Resource and data management service architecture

Deliverable D2.3

FFG – IKT der Zukunft
SHAPE Project
2014 – 845638

Table 1 Document Information

Project acronym:	SHAPE
Project full title:	Safety-critical Human- & dAta-centric Process management in Engineering projects
Work package:	2
Document number:	2.3
Document title:	Resource and data management service architecture
Version:	1
Delivery date:	01 April 2016 (M3)
Actual publication date:	01 April 2016
Dissemination level:	Public
Nature:	Report
Editor(s) / lead beneficiary:	WU Vienna
Author(s):	Giray Havur, Cristina Cabanillas, Jan Mendling, Axel Polleres
Reviewer(s):	Alois Haselboeck, Cristina Cabanillas

Contents

1 Introduction

This document is part of work package 2 (WP2) on *Semantic Models for Mining & Monitoring Process Relevant Data* of the SHAPE project[1]. It reports the work performed under milestone 2.3 *Resource and data management service architecture*. More specifically,

- We investigate and report on how the reasoning service communicates with the business process management system Camunda, and
- We extend the resource allocation encoding for resource management that we described in D2.2 and in [15] towards meeting the needs of the SHAPE project.

In particular, the content of this deliverable is structured as follows: Section 2 details the reasoning service architecture. Section 3 describes an extended version of our resource allocation encoding in iASP [15] for supporting allocation of both human and material resources. Section 4 concludes the deliverable by remarking ongoing and future work.

2 Reasoning Service Architecture

In this section we detail our reasoning service architecture. Figure 1 shows the overall framework for process management in complex engineering projects and how our *reasoning module* is connected to other components in a BPMS environment. The reasoning module uses our Engineering Domain Ontology [6] as input for reasoning tasks. The Engineering Domain Ontology consists of:

- Process model,
- Organizational model,
- Infrastructure model, and
- Regulation model.

The main task the reasoning module performs is scheduling of activities in a business process, and allocating required resources to activities (cf. Section 3) while taking into account the regulation data consist of the requirements, i.e. standards and norms, detailed in [16]. Therefore, the reasoning module lead to results that are *compliant* to such requirements that ensures safety in engineering projects (e.g. Compliance rules derived from EN50126 [13]).

[1] https://ai.wu.ac.at/shape-project/

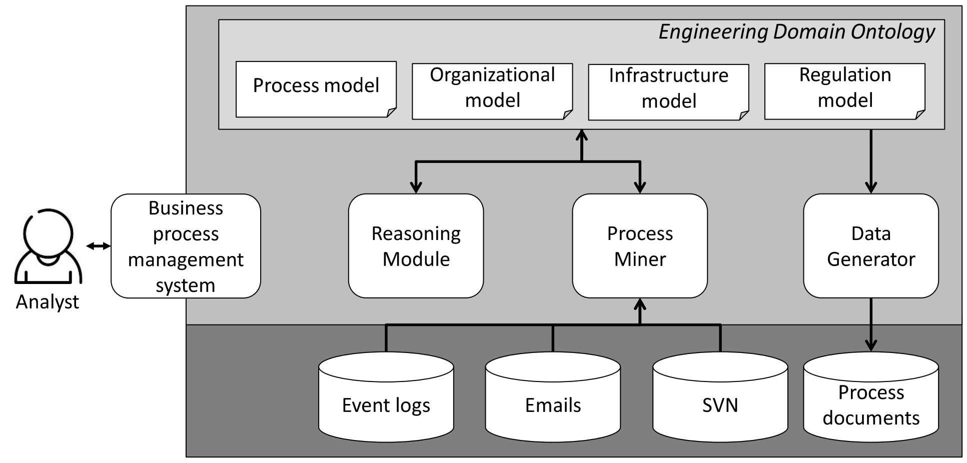


Figure 1 Overall framework for process management in complex engineering projects

3 Customizable Resource Allocation in BPMS

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 [28]. 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 [23].

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 [27].

In this section, 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 [18] 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

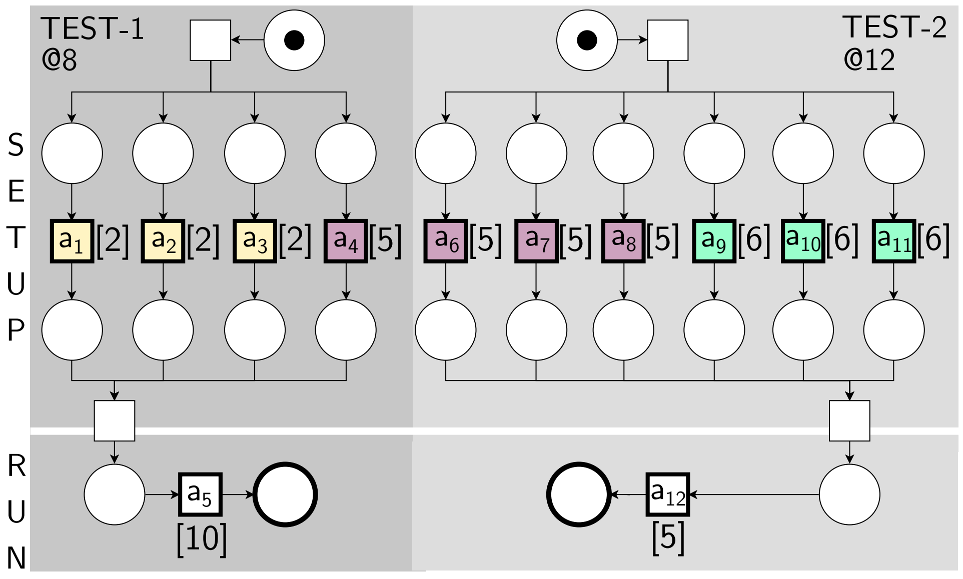


Figure 2 Workflow for two projects

(ASP), a formalism from logic programming that has been found to scale well for solving problems as the one we tackle [15]. 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.

3.1 Motivation

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

3.1.1 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 2 depicts two process models representing the setup and run phases of two tests. We use (timed) Petri nets [22] 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 between the closed intervals $[0,7]$,$[19,31]$,$[43,55]$, and $[67,79]$.

For completing tests, the non-human resources available in the organization include 13 units of space distributed into 2 laboratories (Table 2) and several units of 3 types of hardware (Table 3). The human resources of the company are specialized in the execution of specific phases of the two testing projects, whose activities they are able to complete in a specific time. Table 4 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 5. 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.

3.1.2 Insights

The resource allocation problem 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

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

Table 2 Available space in labs

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

Table 3 Available hardware (HW)

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

Table 4 Specialization of employees

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). Table 4 is provided instead.; (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 [29]. 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 [5].

3.2 Conceptualization of the Resource Allocation Problem

Fig. 3 illustrates our conceptualization of the resource allocation problem. We divide it into three complexity layers related to the aforementioned dependencies

	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 5 Activity requirements

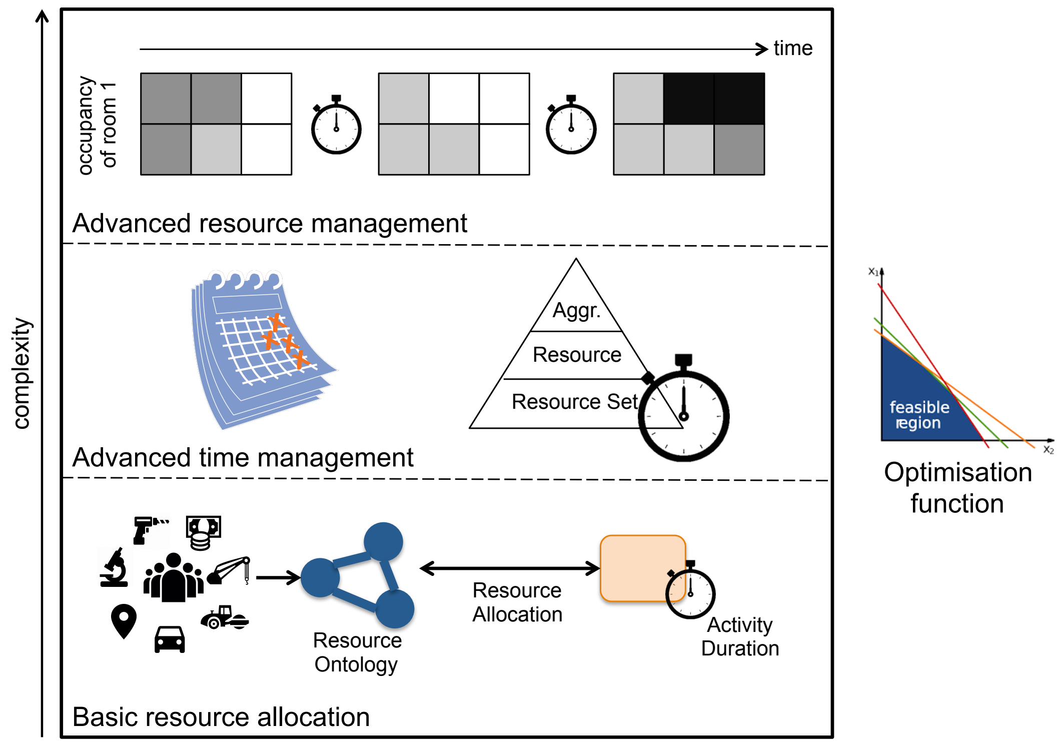


Figure 3 Resource allocation in business processes

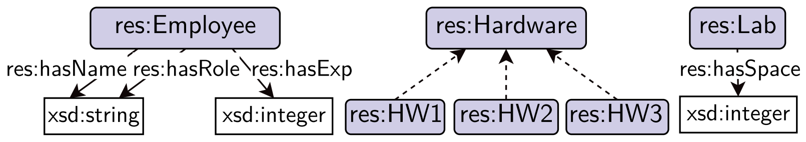
and resource conflicts. Optimization functions can be applied to all types of allocation problems. This model has been defined from the characteristics identified in the industry scenario as well as in related literature [21]. We describe each complexity layer in the following sections.

3.2.1 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.

3.2.1.1 Resource Ontology

In order to enable the integration with semantic technologies for an automatic resource allocation, we suggest the use of an ontology for modeling the organizational information. Fig. 4 illustrates a sample resource ontology using the RDF schema (RDFS) [4], 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



```
: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.
```

Figure 4 Resource ontology and example instantiation

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 [2]. This instantiation represents Tables 2-4 of the industry scenario.

3.2.1.2 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 [20]) or can be modelled externally. In either case, it has to be accessible by the allocation algorithm.

3.2.1.3 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 [7] 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 [7, 35, 34, 8]. 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 3.1.1.

3.2.2 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 status, such as available, unavailable, occupied/busy or blocked [21]. 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

[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.

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.

3.2.3 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. For instance, in Fig. 3 there is a resource *room 1* whose occupancy changes over time. This so called *cumulative resources* are hence characterized by their *dynamic* attributes and they can be allocated to more than one activity at a time.

3.2.4 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 [36] 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 [26] 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 [11].

3.3 Implementation with ASP

Answer Set Programming (ASP) [14] 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:

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

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* $ground(r)$, i.e., the set of its ground instantiations obtained by replacing the variables with all possible constants occurring in Π. Likewise, we denote by $ground(\Pi)$ 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. [3] for details).

ASP Solvers typically first compute a subset of $ground(\Pi)$ and then use a DPLL-like branch and bound algorithm to find answer sets for this ground program. We use the ASP solver *clasp* [14] for our experiments as it has proved to be one of the most efficient implementations available [9].

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 [14] to indicate preferences between possible answer sets:

$$\#minimize\ \{A_1 : Body_1 = w_1, \ldots, A_m : Body_m = w_m @ p\}$$

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) [14] 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 [15]. 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 in Appendix A.

3.3.1 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 [15, 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 3.2.1.1). Note that, any resource ontology described in RDF(S) can be easily incorporated/translated into ASP [12]. 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 [7]. 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 $\mathtt{b}_2$.

3.3.2 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`.

As one activity can be allocated to a group of resources (cf. Section 3.2.2), 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`}.

3.3.3 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 *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 resources consumed or generated. For instance, `requirement(`$\mathtt{a}_1$`,lab_space,-1)` consumes 1 unit of lab space when a_1 is allocated, whereas `requirement(`$\mathtt{a}_{12}$`,rs_ex2,6)` 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(`$\mathtt{a}_1$`,`$\mathtt{a}_2$`,id,n)` blocks `n` amount of resources in the resource set `id` from the beginning of $\mathtt{a}_1$ to beginning of $\mathtt{a}_2$.

3.3.4 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. 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 Appendix A, a time optimal solution for our industry scenario is depicted in Fig. 5. The final allocation of resources to each activity a_i is as follows:

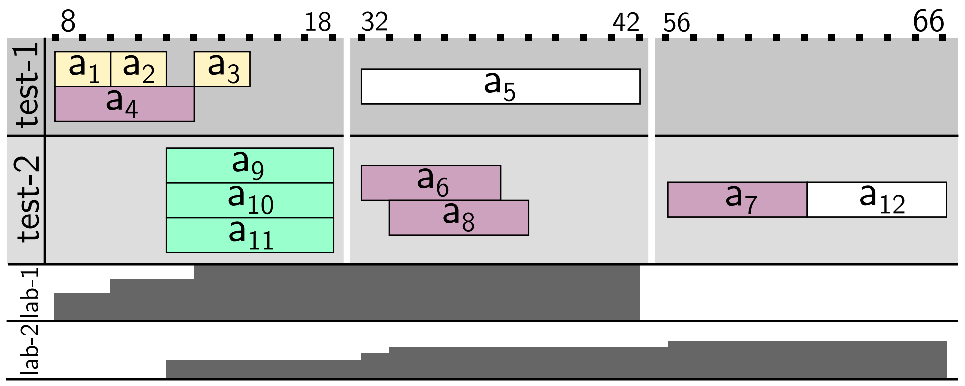


Figure 5 Optimal resource allocation for our industry scenario

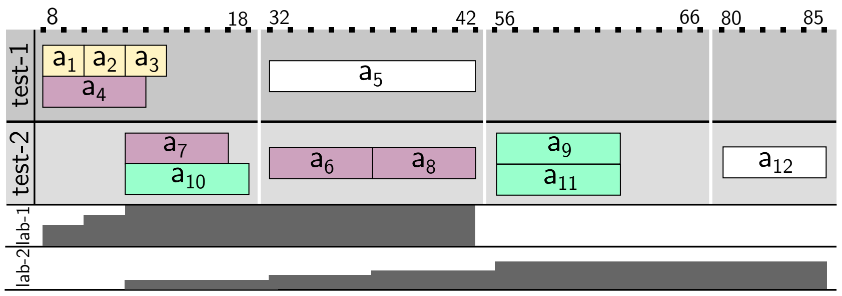


Figure 6 A greedy (suboptimal) resource allocation for our industry scenario

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

3.3.5 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 5) against a greedy allocation, depicted in Fig. 6. 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 4. For each

	Optimal(Fig. 5)	Greedy (Fig. 6)
TET	30	35
AEU	0.61	0.54

Table 6 Result quality comparison

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. 6, $s(8) = \{Glen, Drew, Evan, Mary, Amy\}$. 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 6 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 [15] for scalability of our technique, where we demonstrated that ASP performs well for resource allocation in the BPM domain.

3.4 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 7 collects a set of recent, representative approaches of three related domains: operating room scheduling [10, 31, 25], project scheduling [32, 19, 33] and resource allocation in business processes [36, 30, 15]. The features described in Section 3.2 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:

[3] We have adopted the vocabulary used in BPM for resource allocation [36, 30].

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

Table 7 Representative approaches related to resource allocation

(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 [25]. It is a property of the surgery scheduling problem that some resources, such as the operating rooms, can only be used in one project at a time [25], so cardinality is disregarded [10, 31]. However, it is important to take into account resource availability. The most expressive approach in this domain [25] 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 [32, 19] (cf. Section 3.2.2). However, they assume a constant per-period availability of the resources [33], hence calendars are overlooked. The project scheduling problem has been repeatedly addressed with formalisms like linear integer programming (LIP) [32] and constraint programming (CP) [19], yet ad-hoc solutions also exist [33].

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 [1] and the actual needs (cf. Section 3.1.1). 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 [17], the existing techniques tend to search either for a feasible solution without applying any optimizations [30]; 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 [36]. Nonetheless, recently it was shown that global optimization is possible at a reasonable computational cost [15]. 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 [24]).

4 Summary and Future Work

In this deliverable we provided an overview on reasoning architecture (cf. Section 2) and enhanced/generalized our resource allocation technique for the scope of SHAPE project (cf. Section 3). As future work we aim at digging into architectural details and implementing a demo that brings together functionalities of different work packages in Camunda environment, where the reasoning tasks will play a central role.

Appendices

A iASP Encoding for Automated Resource Allocation

We provide the improved and enhanced iASP encoding version for automated resource allocation.

Listing 1 Static knowledge

```
#program base.

% default duration of a transition
firingDelay(A,B,0) :- not activityDuration(A,B,_),
                      activityTransition(A,B).
firingDelay(A,B,D) :- activityDuration(A,B,D).


% default resource-activity duration preference handling
defaultRABD(R,A,B,D) :- humanResource(R),
                        resourceActivityDuration(R,A,B,D).
defaultRABD(R,A,B,D) :- humanResource(R), resourceSet(R,Q),
                        rsActivityDuration(Q,A,B,D),
                        not resourceActivityDuration(R,A,B,_).
defaultRABD(R,A,B,D) :- humanResource(R), firingDelay(A,B,D),
                        not resourceActivityDuration(R,A,B,_),
                        not rsActivityDuration(_,A,B,_).
```

Listing 2 Cumulative knowledge

```
#program cumulative(s).

%%% PETRI NET DYNAMINCS %%%

% generate action fire
{fire(T,B,I,s) : inPlace(P,T,B), bpInstance(B,I)}.

% fire precondition: if no token at preceding P, then can't fire
:- fire(T,B,I,s), bpInstance(B,I), inPlace(P,T,B),
   not tokenAt(P,B,I,s).

% fire effect: if fire at s-1, token at next place
tokenAt(P,B,I,s) :- fire(T,B,I,s-1), outPlace(P,T,B), bpInstance(B,I).

% fire constraint only 1 succeeding transition can fire
:- inPlace(P,T1,B), inPlace(P,T2,B), T1!=T2, fire(T1,B,I,s),
   fire(T2,B,I,s), bpInstance(B,I).
```

```
% inertia: tokenAt(P,s): if not fire token remains at its place
consumeToken(P,B,I,s) :- inPlace(P,T,B), fire(T,B,I,s),
                         bpInstance(B,I).
tokenAt(P,B,I,s) :- tokenAt(P,B,I,s-1), not consumeToken(P,B,I,s-1).

%%% TIME MANAGEMENT %%%

% Max time
maxTimeAtInPlace(P,T,B,I,s) :- inPlace(P,T,B),
                            not greaterTimeExistsAtInPlace(P,T,B,I,s),
                            fire(T,B,I,s), bpInstance(B,I).
greaterTimeExistsAtInPlace(P1,T,B,I,s) :- inPlace(P1,T,B),
         inPlace(P2,T,B), fire(T,B,I,s), timeAt(P1,C1,B,I,s),
         timeAt(P2,C2,B,I,s), P1!=P2, C1<C2, bpInstance(B,I).

% fire effect on time (T is not an activity)
timeAt(P2,X,B,I,s) :- not activityTransition(T,B), fire(T,B,I,s-1),
                   maxTimeAtInPlace(P,T,B,I,s-1), timeAt(P,X,B,I,s-1),
                   outPlace(P2,T,B), bpInstance(B,I).

% fire effect on time (T is an activity)
timeAt(P2,C2,B,I,s) :- activityTransition(T,B),
                       allocate(R,T,C1,C2,B,I,s-1), fire(T,B,I,s-1),
                       outPlace(P2,T,B), bpInstance(B,I).

% no fire wait
timeAt(P,C,B,I,s) :- timeAt(P,C,B,I,s-1), inPlace(P,T,B),
                     not activityTransition(T,B),
                     not consumeToken(P,B,I,s-1), bpInstance(B,I).

% time relaxation: if activity : relaxation is possible
timeAt(P,C+1,B,I,s)  :- timeAt(P,C,B,I,s-1), inPlace(P,T,B),
                        activityTransition(T,B),
                        not consumeToken(P,B,I,s-1), bpInstance(B,I).

%%% RESOURCE ALLOCATION %%%

% allocate requirements to activities
N{allocate(R,A,C,C+D,B,I,s): defaultD(A,B,D,s), resourceSet(R,Q)}N :-
  requirement(A,Q,N), consumeToken(P1,B,I,s), inPlace(P1,A,B),
  timeAt(P1,C,B,I,s), bpInstance(B,I).

% can not allocate same resource to another activity at the same time
:- allocate(R,A1,C1,C2,B,I,S1), allocate(R,A2,C1,C3,B,I,S2), C1<C2,
   C1<C3, A1!=A2.
:- allocate(R,A1,C1,C2,B,I1,S1), allocate(R,A2,C1,C3,B,I2,S2),
   C1<C2, C1<C3, A1!=A2, I1!=I2.
```

```
:- allocate(R,A1,C1,C2,B1,I1,S1), allocate(R,A2,C1,C3,B2,I2,S2),
   C1<C2, C1<C3, B1!=B2.

% can not allocate same resource to same activity at the same time in
% another instance
:- allocate(R,A,C1,C2,B,I1,S1), allocate(R,A,C1,C3,B,I2,S2), C1<C2,
   C1<C3, I1!=I2.
:- allocate(R,A,C1,C2,B1,I1,S1), allocate(R,A,C1,C3,B2,I2,S2), C1<C2,
   C1<C3, B1!=B2.

% can not allocate when allocatements overlap
:- allocate(R,T,Y1,Y2,B1,I1,S1), allocate(R,T2,X1,X2,B2,I2,S2), X1>Y1,
   X1<Y2.
:- allocate(R,T,Y1,Y2,B1,I1,S1), allocate(R,T2,X1,X2,B2,I2,S2), X2<Y2,
   X2>Y1.

% multiple allocations must start at the same time.
:- allocate(R,A,C1,C2,B,I,s), allocate(R1,A,C3,C4,B,I,s), R!=R1,
   C1!=C3.

% default time is maximum time among assignments
defaultD(A,B,D,s) :- not greaterDExists(A,B,D,s),
                     defaultRABD(R,A,B,D).
greaterDExists(A,B,D1,s) :- defaultRABD(R1,A,B,D1), R1!=R2, D1<D2,
                            defaultRABD(R2,A,B,D2).

% consumable resources
% integer attribute changes
stepChange(Q,N,A,B,I,s) :- requirementC(A,Q,N), inPlace(P1,A,B),
                           consumeToken(P1,B,I,s), timeAt(P1,C,B,I,s),
                           bpInstance(B,I).
1{valueChange(A,B,I,R,N,s): resourceValueSet(R,V,Q,s)}1 :-
                                                   stepChange(Q,N,A,B,I,s).

% sum all value cahnges and apply them
resourceValueSet(R,V+N,Q,s) :- resourceValueSet(R,V,Q,s-1),
               N=#sum{M,A,R,valueChange : valueChange(A,B,I,R,M,s-1)}.

% an integer attribute can not be less than 0
:- resourceValueSet(R,V,Q,s), V<0.

% a value change can not make a resource attribute less than zero
:- resourceValueSet(R,V,Q,s), valueChange(_,B,I,R,M,s), M<0, -M>V.


% block resource
N{block(A,R,C1,s):resourceSet(R,Q)}N :- requirementB(A,A1,Q,N),
     consumeToken(P1,B,I,s-1), inPlace(P1,A,B), timeAt(P1,C1,B,I,s-1).
% unblock resource
unblock(A1,R,C2,s) :- block(A,R,C,s-1), resourceSet(R,Q),
                 requirementB(A,A1,Q,_), allocate(_,A1,C1,C2,B,I,s-1).

% inertia
block(A,R,C1,s) :- not unblock(A1,R,_,s-1), block(A,R,C1,s-1),
                   requirementB(A,A1,Q,N).
unblock(A1,R,C1,s) :- not block(A,R,_,s-1), unblock(A1,R,C1,s-1),
                      requirementB(A,A1,Q,N).

% can not block the same resource if not unblocked
:- block(A1,R,C1,s), requirementB(A1,A3,Q,N), not unblock(A3,R,C3,s),
   block(A2,R,C2,s), A1!=A2, C3=C1..C2, C2>=C1.

% can not allocate a resource if not unblocked
:- block(A1,R,C1,s), requirementB(A1,A2,Q,N), not unblock(A2,R,C2,s),
   allocate(R,A3,C3,_,B,I,s), C2=C1..C3.


% separate/bind duties
% strict separation of duties
notallocateStrictly(R,A1,B1,C2,s) :- strictlySeparated(A,B,A1,B1),
                                     allocate(R,A,_,C2,B,I,s-1).
:- allocate(R,A,C1,C2,B,I,S), notallocateStrictly(R,A,B,C,S1), C1>=C.

% weak separation of duties
notallocateWeakly(R,A1,B1,C2,s) :- weaklySeparated(A,B,A1,B1),
                                   allocate(R,A,_,C2,B,I,s-1).
notallocateWeakly(R,A,B,C,s) :- notallocateWeakly(R,A,B,C,s-1),
                                not allocate(_,A,_,_,B,_,s-1).
:- allocate(R,A,C1,C2,B,I,s), notallocateWeakly(R,A,B,C,s), C1>=C.

% strict binding of duties
mustallocateStrictly(R,A1,B1,C2,s) :- strictlyBinded(A,B,A1,B1),
                                      allocate(R,A,_,C2,B,I,s-1).
:- allocate(R1,A,C1,C2,S,B,I), mustallocateStrictly(R,A,B,C,S1),
   C1>=C, R1!=R.

% weak binding of duties
mustallocateWeakly(R,A1,B1,C2,s) :- weaklyBinded(A,B,A1,B1),
                                    allocate(R,A,_,C2,B,I,s-1).
mustallocateWeakly(R,A,B,C,s) :- mustallocateWeakly(R,A,B,C,s-1),
                                 not allocate(R,A,_,_,B,_,s-1).
:- allocate(R1,A,C1,C2,B,I,s), mustallocateWeakly(R,A,B,C,s), C1>=C,
   R1!=R.
```

```
% break calendar
:- allocate(R,A,C1,C2,S,B,I), resourceSet(R,Q), break(Q,C3,C4),
   C3>=C1, C3=<C2.
:- allocate(R,A,C1,C2,S,B,I), resourceSet(R,Q), break(Q,C3,C4),
   C4>C1, C4<C2.
```

References

1 Michael Arias, Eric Rojas, Jorge Munoz-Gama, and Marcos Sepúlveda. A Framework for Recommending Resource Allocation based on Process Mining. In *BPM 2015 Workshops (DeMiMoP)*, page In press, 2015.

2 David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. Turtle – Terse RDF Triple Language. W3C Candidate Recommendation, February 2014. https://www.w3.org/TR/turtle/.

3 Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

4 Dan Brickley and R.V. Guha. RDF Schema 1.1. W3C Recommendation, February 2014. http://www.w3.org/TR/rdf-schema/.

5 Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing disjunctive datalog by constraints. *IEEE Trans. on Knowledge and Data Engineering*, 12(5):845–860, 2000.

6 Cristina Cabanillas, Alois Haselböck, Jan Mendling, Axel Polleres, Simon Sperl, and Simon Steyskal. Engineering Domain Ontology: Base Regulations and Requirements Description. Project deliverable, Vienna University of Economics and Business, Austria, 2015.

7 Cristina Cabanillas, Manuel Resinas, Adela del Río-Ortega, and Antonio Ruiz-Cortés. Specification and Automated Design-Time Analysis of the Business Process Human Resource Perspective. *Inf. Syst.*, 52:55–82, 2015.

8 Cristina Cabanillas, Manuel Resinas, Jan Mendling, and Antonio Ruiz Cortés. Automated team selection and compliance checking in business processes. In *ICSSP*, pages 42–51, 2015.

9 Francesco Calimeri, Martin Gebser, Marco Maratea, and Francesco Ricca. Design and results of the fifth answer set programming competition. *Artificial Intelligence*, 231, 2016.

10 Pedro M Castro and Inês Marques. Operating room scheduling with generalized disjunctive programming. *Computers & Operations Research*, 64:262–273, 2015.

11 Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. Answer set planning under action costs. *J. Artif. Intell. Res. (JAIR)*, 19:25–71, 2003.

12 Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, and Axel Polleres. Rules and Ontologies for the Semantic Web. In *Reasoning Web 2008*, volume 5224, pages 1–53. San Servolo Island, Venice, Italy, 2008.

13 Julia Fuchsbauer. How to manage processes according to en50126. Bachelor thesis, Vienna University of Economics and Business, July 2015.

14 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Morgan & Claypool Publishers, 2012.

15 Giray Havur, Cristina Cabanillas, Jan Mendling, and Axel Polleres. Automated Resource Allocation in Business Processes with Answer Set Programming. In *BPM Workshops (BPI)*, page In press, 2015.

16 Tudor Ionscu and Alois Haselböck. Workflow Processing and Verification for Safety Critical Engineering – Detailed Design and Integrating Feedback of Experts. Project deliverable, Siemens, 2016.

17 Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 17:51–85, 2012.

18 Ronny Mans, Nick C. Russell, Wil M. P. van der Aalst, Arnold J. Moleman, and Piet J. M. Bakker. Schedule-aware workflow management systems. *Trans. Petri Nets and Other Models of Concurrency*, 4:121–143, 2010.

19 Wail Menesi, Mohamed Abdel-Monem, Tarek Hegazy, and Zinab Abuwarda. Multi-objective schedule optimization using constraint programming. In *ICSC15*, 2015.

20 OMG. BPMN 2.0. Recommendation, OMG, 2011.

21 Chun Ouyang, Moe Thandar Wynn, Colin Fidge, Arthur H.M. ter Hofstede, and Jan-Christian Kuhr. Modelling complex resource requirements in Business Process Management Systems. In *ACIS 2010*, 2010.

22 Louchka Popova-Zeugmann. Time Petri Nets. In *Time and Petri Nets*, pages 139–140. Springer Berlin Heidelberg, 2013.

23 Hajo A. Reijers, Irene T. P. Vanderfeesten, and Wil M. P. van der Aalst. The effectiveness of workflow management systems: A longitudinal study. *Int J. Information Management*, 36(1):126–141, 2016.

24 Julia Rieck and Jürgen Zimmermann. Exact methods for resource leveling problems. In *Handbook on Project Management and Scheduling Vol. 1*. Springer, 2015.

25 Atle Riise, Carlo Mannino, and Edmund K Burke. Modelling and solving generalised operational surgery scheduling problems. *Computers & Operations Research*, 66:1–11, 2016.

26 Riivo Roose. Automated Resource Optimization in Business Processes. MSc. Thesis, 2012.

27 Michael Rosemann and Jan vom Brocke. The six core elements of business process management. In *Handbook on Business Process Management 1*, pages 105–122. Springer, 2015.

28 Geary A Rummler and Alan J Ramias. A framework for defining and designing the structure of work. In *Handbook on Business Process Management 1*, pages 81–104. Springer, 2015.

29 David S. Johnson and Michael R. Garey. Computers and Intractability: A Guide to the Theory of NP-Completeness. *WH Free. Co., San Fr*, 1979.

30 Pinar Senkul and Ismail H. Toroslu. An Architecture for Workflow Scheduling Under Resource Allocation Constraints. *Inf. Syst.*, 30(5):399–422, July 2005.

31 Thiago AO Silva, Mauricio C de Souza, Rodney R Saldanha, and Edmund K Burke. Surgical scheduling with simultaneous employment of specialised human resources. *European Journal of Operational Research*, 245(3):719–730, 2015.

32 Ming-Fung Francis Siu, Ming Lu, and Simaan AbouRizk. Methodology for crew-job allocation optimization in project and workface scheduling. In *ASCE*, pages 652–659, 2015.

33 Arno Sprecher and Andreas Drexl. Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm1. *European Journal of Operational Research*, 107(2):431 – 450, 1998.

34 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.

35 Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Inf. Syst.*, 30(4):245–275, 2005.

36 W.M.P. van der Aalst. Petri net based scheduling. *Operations-Research-Spektrum*, 18(4):219–229, 1996.