria
{firstname.lastname}@wu.ac.at

Abstract. Business Process Management Systems (BPMS) facilitate the execution of business processes by coordinating all involved resources. Traditional BPMS assume that these resources are *independent* from one another, which justifies a greedy allocation strategy of offering each work item as soon as it becomes available. In this paper, we develop a formal technique to derive an optimal schedule for work items that have *dependencies* and *resource conflicts*. We build our work on Answer Set Programming (ASP), which is supported by a wide range of efficient solvers. We apply our technique in an industry scenario and evaluate its effectiveness. In this way, we contribute an explicit notion of resource dependencies within BPMS research and a technique to derive optimal schedules.

Keywords: Answer Set Programming, optimality, resource allocation, resource requirements, work scheduling

1 Introduction

Business Process Management Systems (BPMS) have been designed as an integral part of the business process management (BPM) lifecycle by coordinating all resources involved in a process including people, machines and systems [1]. At design time, BPMS take as input a business process model enriched with technical details such as role assignments, data processing and system interfaces as a specification for the execution of various process instances. In this way, they support the efficient and effective execution of business processes [2].

It is an implicit assumption of BPMS that work items are *independent* from one another. If this assumption holds, it is fine to put work items in a queue and offer them to available resources right away. This approach of resource allocation, can be summarized as a greedy strategy. However, if there are *dependencies* between work items, this strategy can easily become suboptimal. Some domains like engineering or healthcare have a rich set of activities for which various resources, human and non-human, are required at the same time. Resource conflicts have often the consequence that working on one work item blocks resources such that other work items cannot be worked on. This observation emphasizes the need for techniques to make better use of existing resources in business processes [3].

* Funded by the Austrian Research Promotion Agency (FFG) grant 845638 (SHAPE).

In this paper, we address current limitations of BPMS with respect to taking such resource constraints into account. We extend prior research on the integration of BPMS with calendars [4] to take dependencies and resource conflicts between work items into account. We develop a technique for specifying these dependencies in a formal way in order to derive a globally optimal schedule for all resources together. We define our technique using Answer Set Programming (ASP), a formalism from logic programming that has been found to scale well for solving problems as the one we tackle [5]. We evaluate our technique using an industry scenario from the railway engineering domain. Our contribution to research on BPMS is an explicit notion of dependence along with a technique to achieve an optimal schedule.

The paper is structured as follows. Section 2 presents and analyzes an industry scenario. Section 3 conceptually describes the resource allocation problem. Section 4 explains our ASP-based solution and how it can be applied to the industry scenario. Section 5 evaluates the solution. Section 6 discusses related work. Section 7 summarizes the conclusions of the work and the future steps.

2 Motivation

In the following, we describe an industry scenario that leads us to a more detailed definition of the resource allocation problem and its complexity.

2.1 Industry Scenario

A company that provides large-scale technical infrastructure for railway automation requires rigorous testing for the systems deployed. Each system consists of different types and number of hardware that are first set up in a laboratory. This setup is executed by some employees specialized in different types of hardware. Afterwards, the simulation is run under supervision.

Figure 1 depicts two process models representing the setup and run phases of two tests. We use (timed) Petri nets [6] for representing the processes. The process activities are represented by transitions (a_i). The number within square brackets next to the activities indicates their (default maximum) duration in generic time units (TU). The numbers under process names indicate the starting times of the process executions: 8 TU for Test-1 and 12 TU for Test-2. The processes are similar for all the testing projects but differ in the activities required for setting up the hardware as well as in the resource requirements associated with them. Certain resources can only be allocated to activities during working periods, i.e., we want to enforce time intervals (so called *breaks*) where some resources are not available. In our scenario, no resource is available in the intervals $[0, 8)$,$[19, 32)$,$[43, 56)$, and $[67, 80)$.

For completing tests, the available non-human resources in the organization include 13 units of space distributed into 2 laboratories (Table 1) and several units of 3 types of hardware (Table 2). The human resources of the company are specialized in the execution of specific phases of the two testing projects, whose

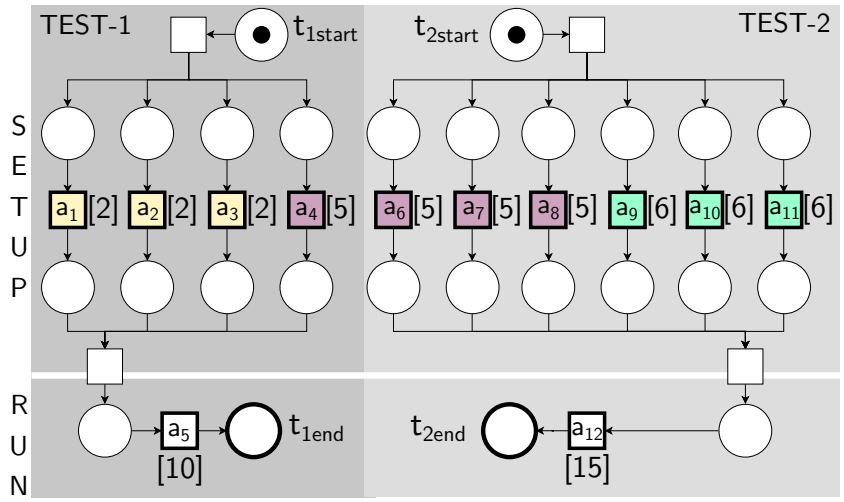


Fig. 1: Workflow for two projects

activities are able to complete in a specific time. Table 3 shows available resources in different process phases and therefore, their ability to conduct certain activities along with their years of experience in the company in square brackets.

The requirements on the use of such resources in the process activities are shown in Table 4. Each process activity requires a specific set of resources for its completion. For instance, three of the activities involved in the setup of Test-1 require 1 employee working on 1 unit of the hardware HW-1 in a laboratory; 1 setup activity requires 1 employee working on 1 unit of the hardware HW-2 in a laboratory; and the run activity requires 4 employees. Besides, a test can only be executed if the whole setup takes place in the same laboratory.

The aim in this scenario is to optimize the overall execution time of simultaneous tests and consequently, the space usage in the laboratories.

	$LAB-1$	$LAB-2$
Space	4	9

Table 1: Available space in labs

$Type$	$Units$
$HW1$	hw1a, hw1b, hw1c
$HW2$	hw2a, hw2b, hw2c, hw2d
$HW3$	hw3a, hw3b, hw3c

Table 2: Available hardware (HW)

	$Test-1$		$Test-2$	
	$Setup$	Run	$Setup$	Run
Glen[7]	✓	✓		
Drew[7]		✓		
Evan[3]		✓		
Mary[5]		✓	✓	
Kate[6]			✓	✓
Amy[8]	✓		✓	✓

Table 3: Specialization of employees

2.2 Insights

The resource allocation problem[1] deals with the assignment of resources and time intervals to the execution of activities. The complexity of resource allocation in BPM arises from coordinating the explicit and implicit dependencies across a broad set of resources and activities of processes as well as from solving potential conflicts on the use of certain resources. As we observe in our industry scenario, such dependencies include, among others: (i) resource requirements, i.e., the characteristics of the resources that are involved in an activity (e.g., roles or skills) (cf. Table 3); (ii) temporal requirements. For instance, the duration of the activities may be static or may depend on the characteristics of the set of resources involved in it, especially for collaborative activities in which several employees work together (such as for the activities of the run phase of a testing process). Furthermore, resource availability may not be unlimited (e.g., break calendars). In addition, resource conflicts may emerge from interdependencies between requirements, e.g., activities might need to be executed within a specific setting which may be associated with (or share resources with) the setting of other activities (e.g., all the setup activities of a testing process must be performed in the same laboratory).

A resource allocation is *feasible* if (1) activities are scheduled with respect to time constraints derived from activity durations and control flow of the process model, and (2) resources are allocated to scheduled activities in accordance with resource availability and resource requirements of activities. This combinatorial problem for finding a feasible resource allocation under constraints is an *NP-Complete* problem [7]. However, organizations generally pursue an optimal allocation of resources to process activities aiming at minimizing overall execution times or costs, or maximizing the usage of the resources available. In presence of objective functions the resource allocation problem becomes Δ_2^P [8].

3 Conceptualization of the Resource Allocation Problem

Fig. 2 illustrates our conceptualization of the resource allocation problem. We divide it into three complexity layers related to the aforementioned dependencies

[1] Commonly referred as *scheduling*.

	Activities	*Requirements*
Test-1	$a_1 - a_3$	1 *Employee*:Setup-1, 1 *Hardware*:HW-1, 1 *Lab*:a_1-a_4 same lab
	a_4	1 *Employee*:Setup-1, 1 *Hardware*:HW-2, 1 *Lab*:a_1-a_4 same lab
	a_5	4 *Employee*:Run-1, after execution(a.e.) release the lab for a_1-a_4
Test-2	$a_6 - a_8$	1 *Employee*:Setup-2, 1 *Hardware*:HW-2, 1 *Lab*:a_6-a_{11} same lab
	$a_9 - a_{11}$	1 *Employee*:Setup-2, 1 *Hardware*:HW-3, 1 *Lab*:a_6-a_{11} same lab
	a_{12}	2 *Employee*:Run-2 (hasExp>5), a.e. release the lab for a_6-a_{11}

Table 4: Activity requirements

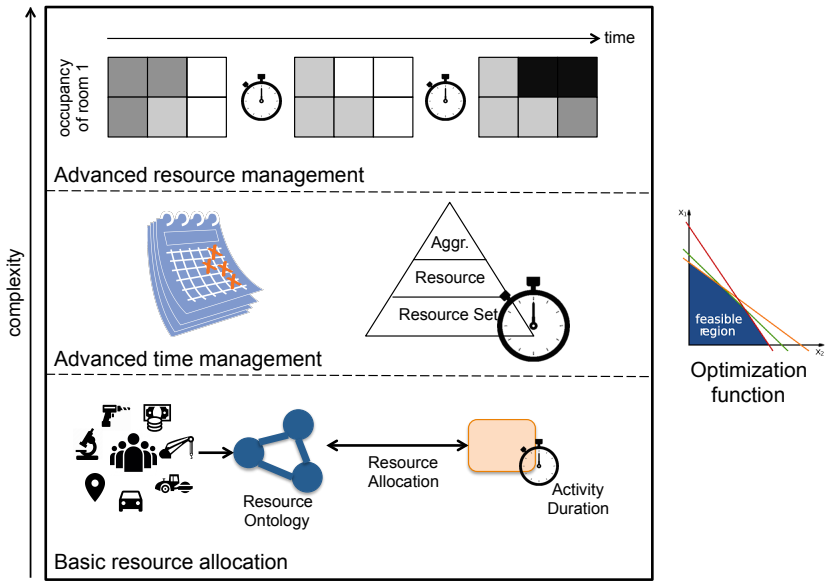
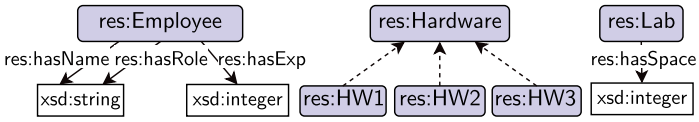


Fig. 2: Resource allocation in business processes




```
:glen a res:Employee; res:name "Glen"; res:hasExp 7;
      res:hasRole "Setup_Test1", "Run_Test1" .
:drew a res:Employee; res:name "Drew";
      res:hasExp 7; res:hasRole "Run_Test1" .
 ...
:lab1 a res:Lab, res:hasSpace 4 .
:lab2 a res:Lab, res:hasSpace 9 .
:hw1a a res:HW1. :hw1b a res:HW1. :hw1c a res:HW1.
:hw2a a res:HW1. :hw2b a res:HW2. :hw2c a res:HW2. :hw2d a res:HW2.
:hw3a a res:HW3. :hw3b a res:HW3. :hw3c a res:HW3.
```

Fig. 3: Resource ontology and example instantiation

and resource conflicts. Optimization functions can be applied to all types of allocation problems. This model has been defined from the characteristics identified in our industry scenario as well as in related literature [9].

3.1 Basic Resource Allocation

Three elements are involved in a basic resource allocation, namely: a model that stores all the information required about the resources available, information about the expected duration of the process activities, and a language for defining the restrictions that characterize the allocation.

Resource Ontology As a uniform and standardized representation language, we suggest the use of RDF Schema (RDFS) [10] to model organizational information and resources. Fig. 3 illustrates a sample RDFS ontology, in which a *resource* is characterized by a *type* and can have one or more *attributes*. In particular, any resource type (e.g. *Employee*) is a subclass of rdfs:Resource. The attributes are all of type rdf:Property; domain (rdfs:domain) and range of attributes are indicated with straight arrows labeled with the attribute name, whereas dashed arrows indicate an rdfs:subclassOf. There are three different types of resources: *Employee*, *Hardware* and *Lab*, where *Hardware* has three resource subtypes. Employees have attributes for their name (*hasName*), role(s) (*hasRole*) and experience level (*hasExp*) in the organization (number of years). Labs provide a certain amount of space for experiments (*hasSpace*). An instantiation of the ontology is described at the bottom of the figure using the RDF Turtle syntax [11]. This instantiation represents Tables 1-3 of the industry scenario.

Activity Duration Resource allocation aims at properly distributing available resources among running and coming work items. The main temporal aspect is determined by the expected duration of the activities. The duration can be predefined according to the type of activity or calculated from previous executions, usually taking the average duration as reference. This information can be included in the executable process model as a property of an activity (e.g. with BPMN [12]) or can be modelled externally. In either case, it has to be accessible by the allocation algorithm.

Resource Allocation Resource allocation can be seen as a two-step definition of restrictions. First, the so-called *resource assignments* must be defined, i.e., the restrictions that determine which resources can be involved in the activities [13] according to their properties. The outcome of resource assignment is one or more[2] *resource sets* with the set of resources that can be potentially allocated to an activity at run time. The second step assigns cardinality to the resource sets such that different settings can be described, e.g. for the execution of activity a_1, 1 employee with role *setup-1*, 1 hardware of type *HW2*, and 1 unit space of a laboratory are required.

There exist languages for assigning resource sets to process activities [13–16]. However, cardinality is generally disregarded under the assumption that only one resource will be allocated to each process activity. This is a limitation of current BPMS that prevents the implementation of industry scenarios like the one described in Section 2.1.

3.2 Advanced Time Management

This layer extends the temporal aspect of resource allocation by taking into account that: (i) resource availability affects allocation, and that (ii) the resource sets allocated to an activity may affect its duration. Regarding resource availability, calendars are an effective way of specifying different resource availability

[2] Since several sets of restrictions can be provided, e.g. for activity a_1 resources with either role r_1 or skill s_1 are required.

status, such as available, unavailable, occupied/busy or blocked [9]. Such information must be accessible by the resource allocation module. As for the variable activity durations depending of the resource allocation, three specificity levels can be distinguished:

- *Resource-set-based duration*, i.e., a triple $(activity, resourceSet, duration)$ stating the (minimum/average) amount of time that it takes to the resources within a specific resource set (i.e., cardinality is disregarded) to execute instances of a certain activity. For instance, $(a_1, technician, 6)$ specifies that people with the role *technician* need (at least/on average) 6 TU to complete activity a_1, assuming that *technician* is an organisational role.
- *Resource-based duration*, i.e., a triple $(activity, resource, duration)$ stating the (minimum/average) amount of time that it takes to a concrete resource to execute instances of a certain activity. For instance, $(a_1, John, 8)$ specifies that *John* needs (at least/on average) 8 TU to complete activity a_1.
- *Aggregation-based duration*, i.e., a triple $(activity, group, duration)$ stating the (minimum/average) amount of time that it takes to a specific group to execute instances of a certain activity. In this paper, we use *group* to refer to a set of human resources that work together in the completion of a work item, i.e., cardinality is considered. Therefore, a *group* might be composed of resources from different resource sets which may not necessarily share a specific resource-set-based duration. An aggregation function must be implemented in order to derive the most appropriate duration for an activity when a group is allocated to it. The definition of that function is up to the organization. For instance, a group might be composed of $(John, Claire)$, where *John* has an associated duration of 8 TU for activity a_1 and *Claire* does not have a specific duration but she has role *technician*, with an associated duration of 6 TU for activity a_1. Strategies for allocating the group to the activity could be to consider the maximum time needed for the resources involved (i.e., 8 TU), or to consider the mean of all the durations (i.e., 7 TU) assuming that the joint work of two people will be faster than one single resource completing all the work.

3.3 Advanced Resource Management

The basic resource allocation layer considers resources to be *discrete*, i.e. they are either fully available or fully busy/occupied. This applies to many types of resources, e.g. people, software or hardware. However, for certain types of non-human resources, availability can be partial at a specific point in time. Moreover, they may have other *fluent* attributes. For instance, *cumulative resources* are hence characterized by their *dynamic* attributes and they can be allocated to more than one activity at a time, e.g. in Fig. 2 there is a resource *room 1* whose occupancy changes over time.

We use the ASP solver *clasp* [17] due to its efficiency for our experiments. This allows us to use integer variables as attributes. There are also other extensions of ASP such as FASP [18] that adds the power to model continuous variables.

3.4 Optimization Function

Searching for (the existence of) a feasible resource allocation ensures that all the work items can eventually be completed with the available resources. However, typically schedules should also fulfill some kind of optimality criterion, most commonly completion of the schedule in the shortest possible overall time. Other optimization criteria may involve for instance costs of the allocation of certain resources to particular activities, etc.

Given such an optimization criterion, there are greedy approaches [19] providing a substantial improvements over choosing any feasible schedule, although such techniques depend on heuristics and may not find a globally optimal solution for complex allocation problems.

We refer to [20] for further information on various optimization functions, but emphasize that our approach will in principle allow arbitrary optimization functions and finds optimal solutions – similar in spirit to encodings of cost optimal planning using ASP [21].

4 Implementation with ASP

Answer Set Programming (ASP) [17] is a declarative (logic-programming-style) paradigm. Its expressive representation language, ease of use, and computational effectiveness facilitate the implementation of combinatorial search and optimization problems (primarily *NP-hard*). Modifying, refining, and extending an ASP program is uncomplicated due to its strong declarative aspect.

An *ASP program* Π is a finite set of rules of the form:

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n. \tag{1}$$

where $n \geq m \geq 0$ and each $A_i \in \sigma$ are (function-free first-order) atoms; if A_0 is empty in a rule r, we call r a constraint, and if $n = m = 0$ we call r a fact.

Whenever A_i is a first-order predicate with variables within a rule of the form (1), this rule is considered as a shortcut for its *grounding* $\mathit{ground}(r)$, i.e., the set of its ground instantiations obtained by replacing the variables with all possible constants occurring in Π. Likewise, we denote by $\mathit{ground}(\Pi)$ the set of rules obtained from grounding all rules in Π. Sets of rules are evaluated in ASP under the so-called stable-model semantics, which allows several models, so called *answer sets* (cf. [22] for details).

ASP Solvers typically first compute a subset of $\mathit{ground}(\Pi)$ and then use a DPLL-like branch and bound algorithm to find answer sets for this ground program. We use the ASP solver *clasp* [17] for our experiments as it has proved to be one of the most efficient implementations available [23].

As syntactic extension, in place of atoms, *clasp* allows set-like *choice expressions* of the form $E = \{A_1, \ldots, A_k\}$ which are true for any subset of E; that is, when used in heads of rules, E generates many answer sets, and such rules are often referred to as *choice rules*. Another extension supported in *clasp* are optimization statements [17] to indicate preferences between possible answer sets:

$$\#\mathit{minimize}\ \{A_1 : \mathit{Body}_1 = w_1, \ldots, A_m : \mathit{Body}_m = w_m @ p\}$$

associates integer weights (defaulting to 1) with atoms A_i (conditional to $Body_i$ being true), where such a statement expresses that we want to find only answer sets with the smallest aggregated weight sum; again, variables in $A_i : Body_i = w_i$ are replaced at grounding w.r.t. all possible instantiations. Several optimization statements can be introduced by assigning the statement a priority level p. Reasoning problems including such weak constraints are Δ_2^P-complete.

Finally, many problems conventiently modelled in ASP require a boundary parameter `k` that reflects the size of the solution. However, often in problems like planning or model checking this boundary (e.g. the plan length) is not known upfront, and therefore such problems are addressed by considering one problem instance after another while gradually increasing this parameter `k`. Re-processing repeatedly the entire problem is a redundant approach, which is why incremental ASP (iASP) [17] natively supports incremental computation of answer sets; the intuition is rooted in treating programs in program slices (extensions). In each incremental step, a successive extension of the program is considered where previous computations are re-used as far as possible.

A former version of our technique is detailed in [5]. We enhance our encoding in three folds: (1) basic resource allocation supporting multiple business processes with multiple running instances, (2) definition of *advanced resource management* concepts, and (3) definition of *advanced time management* concepts. The entire ASP encoding can be found at `http://goo.gl/Q7B2t4`.

4.1 Basic Resource Allocation

This program schedules the activities in business processes described as timed Petri nets (cf. the generic formulation of 1-safe Petri Nets [5, Section 4]) and allocates resources to activities with respect to activity-resource requirements. To achieve this, the program finds a firing sequence between initial and goal places of given processes, schedules the activities in between, and allocates resources by complying with resource requirements. In our program, a firing sequence is represented as predicates `fire(a,b,i,k)`, which means that an activity `a` of a business process `b` in instance `i` is fired at step `k`. Starting time of each activity in the firing sequence is derived from the time value accumulated at the activity's input place `p`. A time value at a place `p` is represented by the predicate `timeAt(p,c,b,i,k)`, where `c` is the time value.

A *resource set* is defined as a rule that derives the members of the set that satisfy a number of properties. These properties can be class memberships or resource attributes defined in resource ontology(cf. Section 3.1). Note that, any resource ontology described in RDF(S) can be easily incorporated/translated into ASP [24]. A resource set is represented with the predicate `resourceSet(R,id)`, where `R` is a set of discrete resources and `id` is the identifier of the set. We explain the following resource sets following our industry scenario:
All employees that can take part in the setup phase of Test-1:

```
resourceSet(R,rs_set1):-employee(R), hasRole(R,setup1).
```

All employees that can take part in the run phase of Test-2 and have a working experience greater than 5 years:

```
resourceSet(R,rs_ex2):-employee(R), hasRole(R,run2), hasExp(E), E>5.
```

All hardware resources of type HW2:
`resourceSet(R,rs_h2):-hardware2(R).`

After defining resource sets, we define *resource requirements* of an activity `a` with the predicate `requirement(a,id,`n`)` where `id` refers to a specific resource set and n is the number of resources that activity `a` requires from this set. For instance, `requirement(`$\mathtt{a}_{12}$`,rs_ex2,2)` means that activity $\mathtt{a}_{12}$ requires 2 resources from the resource set `rs_ex2`. The resource requirements that we support include typical access-control constraints [13]. In particular, *Separation of duties (SoD)* and *binding of duties(BoD)* are implemented in our program by using the predicate `separateDuties(`$\mathtt{a}_1,\mathtt{b}_1,\mathtt{a}_2,\mathtt{b}_2$`)`, which separates the resources allocated to the activity $\mathtt{a}_1$ of process $\mathtt{b}_1$ from the resources allocated to $\mathtt{a}_2$ of $\mathtt{b}_2$; and `bindDuties(`$\mathtt{a}_1,\mathtt{b}_1,\mathtt{a}_2,\mathtt{b}_2$`)`, which binds the resources allocated to the activity $\mathtt{a}_1$ of process $\mathtt{b}_1$ with the resources allocated to $\mathtt{a}_2$ of $\mathtt{b}_2$.

4.2 Advanced Time Management

Default durations of activities are defined in the timed Petri nets and represented as `activityDuration(T,D)` in our program. This default duration can be overwritten by `d` when any resource `r` that belongs to a resource set `rs` is assigned to a certain activity `a` of the process `b` by using the predicate `rSetActDuration(rs,a,b,d)`. In a similar fashion, the default duration can be overwritten by a new value `d` when a certain resource `r` is assigned to a certain activity `a` of the process `b` by using the predicate `resActDuration(r,a,b,d)`. The order ($>$) preferred in activity time is `resActDuration`$>$`rSetActDuration`$>$`activityDuration`. This is especially useful when a resource or a resource set is known to execute a particular activity in a particular amount of time, which can be different from the default duration of the activity.

As one activity can be allocated to a group of resources (cf. Section 3.2), an aggregation method might be needed. Our default aggregation method identifies the maximum duration within the group and uses it for allocation. This method can be modified with different aggregation options that fit in the purpose of allocation scenario.

In many real-life projects, certain resources are only available during the working periods (a.k.a. *break calendars*). We model this by `break(rs,` $\mathtt{c}_1,\mathtt{c}_2$`)` that forbids allocation of resources in the resource set `rs` between time $\mathtt{c}_1$ and $\mathtt{c}_2$, where $\mathtt{c}_1 < \mathtt{c}_2$.

For business process instances and their activities, (optionally, max. or min.) starting or ending times can be defined using the following predicates:
`actStarts(o,a,b,i,c)`, i.e. activity `a` in business process `b` of instance `i`, starts <o> at `c`; `actEnds(o,a,b,i,c)`, i.e. activity `a` in business process `b` of instance `i`, ends <o> at `c`; `bpiStarts(o,b,i,c)`, i.e. business process `b` of instance `i`, starts <o> at `c`; `bpiEnds(o,b,i,c)`, i.e. business process `b` of instance `i`, ends <o> at `c`; where `o`$\in$ {`strictly,earliest,latest`}.

4.3 Advanced Resource Management

A cumulative resource has an integer value attribute describing the state of the resource. This value can increase or decrease when the resource is *consumed* or

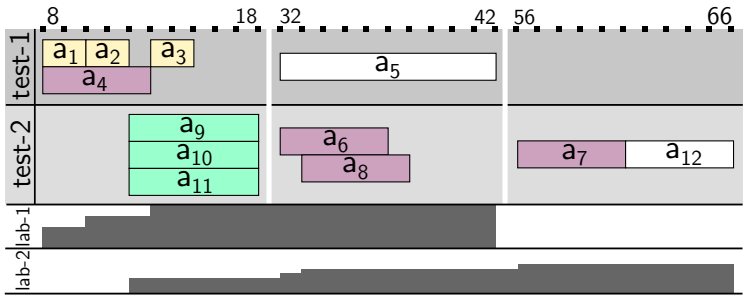


Fig. 4: Optimal resource allocation for our industry scenario

generated by an activity requiring it. Definition of cumulative resource sets have one extra term for this reason: `resourceSet(R,V,id)`, where `R` is the set of cumulative resources, `V` is the set of their initial value and `id` is the identifier of the resource set. For example:
Lab space set:

```
resourceSet(R,V,lab_space):-lab(R),hasSpace(R,V).
```

Resource requirements are defined like for discrete resources, where n is the amount of resource consumed or generated. For instance, `requirement(`a_1`,lab_space,1)` consumes 1 unit of lab space when a_1 is allocated, whereas `requirement(`a_{12}`,lab_space,-6)` releases 6 units of space by the time a_{12} is completed.

Resource blocking functionality allows us to block some resources between the execution of two activities in a process. A blocked resource is not allowed to be allocated by an activity in this period. `block(`a_1`,`a_2`,id,n)` blocks `n` amount of resources in the resource set `id` from the beginning of a_1 to beginning of a_2.

4.4 Optimization Function

As aforementioned, the ASP solver *clasp* allows defining objectives as cost functions that are expressed through a sequence of `#minimize` statements. In our encoding, we ensure time optimality of our solutions using a minimization statement. The incremental solver finds an upper-bound time value c_{upper} at step `k`. A time optimal solution is guaranteed at step `k'` where `k'`$= c_{upper}/min($`D`$)$, `D` is the set of activity durations. In a similar way, any objective that is quantified with an integer value (e.g. cost objectives, resource leveling, etc.) could be introduced. When there is more than one objective, they should be prioritized.

Taking into account all the aforementioned functionality, using the encoding summarized above and detailed in `http://goo.gl/Q7B2t4`, a time optimal solution for our industry scenario is depicted in Fig. 4. The final allocation of resources to each activity a_i is as follows:

a_1 {Amy,hw1a,lab-1(1)}
a_2 {Amy,hw1b,lab-1(1)}
a_3 {Glen,hw1c,lab-1(1)}
a_4 {Glen,hw2b,lab-1(1)}
a_5 {Glen,Drew,Ewan,Mary,lab-1(-4)}
a_6 {Kate,hw2c,lab-2(1)}
a_7 {Mary,hw2a,lab-2(1)}
a_8 {Amy,hw2d,lab-2(1)}
a_9 {Amy,hw3c,lab-2(1)}
a_{10} {Mary,hw3b,lab-2(1)}
a_{11} {Kate,hw3a,lab-2(1)}
a_{12} {Kate,Amy,lab-2(-6)}

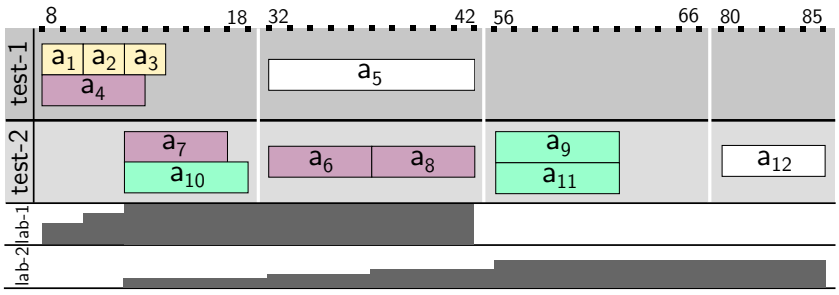


Fig. 5: A greedy (suboptimal) resource allocation for our industry scenario

	Optimal(Fig. 4)	Greedy (Fig. 5)
TET	30	35
AEU	0.61	0.54

Table 5: Result quality comparison

5 Evaluation

Our resource allocation technique not only finds an optimal schedule for activities in our industry scenario but also consequently optimizes the resource utilization. We show the improvement in result quality by comparing an optimal allocation of the scenario (cf. Fig 4) against a greedy allocation, depicted in Fig. 5. We use the following two criteria for this comparison:

1. Total execution time (TET) corresponds to the end time of the last activity for each process (e.g. a_5 for process Test-1).

2. Average employee utilization (AEU): For any time unit $c \in C$, c_{start} is the start time, c_{end} is the end time of process execution, $c_{start} \leq c \leq c_{end}$, a function $s : c \rightarrow R_b$ returns an ordered set of billable employees R_b respecting Table 3. For each element $s \in R_b$ a function $w_c : r \rightarrow \{0, 1\}$ returns whether the employee r is working at time c. In other words, we first sum the ratio between the number of employees allocated and the total number of employees that potentially can take part at each time unit, and normalize this sum using the overall execution time. *AEU* is calculated as described by (2).

$$AEU = \frac{\sum_{i=c_{start}}^{c_{end}} \frac{\sum_{r \in s(i)} w_c(r)}{|s(i)|}}{c_{end} - c_{start}} \tag{2}$$

For instance, in Fig. 5, $s(8) = \{Glen, Drew, Evan, Mary, Amy\}$. Note that $Kate$ is not in the set since she only takes part in Test-2 and Test-2 instances have not started due to the deadline constraint `bpStarts(earliest,test-2,12)`. At time 8, only $w_c(Amy)$ and $w_c(Glen)$ have value of 1.

Table 5 summarizes the results obtained using the two aforementioned criteria for the two allocation strategies. The execution of our industry scenario finishes 5 TU before under optimal allocation, which corresponds to 14% of time usage improvement while *AEU* improves 7%. We refer the reader to [5] for scal-

	Basic Resource Allocation		Advanced Time Management		Advanced Res. Mgmt.		
Approach	Res. Type	A. Level	Calendar	Aggreg.	Dynamism	Objective	Formalism
[25]	Both	Low	✓	-	-	Usage	MIP
[26]	Both	Medium	✓	-	-	Usage	IP
[27]	Both	High	✓	-	-	Any	Ad-hoc
[28]	Both	Medium	-	✓	-	Time&usage	LIP
[29]	Both	Medium	-	✓	-	Time&usage	CP
[30]	Both	Medium	-	✓	-	Makespan	Ad-hoc
[31]	Both	Medium	-	✓	-	-	CP
[19]	Both	Medium	-	✓	-	Makespan	Petri N.
[5]	Human	Medium	-	✓	-	Time	ASP

Table 6: Representative approaches related to resource allocation

ability of our technique, where we demonstrated that ASP performs well for resource allocation in the BPM domain.

6 Related Work

Resource allocation has been extensively explored in various domains for addressing everyday problems, such as room, surgery or patient scheduling in hospitals, crew-job allocation or resource leveling in organizations. Table 6 collects a set of recent, representative approaches of three related domains: operating room scheduling [25–27], project scheduling [28–30] and resource allocation in business processes [5, 19, 31]. The features described in Section 3 are used for comparing them[3]. Specifically, column *Res. Type* specifies the type(s) of resource(s) considered for allocation (human, non-human or both); column *A. Level* indicates the expressiveness of the restrictions that can be defined for the allocation, among: (i) low, when a small range of resource assignment requirements are considered *and* only one individual of each resource type (e.g., one person and one room) is allocated to an activity, i.e., cardinality is disregarded; (ii) medium, when a small range of resource assignment requirements are considered *or* cardinality is disregarded; and (iii) high, when flexible resource assignment *and* cardinality are supported; column *Calendar* refers to whether information about resource availability is taken into account (a blank means it is not); column *Aggreg.* indicates whether the execution time of an activity is determined by the resources involved in it; column *Advanced Res. Mgmt.* shows the support for cumulative resources that can be shared among several activities at the same time; column *Objective* defines the variable to be optimized; and column *Formalism* specifies the method used for resolving the problem.

The concept of process is not explicitly mentioned in the operating room scheduling problem. Traditional approaches in this field tended to adopt a two-step approach which, despite reducing the problem complexity, failed to ensure optimal or even feasible solutions [27]. It is a property of the surgery scheduling

[3] We have adopted the vocabulary used in BPM for resource allocation [19, 31].

problem that some resources, such as the operating rooms, can only be used in one project at a time [27], so cardinality is disregarded [25,26]. However, it is important to take into account resource availability. The most expressive approach in this domain [27] is an ad-hoc algorithm, whereas integer programming (IP) stands out as a formalism to efficiently address this problem.

Project scheduling consists of assigning resources to a set of activities that compose a project, so the concept of workflow is implicit. The approaches in this domain support cardinality for resource allocation but they rely on only the resource type for creating the resource sets assigned to an activity. These approaches implement the so-called *resource-time tradeoff*, which assumes that activity completion is faster if two resources of the same type work together in its execution [28, 29] (cf. Section 3.2). However, they assume a constant per-period availability of the resources [30], hence calendars are overlooked. The project scheduling problem has been repeatedly addressed with formalisms like linear integer programming (LIP) [28] and constraint programming (CP) [29], yet ad-hoc solutions also exist [30].

Finally, in the domain of BPM, the state of the art in resource allocation does not reach the maturity level of the other domains despite the acknowledged importance of the problem [32] and the actual needs (cf. Section 2.1). Similar to project scheduling, a constant availability of resources is typically assumed. In addition, due to the computational cost associated to joint resource assignment and scheduling problems [33], the existing techniques tend to search either for a feasible solution without applying any optimizations [31]; or for a local optimal at each process step using a greedy approach that might find a feasible but not necessarily a globally optimal solution [19]. Nonetheless, recently it was shown that global optimization is possible at a reasonable computational cost [5]. Moreover, driven by the limitations of current BPMS, which tend to disregard collaborative work for task completion, cardinality has been unconsidered for allocation, giving rise to less realistic solutions.

In general, the optimization function depends on the problem and the objective of the approach but it is generally based on minimizing time, makespan or cost, or making an optimal use of the resources (a.k.a. resource leveling [34]).

7 Conclusions and Future Work

In this paper we have conceptualized the complex problem of resource allocation under realistic dependencies that affect resources and activities as well as potential conflicts that may arise due to simultaneous requirement of resources. Our implementation based on ASP and its evaluation show that optimal solutions for this problem are possible, which extends the state of the art in BPM research and could contribute to extend the support in existing BPMS. ASP has proved to scale well [23] and can be easily integrated with RDF ontologies [24].

It is not the aim of this work to provide an end-user-oriented but an effective solution. In order to reasonably use our ASP implementation with a BPMS, it is required: (i) to map the notation used for process modeling along with the durations associated with the activities to (timed) Petri nets, for which several

techniques have been designed [35]; and (ii) the integration of languages for defining all the requirements which could be used by non-technical users in the system as well as their mapping to ASP. However, to the best of our knowledge, there is not yet such an expressive end-user-oriented language but languages that allow a partial definition of the requirements [14, 16].

As future work we plan to compare our technique with existing approaches on other optimal resource allocation techniques, explore the preemptive resource allocation as well as to apply our technique in other domains.

References

1. G. A. Rummler and A. J. Ramias, "A framework for defining and designing the structure of work," in *Handbook on Business Process Management 1*, pp. 81–104, Springer, 2015.
2. H. A. Reijers, I. T. P. Vanderfeesten, and W. M. P. van der Aalst, "The effectiveness of workflow management systems: A longitudinal study," *Int J. Information Management*, vol. 36, no. 1, pp. 126–141, 2016.
3. M. Rosemann and J. vom Brocke, "The six core elements of business process management," in *Handbook on Business Process Management 1*, pp. 105–122, Springer, 2015.
4. R. Mans, N. C. Russell, W. M. P. van der Aalst, A. J. Moleman, and P. J. M. Bakker, "Schedule-aware workflow management systems," *Trans. Petri Nets and Other Models of Concurrency*, vol. 4, pp. 121–143, 2010.
5. G. Havur, C. Cabanillas, J. Mendling, and A. Polleres, "Automated Resource Allocation in Business Processes with Answer Set Programming," in *BPM Workshops (BPI)*, p. In press, 2015.
6. L. Popova-Zeugmann, "Time Petri Nets," in *Time and Petri Nets*, pp. 139–140, Springer Berlin Heidelberg, 2013.
7. D. S. Johnson and M. R. Garey, "Computers and Intractability: A Guide to the Theory of NP-Completeness," *WH Free. Co., San Fr*, 1979.
8. F. Buccafurri, N. Leone, and P. Rullo, "Enhancing disjunctive datalog by constraints," *IEEE Trans. on Knowledge and Data Engineering*, vol. 12, no. 5, pp. 845–860, 2000.
9. C. Ouyang, M. T. Wynn, C. Fidge, A. H. ter Hofstede, and J.-C. Kuhr, "Modelling complex resource requirements in Business Process Management Systems," in *ACIS 2010*, 2010.
10. D. Brickley and R. Guha, "RDF Schema 1.1." W3C Recommendation, Feb. 2014. `http://www.w3.org/TR/rdf-schema/`.
11. D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers, "Turtle – Terse RDF Triple Language." W3C Candidate Recommendation, Feb. 2014. `https://www.w3.org/TR/turtle/`.
12. OMG, "BPMN 2.0," recommendation, OMG, 2011.
13. C. Cabanillas, M. Resinas, A. del Río-Ortega, and A. Ruiz-Cortés, "Specification and Automated Design-Time Analysis of the Business Process Human Resource Perspective," *Inf. Syst.*, vol. 52, pp. 55–82, 2015.
14. W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: Yet Another Workflow Language," *Inf. Syst.*, vol. 30, no. 4, pp. 245–275, 2005.
15. L. J. R. Stroppi, O. Chiotti, and P. D. Villarreal, "A BPMN 2.0 Extension to Define the Resource Perspective of Business Process Models," in *CIbS'11*, 2011.

16. C. Cabanillas, M. Resinas, J. Mendling, and A. R. Cortés, "Automated team selection and compliance checking in business processes," in *ICSSP*, pp. 42–51, 2015.
17. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*. Morgan & Claypool Publishers, 2012.
18. D. Van Nieuwenborgh, M. De Cock, and D. Vermeir, "Fuzzy answer set programming," in *Logics in Artificial Intelligence*, pp. 359–372, Springer, 2006.
19. W. van der Aalst, "Petri net based scheduling," *Operations-Research-Spektrum*, vol. 18, no. 4, pp. 219–229, 1996.
20. R. Roose, "Automated Resource Optimization in Business Processes." MSc. Thesis.
21. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres, "Answer set planning under action costs," *J. Artif. Intell. Res. (JAIR)*, vol. 19, pp. 25–71, 2003.
22. G. Brewka, T. Eiter, and M. Truszczyński, "Answer set programming at a glance," *Communications of the ACM*, vol. 54, no. 12, pp. 92–103, 2011.
23. F. Calimeri, M. Gebser, M. Maratea, and F. Ricca, "Design and results of the fifth answer set programming competition," *Artificial Intelligence*, vol. 231, 2016.
24. T. Eiter, G. Ianni, T. Krennwallner, and A. Polleres, "Rules and Ontologies for the Semantic Web," in *Reasoning Web 2008*, vol. 5224, pp. 1–53, 2008.
25. P. M. Castro and I. Marques, "Operating room scheduling with generalized disjunctive programming," *Computers & Operations Research*, vol. 64, pp. 262–273, 2015.
26. T. A. Silva, M. C. de Souza, R. R. Saldanha, and E. K. Burke, "Surgical scheduling with simultaneous employment of specialised human resources," *European Journal of Operational Research*, vol. 245, no. 3, pp. 719–730, 2015.
27. A. Riise, C. Mannino, and E. K. Burke, "Modelling and solving generalised operational surgery scheduling problems," *Computers & Operations Research*, vol. 66, pp. 1–11, 2016.
28. M.-F. F. Siu, M. Lu, and S. AbouRizk, "Methodology for crew-job allocation optimization in project and workface scheduling," in *ASCE*, pp. 652–659, 2015.
29. W. Menesi, M. Abdel-Monem, T. Hegazy, and Z. Abuwarda, "Multi-objective schedule optimization using constraint programming," in *ICSC15*, 2015.
30. A. Sprecher and A. Drexl, "Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm1," *European Journal of Operational Research*, vol. 107, no. 2, pp. 431 – 450, 1998.
31. P. Senkul and I. H. Toroslu, "An Architecture for Workflow Scheduling Under Resource Allocation Constraints," *Inf. Syst.*, vol. 30, pp. 399–422, July 2005.
32. M. Arias, E. Rojas, J. Munoz-Gama, and M. Sepúlveda, "A Framework for Recommending Resource Allocation based on Process Mining," in *BPM 2015 Workshops (DeMiMoP)*, p. In press, 2015.
33. M. Lombardi and M. Milano, "Optimal methods for resource allocation and scheduling: a cross-disciplinary survey," *Constraints*, vol. 17, pp. 51–85, 2012.
34. J. Rieck and J. Zimmermann, "Exact methods for resource leveling problems," in *Handbook on Project Management and Scheduling Vol. 1*, Springer, 2015.
35. N. Lohmann, E. Verbeek, and R. Dijkman, "Petri Net Transformations for Business Processes - A Survey," *Transactions on Petri Nets and Other Models of Concurrency II*, vol. 2, pp. 46–63, 2009.