State-of-the art on existing models for processes, resources, constraints and security, and their underlying formalisms

Deliverable D2.1

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:	2.1
Document title:	State-of-the art on existing models for processes, resources, constraints and security, and their underlying formalisms
Version:	1
Delivery date:	01 April 2015 (M1)
Actual publication date:	————
Dissemination level:	Public
Nature:	Report
Editor(s) / lead beneficiary:	WU Vienna
Author(s):	Cristina Cabanillas, Giray Havur, Jan Mendling, Axel Polleres, Alexander Wurl, Simon Steyskal
Reviewer(s):	Tudor Ionescu, Jan Mendling

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 work performed under Task 2.1 *State-of-the art on existing models for processes, resources, constraints and security, and their underlying formalisms*. Its goal is to provide an overview of approaches addressing different aspects of Business Process Management (BPM) that are interesting in the scope of SHAPE.

In particular, the content of this deliverable is structured as follows: Section 2 introduces business process modeling and the main perspectives that must be considered in the BPM life cycle (cf. Deliverable 3.1 [**?**] for an overview of the BPM life cycle). Section 2 delves into the formalization of business processes and constraints modeled with them. Afterwards, we focus on the business process resource perspective, crucial in the context of SHAPE as seen in the project requirements described in Deliverable 4.1 [13]. Section 4 addresses automated reasoning on the allocation of resources to business process activities and presents a summary of automated reasoners that could be used for that purpose together with an approach based on Answer Set Programming (ASP). That section also describes resource analysis in business processes and an approach to automate resource-related analysis operations based on Description Logics (DLs). Section 5 presents a survey on latest business process management systems and suites and Section 6 concludes the deliverable.

Regarding publications, this first deliverable of WP2 has yielded the following paper submissions so far:

- Giray Havur, Cristina Cabanillas, Jan Mendling, and Axel Polleres. *Automated Resource Allocation in Business Processes with Answer Set Programming*. Submitted to BPM 2015.
- Simon Steyskal and Axel Polleres. *Towards Formal Semantics for ODRL Policies*. Submitted to RuleML 2015.

2 Business Process Modeling

A business process is a collection of activities that are executed in a logical order along time to achieve a defined goal within an organisational and technical environment. To do so, they take one or more kinds of input and create an output

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

that is of value to the customer or the market [31, 18, 69]. As stated in [60, 19], the basis of BPM is the explicit representation of business processes. Representing a business process helps to discover weaknesses related to the order in which activities are performed, the information that is handled, and any other issue involved in business process execution. Hence, an easy-to-understand-and-use, editable, and executable mechanism to represent business processes is convenient. *Business process models* are defined for that purpose, that is, a *business process model* is the representation of the activities, documents, people and all the elements involved in a process, as well as the execution constraints between them [69]. It serves as a starting point to be analysed and improved before, during and after execution.

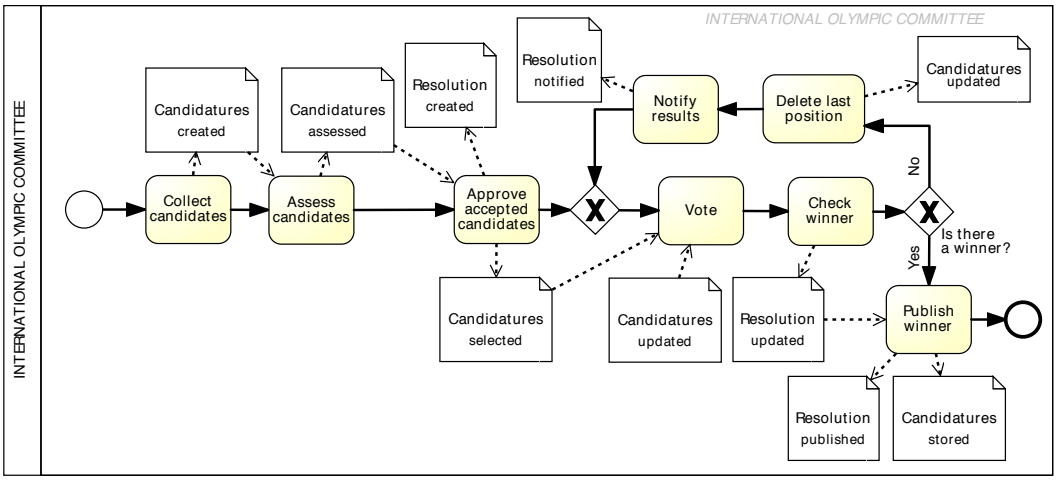


Figure 1 BPMN model of a process to select the venue place for the Olympic Games, taken from [11]

Let us exemplify all this with an example described in [11]. The process model depicted in Figure 1 represents the process to select the venue place for the Olympic Games, a procedure known worldwide that has been simplified for the sake of clarity. The International Olympic Committee is in charge of this process. In a nutshell, this committee first receives the applications of the cities that want to organise the Olympic Games. Each city is evaluated, so that only those cities that meet all the requirements pass to the approval step, where the candidates still need a final approval by the committee in oder to participate in the voting rounds. Once the list of candidates is ready, a first round of secret voting is carried out. If there is consensus and only one city is selected, then the winner venue is published. Otherwise, the least voted city is eliminated from the list of candidates and a new voting round is performed. This is repeated until there are only two cities left. Then, the city with a greatest number of votes wins.

The core elements in a business process are the activities and their execution or-

der. However, there are other elements also involved in business processes, which must be also considered when designing and modelling the processes. These elements are called *business process perspectives* or *business process dimensions*. There are typically five perspectives in business processes [19, 22, 69], which were summarised in [11] as follows:

The functional perspective provides a *description* of all the activities to be performed in a business process. It usually consists of a textual definition of the activity, its goal, the elements involved in its execution, and any other restriction that needs to be taken into account for its performance. According to Business Process Modelling Notation (BPMN) [45], there are two types of activities: atomic activities are called *Tasks*, and decomposable activities are known as *Sub-Processes*. Activities can be executed by a system (*automated*) or by people (*manual*). Furthermore, all activities share common attributes and behaviour such as states and state transitions, that is, an activity has a *life cycle* generally characterizing its operational semantics. There is not consensus in literature on the states of the life cycle of a process activity. Several proposals can be found in [45, 69, 51].

The control flow perspective (also known as (a.k.a.) *behavioural dimension*) specifies the *control flow* dependencies between these activities, that is, the order in which the activities of a process must be performed, e.g. some candidate cities must be selected *before* carrying out a voting round. The process modelling languages usually represent the control flow by means of arrows that connect the process elements (i.e. activities, control flow structures, events) to define the dynamic behaviour of the process.

The resource perspective (a.k.a. *organisational dimension*) focuses on the *people, roles, organisational units and any other entities of the organisational model of a company* that are involved in a process, e.g. how the members of the International Olympic Committee are structured in the organisational model. That is, this dimension is focused on the management of the *resources* that participate in a process, in particular the *human resources* involved in the process activities. The people interacting with the process should also be modelled and associated to the process diagram in order to enable the automation of the allocation of work to specific persons at run time. We delve into how resources can be specified in business processes in Section 4.

The data perspective (a.k.a. *informational dimension*) defines the information that must be produced or consumed by activities, that is, the *data* handled in the process, e.g. a document containing the description of the candidate cities, or a report with the summary of each voting round during the selection process.

The execution of process activities may involve managing a large amount of data that can flow throughout the process, which can and should also be represented in the process models. Notations such as BPMN provide two forms of modelling data [45]. First, as *data objects* attached to the process activities or to control flow elements. Each data object can appear multiple times in a process diagram, but each of these appearances references the same data object instance. Optionally, data objects can be associated to *states* (a.k.a. *data states*), representing the state of the information they contain (cf. Figure 1). Second, as *data stores* representing *repositories* from which activities can take information, and where activities can leave information produced as a result of the action performed. These repositories are supposed to be accessible by the process activities and/or the human resources in charge of manipulating the data used in the activities.

The technical perspective makes reference to the different tools or machines that may be required in order to perform certain activities, e.g. the voting may be done electronically so that the votes can be counted automatically. Thus, it is related to the Business Process Management System (BPMS) where the process is deployed and enacted (cf. Section 5 for information about BPMSs).

There is consensus about the aforementioned business process perspectives. Nonetheless, other aspects of processes can also be seen as process perspectives, since they cover the entire BPM life cycle as well. For instance, policies and regulations that constitute the source for compliance checking must be enforced and modeled at design time and implemented at run time and they can be checked before, during of after process execution [**?**].

3 Formalizing Business Processes and Constraints

In this section, we discuss the formalization of business processes and related constraints. The modeling notation used in the Figure 1 is BPMN, the *de facto* standard for process modelling [45]. Apart from BPMN, there are other notations that allow the definition of process models, e.g. Event-driven Process Chain (EPC), Unified Modeling Language (UML) Activity Diagrams, Business Process Execution Language (BPEL) or Web Service Business Process Execution Language (WS-BPEL), Petri Nets, Workflow (WF) Nets and Yet Another Workflow Language (YAWL). All of them support the definition of activities, decision points with alternative paths of execution, event handling, and other elements and structures required to model a process [3]. EPC [52] is a modelling language to specify the temporal and logical relationships between activities of a process. UML is a language pro-

posed by the Object Management Group (OMG) for providing a standard way to visualize the design of a system [25, 49], especially focused on the development of software systems. Since it was not developed specifically for process modelling, business processes are modelled by extending the Activity Diagrams provided by the language by means of the so-called UML profiles. WS-BPEL [1] is a language that allows the specification of Executable and Abstract business processes based on Web services[2]. It is aimed at facilitating the expansion of automated process integration in both the intra-corporate and the business-to-business spaces. Petri Nets [58, 43] are a special form of graphs or finite automata that can be used to represent processes. For the representation, validation and verification of business processes, we can use WF Nets, a subclass of Petri Nets suitable for dealing with complex processes[57, 30]. YAWL [59] extends Petri Nets and WF Nets for process modelling with constructs to address the multiple process instances, advanced synchronization, and cancellation patterns. A Petri net example is shown in Fig. refpetrimapping.

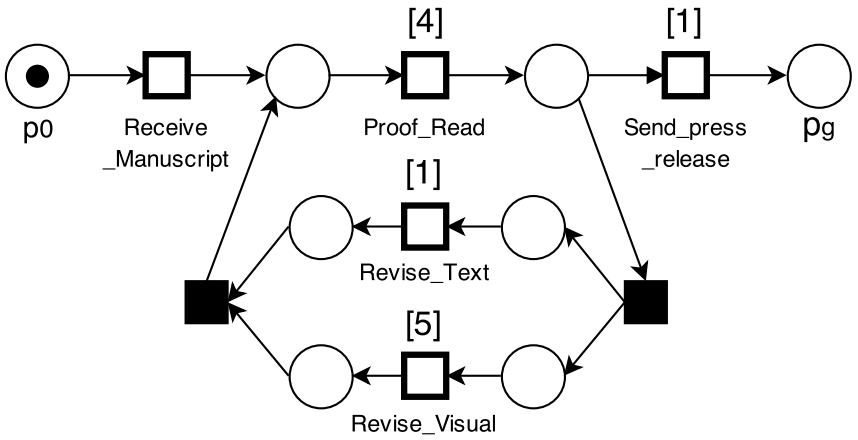
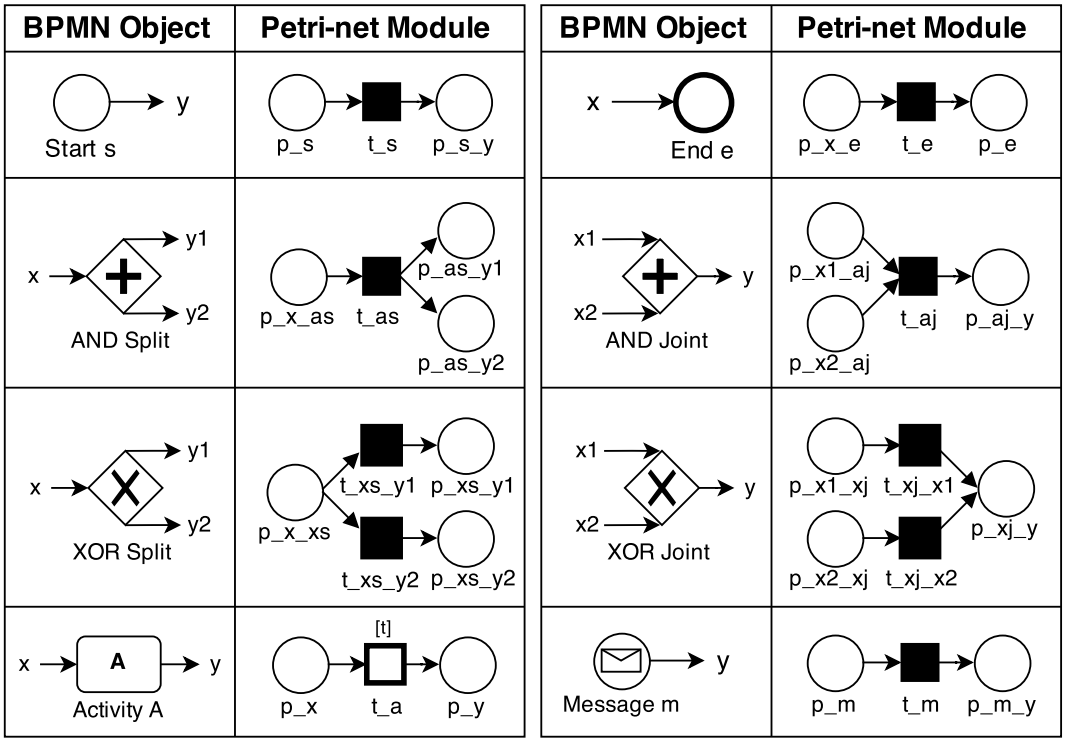


Figure 2 Example timed Petri net

Moreover, there are translations between different modeling notations and Petri nets. For instance, the mapping from BPMN to Petri nets is provided in [20] and summarized in Fig. 3.

Processes can collaborate with other processes and can cross organisational boundaries. The interaction between processes developed in two different organisations is usually performed by means of bidirectional message interchange. In BPMN, this scenario is called *collaboration*. Please, notice that we will use the BPM

[2] According to WS-BPEL [1], "Executable business processes model actual behaviour of a participant in a business interaction, while Abstract business processes are partially specified processes that are not intended to be executed".



Note: x, x1 or x2 represents an input object, y, y1 or y2 represents an output object and [t] represents firing delay.

Figure 3 Mapping from BPMN to Petri nets, adapted from [20]

vocabulary provided by BPMN [45] throughout this document.

3.1 Resource Constraints

There are different kinds of constraints that can be applied on resources in business processes. For instance, the organizational model described in Figure 4 relates roles and resources (*Role* × *Resource*).

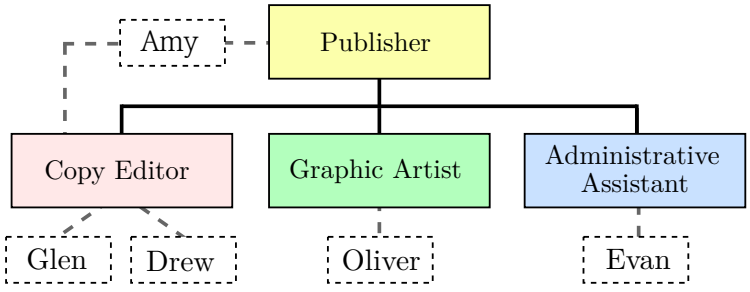


Figure 4 Example organizational model (*Role* × *Resource*)

Activities in a business process are mostly executed by a *Resource* associated with a predefined *Role*. Therefore, we introduce here another relation between

roles and activities ($Role \times Activity$). Using these two relations, resource allocation for a business model can be easily performed by simply assigning an available member who has a role that is in relation with the activity to be executed.

Resources can also be associated with their performances (i.e. Completion time) on specific activities in a business process. Such a relation can be formalized with a triple ($Resource \times Activity \times Performance$). Similarly, a relation for describing the cost of assigning a resource to an activity with a triple ($Resource \times Activity \times Cost$). Such relations can be used to optimize overall completion time of business processes, overall cost of business processes, etc.

3.2 Security and Compliance Constraints (policies)

In the following, we will propose a potential formal representation format for modeling security and compliance constraints. More precisely, we define an abstract syntax as well as formal semantics for the well-known Open Digital Rights Language (ODRL) [36].

Table 2 represents the abstract syntax of ODRL, which can be read as follows:

- text in **bold** represents non-terminal symbols
- text in `typewriter` represents terminal symbols
- text in *italic* represents functions and identifiers
- A^* indicates zero or more occurrences of symbol A
- A^+ indicates one or more occurrences of symbol A
- A? indicates zero or one occurrence of symbol A

A *Policy* contains at least one *PermissionRule* or *ProhibitionRule* and has an associated ODRL *ConflictResolutionStrategy* which is either *permit overrides* (`perm`), *prohibition overrides* (`prohibit`), or *no conflicts allowed* (`invalid`). A *Policy* is applicable, if at least one of the *Rules* contains matches with the request.

A *ProhibitionRule* defines the prohibition of performing an *Action* on an asset by a particular role which are both declared in the *RuleMatch* component of the *ProhibitionRule*. When its *RuleMatch* and *Action* components match a particular request, the applicability of the *ProhibitionRule* can be further constrained by a set of *Constraints*. *Constraints* are represented as boolean formulas that compare a *status* according to an *operator*[3] with a respective *bound*. The *status* of a particular *Constraint* is provided by a respective *Proof* or *DutyProof* that serve as input for the *Constraint*.

[3] Note, that we do not take set operators into account, but see them as a potential extension for further work

	ODRL Policy Components	
Policy	$\mathcal{P}$	::= $\mathcal{P}_{id}$ = [$\langle(\mathcal{PRR}_{id}\|\mathcal{PER}_{id})^{+}\rangle$,$\mathcal{ALG}$]
ProhibitionRule	$\mathcal{PRR}$	::= $\mathcal{PRR}_{id}$ = [$\mathcal{RM}$, $\mathcal{A}$, $\mathcal{CONS}$]
PermissionRule	$\mathcal{PER}$	::= $\mathcal{PER}_{id}$ = [$\mathcal{RM}$, $\mathcal{A}$, $\langle\mathcal{DUR}^{*}_{id}\rangle$, $\mathcal{CONS}$]
DutyRule	$\mathcal{DUR}$	::= $\mathcal{DUR}_{id}$ = [$\mathcal{RM}$, $\mathcal{A}$, $\mathcal{CONS}$]
ConstraintSet	$\mathcal{CONS}$	::= $\mathcal{CONS}_{id}$ = $\langle\mathcal{CON}^{*}_{id}\rangle$
Constraint	$\mathcal{CON}$	::= $\mathcal{CON}_{id}$ = $f^{bool}(status(a), operator(o), bound(a))$
RuleMatch	$\mathcal{RM}$	::= $\mathcal{RM}_{id}$ = $\langle\mathcal{M}^{+}\rangle$
Match	$\mathcal{M}$	::= $\mathcal{M}_{id}$ = $\phi(a)$
Action	$\mathcal{A}$	::= $\mathcal{A}_{id}$ = *action(a)*
	$\phi(a)$	::= *party(a)* | *asset(a)*
	a	::= *value*
	o	::= `eq` | `neq` | `lt` | `lteq` | `gt` | `gteq`
ConflictRes.Strat.	$\mathcal{ALG}$	::= `perm` | `prohibit` | `invalid`
	Query & Proof	
QueryRequest	$\mathcal{Q}$	::= $\langle$*party(a)*?, *action(a)*, *asset(a)*$\rangle$
DutyTarget	$\mathcal{DT}$	::= $\mathcal{DT}_{id}$ = $\langle$*party(a)*?, *action(a)*, *asset(a)*?$\rangle$
DutyProof	$\mathcal{DPF}$	::= $\mathcal{DPF}_{id}$ = [$\mathcal{DT}$,$\mathcal{CON}_{id}$,*status(a)*]
Proof	$\mathcal{PF}$	::= $\mathcal{PF}_{id}$ = [$\mathcal{CON}_{id}$,*status(a)*]
ProofSet	$\mathcal{PFS}$	::= $\langle(\mathcal{DPF}_{id}\|\mathcal{PF}_{id})^{*}\rangle$

Table 2 Abstract Syntax of ODRL

PermissionRules are similarly defined as *ProhibitionRules*, but instead of prohibiting the execution of an *Action* they permit it. Furthermore, a sequence of *DutyRules* can be associated with *PermissionRules*. All associated *DutyRules* must be fulfilled in order for the respective *PermissionRule* to become valid.

A *QueryRequest* contains a particular access request that consists of an action and the respective asset it should be performed on, as well as optional information about the party which shall be performing the action.

Basic Semantics of ODRL Policies

The following section proposes a possible interpretation of the formal semantics of ODRL which differs from earlier approaches defined in [47, 35], as it proposes semantics for the entire core model of ODRL instead of only capturing parts of it.

Match and RuleMatch

Let $\mathcal{M}$ be either a *Match* or a *RuleMatch* component and let $\mathcal{QDT}$ either be a set of possible *QueryRequests* or *DutyTargets*. A match semantic function is a map-

ping $[[\mathcal{M}]] : \mathcal{QDT} \rightarrow \{\mathsf{m}, \mathsf{nm}\}$, where m and nm denote match and no match respectively.

A certain *Match* component (i.e. the attribute value it represents) matches, whenever it is part of a particular *Query* or *DutyTarget*.

$$[[\mathcal{M}_{id}]](\mathcal{QDT}) = \begin{cases} \mathsf{m} & \text{if } \mathcal{M} \in \mathcal{QDT} \\ \mathsf{nm} & \text{if } \mathcal{M} \notin \mathcal{QDT} \end{cases} \tag{1}$$

A *RuleMatch* component (i.e. a set of *Match* components) only matches, if all of its *Match* components are evaluated to m.

$$[[\mathcal{RM}_{id}]](\mathcal{QDT}) = \begin{cases} \mathsf{m} & \text{if } \forall i : [[\mathcal{M}_i]](\mathcal{QDT}) = \mathsf{m} \\ \mathsf{nm} & \text{if } \exists i : [[\mathcal{M}_i]](\mathcal{QDT}) = \mathsf{nm} \end{cases} \tag{2}$$

Action

Let $\mathcal{A}$ be an *Action* component and let $\mathcal{QDT}$ either be a set of possible *QueryRequests* or *DutyTargets*. An action semantic function is a mapping $[[\mathcal{A}]] : \mathcal{QDT} \rightarrow \{\mathsf{m}, \mathsf{broadm}, \mathsf{narm}, \mathsf{reqm}, \mathsf{partm}, \mathsf{nm}\}$, where m denotes match, broadm match of broader action, narm match of narrower action, reqm match of requiring action, partm match of required action, and nm denotes no match.

A certain *Action* component (i.e. the action it represents) matches, whenever it is part of a particular *Query* or *DutyTarget* or if an equivalent action is part of a particular *Query* or *DutyTarget*. Otherwise, it either evaluates to broadm if it is related to a broader action that is part of the *Query* or *DutyTarget*, or to narm if it is related to a narrower action that is part of the *Query* or *DutyTarget*, or to partm if it is related to an action that is part of the *Query* or *DutyTarget* and this action requires the *Action* component for its execution, or to reqm if it requires another action for its execution and this required action is part of the *Query* or *DutyTarget*,

or to nm otherwise.

$$[[\mathcal{A}_{id}]](\mathcal{QDT}) = \begin{cases} \mathsf{m} & \text{if } \mathcal{A}_{id} \in \mathcal{QDT} \text{ or} \\ & \exists i : \mathsf{equals}(\mathcal{A}_{id}, \mathcal{A}_i) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{narm} & \text{else if } \exists i : \mathsf{broader}(\mathcal{A}_i, \mathcal{A}_{id}) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{broadm} & \text{else if } \exists i : \mathsf{broader}(\mathcal{A}_{id}, \mathcal{A}_i) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{partm} & \text{else if } \exists i : \mathsf{requires}(\mathcal{A}_i, \mathcal{A}_{id}) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{reqm} & \text{else if } \exists i : \mathsf{requires}(\mathcal{A}_{id}, \mathcal{A}_i) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{nm} & \text{otherwise} \end{cases} \tag{3}$$

Constraint and ConstraintSet

Let $\boldsymbol{\mathcal{CON}}$ be a *Constraint* component, $\boldsymbol{\mathcal{CONS}}$ a *ConstraintSet* component, and let $\boldsymbol{\mathcal{PFS}}$ be a *ProofSet*. A constraint semantic functions is a mapping $[[\boldsymbol{\mathcal{CON}}]] : \boldsymbol{\mathcal{PFS}} \rightarrow \{\mathsf{t}, \mathsf{f}\}$, where t and f indicate whether the boolean formula represented by $\boldsymbol{\mathcal{CON}}$ holds, given $\boldsymbol{\mathcal{PFS}}$ as input.

This boolean formula is evaluated, if provided *ProofSet* contains a *Proof* that is associated with the respective *Constraint* of the formula. If no associated *Proof* exists, it is evaluated to f.

$$[[\mathcal{CON}_{id}]](\mathcal{PFS}) = \begin{cases} f^{bool}(\mathcal{PF}_i, operator(o), bound(a)) & \text{if } \exists i : \mathcal{PF}_i \in \mathcal{PFS} \wedge \\ & i = id \\ \mathsf{f} & \text{otherwise} \end{cases} \tag{4}$$

A *ConstraintSet* component (i.e. a set of *Constraint* components) only evaluates to t, if all of its *Constraint* components are evaluated to t or the *ConstraintSet* is empty, i.e. there does not exist any associated *Constraints* at all.

$$[[\mathcal{CONS}]](\mathcal{PFS}) = \begin{cases} \mathsf{t} & \text{if } \forall i : [[\mathcal{CON}_i]](\mathcal{PF}_i) = \mathsf{t} \text{ or} \\ & \mathcal{CONS} = \emptyset \\ \mathsf{f} & \text{if } \exists i : [[\mathcal{CON}_i]](\mathcal{PF}_i) = \mathsf{f} \end{cases} \tag{5}$$

DutyRule

Let $\boldsymbol{\mathcal{DUR}}$ be a *DutyRule* component of the form $\mathcal{DUR}_{id} = [\mathcal{RM}, \mathcal{CONS}]$ and let $\boldsymbol{\mathcal{PFS}}$ be a *ProofSet*. A duty rule semantic function is a mapping $[[\boldsymbol{\mathcal{DUR}}]]$:

$\mathcal{PFS} \rightarrow \{\mathsf{t}, \mathsf{f}\}$, where t represents the fulfilment of $\mathcal{DUR}$, and f the exact opposite.

A *DutyRule* component evaluates to t, if there exists at least one *DutyProof* in the *ProofSet* whose *DutyTarget* matches with the *RuleMatch* component of the respective *DutyRule*, and its *ConstraintSet* component returns true. It evaluates to f in any other case.

$$[[\mathcal{DUR}_{id}]](\mathcal{PFS}) = \begin{cases} \mathsf{t} & \text{if } \exists i : \mathcal{DPF}_i \in \mathcal{PFS} \wedge [[\mathcal{RM}_{id}]](\mathcal{DT}_{id}) = \mathsf{m} \wedge \\ & [[\mathcal{A}_{id}]](\mathcal{DT}_{id}) = \mathsf{m} \wedge [[\mathcal{CONS}]](\mathcal{PFS}) = \mathsf{t} \\ \mathsf{f} & \text{otherwise} \end{cases} \quad (6)$$

PermissionRule

Let $\mathcal{PER}$ be a *PermissionRule* component of the form $\mathcal{PER}_{id} = [\mathcal{RM}, \mathcal{A}, \langle \mathcal{DUR}^*_{id} \rangle, \mathcal{CONS}]$ and let $\mathcal{Q}$ be a set of possible *QueryRequests*. A permission rule semantic function is a mapping $[[\mathcal{PER}]] : \mathcal{Q} \rightarrow \{\mathsf{permission}, \mathsf{cper}, \mathsf{cpro}, \mathsf{na}, \mathsf{nm}\}$, where permission represents permission of $\mathcal{Q}$, cper denotes conditional permission of $\mathcal{Q}$, cpro indicates conditional prohibition of $\mathcal{Q}$, and na, nap represent not active and not applicable respectively.

A *PermissionRule* component evaluates to permission, if its *RuleMatch* component matches with $\mathcal{Q}$, its *ConstraintSet* component returns true, and if it has no associated duties. It evaluates to cpro if its *RuleMatch* component matches with $\mathcal{Q}$, its *ConstraintSet* component returns true, but it has at least one associated *DutyRule* component that evaluates to false given a specific *ProofSet* as input. It evaluates to cper if its *RuleMatch* component matches with $\mathcal{Q}$, its *ConstraintSet* component returns true, and all associated *DutyRule* components evaluate to true given a specific *ProofSet* as input. Finally, a *PermissionRule* component evaluates to na if its *RuleMatch* component matches with $\mathcal{Q}$ but its *ConstraintSet* component returns

false, and it evaluates to nap if its *RuleMatch* component does not match with $\mathcal{Q}$.

$$[[\mathcal{PER}_{id}]](\mathcal{Q}) = \begin{cases} \text{permission} & \text{if } [[\mathcal{RM}_{id}]](\mathcal{Q}) = \text{m}, [[\mathcal{A}_{id}]](\mathcal{Q}) \neq \text{nm}, \\ & [[\mathcal{CONS}]](\mathcal{Q}) = \text{t and } \langle \mathcal{DUR}^*_{id} \rangle = \emptyset \\ \text{cpro} & \text{if } [[\mathcal{RM}_{id}]](\mathcal{Q}) = \text{m}, [[\mathcal{A}_{id}]](\mathcal{Q}) \neq \text{nm}, \\ & [[\mathcal{CONS}]](\mathcal{Q}) = \text{t and } \exists i : [[\mathcal{DUR}_i]](\mathcal{PF}_i) = \text{f} \\ \text{cper} & \text{if } [[\mathcal{RM}_{id}]](\mathcal{Q}) = \text{m}, [[\mathcal{A}_{id}]](\mathcal{Q}) \neq \text{nm}, \\ & [[\mathcal{CONS}]](\mathcal{Q}) = \text{t and } \forall i : [[\mathcal{DUR}_i]](\mathcal{PF}_i) = \text{t} \\ \text{na} & \text{if } [[\mathcal{RM}_{id}]](\mathcal{Q}) = \text{m}, [[\mathcal{A}_{id}]](\mathcal{Q}) \neq \text{nm and} \\ & [[\mathcal{CONS}]](\mathcal{Q}) = \text{f} \\ \text{nap} & \text{otherwise} \end{cases} \quad (7)$$

ProhibitionRule

Let $\boldsymbol{\mathcal{PRR}}$ be a *ProhibitionRule* component of the form $\mathcal{PRR}_{id} = [\mathcal{RM}, \mathcal{A}, \mathcal{CONS}]$ and let $\boldsymbol{\mathcal{Q}}$ be a set of possible *QueryRequests*. A prohibition rule semantic function is a mapping $[[\boldsymbol{\mathcal{PRR}}]] : \boldsymbol{\mathcal{Q}} \rightarrow \{\text{prohibition, na, nm}\}$, where prohibition represents the prohibition of $\boldsymbol{\mathcal{Q}}$, na denotes that $\boldsymbol{\mathcal{PRR}}_{id}$ is not active, and nap represents not applicable.

A *ProhibitionRule* component evaluates to prohibition, if its *RuleMatch* component matches with $\boldsymbol{\mathcal{Q}}$ and its *ConstraintSet* component returns true. It evaluates to na if its *RuleMatch* component matches with $\boldsymbol{\mathcal{Q}}$ but its *ConstraintSet* component returns false, and it evaluates to nap if its *RuleMatch* component does not match with $\boldsymbol{\mathcal{Q}}$ (i.e. the rule is not applicable).

$$[[\mathcal{PRR}_{id}]](\mathcal{Q}) = \begin{cases} \text{prohibition} & \text{if } [[\mathcal{RM}_{id}]](\mathcal{Q}) = \text{m}, [[\mathcal{A}_{id}]](\mathcal{Q}) \neq \text{nm and} \\ & [[\mathcal{CONS}]](\mathcal{PF}) = \text{t} \\ \text{na} & \text{if } [[\mathcal{RM}_{id}]](\mathcal{Q}) = \text{m}, [[\mathcal{A}_{id}]](\mathcal{Q}) \neq \text{nm and} \\ & [[\mathcal{CONS}]](\mathcal{PF}) = \text{f} \\ \text{nap} & \text{otherwise} \end{cases} \quad (8)$$

Policy

Let $\boldsymbol{\mathcal{P}}$ be a *Policy* component and let $\boldsymbol{\mathcal{Q}}$ be a set of possible *QueryRequests*. A policy semantic function is a mapping $[[\boldsymbol{\mathcal{P}}]] : \boldsymbol{\mathcal{Q}} \rightarrow \{\text{permission, prohibition, cpro, na, nm}\}$,

where permission represents permission of $\mathcal{Q}$, prohibition represents prohibition of $\mathcal{Q}$, cpro indicates conditional prohibition of $\mathcal{Q}$, and na, nap represent not active and not applicable respectively.

Given a *Policy* $\mathcal{P}_{id}$ of the form $\mathcal{P}_{id} = [\mathcal{RM}, \mathcal{A}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{ALG}}]$ where *Rule* $\mathcal{R}$ either represents a *ProhibitionRule* or a *PermissionRule*, $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$ be the set of all *Rules* of $\mathcal{P}_{id}$, and $\boldsymbol{\mathcal{ALG}}$ denotes the conflict resolution strategy of the *Policy*. $\mathcal{P}_{id}$ is not active, if all $\mathcal{R}$ in $\mathcal{P}_{id}$ are evaluated to na. $\mathcal{P}_{id}$ is not applicable (nap), if all $\mathcal{R}$ in $\mathcal{P}_{id}$ are evaluated to nap. If there is at least one $\mathcal{R}$ in $\mathcal{P}_{id}$ which is neither evaluated to na nor nap, $\mathcal{P}_{id}$ is evaluated to the result returned by the respective conflict resolution strategy $\boldsymbol{\mathcal{ALG}}$ that takes those *Rules* as input.

$$[[\mathcal{P}_{id}]](\mathcal{Q}) = \begin{cases} \mathsf{na} & \text{if } \forall i : [[\mathcal{R}_i]](\mathcal{Q}) = \mathsf{na} \\ \mathsf{na} & \text{if } \exists i : \neg([[\mathcal{R}_i]](\mathcal{Q}) = (\mathsf{permission}|\mathsf{prohibition})) \text{ and} \\ & \exists j : [[\mathcal{R}_j]](\mathcal{Q}) = \mathsf{na} \\ \mathsf{nap} & \text{if } [[\mathcal{RM}_{id}]](\mathcal{Q}) = \mathsf{nm} \text{ and } [[\mathcal{A}_{id}]](\mathcal{Q}) = \mathsf{nm} \\ \otimes_{\boldsymbol{\mathcal{ALG}}}(\boldsymbol{\mathcal{R}}) & \text{otherwise} \end{cases} \tag{9}$$

4 Reasoning over Business Processes and Constraints

Process models that include information related to all the business process perspectives (cf. Section 2) constitute an analysis source to extract information and infer knowledge in an automated way.

There is vast literature on the analysis of the control flow to check, for instance, whether a business process has deadlocks or livelocks [58]. Most of the approaches rely on Petri nets [43] as process representation formalism due to its well-defined semantics. The so-called behavioral profiles [68] describe the different relations between process activities and have been used for several purposes, e.g., for process model abstraction [53] or to check process consistency [66], process compliance [67] and model equivalence [65].

Other approaches target the analysis of the data perspective [41]. Approaches for the automatic generation of a data-centered view of processes [16], data-aware checking of conformance between two process models [42] or data-aware business process model abstraction [32] are examples of work performed in this regard.

However, less attention has been paid so far to the business process resource[4]

[4] We will mainly refer to human resources.

perspective, which, nonetheless, is crucial in the correct execution of processes. In fact, it is one of the main concerns in the scope of SHAPE, as can be seen in the requirements described in Deliverable 4.1 [13]. There are two fundamental actions related to resource management in business processes:

- *Resource assignment* is the definition of conditions or constraints that specify the set of resources that are allowed to take part in a process activity, known as *potential participants* or *potential performers*. The kind of participation can be given by so-called task duties [11], which define the degree of involvement of a resource in an activity. For instance, For instance, the Generic Human Roles defined in BPEL4People [2] describe a resource responsible for the execution of an activity, a resource in charge of approving the work performed, and a resource being informed of the completion of an activity. The so-called RACI matrices [54]) extend this set of task duties to include resources helping in the execution of an activity by providing information required.
- *Resource allocation* is the selection of *actual participants* or *actual performers* among the resources assigned to an activity. There are several allocation mechanisms, such as those detailed in the Workflow Resource Patterns [50]. Prioritization mechanisms may help to decide who must take part in an activity in a specific moment [12].

Unlike resource assignment, which has been often addressed by researchers in the last years [55, 56, 14], resource allocation and work scheduling has tended to be disregarded. In the following, we describe an approach for automated resource allocation in business processes along with a mechanism to automatically analyze the resource perspective at design time (i.e., before a process is executed) that complements the analyses of the other perspectives (i.e., control flow and data) mentioned above. In addition, we will briefly describe how security and policies can be enforced in business process models such that we can automatically check whether processes comply with regulations, laws and policies a process-oriented organization may be subject to.

4.1 Resource Allocation/Work Distribution

Human resources are key elements in business process management as they are responsible for process execution or supervision. Companies must ensure they have the necessary human resources[5] to carry out all the business activities. At

[5] From now on *resources* for the sake of brevity.

best, all the activities of the process instances running concurrently can be allocated to suitable resources so that processes can be completed in an acceptable amount of time. However, lack of resources or an suboptimal work schedule may produce delayed work, leading to a reduced quality in the provided services, an increase of the time required to finish a process and, potentially, an increase in the costs of hiring or re-organising the available resources.

In this section, we address the problem of allocating the resources available in a company to the activities in the running process instances in a time optimal way, i.e., such that process instances are completed in the minimum amount of time. Our approach lifts limitations of prior research pursuing similar goals, which assumes simplified non-cyclic processes and does not necessarily search for an optimal resource allocation [62, 48].

We represent business processes as timed Petri nets in this Section. Therefore a brief preliminary about Petri nets and timed Petri nets are included as follows.

4.1.1 Petri Nets

Petri nets [44] provide a diagrammatic tool to model concurrency and synchronization in distributed systems. A *Petri net* is defined as a 4-tuple $N = \langle P, T, F, M_0 \rangle$, where P is a finite set of *places*, T is a finite set of *transitions*, with $P \cap T = \emptyset$, $F \subset (P \times T) \cup (T \times P)$ describes a bipartite graph. A *marking*(state) $M : P \to \mathbb{Z}^+$ assigns to each place a non-negative integer, denoting number of tokens in places, where M_0 is the *initial marking*. Each arc $f \in F$ either connects a place to a transition $(P \times T)$ or a transition to a place $(T \times P)$. The set of places on incoming arcs of a transition $t \in T$ is called input place set $\bullet t$ and the set of places on outgoing arcs of t is called output place set $t\bullet$. As a complimentary definition we can also talk about the input transition set $\bullet p$ of a place and vice versa. A transition may fire when all input places $p \in \bullet t$ have tokens. When a transition t fires, written $\xrightarrow{t}$, it consumes all tokens in places $\bullet t$ and it produces tokens in each place $p \in t\bullet$. Petri nets whose places can contain at most one token in any state are called *1-safe Petri nets*. A marking M_k is *reachable* from a marking M_{k-1} in *one step* if $M_{k-1} \xrightarrow{t_{k-1}} M_k$. A firing sequence of transitions $\overrightarrow{\sigma} = \langle t_1 t_2 ... t_n \rangle$ changes the state of the Petri net at each firing: $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{\cdots} M_n$. Two important properties of Petri nets that we need with the scope of this paper are *soundness* and *free-choiceness*. Soundness means that from every reachable state, a proper final state can be reached in N. Thus, a successful execution is guaranteed. In a free-choice Petri net, every two transitions t_1 and t_2, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$. Further properties of Petri nets are detailed in [63].

Timed Petri Nets. Several models have been developed to handle time in Petri nets [6, 61, 46]. We adopt the concept of timed Petri nets described in [46], which associates a firing finite duration $c(t)$ with each transition $t \in T$ of the net: a *timed Petri net* is defined by a 5-tuple $N_T = \langle P, T, F, c, M_0 \rangle$ such that $\langle P, T, F, M_0 \rangle$ is a Petri net and $c : T \to \mathbb{N}$ is a function that assigns firing delays to every transition $t \in T$. Fig. 2 shows an example of a timed Petri net: circles represent places, squares represent transitions, and numbers in brackets on transitions denote firing delays. Filled squares denote "silent" transitions that have no firing delays, i.e., $c(t) = 0$. However, note that also normal transitions that correspond to activities can have no delay, e.g., t_m in Fig. 2.

4.1.2 Automated Reasoners

For automated resource allocation, we need an efficient automated reasoner. There are four admissible alternatives supporting different formalisms that are appropriate for our purpose. Namely, they are

Flora-2: $Flora - 2$ is an advanced object-oriented knowledge representation and reasoning system. The formalism it requires is a dialect of F-logic [38] with numerous extensions, including meta-programming in the style of HiLog [17], logical updates in the style of Transaction Logic [8], and defeasible reasoning [64].

STeLP: *STeLP* is a solver for Answer Set Programming with temporal operators [10]. Taking as an input a particular kind of logic program with modal operators, STeLP obtains its set of temporal equilibrium models that are represented in terms of a deterministic Büchi automaton capturing the complete program behaviour.

clingo: *clingo* is an answer set solver for normal and disjunctive logic programs [28]. It combines the high-level modelling capacities of Answer Set Programming (ASP) [9, 27, 5, 39], which is a declarative knowledge representation and reasoning formalism, with state-of-the-art techniques from the area of Boolean constraint solving.

dlvhex: *dlvhex* system is a logic-programming reasoner for computing the models of so-called HEX-programs [23], which are an extension of answer-set programs towards integration of external computation sources.

We formalise our domain in ASP and used *clingo* as automated reasoner. This decision is made due to high performance of *clingo* [34] and the big community size of ASP. Further investigations on problem-based performance evaluation between these reasoners are planned as a future work. Now, we describe the syntax and the semantics of ASP.

4.1.3 ASP

Answer Set Programming (ASP) is a declarative programming paradigm oriented towards solving combinatorial search problems along with declaratively defined background knowledge in the form of rules.

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

$$A_0 :- A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n. \tag{10}$$

where $n \geq m \geq 0$ and each $A_i \in \sigma$ is an (first-order) atom, with the exception of A_0, which can be either an atom or bottom $\bot$. A rule is called a *fact* if $m{=}n{=}0$ and a *constraint* if A_0 is $\bot$. We generally omit the $:-$ sign for facts, and the $\bot$ sign for constraints. In (10), *not* is meant to stand for "default negation" (sometimes referred to as "negation as failure"). Consider an example program $\Pi_1 = \{lights\_on.\ shop\_open :- lights\_on, not\ door\_locked.\}$ With these two rules, intuitively, *lights_on* should be derived since it is given as a fact, plus *shop_open* should also be derived, because *lights_on* is derivable and there is no reason to assume *door_locked*, as it is not derivable from any other rule. While this example considers only propositional atoms in rules, whenever A_i is a first-order predicate with variables within a rule of the form (10), 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. For instance, let $\Pi_2 = \{p(1).\ p(2).\ q(X) :- p(X).\}$ then $ground(\Pi_2) = \{p(1).\ p(2).\ q(1) :- p(1).\ q(2) :- p(2).\}$.

Semantics. An interpretation I of a ground ASP program Π is a set of atoms. A rule is *satisfied* by an interpretation I if the head (A_0) of the rule is true whenever the *body* of the same rule is true with respect to I, that is $A_1, \ldots, A_m \in I$ and $A_{m+1}, \ldots, A_n \notin I$. An interpretation I satisfies a program Π if it satisfies all the rules in the program, and it is called a *model* of Π. An *answer set* of a program Π that does not contain negation as failure is defined to be the unique subset-minimal Herbrand model (i.e., intuitively, a subset of all rule heads) of Π. Now consider a program Π that may contain negation. A model I is an *answer set* for Π if it is an answer set of the reduct Π^I of Π w.r.t. I, defined as the set of rules $\{A_0 :- A_1, \ldots, A_m. \mid \{A_{m+1}, \ldots, A_n\} \cap I = \emptyset\}$ for all rules of the form (10) in Π. For instance, for Π_1 above the unique answer set is $\{lights\_on,\ shop\_open\}$. Note that programs can have several answer sets, e.g. $\Pi_3 = \Pi_1 \cup \{door\_locked :- lights\_on, not\ shop\_open\}$ would have the additional answer set $\{lights\_on,\ door\_locked\}$.

Incremental ASP (iASP). Many problems conveniently 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 in a planning problem) is not known upfront, and therefore such problems are addressed by considering one problem instance after another while gradually increasing the parameter k. However, re-processing repeatedly the entire problem is a redundant approach, which is why an extension of *clingo* natively supports incremental computation of answer sets (iASP)[26, 28]; the intuition is rooted in treating programs in program slices (extensions). Each time the parameter is increased, a successive extension of the program is considered altogether with existing program slices. Redundancy in solving programs that require an incremental parameter can be reduced in this incremental approach.

An iASP program is a triple $(B, A[k], Q[k])$, where B describes the *static* knowledge, and $A[k]$ and $Q[k]$ are ASP programs parameterized by a single parameter k that ranges over positive integers. In the iterative answer set computation of iASP, while the knowledge derived from the rules in $A[k]$ accumulates as k increases, the knowledge obtained from $Q[k]$ is only considered for the latest value of k. For this reason, $A[k]$ and $Q[k]$ are named *cumulative* knowledge and *volatile* knowledge, respectively. More formally, an iASP solver computes in each iteration i

$$\Pi[i] = B \cup \bigcup_{1 \leq j \leq i} A[k/j] \cup Q[k/i]$$

until an answer set for some (minimum) integer $i \geq 1$ is found. We will demonstrate next, how iASP can be successfully used to model and solve various variants of resource allocation problems in business process management.

The bottom layer is the generic iASP encoding Π_N for finding a firing sequence between initial and goal markings of a 1-safe Petri net N. This provides a marking of N at each value of parameter k. On a second layer we extend Π_N towards Π_T to encode timed Petri Nets, i.e., we support business processes encoded as timed Petri nets whose activities can have a duration. Consequently, this encoding cannot only compute possible markings, but also the overall duration for a firing sequence. In other words, now we also know about the value of the overall time spent time at a firing sequence of length k. In the upper layer Π_R, we include rules and constraints about resources in order to encode an iASP program that allocates activities to available resources for a certain period of time.

Please, note some general assumptions that we make about the structure of a resource allocation problem: (i) no resource may process more than one activity at a time; (ii) each resource is continuously available for processing; (iii) no preemption, i.e., each activity, once started, must be completed without interruptions;

$A_N[k]$:

$\{\texttt{fire(T,k,I)} : \texttt{inPlace(P,T)}, \texttt{instance(I)}\}.$ (11)

$\texttt{:-fire(T,k,I)}, \texttt{instance(I)}, \texttt{inPlace(P,T)}, \texttt{not tokenAt(P,k,I)}.$ (12)

$\texttt{tokenAt(P,k,I):-fire(T,k}-1\texttt{,I)}, \texttt{outPlace(P,T)}, \texttt{instance(I)}.$ (13)

$\texttt{:-inPlace(P,T1)}, \texttt{inPlace(P,T2)}, \texttt{T1!} = \texttt{T2}, \texttt{fire(T1,k,I)}, \texttt{fire(T2,k,I)},$ (14)
$\texttt{instance(I)}.$

$\texttt{consumeToken(P,k,I):-inPlace(P,T)}, \texttt{fire(T,k,I)}, \texttt{instance(I)}.$ (15)

$\texttt{tokenAt(P,k,I):-tokenAt(P,k}-1\texttt{,I)}, \texttt{not consumeToken(P,k}-1\texttt{,I)}.$ (16)

$Q_N[k]$:

$\texttt{:-not tokenAt(p}_g\texttt{,k,I)}, \texttt{instance(I)}.$ (17)

Figure 5 1-safe Petri net formulation in iASP

and (iv) the processing times are independent of the schedule, and they are known in advance. These assumptions are common in related approaches [62].

4.1.4 Π_N : A Generic Formulation of 1-safe Petri Nets

Based on the notions introduced in preliminary section for Petri nets, we formalise the firing dynamics of 1-safe Petri net $N = \langle P, T, F, M_0 \rangle$ in an iASP program $(B_N, A_N[k], Q_N[k])$. Given a goal state M_k, which for the sake of simplicity we assume to be defined in terms of a single goal place p_g, the aim is to find a shortest possible firing sequence in number of firings $\overrightarrow{\sigma} = \langle t_1 t_2 ... t_k \rangle$ that does not violate the constraints, from M_0 to M_k. The formulation is shown in Fig. 5.

B_N: $N = \langle P, T, F, M_0 \rangle$ is represented by facts using predicates $\texttt{inPlace}_N\texttt{(p,t)}$ and $\texttt{outPlace}_N\texttt{(p,t)}$ that encode F. We encode different instances i of N by the unary predicate $\texttt{instance}_N$, which allows us to run the allocation problem against different instances of the same process; initial markings of instance M_{0i} are defined as facts via predicate $\texttt{tokenAt}_N\texttt{(P}_0\texttt{,k}_0\texttt{,i)}$ where for each $p \in P_0$, $M_0(p) = 1$.[6]

$A_N[k]$: Rule (11) guesses all subsets of possible firing actions for each instance of N. Constraint (12) ensures that any transition t is fired only if all input places in $\bullet t$ have tokens. Rule (13) models the effect of the action `fire` on output places by assigning a token to each output place in the step following the firing. Constraint (14) prohibits concurrent firings of transitions $t \in p\bullet$. Rules (15) and (16) preserve tokens at place p in successive steps if none of the transitions $t \in p\bullet$ fires.

$Q_N[k]$: Finally, constraint (16) in Fig. 5 enforces a token to reach the goal place

[6] Since in the following we only consider instances of the same Petri Net, we will drop the suscript N in the predicates.

p_g (for all instances $i \in I$). The computation ends as soon as this constraint is not violated in an iteration of the iASP program, i.e., it computes the minimally necessary number of iterations k to reach the goal state.

4.1.5 Π_T : Activity Scheduling using Timed Petri Net

In order to model activity durations, we extend the above iASP encoding towards Timed Petri nets: that is, Π_N is enhanced with the notion of time in Π_T. By doing so, $\Pi_N \cup \Pi_T$ becomes capable of scheduling activities in instances of a timed Petri net N_T.

B_T: We expand the input of Π_N with facts related to time and with the rules that are independent from the parameter k. For each fact $\texttt{tokenAt}(\texttt{p}_0, \texttt{k}_0, \texttt{i})$ previously defined we add in B_T a fact $\texttt{timeAt}(\texttt{p}_0, \texttt{c}_0, \texttt{k}_0, \texttt{i})$ where c_0 is the initial time at p_0. In order to distinguish activity transitions and ("silent") non-activity transitions[7], we add facts `activity(t)` for all activities. Durations of activities are specified with facts `timeActivity(t,c)` where t is an activity and $c \in \mathbb{Z}^+$. The remainder of B_T is given by rules (18)+(19) in Fig. 6: rule (18) defines firing delays of each transition in N and rule (19) assigns duration zero to activity transitions per default, where the delay is not otherwise specified.

$A_T[k]$: Rule (22) defines the effect of action `fire` on `timeAt` for all output places $t\bullet$ where t is a *non-activity* transition. In this case, the maximum time among the input places, which is computed by rules (20) and (21), is propagated over all output places. As opposed to (22), rule (23) defines the effect of action `fire` on `timeAt` for *activity* transitions. Time value derived in rule (23) for the next step is the sum of the maximum time value at the input places and the value of the activity duration. Rule (24) conserves the time value of a place in the succeeding step k in case the transition does not fire at step $k-1$.

$Q_T[k]$: On top of $Q_N[k]$, an optimization statement (25) is added for computing answer sets with the minimum time cost.

4.1.6 Π_R : Resource Allocation

In the last layer of our iASP program, Π_R, we additionally formalise resources and related concepts. $\Pi_N \cup \Pi_T \cup \Pi_R$ allow allocating resources to activities for a time optimal execution of all defined instances of N_T.

---

[7] Recall: in Petri nets representing business processes, activity transitions are empty squares while silent transitions are represented in filled squares (cf. Fig. 2).

B_T :

```
firingDelay(T,C):-timeActivity(T,C).                                    (18)
firingDelay(T,0):-not timeActivity(T,_),activity(T).                    (19)
```

$A_T[k]$:

```
greTimeInPlace(P1,T,k,I):-inPlace(P1,T),inPlace(P2,T),fire(T,k,I),      (20)
                          timeAt(P1,C1,k,I),timeAt(P2,C2,k,I),P1!=P2,
                          C1<C2,instance(I).
maxTimeInPlace(P,T,k,I):-inPlace(P,T),not greTimePlace(P,T,k,I),        (21)
                         fire(T,k,I),instance(I).
timeAt(P2,C,k,I):-not activity(T),fire(T,k-1,I),outPlace(P2,T),         (22)
                  maxTimeInPlace(P,T,k-1,I),timeAt(P,C,k-1,I),
                  instance(I).
timeAt(P2,C,k,I):-activity(T),fire(T,k-1,I),outPlace(P2,T),             (23)
                  maxTimeInPlace(P,T,k-1,I),timeAt(P,C,k-1,I),
                  firingDelay(T,D),C=C1+D,instance(I).
timeAt(P,C,k,I) :-not consumeToken(P,k-1,I),inPlace(P,T),               (24)
                  timeAt(P,C,k-1,I),instance(I).
```

$Q_T[k]'$:

```
#minimize{timeAt(p_g,C,k,I) : instance(I)=C}                            (25)
```

Figure 6 Scheduling extension

B_R**:** The facts related to resources and organisational models are defined in the input of Π_T. An example organisational model is shown in Fig. 4. Facts `hasRole(r,l)` relates a resource r to a role l. Activities are related to a role via facts of the form `canExecute(l,t)`, which means that a role l is allowed to performing an activity t. An optional estimated duration for a resource to execute an activity can be defined by `timeActivityResource(t,l,c)`. Similarly an optional estimated duration for a role per activity can be defined by `timeActivityRole(t,l,c)`. Both can override the default `timeActivity(t,c)`. In particular, the order ($>$) preferred in resource-time allocation is `timeActivityResource>timeActivityRole>timeActivity`. This is especially useful when a resource or a role is known to execute a particular activity in a particular amount of time, which can be longer or shorter than the default duration of the activity. In our program (cf. Fig. 7) this preference computation is encoded in rules (26)-(30). Rule (26) and (27) are projections of optionally defined activity execution durations. Rules (28)-(30) derive correct execution duration for resource-activity pairs considering both mandatory and optional duration knowledge.

$A_R[k]$**:** In the iterative part, rule (30) allocates a resource r to an activity t from

B_R :

existsTimeActivityResource(T,R):-timeActivityResource(T,R,C). (26)

existsTimeActivityRole(T,L):-timeActivityRole(T,L,C),hasRole(R,L). (27)

takesTime(T,R,C):-timeActivityResource(T,R,C). (28)

takesTime(T,R,C):-timeActivityRole(T,L,C),hasRole(R,L),canExecute(L,T), (29)
not existsTimeResource(T,R).

takesTime(T,R,C):-firingDelay(T,C),hasRole(R,L),canExecute(L,T), (30)
not existsTimeActivityResource(T,R),
not existsTimeActivityRole(T,L).

$A_R[k]$:

{assign(R,T,C,C2,k,I) : takesTime(T,R,C),C2 = C + D}:-inPlace(P1,T), (31)
timeAt(P1,C,k,I),activity(T),instance(I).

timeAt(P2,C2,k,I):-activity(T),assign(R,T,C1,C2,k − 1,I), (23)*
fire(T,k − 1,I),outPlace(P2,T),instance(I).

assigned(T,k,I):-assign(R,T,C1,C2,k,I). (32)

:-not assigned(T,k,I),fire(T,k,I),activity(T),instance(I). (33)

:-assign(R,T,C1,C2,K,I),assign(R1,T,C3,C4,K,I),R! = R1. (34)

:-assign(R,T1,C1,C2,K1,I1),assign(R,T2,C1,C2,K2,I2),C1! = C2,T1! = T2. (35)

:-assign(R,T,C1,C2,K1,I1),assign(R,T,C1,C2,K2,I2),C1! = C2,I1! = I2. (36)

:-assign(R,T1,C1,C2,K1,I1),assign(R,T2,C1,C2,K2,I2), (37)
C1! = C2,I1! = I2,T1! = T2.

:-assign(R,T,B1,B2,K1,I),assign(R,T2,A1,A2,K2,I2),A1 > B1,A1 < B2. (38)

:-assign(R,T,B1,B2,K1,I),assign(R,T2,A1,A2,K2,I2),A2 < B2,A2 > B1. (39)

Figure 7 Allocation extension

time c to time $c2$. Note that, for handling optional execution durations, rule (23) from Fig. 6 is replaced by rule (23)*. Rule (32) along with constraint (33) prohibits any firing of an activity transition that is not allocated to a resource. Constraint (34) ensures that an activity cannot be assigned to more than one resource. Constraints (35)-(37) guarantee that only one resource is assigned to one activity at a time. Constraints (38) and (39) prevent a resource to be assigned when it is busy.

4.1.7 Time Relaxation

In case a resource is busy at the time when s/he is required for another activity, our program would be unsatisfiable as it is. We add rules (40) and (41) (cf. Fig. 8) into $A_T[k]$ for allowing the demanding activity to wait until the required resource is available again.

$A_T[k]'$:

$$\texttt{relaxationAt}(\texttt{P},\texttt{C}+1,\texttt{k},\texttt{I})\texttt{:-timeAt}(\texttt{P},\texttt{C},\texttt{k}-1,\texttt{I}),\texttt{inPlace}(\texttt{P},\texttt{T}),\texttt{activity}(\texttt{T}), \quad (40)$$
$$\texttt{not consumeToken}(\texttt{P},\texttt{k}-1,\texttt{I}),\texttt{instance}(\texttt{I}).$$

$$\texttt{timeAt}(\texttt{P},\texttt{C},\texttt{k},\texttt{I})\texttt{:-relaxationAt}(\texttt{P},\texttt{C},\texttt{k},\texttt{I}). \quad (41)$$

Figure 8 Time relaxation for optimality

The extended version of this section is submitted to *Business Process Management 2015*. Therefore, an example scenario and the performance evaluation can be found in [33].

4.2 Resources Analysis

We have identified and formally defined a set of Person-Activity analysis operations related to how resources are involved in process activities, which can be divided into three categories: basic operations, consistency checking operations and criticality checking operations. We next summarize them together with a mechanism to automatically execute them based on Description Logics(DLs). The complete information can be found in [14].

4.2.1 Person-Activity Analysis Operations

All the operations have been defined to be as reusable as possible.

4.2.1.1 Basic Person-Activity Operations

These operations analyse the relations between the activities of a process and the people who can perform them according to the resource assignments. There are four basic person-activity operations, one of which (Potential Participants) has already been identified in the literature.

Potential Participants (PP) The PP operation takes an activity and a task duty and returns the people who are candidates to perform that specific task duty for the activity specified. Thus, at design time, a person is a potential participant of an activity for a specific task duty if there is *some* process instance in which she can be an actual performer of that task duty[8].
Although obtaining the potential participants of an activity is sometimes straightforward, the presence of access-control constraints in processes may make it

[8] Note that from this definition, *participant* and *performer* can be used as synonyms in this context.

significantly more difficult, especially when they affect loops. We refer the reader to [14] for further information.

This operation serves for studying or checking whether people are involved in specific types of activities as well as for detecting security problems derived from an incorrect assignment of permissions in terms of activity execution, i.e., a person who was supposed to be involved in an activity but cannot take part in it due to the assignment. It is also useful to detect activities that can be assigned to the same resources and, hence, are candidates for aggregation when creating an executable BP model [21]. Furthermore, typical operations for set comparison used in Set Theory [24] can be applied to this operation, e.g., to determine whether the potential participants in two given activities are exactly the same resources.

Potential Activities (PA) The PA operation lists the activities that may be allocated to one resource with regard to a specific task duty during a process instance execution. It takes the identity of a specific person and the task duty to be checked, and it returns the activities that can be potentially allocated to this person for that task duty.

This operation is useful to provide people with a personalised list of all of the activities they may be involved in or to identify the requirements for someone who is going to substitute a certain person in the organisation. It is also useful to detect the degree of involvement of a person in a process in terms of the number of activities in which she can take part. Moreover, similar to potential participants, typical operations for set comparison can also be used to determine, for instance, whether the set of activities that can be allocated to a specific person is a subset of the set of activities potentially allocated to another person.

Non-potential Activities (NPA) The NPA operation takes a person and a task duty and calculates the activities in which she *cannot* perform that task duty, if any.

This operation is useful when one is interested in increasing the responsibilities of a person in the organisation. The outcome of this operation is a set of activities whose resource assignments are candidates to be changed to include the resource at hand.

Non-participants (NP) The NP operation takes an activity and a task duty and returns the people who can never participate in the activity performing that task duty, if any.

This operation is a way to quickly detect the relationship between people and processes in an organisation, making it easier to ensure that certain resources

do not have access to processes that are not aligned with their duties or responsibilities in the company. Such duties may be defined in the form of access-control policies of people to specific types of processes or activities.

4.2.1.2 Consistency Checking Person-Activity Operation

This category of operations includes just one operation focused on checking whether for all activities of the process there is at least one person who is allowed to perform the task duty for any execution of the activity. Specifically, the *consistency checking (CC)* operation takes a task duty and returns whether the process model is consistent with regard to that task duty, i.e., if it is always possible to find a potential participant for an activity during any execution of the process for that task duty. This definition is based on the definition of consistency introduced in [7], although it has been extended to address task duties.

An inconsistent process may result in behavioural problems at run time because there may not be anyone to whom some task duty can be allocated in case the activity needs to be executed in a process instance. Therefore, this operation is fundamental to ensure the correct operation of the business process resource perspective, as it detects situations in which the process could fall into a deadlock.

4.2.1.3 Criticality Checking Person-Activity Operations

Apart from consistency, one aspect that is relevant to resource assignment is checking whether there is only one person who is authorised to perform a certain activity of the process. Identifying these people is useful for reducing the vulnerability of the organisation to failure, which, according to Malone et al. [40], is strongly related with the possibility to replace one resource with another. The two novel operations introduced next detect weak points of a process in the face of resource unavailability.

Critical Participants (CP) One or more people are critical participants of a process if they have to be allocated to one or more activities because there are no more potential participants for them. The CP operation takes a task duty and returns the members of the organisation who are critical in the execution of a process for that task duty.

The simplest case is when there is only one potential participant for an activity. However, this operation also has to take into account situations that may appear in the presence of access control constraints. We refer the reader to [14] for further information.

A process with a critical participant for task duty Responsible is a process whose execution may eventually depend on one unique person. This fact may make the organisation vulnerable in the sense that it may depend on one specific person to complete one of its business processes. Therefore, this operation is useful for identifying those people who have this particular relevance in the organisation. Furthermore, it is also useful as a mechanism to identify potential bottlenecks without the need to gather and analyse process execution logs.

Critical Activities (CA) An activity is a critical activity for a given task duty if it has only one potential participant for that task duty. The CA operation takes a person and a task duty and returns the critical activities in which that person is involved with the given task duty.

Detecting the activities of a process that can only be performed by one person helps pinpoint potential bottlenecks without the need to gather and analyse process execution logs. It is also useful for obtaining the activities whose resource assignments should be modified temporarily or permanently when a specific person is unavailable for a specific (or indefinite) period of time to avoid process deadlocks.

4.2.2 Automated Analysis at Design Time

We show that the automated analysis of the business process resource perspective is possible with an implementation of the analysis operations described in Section 4.2.1 based on DLs and target at design-time analysis, i.e., with static information before a process is executed. To that end, all the required information must be included in a DL-based Knowledge Base (KB). Specifically, a KB comprises two components, the *TBox* and the *ABox*. The TBox describes *terminology*, i.e., the KB in the form of *concepts* and *property* definitions, and their relations; the ABox contains *assertions* about individuals using the terms from the TBox. For our purpose, the following mappings to DLs are required: the mapping of the organisational information of the company, i.e., an organizational model with roles, positions, resource skills, and the like; the mapping of the process elements, i.e., the process model with resource-related information being analyzed; and the mapping of the resource assignment conditions (cf. Section 4) from the resource assignment language used. We rely on the mappings explained in [14], which are summarized in Deliverable D4.1 [13]. We use Resource Assignment Language (RAL) [15] to define the resource assignment conditions because it is more expressive than other notations, as shown in [11, 14].

Before defining the analysis operations in terms of standard DL reasoning operations, it is necessary to introduce the DL-based KB that will be used.

▶ **Definition 1** (DL-based knowledge base $\mathcal{K}_C$)**.** Let O be an organisational model, bp be a business process, and ρ be a resource assignment for the activities of bp. $\mathcal{K}_C$ is a DL-based KB obtained after mapping the elements of O, bp, and ρ into DLs using the mappings described in [14] and summarized in Deliverable D4.1 [13], and including the following axioms:

1. For every activity a in the business process that is not in a loop: $\{a\} \sqsubseteq \leq 1 isOfType^-$
2. For every activity a in the business process: $\{a\} \sqsubseteq \geq 1 isOfType^-$

Equipped with the KB $\mathcal{K}_C$, the person-activity analysis operations can be formulated in terms of standard DL reasoning tasks that are implemented by most DL reasoners. In particular, the following DL reasoning tasks are used.

- Concept subsumption, which is the problem of deciding whether a concept C_1 is subsumed by another concept C_2 with respect to a KB $\mathcal{K}$. In particular, we are interested in obtaining all concepts that are subsumed by a concept C_1 and denote this reasoning task as $subconcepts_{\mathcal{K}}$.
- Concept retrieval, which is the problem of computing the set containing exactly every instance of a concept C with respect to a KB $\mathcal{K}$. We denote this reasoning task as $individuals_{\mathcal{K}}$.
- Consistency, which is the problem of deciding whether a KB $\mathcal{K}$ is consistent. We denote this reasoning task as $consistent_{\mathcal{K}}$.

4.2.2.1 Basic Person-Activity Analysis Operations

The non-participants of an activity a for task duty d are those people p for which there is no $i_a \in AI_a$ such that $d(i_a, p)$, i.e., those people p such that $p \in Person \sqcap \neg\exists d^-.AI_a$. This corresponds to the concept retrieval reasoning task, and hence, the non-participants operation can be expressed in terms of a DL reasoner as follows:

$$NP(a^{bp}, d^{bp}) = individuals_{\mathcal{K}_C}(Person \sqcap \neg\exists d^-.AI_a)$$

Having the non-participants of an activity a for a task duty d, the potential participants of a for task duty d can be obtained as those people who are not non-participants of a for task duty d because for any person p and task duty d, it holds that $PP(a,d) \cup NP(a,d) \equiv Person$, and $PP(a,d) \cap NP(a,d) = \emptyset$.

The same approach can be followed for the operations that obtain the activities in which a person can participate. The non-potential activities of a person p for

task duty d are those activities for which there is no $i_a \in AI_a$ such that $d(i_a, p)$. Therefore, an activity a is a non-potential activity of a person p regarding a task duty d if its activity instances $AI_a \sqsubseteq ActivityInstance \sqcap \neg\exists d.\{p\}$. This corresponds with the concept subsumption reasoning task as follows:

$$NPA(p^{bp}, d^{bp}) = subconcepts_{\mathcal{K}_C}(ActivityInstance \sqcap \neg\exists d.\{p\})$$

Finally, similar to potential participants, the potential activities of a person p for a task duty d can be obtained as those activities of the process that are not amongst its non-potential activities.

4.2.2.2 Consistency Checking Person-Activity Operations

As proven in [14], checking the consistency of a business process is equivalent to checking its so-called α-consistency. Furthermore, the α-consistency of a process can be computed by checking the consistency of $\mathcal{K}_C^1$.

As a result, the consistency checking operation can be expressed in terms of the consistency reasoning task as follows:

$$CC \Leftrightarrow consistent_{\mathcal{K}_C^1}$$

4.2.2.3 Criticality Checking Person-Activity Operations

The two criticality checking person-activity operations can be defined in terms of DL reasoning tasks as follows. A person p is a critical participant for task duty d if there is a subset of activities in the process such that p has to be allocated to task duty d of some activity instance of any of these activities in any possible execution that involves any of them. In other words, a person p is critical if $\mathcal{K}_C$ entails that p participates with task duty d in some activity instance of the process $\mathcal{K}_C \models p \in \exists d^-.ActivityInstance$, which can be easily computed using a DL reasoner by means of the concept retrieval reasoning task:

$$CP(d^{bp}) \equiv individuals_{\mathcal{K}_C}(\exists d^-.ActivityInstance)$$

An activity a is critical for person p and task duty d if p is the only person who can perform task duty d in activity a. In other words, a is critical if $AI_a \sqsubseteq \exists d.\{p\}$. Therefore, to obtain all critical activities of a person, the concept subsumption reasoning task can be used as follows:

$$CA(p^{bp}, d^{bp}) \equiv subconcepts_{\mathcal{K}_C}(\exists d.\{p\})$$

4.3 Security and Compliance in Business Processes

Security or in general compliance in business processes can be understood as consistence of a set of rules in business processes against a set of rules stated in some regulatory documents (e.g. business contracts). Those documents provide general guidance in how parties agreed to work with each other, i.e. ensuring compliance of business processes according to regulatory documents means to check whether the specification and all possible execution paths of a business process comply with a normative document regulating the domain of the process [29].

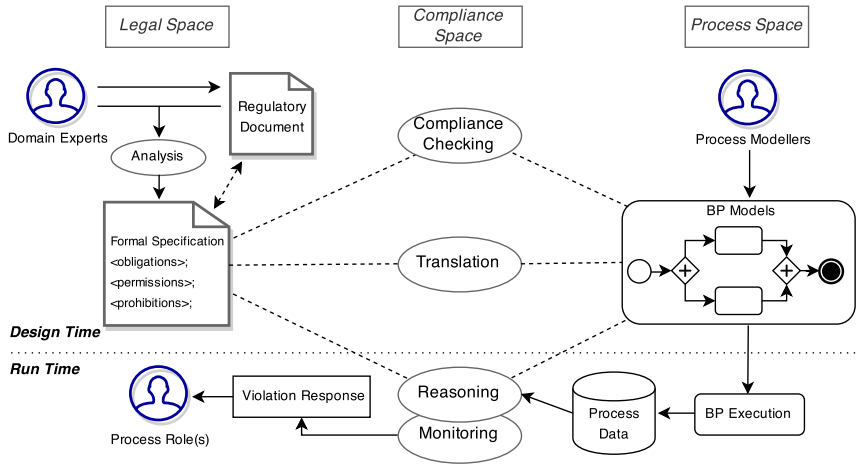


Figure 9 *Compliance Space* as linkage between *Legal Space* and *Business Process Space* taken from [29].

Furthermore, compliance of business processes with regulatory documents is important to integrate the two perspectives of legal and business process spaces which are usually separated from each other (cf. Figure 9).

While in the legal domain, the focus lies on contract negotiation, contract drafting and ensuring the legal validity of contracts according to relevant laws such as business contracts law and various regulations, the business process perspective deals with management science, such as various business process re-engineering approaches.

The compliance space however, is particularly driven by recent regulative and legislative acts, which require the establishment of stronger and more enforceable compliance requirements against the target set of rules. Ensuring compliance of business processes with business contracts is a complex problem which was mostly done through manual checking and analysis of regulatory documents. However, approaches towards formalisation of contract conditions [4], open new

possibilities for an increasing role of tools to support contract analysis (e.g. to identify clause inconsistency), and automated reasoning over compliance rules.

5 Business Process Management Systems/Suites

The development of transaction management (Business Process Management (BPM)) has made rapid progress within the last years. More and more companies use corresponding tools particularly for process automation - Business Process Management Suites (BPMS).
An evaluation of currently existing tools on the market will promote the decision which tool satisfies both formal modelling processes (also in the concern of resource allocation) and integrating these models in a BPMS capable to deal with Siemens environment.

5.1 Criteria for Evaluation

The whole list within all of its details will be available in the next deliverable since the evaluation is not finished yet. But to provide an insight into the extensive evaluation progress, tools are evaluated and selected based on criteria in

- **design time validation** (e.g. is the format of BPMN 2.0 supported, and process realization)
- **runtime validation** (e.g. process execution with deadlocks)
- **execution support in resource allocation & scheduling** (e.g. how is it supported to allocate resources to tasks, how can constraints on resource allocation be modelled and how is it supported to schedule allocations)
- **adaptivity** (e.g. changes at runtime - how are they supported, through which API the tool is extensible)

5.2 Notation in BPMS

The notation of business processes is the first level for Business Process Modeling Suites (BPMS) to design processes in a human-readable way and make them interoperable with different BPMS at the same time. Since there exist some methods for converting formal models of processes to a notation interoperable with BPMS, this will be investigated in the next deliverable.
BPMN [45] is one of the most advanced and widely used representation of business processes. In its actual form, BPMN 2.0 is the most recent release to enable a metamodel and interchange format beside the notation. This is a critical point

to decide which BPMS (Business Process Modeling Suites) to use for further process representation and execution. For instance, the tool Bizagi claims to support BPMN 2.0. But instead using the BPMN2 XML standard Bizagi creates an XML file only readable for Bizagi so that the advantage of BPMN 2.0 to interchange diagrams easily is not given. The focus on this tool definitely lies on the usability of the software. On the other hand, this is exactly what makes this tool remarkable, the user is guided through a convenient wizard step by step. So the great advantage of BPMN 2.0 for BPMSs is the handling of imports and exports because BPMN 2.0 allows saving the model as a BPMN2 XML file which is accepted from tools like Bonita, Camunda (formerly known as Activiti) and jBPM. With a validation module, syntax errors are detected, which would produce an erroneous BPMN2 XML file.

5.3 Extensions

Most of the BPMSs have a web service API to extend the process externally. Due to the possibility of interchanging the metadata of a process diagrams via BPMN2 XML format, this can also be seen as an extension. Table 3 visualizes an overview of extensions of tools.

5.4 Some Suites evaluated

According to the evaluation criteria tools will evaluated and selected. To obtain an overview what tools exist on market, the study [37] which is a mix of open source and enterprise tools evaluation give an insight of strengths and weaknesses of BPMS.

The **Adobe** solution is recommended if a convenient and appealing process execution for end users is important and the processing of PDF documents across different input and output channels represents the central concern of the BPM project. The solution is less recommended when issues such as process controlling or technical process modeling with BPMN are relevant, or at least an exchange is requested with appropriate modeling tools.
The solution of **Axon Active** is recommended when a powerful and intuitive process implementation using BPMN is required in the development environment, while professional and technical models should be supported. The solution, however, is less recommended, for example, when an appealing user interface for end users without previous adaptation ("out of the box") must be available, or when a special task on BPM Governance aspects (such as rights management and life

Table 3 API - Extensions

Suite	Notation	API
Adobe	BPMN, but not fully supported	Service invocation via Java utilizes Remote Method Invocation (RMI) and utilizes a strongly typed API
Axon Active	BPMN 2.0 on graphically standard but not stored in BPMN2 XML file	SAP, MS-SharePoint
Bizagi	BPMN 2.0 on graphically standard but not stored in BPMN2 XML file	The information exchange between Bizagi and the external system is performed through JSON files. RESTful services and SOAP Web services allow systems and portals to integrate
Bosch SI	BPMN 2.0 stored in XML	HTTP, WS, REST
Camunda	BPMN 2.0 is used; Graph is stored in BPMN2 XML file.	Java makes it very flexible and extendable. Powerful REST API
Prologics	BPMN but focus on designing even with Gantt diagram; Word, Excel, XML export	MS-SharePoint, SAP Integration
SAP	BPMN 2.0 is used; Graph is stored in BPMN2 XML file.	Java makes it very flexible and extendable. Powerful REST API
Vitria	BPMN 2.0 on graphically standard but not stored in BPMN2 XML file	Social Acitivity: Twitter, Facebook, WebServices: ESB
Bonita	BPMN 2.0 is used; Graph is stored in BPMN2 XML file.	REST API
IBM	BPMN 2.0 is used; Graph is stored in BPMN2 XML file.	REST API
ProcessMaker	BPMN	Nothing available because it is a Web UI
jBPM	BPMN 2.0 is used; Graph is stored in BPMN2 XML file.	REST API

cycle management) is located.

The solution of **BizAgi** is recommended if a comfortable, simple yet powerful (functional and technical) process modeling, process implementation and process execution is desired based on BPMN. The user interface of the modeling tools is leaning against other MS Office, which facilitates entry. The solution is also recommended if a simple integration of external systems or good possibilities of Controlling are desired options. The solution is known, however, to recommend less if comprehensive delegation capabilities are needed or when short response

times within the development environment a relevant selection criterion represent.

The solution from **Bosch SI** is recommended for high standards of BPM governance (such as rights management and life cycle management) or when runtime management (in particular task delegation) is important. Various adapters and a very flexible customizable end-user portal are further essential aspects. Less recommendable is the solution when a separation of professional and technical models is wanted but a common language should be used, since in addition to BPMN its own proprietary language for the implementation process is used. Even for the more complicated use of the development environment, extensive development knowledge should be available, what the solution can thus appear less suitable for business users.

The solution of **Camunda** is recommended when highly individual, performance standards compliant solutions with Java and based on BPMN models are to be developed, with the possibility of this even free of charge to do on open-source basis. The solution is recommended if a BPM Suite is needed to be integrated into an existing application. The solution is less recommended if the customer provides no Java developers, as this is an essential prerequisite. Many things are only feasible if it they are implemented individually using Java.

The solution of **PROLOGICS** is recommended if special requirements to BPM governance (such as rights management and life cycle management) are needed and a separation of professional and technical modeling is desired. The solution is also recommended when controlling process or a good development and test environment is required. In addition, it is useful if a strong integration with Microsoft products is sought and a high degree of usability is required in the development environment. The solution is less recommended if the suite is to be operated outside the Microsoft world or clustering functionality is required.

The **SAP** solution is recommended if BPM governance and control management play an important role or there are high administrative requirements. Numerous adapters to external systems and thus high flexibility when integrating and good interaction with other SAP products are further essential advantages. The solution is less recommended when simple process implementation and usability / user experience are particularly important.

The solution of **Vitria** recommended when process control, flexible mash-ups and skills for complex event processing as well as a web-based development are important requirements. The solution, however, is less recommendable when ease of process implementation, usability/user experience, as well as adequate runtime management are required.

To sum up, process modeling, process execution and the integration are realized by systems very well in most products, but there is still potential to improve in the areas of runtime management and process controlling. Also in the area of process execution and BPM-Governance almost all supplier have potential to improve. Concerning this project where the transformation from formal modelled processes to a BPMN notation is highly required, that part cannot be done from BPMS since their starting point is at most a standardized notation like BPMN 2.0. The step to convert into BPMN for executing at runtime by any APIs will be investigated in the next deliverable.

6 Summary

In this deliverable we have investigated the state-of-the-art on existing models for processes, resources, constraints and security, and their underlying formalisms. As a result of the work, we outlined some of the latest literature on business process modeling (cf. Section 2), different formalizations and reasoning techniques (cf. Section 3 and 4, respectively) as well as a brief survey on latest business process management systems and suites (cf. Section 5).

For the next deliverable we plan to tackle other requirements of SHAPE described in Deliverable 4.1 [13] that concern WP2. Specifically, we will try some of the formalisms presented in this deliverable to solve the same problems addressed with ASP so far and we will extend our approach to allocate not only individual resources but also teams to process activities.

References

1 Web Services Business Process Execution Language v2.0. Technical report, OASIS, 2007.

2 WS-BPEL Extension for People (BPEL4People). Technical report, OASIS, 2009.

3 V. Andrikopoulos, S. Benbernou, M. Bitsaki, O. Danylevych, M.S. Hacid, W.J. van den Heuvel, D. Karastoyanova, B. Kratz, F. Leymann, M. Mancioppi, K. Mokhtari, C.N. Nikolaou, M.P. Papazoglou, and B. Wetzstein. Survey on Business Process Management. Technical report, 2008.

4 Tara Athan, Harold Boley, Guido Governatori, Monica Palmirani, Adrian Paschke, and Adam Wyner. Oasis legalruleml. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law*, ICAIL '13, pages 3–12, New York, NY, USA, 2013. ACM.

5 Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.

6 B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Soft. Eng.*, 17(3):259–273, Mar 1991.

7 Elisa Bertino, Elena Ferrari, and Vijay Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2:65–104, February 1999.

8 Anthony J Bonner and Michael Kifer. A logic for programming database transactions. In *Logics for databases and information systems*, pages 117–166. Springer, 1998.

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

10 Pedro Cabalar and Martín Diéguez. Stelp–a tool for temporal answer set programming. In *Logic Programming and Nonmonotonic Reasoning*, pages 370–375. Springer, 2011.

11 Cristina Cabanillas. *Enhancing the Management of Resource-Aware Business Processes*. PhD thesis, University of Seville, December 2012.

12 Cristina Cabanillas, José María García, Manuel Resinas, David Ruiz, Jan Mendling, and Antonio Ruiz Cortés. Priority-Based Human Resource Allocation in Business Processes. In Samik Basu, Cesare Pautasso, Liang Zhang, and Xiang Fu, editors, *ICSOC*, volume 8274 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2013.

13 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.

14 Cristina Cabanillas, Manuel Resinas, and Antonio Ruiz Cortés. Specification and Automated Design-Time Analysis of the Business Process Human Resource Perspective. *Inf. Syst.*, page In press., 2015.

15 Cristina Cabanillas, Manuel Resinas, and Antonio Ruiz-Cortés. RAL: A High-Level User-Oriented Resource Assignment Language for Business Processes. In *Business Process Management Workshops (BPD'11)*, pages 50–61, 2011.

16 Cristina Cabanillas, Manuel Resinas, Antonio Ruiz-Cortés, and Ahmed Awad. Automatic Generation of a Data-Centered View of Business Processes. In *CAiSE*, volume 6741, pages 352–366, 2011.

17 Weidong Chen, Michael Kifer, and David S Warren. Hilog: A foundation for higher-order logic programming. *The Journal of Logic Programming*, 15(3):187–230, 1993.

18 Thomas H. Davenport. *Process innovation: reengineering work through information technology*. Harvard Business School Press, Boston, MA, USA, 1993.

19 Gero Decker. *Design and Analysis of Process Choreographies*. PhD thesis, University of Potsdam, 2009.

20 Remco M Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in bpmn. *Information and Software Technology*, 50(12):1281–1294, 2008.

21 Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.

22 Johann Eder and Walter Liebhart. Workflow Recovery. In *CoopIS*, pages 124–134, 1996.

23 Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12):1495–1539, 2008.

24 H.B. Enderton. *Elements of Set Theory*. Acad. Press, 1977.

25 UML Revision Task Force. OMG Unified Modeling Language Specification, Version 1.4 (final draft), Februrary 2001.

26 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. Engineering an incremental ASP solver. In *Logic Programming*, pages 190–205. Springer, 2008.

27 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

28 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo= asp+ control: Extended report. Technical report, Technical report, 2014.

29 Guido Governatori, Zoran Milosevic, and Shazia Sadiq. Compliance checking between business process and business contracts. In *Proceedings of the 10th IEEE Conference on Enterprise Distributed Object Computing*, pages 16–20. IEEE Computer Society, October 2006.

30 Katalina Grigorova. Process modelling using Petri nets. In *Proceedings of the 4th international conference conference on Computer systems and technologies: e-Learning*, CompSysTech '03, pages 95–100, 2003.

31 Michael Hammer and James Champy. *Reengineering the corporation: a manifesto for business revolution*. HarperBusiness, New York, 1st ed. edition, 1993.

32 Josefine Harzmann, Andreas Meyer, and Mathias Weske. Deciding Data Object Relevance for Business Process Model Abstraction. In *International Conference on Conceptual Modeling (ER)*, pages 121–129, 2013.

33 Giray Havur, Cristina Cabanillas, Axel Polleres, and Jan Mendling. Automated Resource Allocation in Business Processes with Answer Set Programming. *Business Process Management*, submitted to BPM 2015 (on 2015.03.22).

34 Marijn JH Heule and Torsten Schaub. What's Hot in the SAT and ASP Competitions. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2015.

35 Markus Holzer, Stefan Katzenbeisser, and Christian Schallhart. Towards formal semantics for ODRL. In *Proceedings of the First International Workshop on the Open Digital Rights Language (ODRL), Vienna, Austria, April 22-23, 2004*, pages 137–148, 2004.

36 Renato Iannella, Susanne Guth, Daniel Pähler, and Andreas Kasten. Odrl: Open digital rights language 2.1. W3C ODRL Community Group, 2012. http://www.w3.org/community/odrl/.

37 Fraunhofer IESE. Studie - BPM SUITES 2013. http://www.iese.fraunhofer.de/, 2013.

38 Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, 42(4):741–843, 1995.

39 Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1):39–54, 2002.

40 Thomas W. Malone. Modeling Coordination in Organizations and Markets. *Management Science*, 33(10):1317–1332, October 1987.

41 Andreas Meyer, Sergey Smirnov, and Mathias Weske. Data in Business Processes. *EMISA Forum*, 31(3):5–31, 2011.

42 Andreas Meyer and Mathias Weske. Weak Conformance between Process Models and Synchronized Object Life Cycles. In Xavier Franch, Aditya K. Ghose, Grace A. Lewis, and Sami Bhiri, editors, *ICSOC*, volume 8831 of *LNCS*, pages 359–367. Springer, 2014.

43 T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541 –580, apr 1989.

44 Tadao Murata. Petri nets: Properties, analysis and applications. *IEEE*, 77(4):541–580, 1989.

45 OMG. BPMN 2.0. Recommendation, OMG, 2011.

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

47 Riccardo Pucella and Vicky Weissman. A Formal Foundation for ODRL. *CoRR*, abs/cs/0601085, 2006.

48 A. Rozinat and R. S. Mans. Mining CPN Models: Discovering Process Models with Data from Event Logs. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN*, pages 57–76, 2006.

49 James E. Rumbaugh, Ivar Jacobson, and Grady Booch. *The unified modeling language reference manual*. Addison-Wesley-Longman, 1999.

50 N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Resource Patterns. Technical report, BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

51 Nick Russell, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *CAiSE*, pages 216–232, 2005.

52 August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. *Process Modeling using Event-Driven Process Chains*, pages 119–145. John Wiley and Sons, Inc., 2005.

53 Sergey Smirnov, Matthias Weidlich, and Jan Mendling. Business Process Model Abstraction Based on Synthesis from Well-Structured Behavioral Profiles. *Int. J. Cooperative Inf. Syst.*, 21(1):55–83, 2012.

54 Michael Smith. Role And Responsibility Charting (RACI). In *Project Management Forum (PMForum)*, page 5, 2005.

55 Mark Strembeck and Jan Mendling. Modeling process-related RBAC models with extended UML activity models. *Inf. Softw. Technol.*, 53:456–483, 2011.

56 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.

57 Wil M. P. van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer Berlin / Heidelberg, 1997.

58 Wil M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

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

60 Will .M.P van der Aalst, Arthur H.M. ter Hofstede, and Mathias Weske. Business process Management: A survey. In *Business Process management*, volume 2678, pages 1–12. Springer, 2003.

61 W.M.P. van der Aalst. Interval Timed Coloured Petri Nets and Their Analysis. In *International Conference on Application and Theory of Petri Nets*, pages 453–472, 1993.

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

63 W.M.P. van der Aalst. *Structural characterizations of sound workflow nets*. Eindhoven University of Technology, Department of Mathematics and Computing Science, 1996.

64 Hui Wan, Benjamin Grosof, Michael Kifer, Paul Fodor, and Senlin Liang. Logic programming with defaults and argumentation theories. In *Logic Programming*, pages 432–448. Springer, 2009.

65 Matthias Weidlich, Remco M. Dijkman, and Mathias Weske. Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences. *Comput. J.*, 55(11):1398–1418, 2012.

66 Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. *IEEE Trans. Software Eng.*, 37(3):410–429, 2011.

67 Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, Jan Mendling, and Mathias Weske. Process compliance analysis based on behavioural profiles. *Inf. Syst.*, 36(7):1009–1025, 2011.

68 Matthias Weidlich, Mathias Weske, and Jan Mendling. Change Propagation in Process Models Using Behavioural Profiles. In *International Conference on Services Computing (SCC)*, pages 33–40. IEEE Computer Society, 2009.

69 M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer Verlag, 2012.