Software and Systems Modeling (2020) 19:1415–1441
https://doi.org/10.1007/s10270-020-00820-7

SPECIAL SECTION PAPER

The RALph miner for automated discovery and verification of resource-aware process models

Cristina Cabanillas[1,2] · Lars Ackermann[3] · Stefan Schönig[4] · Christian Sturm[3] · Jan Mendling[2]

Received: 14 December 2018 / Revised: 14 July 2020 / Accepted: 22 July 2020 / Published online: 8 August 2020

Abstract
Automated process discovery is a technique that extracts models of executed processes from event logs. Logs typically include information about the activities performed, their timestamps and the resources that were involved in their execution. Recent approaches to process discovery put a special emphasis on (human) resources, aiming at constructing resource-aware process models that contain the inferred resource assignment constraints. Such constraints can be complex and process discovery approaches so far have missed the opportunity to represent expressive resource assignments graphically together with process models. A subsequent verification of the extracted resource-aware process models is required in order to check the proper utilisation of resources according to the resource assignments. So far, research on discovering resource-aware process models has assumed that models can be put into operation without modification and checking. Integrating resource mining and resource-aware process model verification faces the challenge that different types of resource assignment languages are used for each task. In this paper, we present an integrated solution that comprises (i) a resource mining technique that builds upon a highly expressive graphical notation for defining resource assignments; and (ii) automated model-checking support to validate the discovered resource-aware process models. All the concepts reported in this paper have been implemented and evaluated in terms of feasibility and performance.

Keywords Model checking · Organisational mining · Process mining · Process verification · RALph · Resource assignment · Resource mining

1 Introduction

Process mining extracts relevant information on executed business processes from historical data stored in event logs and analyses it for different purposes [59]. *Process discovery* organises the information extracted in the form of a process model. The richer the data in the event logs, the more facets of the underlying processes that can be discovered. Such event data typically refers to the executed activities, their timestamps and the human resources (for short resources) that were involved. The functional (activities), behavioural (control flow) and organisational (resources) perspectives of business processes can thus be discovered [24]. Most of the recent process mining techniques focus on the two former perspectives and generate textual as well as graphical representations of the discovered processes [23,45]. The target of those approaches have been both routine (or procedural) processes, which are usually modelled with imperative notations

Communicated by Rainer Schmidt and Jens Gulden.

This work was funded by the Austrian Science Fund (FWF)—Grant V 569-N31 (PRAIS); and by MCI/AEI/FEDER, UE—Grant RTI2018-100763-J-100 (CONFLEX).

✉ Cristina Cabanillas
cristinacabanillas@us.es

Lars Ackermann
lars.ackermann@uni-bayreuth.de

Stefan Schönig
stefan.schoenig@ur.de

Christian Sturm
christian.sturm@uni-bayreuth.de

Jan Mendling
jan.mendling@wu.ac.at

[1] University of Seville, Seville, Spain

[2] Vienna University of Economics and Business, Vienna, Austria

[3] University of Bayreuth, Bayreuth, Germany

[4] University of Regensburg, Regensburg, Germany

(e.g., Business Process Model and Notation (BPMN) [39]); and flexible processes, for which declarative notations are preferred (e.g., Declare [58]). A key challenge of approaches that yield rules, such as Declare constraints, based on support thresholds can produce rule sets that are inconsistent. For this reason, the constructed process models require an additional check in order to avoid or fix conflicting constraints [21]. To this end, *model checking* addresses the automatic verification of properties of a model. Regarding activities and control flow, a number of approaches to check process soundness [42] have been developed (e.g., [14,33]).

As far as the organisational perspective is concerned, *resource mining* aims at discovering resource assignment constraints (or rules) specifying who is allowed to execute the process activities depending on the roles, skills and a number of properties defined in organisational models. Such constraints range from simple role-based assignments to complex constraints over several activities, like separation of duties [46,54]. The resulting *resource-aware process models* ideally contain expressive resource assignments including simple as well as complex constraints. First, the expressive power of the generated model depends on the resource assignment language used. Most of the existing resource assignment languages target either imperative or declarative process model notations with differing expressiveness, and so far only textual representations of the output resource assignment constraints have been provided. While benefits of graphical notations have been acknowledged [32,37,44], an approach capable of generating expressive graphical resource-aware process models has been missing. Second, resource-aware model checking aims at analysing properties of business process models related to resources. For example, it checks whether all resource assignment constraints can be satisfied with the available resources, or identifies resources that are critical for process completion. An initial set of analysis operations have been defined [17] but comprehensive support for their automated resolution is missing. Finally, the existing resource mining and resource-aware model-checking approaches have been separately developed relying on different types of resource assignment languages. Hence, support for the automated verification of automatically discovered graphical resource-aware process models is an important research gap.

In this paper, we present the RALph Miner as an integrated solution with a two-fold contribution. First, we define a technique to discover resource-aware process models. It mines the constraints from event logs and represents the resulting resource-aware process models using RALph [16]. RALph is an expressive graphical notation for defining resource assignments, which is independent of the process modelling notation. Second, we introduce a novel model-checking approach that allows for the verification of RALph-aware process models in conjunction with the recent extension of the Declare language for defining multi-perspective process models (MP-Declare [12]). For this purpose, we rely on the Alloy framework [29]. We show how previously defined analysis operations are supported. To validate our solution, the RALph Miner has been implemented and the two techniques have been tested with a real use case from the university domain.

This paper builds on previous work [20] that introduced the RALph Miner as a novel way of mining resource-aware graphical imperative process models together with BPMN. We have developed automated resource-aware process verification capabilities. The RALph Miner is meant to be interactively used by process analysts. To assure an appropriate response time, the performance of the verification technique has been tested in experiments based on example models. The performance of the resource mining approach was assessed in [20].

We have followed the Design Science Research Methodology (DSRM) [40] to develop and evaluate our approach. The *problem identification and motivation phase* is covered by Sect. 2, which exemplifies the research problems addressed. The *objective of the solution phase* is tackled by Sect. 3, which describes the pursued requirements as well as the existing support for them. The *design and development phase* involves Sects. 4–6, which first provide an overview and then explain the different components of our approach, the RALph Miner. Finally, the *demonstration and evaluation phases* are included in Sect. 7, which shows the evaluations conducted with details of the implementations, applications and performance tests. The paper finishes with the conclusions drawn from the work and a discussion of limitations and directions for future work in Sect. 8.

2 Background: resource-aware process discovery and verification

Process participants are responsible for the correct operation of the business processes of an organisation. The specification of *who must do what* is known as *resource assignment* and is usually done based on organisational information. Organisational data include organisational units, positions, roles and characteristics of the people, such as skills. For instance, the research group (organisational unit) depicted in Fig. 1 is structured as a hierarchy of positions. The group is led by a professor (SJ) accountable for the work of two secretaries (KH, RR) and three researchers (BR, SS, CC). It would be reasonable to assign resources to process activities based on their positions, for example.

One of the most frequent activities related to scientific research is the management of trips for attending conferences or giving invited research talks. In a typical business trip management process, a researcher first applies for a business

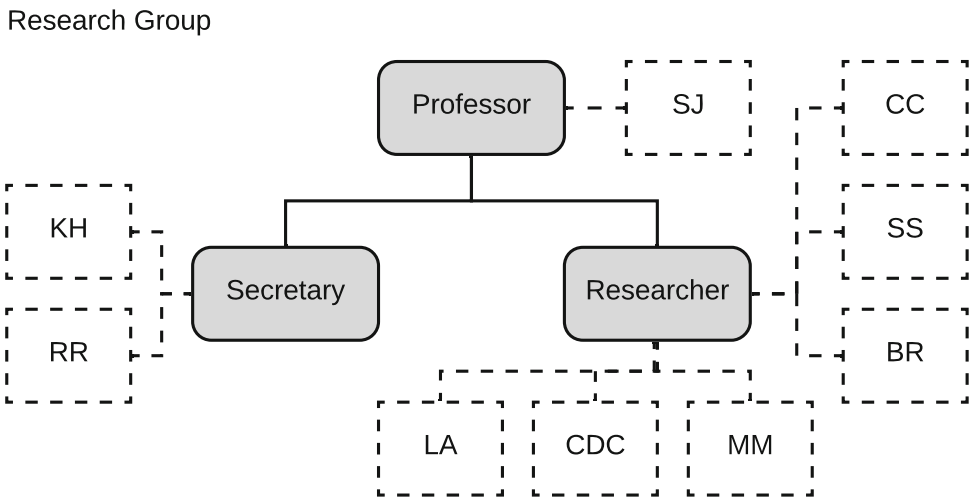


Fig. 1 Hierarchy of positions within a research group

trip, which must be approved by their immediate superior. Once the trip request is approved, the applicant is in charge of booking the accommodation required and of buying the respective transport tickets. Finally, all the documentation is stored by one of the secretaries in order to preserve it for potential future needs (e.g., internal audits). This process description indicates that the process must be always executed in a similar way (procedural process). This kind of process definition is usually specified with imperative process modelling notations like BPMN [39] or Event-driven Process Chains (EPCs) [35]. A more flexible specification of the same process could be in place in another organisational unit, in which the process is described as follows: if a work trip application is approved, the respective trip request has to have been placed eventually before. The approval has to be done by a Professor. When the accommodation has been booked, the documentation is eventually stored in the system. This description, based on rules, leaves room for variation in the execution of the process (flexible process). For example, it is not specified when the transport tickets must be bought or whether they have to be bought at all. Similarly, it could be the case that other activities are performed between the application for the trip and its approval, and between the booking of the accommodation and the documentation storage, respectively. Flexibility also concerns the organisational perspective, in this case. Flexible process definitions are better specified with declarative process modelling notations like Declare [58], DCR Graphs [26] or Declarative Process Intermediate Language (DPIL) [64].

Process executions are stored in event logs (i.e., machine-recorded files that report on the execution of tasks during the enactment of the instances of a given process).

In an event log, every process instance corresponds to a sequence (*trace*) of recorded entries, namely, *events*. The *trace length* corresponds to the number of events the trace consists of. Each event is defined by a set of *attributes*. These attributes typically involve an explicit reference to the enacted task and to the operating resource [59]. For instance, the following excerpt of a business trip event log encoded using the XES format [60] shows the recorded information of the *completion event* of an instance of the activity *Apply for trip* performed by the resource *SS*.

```
<event>
<string key="org:resource" value="SS"/>
<date key="time:timestamp"
     value="2017-08-06T14:58:00.000+01:00"/>
<string key="concept:name" value="Apply for trip"/>
<string key="lifecycle:transition" value="complete"/>
</event>
```

As different activity instances could be executed by different resources, it is necessary to infer the actual resource assignment rules from the event log data. The organisational information is crucial for that purpose. Following up on the previous example, we observed five instances of the activity executed by *SS*, three by *CC* and three by *BR*. The analysis of such data along with the organisational model depicted in Fig. 1 reveals that the activity *Apply for trip* is performed by a resource with the position *Researcher*. The output resource assignment rules should be specified together with the functional and behavioural perspectives of the process in the resulting process model, being it imperative or declarative. Furthermore, the discovery of both simple as well as complex resource assignment rules should be supported. *Resource mining* addresses the automated discovery of resource-aware process models whose expressive power greatly depends on the language used to define the resource assignment constraints.

Verifying the correctness of the output model is crucial in order to avoid inconsistencies at run time. Imagine that resource mining over the event log of our running scenario discovered a binding of duties constraint between activities *Apply for trip* and *Approve application*. This constraint specifies that the two activities must be performed by the same

resource. However, this can never hold true if the former activity is executed by a researcher and the latter by a professor because these sets of resources are disjoint (cf. Fig. 1). Indeed, rule sets that are mined from process executions can be inconsistent if a support threshold below 100% is chosen [21]. Rule sets can also be modified by process analysts before they are put into operation. With the help of *model checking* [22] we can detect inconsistencies and apply the required adjustments or recovery actions.

3 Problem scope and state of the art

In this section, we first define functional and non-functional requirements that we pursue and which have already been used in similar contexts. Afterwards, we analyse the state of the art with respect to such requirements.

3.1 Resource mining and resource-aware model checking requirements

We aim at discovering expressive resource-aware process models, which implies that we have to use an expressive notation for defining resource assignment constraints. The expressiveness of a resource assignment language is often related to the *workflow resource patterns* called *creation patterns* [46], which describe various ways in which resources can be distributed in process activities. The more patterns a mining approach can discover, the higher expressive power it has. Therefore, providing support for discovering the creation patterns leads to requirements for an expressive resource mining (RM) approach, specifically:

(RM1) Automated discovery of Direct Assignments (Dir), i.e., activities assigned to a specific resource.
(RM2) Automated discovery of Role-Based Assignments (Rb), i.e., activities assigned to a specific organisational role.
(RM3) Automated discovery of Capability-Based Assignments (Cb), i.e., activities assigned to a specific capability.
(RM4) Automated discovery of Organisational Assignments (Org), i.e., activities assigned to a resource based on their organisational position and their organisational relationship with other resources.
(RM5) Automated discovery of Separation of Duties (SoD), i.e., the obligation of allocating two tasks to different resources in a given process instance.
(RM6) Automated discovery of Case Handling (CH), i.e., the allocation of all the activity instances within a given process instance to the same resource.
(RM7) Automated discovery of Retain Familiar or Binding of Duties (BoD), i.e., the obligation of allocating an activity instance to the same resource that performed a preceding activity instance within a given process instance.
(RM8) Automated discovery of History-Based Assignments (Hb), i.e., activity instances assigned to resources based on their execution history.

Note that a requirement for the creation pattern *Deferred Assignment* was not included because this pattern is related to run time and cannot be inferred from the log data. Furthermore, the *Authorisation* and *Automatic Execution* patterns were disregarded because they are unrelated to the definition of resource assignment constraints. Note also that the separation of duties and binding of duties patterns are discussed in research on access-control constraints, too [54].

Graphical notations have the advantage of grouping together all the information that is used. They also support a number of perceptual inferences that may be easier to understand for humans than textual specifications [32]. We pursue the graphical representation of the discovered resource assignment constraints and the capability of dealing with procedural as well as flexible processes:

(RM9) Graphical representation of the discovered resource assignments (Gr): the resource assignments can be graphically specified together with the process control flow.
(RM10) Independence of the process modelling notation (Ind): the resource assignments can be defined over imperative as well as declarative process models.

The second group of requirements aim to develop a model-checking technique that allows for the verification of different properties of the output resource-aware process models. Analysis operations to check resource-aware processes have been defined in the Role-Based Access Control (RBAC) and workflow management literature [8,17,54,55,61]. The goal of the model-checking technique is to provide automated design-time support for the largest possible set of operations in order to cover a variety of needs. Providing support for the eight analysis operations that we have found formally defined in the literature (which are more technically described in Sect. 4.3) constitute the model-checking (MC) requirements in this work, in particular:

(MC1) Support for the Potential Participants (PP) operation, which infers the resources that can participate in a process activity given the resource assignments in the process model.
(MC2) Support for the Potential Activities (PA) operation, which infers the activities that can be allocated to a specific resource given a resource-aware process model.

(MC3) Support for the Non-potential Activities (NPA) operation, which infers the activities that can never be allocated to a specific resource given a resource-aware process model.
(MC4) Support for the Non-participants (NP) operation, which infers the resources that can never participate in the a process activity given the resource assignments in the process model.
(MC5) Support for the Satisfiability (SF) operation, which checks whether the available resources can complete a workflow given its resource-aware process model.
(MC6) Support for the Consistency Checking (CC) operation, which checks whether it is *always* possible to complete a workflow given its resource-aware process model.
(MC7) Support for the Critical Participants (CP) operation, which infers the resources that are *necessary* to be able to execute a business process.
(MC8) Support for the Critical Activities (CA) operation, which infers the activities of the process that can cause a deadlock because only one specific resource can execute them.

Finally, we aim to address two non-functional (NF) requirements. The former pursues the development of an integrated approach. Having individual solutions for specific requirements can be beneficial in certain situations (e.g., when it can be assured that only procedural processes will have to be discovered or the analysis functionalities are not needed) but it limits their application under different circumstances. The need of integration has already been described in the literature [15]. The latter non-functional requirement relates to the end users of the approach, specifically, to the need of caring about performance aspects to prevent excessive response times. They are defined as follows:

(NF1) Integrated support for the analysis of the discovered resource-aware process models, i.e., all analysis operations should operate on the graphical and expressive resource-aware process models automatically discovered from event logs of procedural as well as flexible processes.
(NF2) Support for user interaction, i.e., the system must ensure that the end users (process analysts) get the results within a reasonable amount of time (ranging from maximum 1 to 2 seconds).

3.2 Related work

In the last years, a number of techniques for mining the organisational perspective of a process have been developed [11]. Using input data from process event logs, several methods focus on extracting the organisational model or a social network [53] behind a business process, which show the characteristics and relationships among the process participants. There is also increasing interest in analysing resource behaviour and productivity [41] as well as the influence of resources on process performance [27,38,62].

The approaches that are most closely related to our resource mining research problem are those addressing the discovery of creation patterns [46] to enrich process models with resource assignments [65]. Table 1a collects them along with the support provided with respect to the resource mining requirements. Among them, the so-called staff assignment mining approach [43] is able to extract several types of assignment rules based on decision tree learning. The identification of separation and binding of duties, among others, is not addressed. The output is an imperative process model (a Petri net, an EPC or a Heuristic net) and textual resource assignments written as Staff Assignment Rules (SAR).

The approaches classified as role mining [7,13,30] share a focus on organisational roles. Some of them address the pure identification of roles by analysing only the data in the event logs [30]. In this case, resource assignment is not an objective. Others aim at building an RBAC model [5] that includes the discovery of information about roles and permissions as well as role-based assignments associated with the activities [7]. In this case, the assignments are defined within the RBAC model and thus decoupled from the process model. An explicit link to the process model is present in the approach introduced in [13], which uses the Handover of Roles (HooR) principle to enrich a given control-flow model with roles that cluster the process activities under the assumption that each resource has exactly one role. BPMN and its swimlanes [39] are used to show the outcome. This as well as some of the aforementioned methods have been integrated into the ProM tool suite.[1]

None of the previous approaches covers the whole set of creation patterns (RM1–RM8). The DPILMiner was developed to narrow that gap [49]. It implements a three-step framework that can mine not only most of the creation patterns but also patterns that consider the control flow and the resources together. The output is a declarative process model with textual resource assignments defined with DPIL [64]. The History-based Assignment pattern (RM8) is not covered because DPIL does not support the definition of the respective resource assignments.

Business process verification has usually been addressed with model checking techniques [63], which assess the internal correctness of a process model including the satisfaction of a given formula by a model (e.g., binding of duties) [34]. Focusing on resource-aware process verification, the existing support for automatically executing the operations that constitute our model-checking requirements (MC1–MC8)

[1] http://www.promtools.org/.

Table 1 State of the art on resource-aware process discovery and verification: ✓ supported; (–) partly supported; – not supported; n/a not applicable

Approach	Resource mining requirements									
	RM1 Dir	RM2 Rb	RM3 Cb	RM4 Org	RM5 SoD	RM6 CH	RM7 BoD	RM8 Hb	RM9 Gr	RM10 Ind
(a) Support for resource mining										
[43]	✓	✓	✓	✓	–	–	–	–	–	–
[7,30]	–	✓	–	–	–	–	–	–	*n/a*	*n/a*
[13]	–	✓	–	–	–	–	–	–	✓	–
[49]	✓	✓	✓	✓	✓	✓	✓	–	–	–
RALph Miner	✓	✓	✓	✓	✓	✓	✓	–	✓	✓

Approach	Resource-aware model checking requirements							
	MC1 PP	MC2 PA	MC3 NPA	MC4 NP	MC5 SF	MC6 CC	MC7 CP	MC8 CA
(b) Support for resource-aware process verification								
[8,54,55]	✓	–	–	–	✓	✓	–	–
[17]	✓	✓	✓	✓	✓	✓	✓	✓
RALph Miner	✓	✓	✓	✓	✓	(–)	✓	✓

is limited mostly to the Potential Participants, Satisfiability and Consistency Checking operations [8,54,55] (MC1, MC5 and MC6), as depicted in Table 1b. Still, as shown in the table, the eight operations have been implemented in [17]. The implementation uses description logics [6] on a textual resource assignment language called RAL. While this shows the feasibility of description logics to automatically execute some of the operations, description logics apply the so-called *open-world assumption*, assuming that the information in the knowledge base may be incomplete and hence, the absence of a property assertion does not imply the fact being false but unknown. As we are not working with incomplete information, we have to ensure that the data in the knowledge base always be complete, which hinders the implementation.

Regarding the non-functional requirements, the disjoint sets of approaches in the two tables show the lack of an integrated solution targeting both resource mining and the subsequent analysis of the output resource-aware process models (NF1). The DPILMiner has demonstrated a performance that meets NF2 [49]. The existing implementation of the analysis operations has not been tested for performance [17].

Therefore, from the study of the state of the art we draw the following conclusions: (i) the discovery of expressive resource-aware declarative process models is covered by the DPILMiner. The definition of the resource assignments relies on DPIL, a textual declarative process modelling notation that supports several process perspectives, including the organisational perspective. Such assignments are inherently related to DPIL and cannot be used with other notations—hence, RM9 and RM10 remain uncovered; (ii) the automated verification of resource-aware process models has been addressed with description logics for process models annotated with RAL resource assignments despite the difficulties caused by the open-world assumption; and (iii) there is not an integrated approach that enables the automated verification of the expressive resource-aware process models discovered with resource mining because process discovery and verification have been separately addressed. Our goal in this work is to develop a solution that fills the existing gaps.

4 Overview of the RALph miner

To address the pursued requirements, we conceptualise and integrate different techniques to provide full support for integrated resource mining and model checking functionality (NF1) within the so-called RALph Miner. An overview of the approach is depicted in Fig. 2. The baseline is a graphical notation for resource assignment called RALph [16], which is briefly described in Sect. 4.1. On top of it is RALph's semantics, which we have defined in Linear Temporal Logic over finite traces (LTL_f) [36]. LTL_f constitutes the underlying formalism for the two main functionalities developed, namely, resource mining and resource-aware process verification. Basics of LTL_f are described in Sect. 4.2.

As shown in the figure, to develop the required functionalities, our work builds upon two separate technologies and research streams: SQL and Alloy. While any of the two techniques could have been used to address the resource mining and model checking requirements, each of them has specific strengths that make it more suitable for a specific purpose. Following this rationale, we rely on SQL and Alloy for dif-

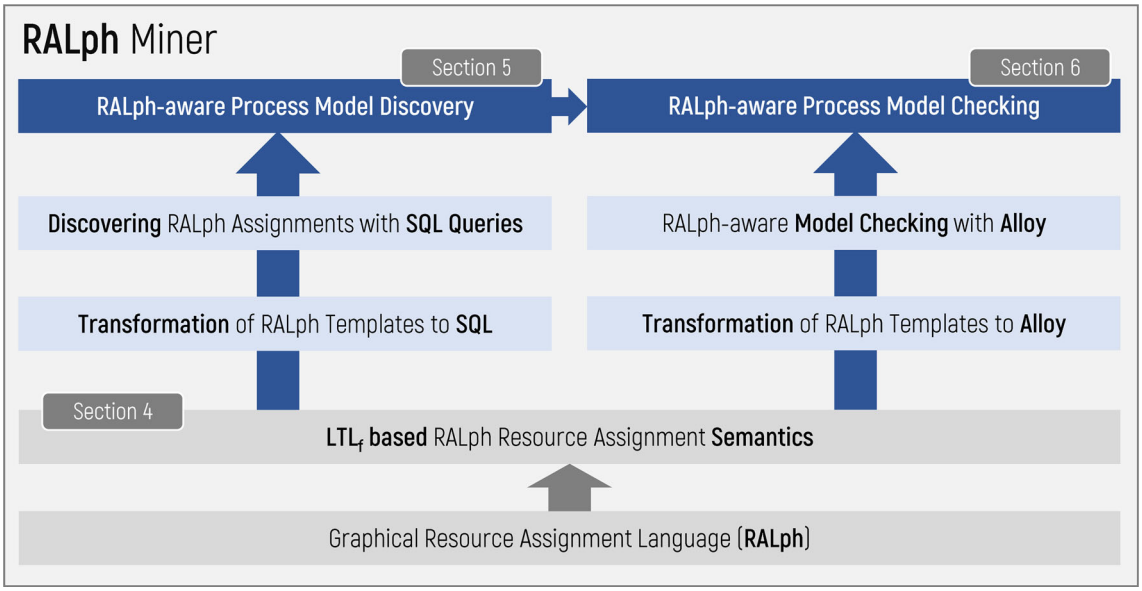


Fig. 2 Overview of the integrated approach

ferent purposes in order to best exploit their capabilities, as will be explained next. Specifically, SQL is used for mining process models (as explained in detail in Sect. 5) and Alloy for checking process models (as described in Sect. 6).

The left hand side of the figure tackles resource mining and thus provides support for requirements RM1–RM10. As aforementioned, we rely on SQL for mining expressive RALph resource assignments from event logs. Our choice is propelled by previous findings that show that the semantics of the resource assignment patterns can be expressed through SQL queries in an efficient way [51]. Furthermore, process event logs can be easily represented or transformed to relational database tables [50,51]. In contrast, encoding complete event logs in Alloy models would not be practical and browsing the extensive search space of event logs with Alloy has proven to be inadequately slow [4].

The right hand side of the figure is concerned with the model-checking approach to address the analysis operations involved in requirements MC1–MC8. Model checking requires both a possibility to represent process models with RALph assignments and a possibility to analyse model characteristics. To the best of our knowledge, an approach to encode resource-aware process models in relational databases or SQL queries has not yet been devised. On the contrary, metamodels for encoding process models in Alloy have been presented in the literature [3,4]. Here, Alloy has been used to generate traces from a given declarative process model. Therefore, we use Alloy as a basis for automatically checking RALph-aware process models. Once RALph-aware process models have been discovered, the resulting model can be checked and varied for consistency with the RALph-aware process model checking approach.

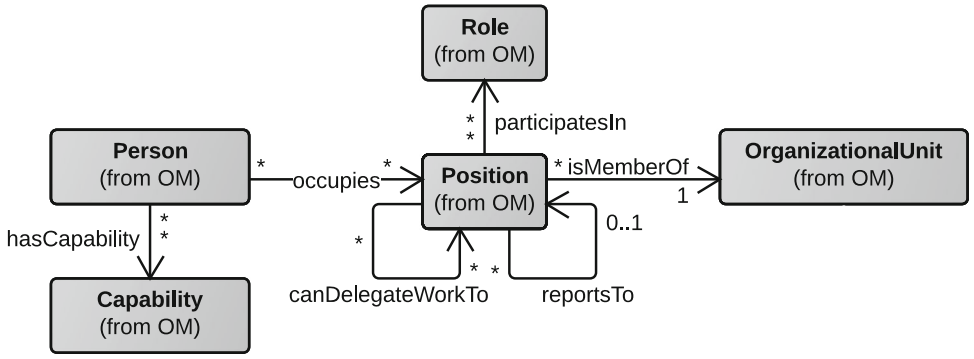
The essential concepts to understand the RALph Miner's functionalities, including RALph, multi-perspective process mining, and resource-aware process verification in terms of the model-checking requirements, are described next.

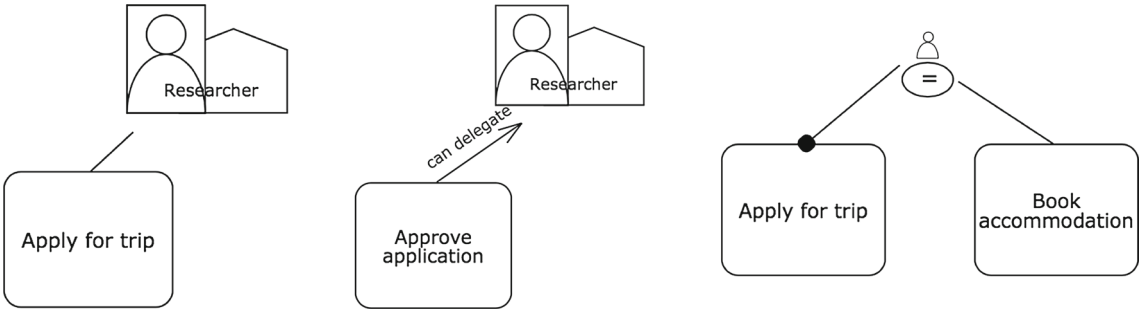
4.1 Target resource assignment language: RALph

RALph is a graphical notation for the definition of resource assignment constraints. It is expressive in terms of the creation patterns described in Sect. 3.1 (cf. [16]), and it has been designed independently of any process modelling notation so it can be combined with both imperative and declarative process models. So far it has been integrated with BPMN [16,20].

RALph assumes an organisational model representing a hierarchy of positions compatible with the metamodel depicted in Fig. 3, similar to other approaches such as [46]. This kind of organisational model includes concepts like position, role, capability and organisational unit, besides person.[2] The RALph notation consists of entities and connectors that enable the graphical modelling of resource assignments in relation to process models. Some of the creation patterns are directly supported with explicit constructs (e.g., Direct, Role-based and Capability-based Distribution) and others implicitly through constructs that aggregate several types of constraints (e.g., RALph history connectors enable the specification of binding and separation of duties within the same or different process instances).

[2] While there is not a consensus on the most common structure of organisational models [28], the organisation ontology defined by the W3C shares many concepts with RALph's metamodel (https://www.w3.org/TR/vocab-org/).





(a) Position-based **(b)** Hierarchy-based **(c)** Binding of duties

Fig. 3 Organisational metamodel used by RALph (taken from [46])

Fig. 4 Examples of RALph assignments with BPMN

Figure 4a illustrates some prominent concepts of RALph. A *Position* entity is connected to an activity to indicate that a person with the position *Researcher* has to apply for a trip. In Fig. 4b, a RALph *hierarchy connector* is used to specify that the approval of the application must be done by someone who can delegate work to researchers (i.e., a researcher's superior). Finally, Fig. 4c depicts a binding of duties between two activities, meaning that activity *Book accommodation* has to be executed by the same person who performed activity *Apply for trip* in that process instance. Note that if we had several assignment rules associated with one activity, the intersection of all of them (AND) would be used to find suitable resources. For more flexibility, RALph provides an *alternative connector* that enables the union of the resource assignment rules (OR). A more detailed description of the RALph notation can be found in [16].

4.2 Multi-perspective process mining

To do process mining, a declarative approach is typically used based on the definition of *constraint templates*. Constraint templates define parametrised classes of properties, and constraints are their concrete instantiations. Constraint templates are used for querying the provided event log to find solutions for the placeholders. A solution (also known as *constraint candidate*), is any combination of concrete values for the placeholders that yields a concrete rule that is satisfied in the event log. This approach has its roots in declarative process modelling notations, most notably in Declare [58]. For instance, a *response* constraint indicates that if activity *A occurs*, activity *B* must eventually *follow*. A template for this constraint parametrises the variable elements of the rule, in this case A and B. By replacing these placeholders with specific activities found in traces of an event log, pairs of activities that fulfil the constraint can be automatically identified. For example, the *response constraint* is satisfied in traces such as $\mathbf{t}_1 = \langle A, A, B, C \rangle$, $\mathbf{t}_2 = \langle B, B, C, D \rangle$, and $\mathbf{t}_3 = \langle A, B, C, B \rangle$, but not in $\mathbf{t}_4 = \langle A, B, A, C \rangle$. In this case, the second occurrence of A is not eventually followed by B. In $\mathbf{t}_2$, it is actually *vacuously satisfied* [31] (i.e., in a trivial way, because A never occurs).

The semantics of the constraints and the templates can be formalised using formal logics, such as LTL_f [36]. Declare has traditionally focused on the functional and behavioural perspectives. Hence, the operators that have been used include:

– The **F**, **X**, **G**, and **U** LTL_f future operators: $\mathbf{F}\psi_1$ means that ψ_1 holds sometime in the future, $\mathbf{X}\psi_1$ that ψ_1 holds in the next position, $\mathbf{G}\psi_1$ that ψ_1 holds forever in the future, and $\psi_1\mathbf{U}\psi_2$ that sometime in the future ψ_2 will hold and until that moment ψ_1 holds (with ψ_1 and ψ_2 LTL_f formulas).

– The **O**, **Y** and **S** LTL$_f$ past operators: $\mathbf{O}\psi_1$ means that ψ_1 held sometime in the past, $\mathbf{Y}\psi_1$ that ψ_1 held in the previous position, and $\psi_1\mathbf{S}\psi_2$ that ψ_1 has held sometime in the past and since that moment ψ_2 holds.

The aforementioned *response* constraint is defined with LTL$_f$ as $\mathbf{G}(A \rightarrow \mathbf{F}B)$.

An *activation activity* of a constraint in a trace is an activity whose execution imposes, because of that constraint, some obligations on the execution of other activities (*target activities*) in the same trace. For example, in the *response* constraint A is an activation activity and B is a target activity, since the execution of A forces B to be executed. An activation of a constraint leads to a *fulfilment* or to a *violation*. Consider again $\mathbf{G}(A \rightarrow \mathbf{F}B)$. In trace $\mathbf{t}_1$, the constraint is activated and fulfilled twice, whereas in trace $\mathbf{t}_3$ it is activated and fulfilled only once. Referring to the formal specification of constraints in LTL$_f$, *activation* ϕ_a is the sub-formula that lies on the left-hand side of the implication operator $\rightarrow$, whereas *target* ϕ_t is the formula that lies on its right-hand side.

The importance of multi-perspective dependencies led to the definition of a multi-perspective version of Declare (MP-Declare) [12]. This version is of interest to us since we aim at defining templates for constraints that relate to the process organisational perspective. Its semantics builds on the notion of *payload* of an event, which is the set of attributes that define it. $e(activity)$ identifies the occurrence of an event in order to distinguish it from the activity name. At the time of a certain event e, its attributes $x_1, \ldots, x_m$ have certain values. $p^e_{activity} = (val_{x1}, \ldots, val_{xn})$ represents its payload. To denote the projection of the payload $p^e_A = (x_1, \ldots, x_n)$ over attributes $x_1, \ldots, x_m$ with $m \leq n$, the shorthand notation $p^e_A[x_1, \ldots, x_m]$ is used. For instance, $p^e_{ApplyForTrip}[Resource]$=SS is the projection of the attribute *Resource* in the event description shown in Sect. 2. Furthermore, the n-ples of attributes x_i are represented as $\mathbf{x}$.

Therefore, the templates in MP-Declare extend standard Declare with additional conditions on event attributes. Specifically, given the events $e(A)$ and $e(B)$ with payloads $p^e_A = (x_1, \ldots, x_n)$ and $p^e_B = (y_1, \ldots, y_n)$, the *activation condition* φ_a, the *correlation condition* φ_c, and the *target condition* φ_t are defined. The activation condition is part of the activation ϕ_a, whilst the correlation and target conditions are part of the target ϕ_t, according to their respective time of evaluation. The *activation* condition is a statement that must be valid when the activation occurs. In the case of the *response* template, the activation condition has the form $\varphi_a(x_1, \ldots, x_n)$, meaning that the proposition φ_a over $(x_1, \ldots, x_n)$ must hold true. The *correlation* condition is a statement that must be valid when the target occurs, and it relates the values of the attributes in the payloads of the activation and the target event. It has the form $\varphi_c(x_1, \ldots, x_m, y_1, \ldots, y_m)$ with $m \leq n$, where φ_c is a propositional formula on the variables of both the payload of $e(A)$ and the payload of $e(B)$. *Target* conditions exert limitations on the values of the attributes that are registered at the moment wherein the target activity occurs. They have the form $\varphi_t(y_1, \ldots, y_m)$ with $m \leq n$, where φ_t is a propositional formula involving variables in the payload of $e(B)$.

4.3 Resource-aware process verification

The design-time analysis of the organisational perspective of business process models aims to ensure a correct utilisation of resources in the processes. Detected potential problems can be overcome by re-defining the processes and/or organisational data before the processes are under execution. Our approach assumes that it is always a single person responsible for the task execution. This is in line with classical workflow concepts. Note that collaborative mining [48] and collaborative resource assignment [18] have also been discussed in the literature. In the following, we describe the analysis operations underlying our model-checking requirements (cf. Sect. 3.1) in terms of their inputs and outputs as defined in the literature. Their applicability is also outlined:

1. *Potential participants (PP):* It takes an activity and a responsibility and returns the people who are candidates to hold that specific responsibility for the activity specified [17]. Thus, at design time, a person is a potential participant of an activity for a specific responsibility if there is *some* process instance in which they can be an actual holder of that responsibility.
   This operation serves for studying 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. It is also useful to detect activities that can be assigned to the same resources and hence, may be aggregated when creating an executable process model [24].
2. *Potential activities (PA):* It lists the activities that may be allocated to one resource with regard to a specific responsibility during a process instance execution. It takes the identity of a specific person and the responsibility to be checked, and it returns the activities that can be potentially allocated to this person for that responsibility [17]. This operation is useful to provide people with a personalised list of all the activities they may be involved in or to identify the requirements for someone who is going to substitute a certain person. It also detects the degree of involvement of a person in a process in terms of the number of activities in which they can take part.
3. *Non-potential activities (NPA):* It takes a person and a responsibility and calculates the activities in which they *cannot* hold that responsibility, if any [17].

This operation is useful when the responsibilities of a person in the organisation might be increased. The resource assignments of the activities returned by this operation might be changed to include that resource.

4. *Non-participants (NP):* It takes an activity and a responsibility and returns the people who can never participate in the activity holding that responsibility, if any [17].
   This operation is a way to quickly detect the relationship between people and processes, helping to ensure that certain resources do not have access to processes that are not aligned with their duties in the company.
5. *Satisfiability (SF):* It takes a responsibility and returns whether the process model is satisfiable with regard to that responsibility (i.e., if it is possible at all to find a potential participant for an activity during an execution of the process for that responsibility) [55,61].
   This operation is essential to know whether it is possible to complete an execution of a process. It helps to identify mistakes in the resource assignments that make it impossible to find suitable resources under any circumstances.
6. *Consistency checking (CC):* It takes a responsibility and returns whether the process model is consistent with regard to that responsibility (i.e., if it is *always* possible to find a potential participant for an activity during any execution of the process for that responsibility) [9,17,54,55].
   This operation is fundamental to ensure the correct operation of the organisational perspective as it detects situations in which the process could fall into a deadlock [56].
7. *Critical participants (CP):* One or more people are critical participants of a business 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 responsibility and returns the members of the organisation who are critical in the execution of a process for that responsibility [17]. This operation is useful for identifying those people on which a process execution may eventually depend. Furthermore, it is a mechanism to identify potential bottlenecks without the need to gather and analyse process execution logs.
8. *Critical activities (CA):* An activity is a critical activity for a given responsibility if it has only one potential holder for that responsibility. The CA operation takes a person and a responsibility and returns the critical activities in which that person is involved with the given responsibility [17].
   This operation is useful to identify 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.

This catalogue of analysis operations that have been used to frame the model-checking requirements does not intend to be a complete list but to constitute a representative set operations that can help to verify properties of a resource-aware process model related to the involvement of resources in the process activities. It covers typical notions that have been discussed, for instance, in the RBAC research community [55,61].

5 Automated discovery of RALph-aware process models

In the following, we describe our resource-mining approach in several steps. In Sect. 5.1 we define a set of templates that we need in order to discover RALph assignment rules. In Sect. 5.2 we describe the metrics we use for discovering the rules. In Sect. 5.3 we explain and exemplify the discovery mechanism, based on SQL queries. Finally, in Sect. 5.4 we delve into the order in which certain queries must be run and how to refine the output RALph-aware process models obtained from the mining.

5.1 RALph assignment templates

Resource assignment modelling languages like RALph are declarative by nature. Therefore, in order to extract RALph-aware process models from event logs, we can rely on existing principles for declarative process mining.

Consider a *Direct Assignment* constraint that reflects a constraint on activity a, demanding a, if executed, to be performed by a specific resource res. The respective template comprises placeholders of type *Activity A* as well as *Resource Res*. In Table 2 we provide all RALph constraint templates that should be discovered by our approach according to RALph's expressive power [16], which involve most of our resource mining requirements.[3] The table shows the constraint templates, the corresponding semantics in LTL_f and the related payload (i.e., the event attribute that is considered when mining for a certain assignment constraint). In case of the *Direct Assignment* template we have to query the event log for constraints of the shape $\mathbf{G}(A \rightarrow (A \wedge \varphi_t(\mathbf{x})))$, where the target condition $\varphi_t(\mathbf{x})$ is of the form $p^e_A[Resource] = val$. To discover *Role-based*, *Capability-based*, *Position-based* and *Unit-based* assignment rules, we query for the same semantics as for *Direct Assignments* but we have to consider different payloads that refer to information stemming from the organisational model (e.g., $p^e_A[Position]$ to discover position-based assignments as described in the example scenario in Sect. 2). A *Binding of Duties* template $\mathbf{G}((A \rightarrow \mathbf{G}(B \rightarrow (B \wedge \varphi_c(\mathbf{x}, \mathbf{y}))))$ reflects constraints on

[3] The Organisational Assignment pattern has been divided into Position-based Assignment, Unit-based Assignment (referred to an organisational unit) and Hierarchy-based Assignment.

Table 2 Semantics of RALph assignment rules

Template	LTL_f Semantics	Related Payload Cond.
Direct Assignment	$\mathbf{G}(A \rightarrow (A \wedge \varphi_t(\mathbf{x})))$	$p_A^e[Resource] = res$
Role-based Assignm.	$\mathbf{G}(A \rightarrow (A \wedge \varphi_t(\mathbf{x})))$	$p_A^e[Role] = r$
Pos.-based Assignm.	$\mathbf{G}(A \rightarrow (A \wedge \varphi_t(\mathbf{x})))$	$p_A^e[Position] = p$
Cap.-based Assignm.	$\mathbf{G}(A \rightarrow (A \wedge \varphi_t(\mathbf{x})))$	$p_A^e[Capability] = c$
Unit-based Assignm.	$\mathbf{G}(A \rightarrow (A \wedge \varphi_t(\mathbf{x})))$	$p_A^e[Unit] = u$
Negated Assignm.	$\mathbf{G}(A \rightarrow (A \wedge \varphi_t(\mathbf{x})))$	$p_A^e[Unit]! = u$
Case Handling	$\forall A(\mathbf{G}(A \rightarrow (A \wedge \varphi_t)(\mathbf{x})))$	$p_A^e[Resource] = res$
Binding of Duties	$\mathbf{G}((A \rightarrow \mathbf{G}(B \rightarrow (B \wedge \varphi_c(\mathbf{x}, \mathbf{y})))$	$p_A^e[Res.] = p_B^e[Res.]$
Separation of Duties	$\mathbf{G}((A \rightarrow \mathbf{G}(B \rightarrow (B \wedge \varphi_c(\mathbf{x}, \mathbf{y})))$	$p_A^e[Res.]! = p_B^e[Res.]$
Hierarchy-based Ass.	$\mathbf{G}((A \rightarrow \mathbf{G}(B \rightarrow (B \wedge \varphi_c(\mathbf{x}, \mathbf{y})))$	$p_A^e[Res.]\ reportsTo\ p_B^e[Res.](*)$

(*) Resp. *canDelegateWorkTo*

activity a and b, demanding b, if executed, to be performed by the same resource as activity a. Here, we query the event log for correlation conditions $\varphi_c(\mathbf{x}, \mathbf{y})$ on the payloads of the events that correspond to both activities a and b with the specific condition that $p_A^e[Resource] = p_B^e[Resource]$.

For subsequent automated discovery, the analyst will select from the set of predefined constraint templates the ones to be discovered depending, for instance, on the type of organisational information available (e.g., only roles, roles and positions, etcetera).

5.2 Metrics for RALph mining

Querying with constraint templates provides for every possible combination of concrete values for the placeholders in the templates the number of satisfactions in the event log. Based on the number of satisfactions, two metrics, *Support* and *Confidence*, are calculated, which express the probability of an assignment constraint to hold in the process. *Support* is the number of fulfilments of a constraint divided by the number of occurrences of the condition of a constraint. The *Confidence* metric scales the support by the fraction of traces in the log wherein the activation condition is satisfied. Constraints are considered valid if their *Support* and *Confidence* measures are above a user-defined threshold. We adopt the most recent definition by Di Ciccio et al. [23]. Here, we only consider the event-based support that is meant to be used for all constraints in which both activation and target events occur.

As defined in [23], we denote the set of *events* in a *trace* $\mathbf{t}$ of an event log L that fulfil an LTL_f formula ψ as $\models_{\mathbf{t}}^e(\psi)$. The set of all the *events* in *log* L that fulfil ψ are denoted as $\models_L^e(\psi)$. Given a resource assignment constraint $\mathcal{Z}$ comprising activation ϕ_a and target ϕ_t, we define the event-based support $\mathcal{S}_L^e$ and the event-based confidence $\mathcal{C}_L^e$ as follows:

$$\mathcal{S}_L^e = \frac{\sum_{i=1}^{|L|} \left|\models_{\mathbf{t}_i}^e (\mathcal{Z})\right|}{\left|\models_L^e (\phi_a)\right|} \tag{1}$$

$$\mathcal{C}_L^e = \frac{\mathcal{S}_L^e \times \left|\models_L^e (\phi_a)\right|}{|L|}. \tag{2}$$

5.3 Discovering RALph assignment rules with SQL

Our proposed RALph mining method builds on the SQL-based process discovery approach described in [51] because of its versatility towards customisation. With SQL queries it is possible to extract relevant process knowledge from event logs stored in a conventional relational database following the *RelationalXES (RXES)* architecture [57]. The database tables in our case include: (1) one event log table capturing the following event attributes: *EventID* (unique identifier for each recorded event), *TraceID* (unique identifier for the corresponding trace), *ActivityID* (name of the corresponding activity the event refers to), *Time* (date and time the event has occurred) as well as *Resource* (identifier of the performing resource); and (2) tables for the relationships in the organisational metamodel (cf. Fig. 3) storing the organisational information, which results in six tables: HasCapability (*Person*, *Capability*), Occupies (*Person*, *Position*), ReportsTo (*Position*, *Position*) - resp. CanDelegateWorkTo -, ParticipatesIn (*Position*, *Role*) and IsMemberOf (*Position*, *Unit*).

The mining technique discovers all constraints of a certain template under the consideration of two thresholds *minSupp* and *minConf* related to the metrics described in Sect. 5.2 by applying conventional database queries without any parsing or data conversion. As an example, we explain the SQL query to extract *Direct Assignment* constraints (RM1).

```
SELECT  'Direct Assignment', A, l1.Resource, [Support],
  [Confidence]
FROM Log l1, [ActivityCombinations] c
WHERE l1.Activity = c.A
```

```
GROUP BY c.A, c.Resource
HAVING [Support] > minSupp AND [Confidence] > minConf
```

In the FROM clause the data source tables are joined together (i.e., the table of the analysed event log where every tuple depicts a single event and, if available, the tables of the *OrganisationalModel*). Furthermore, the clause contains a subquery *ActivityCombinations* that provides a table with the activity combinations that should be checked. Every source table gets an abbreviation assigned to be referable in other clauses (e.g., "l1" for the event log table or "c" for the combination table). The WHERE clause contains the different constraint expressions that have to hold for activities and their events (i.e., the constraint activation condition as well as its fulfilment requirements). After deriving the fulfilments, the tuples are grouped by the set of parameters of the constraint template in the GROUP BY clause. After grouping, the number of tuples corresponding to a certain parameter combination can be extracted using the SQL aggregate function COUNT(*). In addition, a subquery computes the number of occurrences of the condition of the constraint. This way, the *Support* value of each constraint can be derived. The *Confidence* of each parameter combination can be calculated in a similar way. The resulting values of both queries can then be filtered by user-defined thresholds (*minSupp* and *minConf*). In the last step, the query output is selected in the SELECT clause (i.e., the parameter combination and its corresponding *Support* $\mathcal{S}_L^e$ and *Confidence* $\mathcal{C}_L^e$ values). The result set contains tuples for each parameter combination that fulfils the constraint under consideration of the given thresholds. The *Support* value is computed with the subquery below.

```
COUNT(*) /  (SELECT COUNT(*) FROM Log WHERE Activity = A)
```

Analogous, the query for the *Confidence* value is defined. It can be found in [20]. We next show the query to extract *Position-based Assignment* constraints (related to RM4). The FROM, WHERE and GROUP BY clauses of the query are as follows:

```
SELECT  'Position-based Assignment', A, l1.Unit,
  [Support], [Confidence]
FROM Log l1, Position p, [ActivityCombinations] c
WHERE l1.Activity = c.A AND a.Resource = u.Resource
GROUP BY c.A, p.Position
HAVING [Support] > minSupp AND [Confidence] > minConf
```

In this case, in addition to the event log and the activity combinations we also join the table with the resource-positions assignments according to the organisational model in the FROM clause. The query sums up all occurrences of events with respective resources and groups the occurrences with respect to the corresponding position given in the table *Occupies*.

This approach is followed to define SQL queries for all the types of resource assignments that we aim at discovering, in our case, those in Table 2. We provide SQL queries for discovering the set of resource assignment constraints online.[4]

5.4 Alternative connectors and pruning

If with certain *minSupp* and *minConf* thresholds we do not extract any resource assignment rule for a process activity, it could be the case that several resource assignment rules are associated to it with lower frequencies. Consider, for instance, that for an activity *Apply for trip* we could not extract a valid *Position-based Assignment* rule since for no rule candidate $\mathcal{S}_L^e > minSupp$ with (e.g., $minSupp = 0.95$ holds). In this case, however, it could be possible to extract a *Position-based Assignment* rule for *Researcher* with $\mathcal{S}_L^e = 0.5$ and a *Capability-based Assignment* rule for *Can speak English* with $\mathcal{S}_L^e = 0.5$, respectively. This union is modelled with the RALph alternative connector to express that one of the two conditions suffices to find suitable resources. Therefore, alternative connectors are examined at the end of the mining procedure using lower support thresholds and combining the different extracted assignment rules.

The mining method extracts *all* the assignment rules related to each activity. However, when several rules are extracted for one single activity (AND), not all of them might be strictly necessary to understand the process. Specifically, some rules may be implied by *stronger* rules because they are less restrictive and do not provide added value to the current resource assignment expression of an activity. Those rules complicate the understandability of the discovered models and hence, they are unnecessary. The work in [49] identifies two pruning approaches to eliminate unnecessary resource assignment rules: pruning based on organisational rule hierarchies (e.g., *position-based assignment* dominates *direct assignment*) and pruning based on transitive reduction (e.g., for binding of duties rules). The requirement for all pruning operations is that they do not change the meaning of the generated model. These post-processing methods can be applied to the approach at hand in a similar way in order to avoid overloading the output RALph-aware process models with unnecessary assignments that would, on the other hand, worsen their readability.

6 Model checking with RALph and alloy

In this section we describe our approach for performing model checking on the discovered RALph-aware process models. As several process perspectives must be jointly considered (namely, the functional, behavioural and organisational perspectives), we need to specify the process modelling notation used. Since Sect. 5 already introduces an

[4] https://github.com/stefanschoenig/mpdeclaremining.

LTL_f-based semantics for RALph assignment rules, it can be easily combined with MP-Declare (cf. Sect. 4.2). Therefore, we will perform model checking over *RALph-aware MP-Declare process models*.

A process execution technique for MP-Declare based on trace generation has already been investigated in [4]. Its underlying technological basis is the logic framework Alloy [29], which was explicitly designed for model checking in general. Hence, enabling RALph for model checking can be achieved by transforming RALph assignment constraints into Alloy. An Alloy- and simulation-based model checking approach for the multi-perspective, declarative process modelling language DPIL [47,64] showed encouraging results [3].

The remainder of the section is structured as follows. First, the logic framework Alloy is briefly introduced (cf. Sect. 6.1). Afterwards, the organisational metamodel on which RALph is based is reformulated with Alloy (cf. Sect. 6.2). This enables us to transform RALph's templates for assignment rules into Alloy (cf. Sect. 6.3), which then forms the basis for enacting RALph-aware MP-Declare process models (cf. Sect. 6.4) and conducting model checking (cf. Sect. 6.5).

6.1 Alloy in a nutshell

Alloy is a logic-based declarative modelling language for describing software structures by means of constraints [29]. It is complemented by an analysis engine that is based on *constraint solving* and that can be used to check whether a model is *sound*. In Alloy a model is treated as sound if it has at least one *valid instance*. An instance is an exemplary configuration of *atoms* and *relations* which fulfils all given constraints. Atoms and relations correspond to basic entities and their relationships [29, p. 35]. Atoms have three essential properties, namely, they are *(i) indivisible*, *(ii) immutable* and *(iii) uninterpreted*. Indivisible means that they cannot be divided into smaller components, and they are immutable because their properties cannot be changed. They do not have any built-in properties, which makes them uninterpreted. Relations allow for describing composite, mutable and interpreted entities. They are comparable to a table where each entry (tuple) is an atom. Hence, a relation is a set of tuples, where each tuple is a sequence of atoms.

Alloy's analysis engine is able to provide such instances—if there is any. In the scope of the paper at hand, the combination of the Alloy logic and constraint solving features is used to describe the operative semantics of RALph on the one hand side, and to use this semantics for model checking on the other hand side.

Alloy's language is based on a three-fold calculus: first-order, relational calculus and a navigation expression style. The relational calculus forms the basis and is extended by the quantifiers of the first-order calculus. Navigation expressions form sets by traversing relations between quantified variables. In many cases, a constraint can be expressed in each of the three formalisms. In [29] an example is given that describes the following constraint: An address book—described by means of a relation (*address*) from names to addresses—must not map each name to more than one address. This can be represented in each of the three logics (Listing 1).

```
// Predicate calculus
all n: Name, d, d': Address | n->d in
    address and n->d' in address implies d
    = d'

// Navigation expression style
all n: Name | lone n.address

// Relational calculus
no ~address.address - iden
```

Listing 1 Alloy's three logics: Address book example

In Listing 1 the predicate calculus style takes a name and a pair of addresses and states that if two name-address mappings that share the same name are a tuple in *address* they also have to share the same address—or, simplified: The two tuples have to be identical. In the navigation expression style one navigates over relations via quantified variables. In the given example the same constraint can be formulated by navigating from all names to their corresponding address and stating that the navigation end (each time one corresponding address) is quantified by zero or one (*lone*). Finally, in the relational calculus style one can formulate that there is no pair of addresses assigned to names with two different addresses. The examples in the later course of the current section will mostly use a mixture of the different logics for reasons of readability and in order to form the rules in a similar shape as far as possible.

Let us explain those parts of the Alloy notation that are necessary for our purposes. The syntax is described with an Alloy specification that presents a tree data structure (cf. Listing 2). Each Alloy model consists of three parts [29]: a header, a specification part and a command part. The *header* section contains information about modularisation and comprises the module's name (*module treeModule*) and imports of other modules like, for instance, a module that contains basic mathematical constants and operations (*open util/integer*).

```
// Header part
module treeModule
open util/integer

// Specification part
abstract sig Node {
    children: set Node
}

one sig Root extends Node{}

sig Leaf extends Node {
}
{
    #children = 0
```

```
}

fact {
    no n: Node | n in n.^children
    Node in Root.*children
    all n: Node | lone n.~children
}

fun countNodesOfSubtree(n: one Node) : Int {
    #(n.*children)
}

pred example{}

assert rootCheck {
    countNodesOfSubtree[Root] = #Node
}

// Command part
run example for 5 Node
check rootCheck for 5 Node
```

Listing 2 Exemplary Alloy model for tree structures

The subsequent *specification* part contains the software structure definition. More precisely, the following language elements can be used: *signatures*, *facts*, *functions*, *predicates* and *assertions*.

*Signature*s (e.g., *sig Node* in Listing 2) are used to define static structures and are comparable to classes in object-oriented programming languages. Hence, they can contain *fields* (cf. *children*). Furthermore, signatures can be *abstract* and, consequently, are defined to be extended by other signatures which then inherit all properties and constraints of the parent signature. In Listing 2 the abstract signature *Node* is extended by two other signatures: *Root* and *Leaf*. This means that (ignoring the remainder of the specification) root and leaf nodes can have children, too. Each signature can contain *signature facts*, which are constraints that usually restrict the containing signature further. In Listing 2 there is one signature fact (*#children = 0*) which prevents leaf nodes from having child nodes. Signatures can have multiplicities that restrict how often they can be instantiated (*one* means exactly once, *lone* means at most once, and *some* means once or more often).

Signature facts can be always formulated by means of regular *facts*, too. Fact blocks (in *fact*{}) are building blocks to formulate invariants (i.e., they contain conjunctions of constraints which must always be fulfilled in order to provide a valid solution). There is only one fact block in Listing 2, which contains constraints that require that (i) no node can be a child of itself, (ii) each node can be reached traversing the tree starting from the root node, and (iii) all nodes have at most one parent node.

Functions are (like in general-purpose programming languages) reusable pieces of code that can be parametrised and which return some sort of result. In Listing 2 the function *countNodesOfSubtree*, for instance, computes the size of the reflexive closure of the binary relation *children* starting from a given source node (i.e., it returns the number of nodes within a particular subtree). *Predicates* have the same characteristics as functions but always return a Boolean result. Predicates in Alloy can additionally be used as arguments (cf. *run* command in Listing 2) for the commands discussed below.

The last building block of the specification section are the *assertions*, which encode assumptions that are intended to be checked. Assertions are used as command arguments, too. The assertion in Listing 2 contains a test specification which checks whether the size of the reflexive closure of the *children* relation between *Root* and *Node* is equal to the overall number of nodes.

Functions, predicates and assertions are formulated by means of expressions. According to the explanations of the three-fold calculus, Alloy ships with the expressiveness of three logics. This includes that expressions can make use of predicate calculus operators as well as set operators. For instance, in order to compute a set of identical entities found in two different relations, it is possible to use the *intersection* set operator *&*: *X & Y*. This expression evaluates to a set of tuples that are found in both *X* and *Y*.

The *command* part of the code is usually either a *run* or a *check* command. In order to show the usage of both, Listing 2 shows two commands but running Alloy's analysis engine would only execute the *run* command because it stops searching for commands after the first finding. This command type causes the engine to search for instances that fulfil both the predicate that is used as parameter (*example*) and the model from the specification part. In contrast, the *check* command verifies the assertion it is parametrised with (*rootCheck*) and tries to find counterexamples that prove that the assertion is invalid. Both command types must be further configured in terms of the size of the solution space, which means defining a *scope* limit for all signatures that are not restricted with any multiplicity constraint yet. In Listing 2 this means that any tree that is created by executing the *run* command or that is investigated by executing the *check* command has at most five nodes.

For a more detailed and extended description of Alloy we refer to [29].

There are potential alternatives to Alloy. However, Alloy was chosen for several reasons. First, because it was developed exactly for the intended purpose, namely, model checking. Alternatives like B [2], VDM [10] and Z [1] are more focused on proof than on instance finding for a defined scope. This is an advantages since some logics (e.g., first-order) are undecidable in the general case. Additionally, model checking is usually applied iteratively and rather often in the design phase. Alloy was found exactly for this situation. It is based upon the so-called *small scope hypothesis* which states that problems with a model most often occur already in small scenarios. This conforms to the usage scenario of frequent and early testing for models in development state. Furthermore, in contrast to similar approaches, Alloy is fully executable

and supported by an analysis engine. Many of the other languages (i) restrict the particular language to an executable subset and/or (ii) are not scope-complete, which means that exhaustive search within a given scope is not supported.

One could argue that plain satisfiability solving (*SAT solving*) techniques could also be used. Though this is true, Alloy is a language built for making SAT solving issues more readable but when it comes to the analysis part it is translated into a plain SAT solving form (i.e., a Boolean formula). One can then choose among a set of the fastest and most matured SAT solver technologies. Thus, Alloy is just an abstraction layer for the low-level logic formulae allowing for much more compact representations.

Finally, note that Alloy can be used in two modes: (i) for generating positive examples (i.e., for finding instances that fulfil the given model), and (ii) for generating counter examples (i.e., for finding "instances" that violate a given hypothesis, such as an assumption or check criterion). Our approach is based on both of the two modes. Other approaches successfully transformed (declarative) process models into Alloy including transformations of organisational models and rules [3,4,52] for different purposes, such as execution of multi-perspective declarative process models and the generation of artificial event logs.

6.2 Transformation of RALph's organisational metamodel to alloy

Both an Alloy-based metamodel for process execution traces[5] (cf. definition of traces in Sect. 4.2) and a transformation approach for MP-Declare rule templates that are commonly used in literature have already been described in [4]. In order to transform RALph's resource assignment constraints to Alloy, the organisational metamodel (cf. Fig. 3) has to be translated to Alloy beforehand.

```
module orgMM_RALph
open commons

abstract sig Role {}

abstract sig Position {
    participatesIn: set Role,
    isMemberOf: one OrganisationalUnit,
    canDelegateWorkTo: set Position,
    reportsTo: lone Position
}{
    reportsTo != this
}

abstract sig Person extends AssociatedElement {
    occupies: set Position,
    hasCapability: set Capability
}

abstract sig Capability {}
abstract sig OrganisationalUnit {}
```

Listing 3 Organisational metamodel for RALph in Alloy

The transformed organisational metamodel (cf. Listing 3) is encapsulated in a module (*orgMM_RALph*) that makes it reusable. That module, in turn, depends on a module called *commons*, which consists of a signature called *AssociatedElement*, which is a base signature for all the information that should be associated with an event of a process execution trace. Here it provides the means to associate instances of the signature *Person* (which means concrete resources) with those events. This can be done since the signature for events (*HumanTaskEvents* [4]) contains a field (*assoEl*) of type *AssociatedElement*.

Through two fields (*occupies* and *hasCapability*) the relations from *Person* to *Position* and *Capability* are respectively declared. *Position* is further restricted by means of a signature fact that prevents a position from reporting to itself. The remainder of the organisational metamodel from Fig. 3 is transformed analogously. All signatures are abstract. In Alloy, abstract signatures are means to build a classification hierarchy. For instance, a signature extending *Person* extends *AssociatedElement*, too, and thus, can be associated with events of the process execution trace. The subsequent paragraph explains how process execution traces can be defined in Alloy [4].

```
module traceMM
open commons

abstract sig PEvent { pos: disj Int }
abstract sig TaskEvent extends PEvent{
    assoEl: some AssociatedElement
}{ #(Task & assoEl) = 1 }
sig HumanTaskEvent extends TaskEvent {}{
    #(Person & assoEl) = 1
}

abstract sig Task extends AssociatedElement{}

fact{
    one te: TaskEvent | te.pos = integer/min
    all te: TaskEvent | te.pos = integer/min or
        sub[te.pos,1] in TaskEvent.pos
}
```

Listing 4 Process execution trace metamodel in Alloy

Listing 4 shows a metamodel for process execution traces encoded as a reusable module in Alloy (*traceMM*). *PEvent* represents any event that might occur within the execution of a process. *TaskEvent* extends this basic concept by declaring a field for general information that may be associated with an event (*assoEl*). However, the attached constraint requires that the associated information contain an activity (a.k.a. *Task*) that adds the semantics of an activity execution to this type of events. Consequently, *HumanTaskEvent* extends a *TaskEvent* further by requiring the presence of a resource (a.k.a. *Person*) that is considered to be the performer of the associated activity. Since the metamodel for process execution traces (*traceMM*) also depends on the *commons* module,

[5] Since the proposed model checking approach is based on trace generation, it is necessary to define a trace metamodel in Alloy, too. This way, Alloy's satisfiability solving capabilities can be used to generate traces that produce the desired results of the analysis operations shown in Listing 7.

the imported signature *AssociatedElement* can be used as a type of the field *assoEl*. This way, it is possible to declare instances of organisational metamodel signatures (Listing 3) as parts of a *TaskEvent*'s payload, too.

The *fact* block adds two constraints that form an event chain (i.e., it determines the positioning of each event within the process execution trace). Without this block, instances of *traceMM* were unordered sets of events rather than an ordered sequence.

Based on the two metamodels from Listings 3 and 4 the next section shows how to map RALph's assignment templates to Alloy.

6.3 Transformation of RALph's resource assignment templates to alloy

Based on the metamodel transformation discussed in the previous section it is now possible to transform RALph's resource assignment templates (cf. Table 2) into Alloy, too.

As described in Sect. 6.1, Alloy provides two alternatives to declare reusable code fragments: predicates and functions. Since predicates are parametrisable and (unlike functions) always have Boolean results, they are suitable for representing templates. Parametrisation is important since all of the resource assignment templates can be parametrised, too. The Boolean result type of Alloy predicates reflects the binary property of instances of resource assignment templates that state that they can be either fulfilled or not.

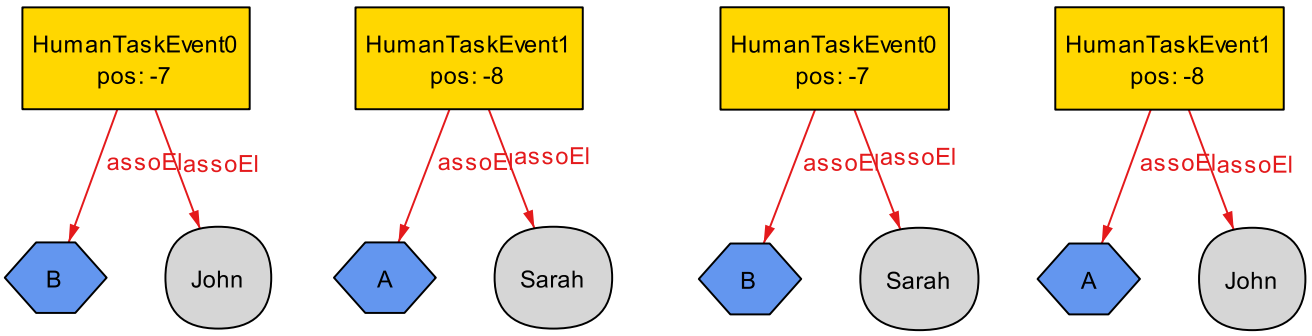
Each template from Table 2 forms one Alloy predicate as shown in Listing 5. Therein, the predicates have the same ordering and naming as the templates in Table 2.

Before describing the predicates it has to be mentioned that there is a slight deviation in the payload definitions between Table 2 and the Alloy implementation given in Listing 3. In Table 2 all elements of the organisational metamodel given in Fig. 3 can be part of the payload of an event. In order to keep the Alloy code concise and readable, we decided to only allow for the person/resource element to be part of the event payload. Since it is possible to reach all other payload information by means of the transitive closure, it does not limit the expressiveness of the chosen implementation.

```
pred directAssignment(t: Task, p: Person) {
  all e: HumanTaskEvent | #(t & e.assoEl) > 0
      implies #(p & e.assoEl) > 0
}

pred roleBasedAssignment(t: Task, r: Role) {
  all e: HumanTaskEvent | #(t & e.assoEl) > 0
      implies #((e.assoEl & Person).occupies.
      participatesIn & r) > 0
}

pred posBasedAssignment(t: Task, p: Position) {
  all e: HumanTaskEvent | #(t & e.assoEl) > 0
      implies #((e.assoEl & Person).occupies &
      p) > 0
}

pred capBasedAssignment(t: Task, c: Capability)
    {
  all e: HumanTaskEvent | #(t & e.assoEl) > 0
      implies #((e.assoEl & Person).
      hasCapability & c) > 0
}

pred unitBasedAssignment(t: Task, u:
    OrganisationalUnit) {
  all e: HumanTaskEvent | #(t & e.assoEl) > 0
      implies #((e.assoEl & Person).occupies.
      isMemberOf & u) > 0
}

pred negUnitBasedAssignment(t: Task, u:
    OrganisationalUnit) {
  all e: HumanTaskEvent | #(t & e.assoEl) > 0
      implies #((e.assoEl & Person).occupies.
      isMemberOf & u) = 0
}

pred bindingOfDuties(t1, t2: Task) {
  all e,f: HumanTaskEvent | #(t1 & e.assoEl) >
      0 and #(t2 & f.assoEl) > 0 implies #(e.
      assoEl & f.assoEl & Person) > 0
}

pred separationOfDuties(t1, t2: Task) {
  all e,f HumanTaskEvent | #(t1 & e.assoEl) > 0
       and #(t2 & f.assoEl) > 0 implies #(e.
      assoEl & f.assoEl & Person) = 0
}

pred hierarchyBasedAssignmentDelegate(t: Task,
    p: Position) {
  all e: HumanTaskEvent | #(e.assoEl & t) > 0
      implies #((e.assoEl & Person).occupies.
      canDelegateWorkTo & p) > 0
}

pred hierarchyBasedAssignmentReport(t: Task, p:
    Position) {
  all e: HumanTaskEvent | #(e.assoEl & t) > 0
      implies #((e.assoEl & Person).occupies.
      reportsTo & p) > 0
}

pred caseHandling(p: Person) {
  all e: HumanTaskEvent | (e.assoEl & Person) =
      p
}
```

Listing 5 RALph's assignment rules in Alloy

All predicates restrict the events that occur in a trace, where an event is already described by the trace metamodel from Listing 4. The current concept of the metamodel only supports events that describe the execution of activities by human resources. However, because of the payload abstraction via the common signature *AssociatedElement* it is possible to extend the metamodel with a custom module for, e.g., non-human resources. The subsequent implications (*implies*) always restrict rule applications to events that describe the execution of the task(s) provided as rule parameter (left part of the implication). All rule templates then use the set operator *&* (a.k.a. *intersection*) to check different properties of the *Person*s that are assigned to those events. For instance, the rule template *unitBasedAssignment* uses the term $\#((e.assoEl \,\&\, Person).occupies.isMemberOf \,\&\, u) > 0$ to restrict the performer of the task t to a *Person* that *occupies* a position which belongs to the provided *OrganisationalUnit* u.



(a) Trace #1 **(b)** Trace #2

Fig. 5 Two exemplary process execution traces

6.4 Alloy-based representation of process models and execution traces

Based on the representation of RALph's resource assignment templates in Alloy (Sect. 6.3) it is possible to represent RALph-aware MP-Declare models in Alloy, too. This is necessary in order to use Alloy for checking those models with respect to the analysis operations defined in Sect. 3.1.

```
// (a) RALph-aware MP-Declare model
one sig A extends Task{}
one sig B extends Task{}

one sig John extends Person{}
one sig Sarah extends Person{}

fact {
    separationOfDuties[A,B]
}

// (b) Given process execution trace
fact {
  #(atPos[A,add[integer/min,0]]
  #(atPos[B,add[integer/min,1]]
}
```

Listing 6 Representing process execution traces in Alloy

Listing 6 shows the Alloy representation of (a) a process model and (b) a (maybe partial) trace of its execution. The process model consists of the two activities *A* and *B* as well as the two persons *John* and *Sarah*. In the first fact block, RALph's resource assignment template *separation of duties* (Table 2) is instantiated and configured with the two activities *A* and *B*. This means that if the two activities are executed within the same process instance, the performers have to be different.

Additionally, the approach can be configured with an empty, partial or complete process execution trace. This allows running all analysis operations (cf. Sect. 3.1) to be evaluated either for an initial or a particular execution state of the process.

More concretely, this means it enables the user to specify a sequence of fixed events forming a trace which describes the progress of a process instance. This process instance can be completed but does not have to. If it is not completed, the trace is referred to as partial.

The given trace in Listing 6 comprises two events and is encoded in an Alloy *fact* block. The first event is restricted in a way that it has to cover information that says that some activity *A* has been performed first. In contrast, the second event describes the execution of some activity *B*. No resource assignments are specified within the trace. A potential use case could be to check the effect of the *separation of duties* for the two given activities. Alloy is able to create examples that are valid regarding the given model. An exemplary[6] result is shown in Fig. 5.

Figure 5 shows two process execution traces that are visualised by means of the graphical representation capabilities of Alloy's analysis engine. Both of the traces show two *HumanTaskEvent*s, respectively, where one describes the execution of activity *A* and the second describes an execution of activity *B*. The only difference is that in Fig. 5a *Sarah* executed activity *A* while John was responsible for activity *B* whereas in Fig. 5b it is the other way round. The Alloy analysis engine exhaustively searches for examples that fulfil the given model and, for the example given in Listing 6, the results given in Fig. 5 are complete. This shows the effect of the *separation of duties* since there is no example where both activities *A* and *B* are executed by the same resource.

Though the approach discussed in [4] does not use RALph, the included principle already describes how MP-Declare models as well as process execution traces can be encoded in Alloy. Although the given examples only show resource assignments, both MP-Declare as well as the trace metamodel allow for specifying more aspects of an event's payload. However, they are not discussed in the paper at hand for the sake of conciseness. Instead, the authors refer to

[6] This example assumes a trace length of two. Since the current paper focuses on model checking rather than example generation the details for the latter are skipped.

examples in [4] that show the involvement of, for instance, the data perspective.

6.5 RALph-aware MP-declare process model checking with alloy

In general, Alloy provides flexibility regarding the model properties that are desired to be checked. The remainder of this section discusses how properties of RALph-aware MP-Declare process models can be checked. For the current paper the model checking approach comprises the design-time execution of the analysis operations that constitute our process verification requirements.

In order to enable Alloy to check RALph-aware MP-Declare process models the desired analysis operations have to be encoded in Alloy, too. Hence, the model checking approach is based upon two main building blocks—(i) a set of Alloy predicates that represent analysis operations and (ii) an Alloy command configuration. Alloy predicates are reusable pieces of code that become active either by using them in a *fact* block or in combination with an Alloy command (cf. Sect. 6.1). The predicates for analysis operations formulate (in general) a contradiction to what the particular analysis operation should compute. Using the Alloy *check* command these contradictory statements are refuted by showing *counter examples* that contain the results of the particular analysis operations. For instance, for determining *Potential Participants* for a given activity, the predicate states that the activity is never executed by any resource. By applying Alloy's check command this statement is refuted by showing exemplary executions (a.k.a. *HumanTaskEvents*) where a valid resource participates. Since Alloy provides a scope-complete analysis,[7] those examples in sum show all potential participants.

The necessary building blocks (i.e. the predicates and the configuration of the check command) are explained in detail in the remainder of this section.

Alloy predicates for the analysis operations. The analysis operations are generally represented as predicates, which makes them (i) parametrisable and (ii) usable with the Alloy commands. Listing 7 shows an Alloy representation for several analysis operations described in Sect. 4.3.

```
// Potential participants (Requirement MC1)
pred PP(t: Task) {
 all e: HumanTaskEvent | #(e.assoEl & t)>0
     implies #(Person & e.assoEl)=0
}

// Potential activities (Requirement MC2)
pred PA(p: Person) {
 all e: HumanTaskEvent | #(e.assoEl & Task)>0
     implies #(p & e.assoEl)=0
}

// Non-potential activities (Requirement MC3):
    Not encoded as a predicate
// Non-Potential participants (Requirement MC4)
    : Not encoded as a predicate

// Critical Participants (Requirement MC7)
pred CP() {
 all p: Person | p in HumanTaskEvent.assoEl
}

// Critical activities (Requirement MC8)
pred CAhelper(t: Task, p: Person) {
 all e: HumanTaskEvent | #(e.assoEl & t) > 0
     implies #(p & e.assoEl) = 1
}

pred CA(p: Person) {
 no t: (HumanTaskEvent.assoEl & Task) |
     CAhelper[t,p]
}
```

Listing 7 RALph analysis operations in Alloy

The predicate for retrieving *Potential Participants (PP)* formulates a statement that each execution of a given task *t* occurs in absence of any resource (a.k.a. *Person*). *Potential activities (PA)* are identified similarly through a statement that each time a given resource *p* is involved in any execution this execution is not associated with any activity. The operations *Non-participants (NP)* for an activity *t* and *Non-potential Activities (NPA)* for a resource *p* do not have to be encoded in Alloy, since *NP* is the relative complement of all resources and all potential participants for *t* (*NP* = *Person* - *PP[t]*) and *NPA* is the relative complement of all activities and the potential activities for *p* (*NPA* = *Task* - *PA[p]*).

Consistency Checking (CC)(MC6) cannot be solved with the proposed Alloy-based approach. The reason is that Alloy is based on instance search (i.e. it solves satisfiability issues by computing examples—if existent—that prove satisfiability). However, for *CC* computing examples of satisfiable cases does not answer the question whether the process model is *always* consistent. One way to achieve this is to search for the opposite, which means searching for *inconsistent* examples. Consequently, the *CC* analysis operation would have to search for examples that *violate* the given specification. Inconsistent here means that it has to compute examples where a resource assignment is not possible. However, due to the constraint that each *HumanTaskEvent* has exactly one associated resource it is not possible to generate those examples. Nevertheless, a relaxed interpretation of this analysis operation can be the general satisfiability of the process model (*MC5*). This is the most native usage scenario of Alloy and can be solved by applying Alloy's run command instead of the check command. If this application leads to one or more examples the model is proven to be satisfiable since it is possible to identify one or more examples that fulfil the model.

[7] Scope-complete here means that Alloy is able to compute all examples that refute the discussed statement that are within the defined maximum trace length.

Critical Participants (CP) are computed by a statement that all resources are always involved in the process. The result of applying the check command would contain all resources that do not necessarily have to be involved in the process and, thus, the result of the analysis operation is the relative complement of all resources and those that are contained in the results of applying the check command for *CP*. Finally, *Critical Activities (CA)* was split into two predicates for readability reasons. The predicate *CAhelper* forms a statement that each time a given activity *t* is executed, a given resource *p* must participate in this execution. The second predicate—*CA*—forms a statement that for a given person *p* there is no task execution for which *CAhelper* is true. Applying the check command refutes the statement contained in *CA* providing counter examples containing activities that can be executed by a resource other than the given resource *p*. Consequently, the critical activities are all activities not contained in these counter examples (i.e., the relative complement of all activities and those contained in the counter example).

Besides these analysis operations, Alloy is able to support additional operations, too. Three examples are given in Listing 8.[8]

```
// Is p in PP (pPP)
pred pPP(p: Person, t: Task) {
 no e: HumanTaskEvent | #(e.assoEl & t)>0 and
     #(p & e.assoEl)=1
}

// Is a specific role/position/capability
    required for the execution of the process?
pred RPC(rpc: (Position + Role + Capability)) {
 rpc in ((HumanTaskEvent.assoEl & Person).
     occupies + (HumanTaskEvent.assoEl & Person
     ).hasCapability + (HumanTaskEvent.assoEl &
      Person).occupies.participatesIn)
}

// Which roles are not involved in the process?
pred NP {
 all po: Position | not RPC[po]
}
```

Listing 8 Additional analysis operations in Alloy

The predicate *pPP* can be used to ascertain whether a given resource *p* is one of the potential participants of a given activity *t*. Therefore, it formulates a statement that no execution of *t* involves *p*. The second exemplary predicate (*RPC*) determines whether a given role, position or capability *rpc* is required for the execution of the process. It contains a statement that checks whether *rpc* is involved in any execution of any activity. If the analysis result provides counter examples this means that *rpc* is not required in order to execute the process. Finally, the third example (*NP*) computes all positions that are not involved in the process.[9] For that reason it reuses the predicate *RPC* and evaluates this predicate for all positions in the model. Hence, applying the check command retrieves all positions that can be involved in the process and, consequently, the final result is the relative complement of all positions and those contained in the analysis result.

Command configuration. All questions mentioned above can be answered via Alloy's language and analysis features. However, each answer is only valid for a given *scope* which describes the size of the solution space. To be more concrete, a general answer for the questions above is not possible. This is because first-order logic is in general undecidable. For that reason, Alloy requires to limit the size of the solution space by means of a scope restriction. Hence, the remainder of this section describes what that scope represents for the current approach and how it is used to configure Alloy's check command.

In Alloy, all signatures that do not have an explicit multiplicity must be restricted regarding the number of times an instance of each of them may occur in a solution produced by Alloy's analysis engine. The example given in Listing 9 shows three signatures. The abstract signature *Person* is already known from the organisational metamodel given in Listing 3. In Alloy, an abstract signature is used for inheritance of fields and signature facts which means that they are usually extended by other signatures. However, if no signature extends this abstract signature, Alloy instantiates the abstract signature instead. Thus, in a model without any involved resources (a.k.a. *Person*s), it is necessary to restrict the scope for *Person* explicitly. In Listing 9, *Max* and *Mary* are two signatures that extend the abstract signature *Person*. Consequently, a scope restriction for *Person* would be superfluous. However, in contrast to *Mary*, *Max* is a signature without an explicitly defined multiplicity and hence, it has to be restricted via the scope, too. *Mary* is a signature with a multiplicity of exactly one, so it does not require any scope restriction. As a result, the same exemplary model *without* the signature *Max* does not require any scope restriction for the signature *Person* since it is already restricted by *Mary*. In short, scope restrictions are necessary for those signatures without any explicit multiplicity.

```
abstract sig Person extends AssociatedElement{...
    }
sig Max extends Person{}
one sig Mary extends Person{}
```

Listing 9 RALph analysis operations in Alloy

Considering the metamodel given in Listing 4, a scope restriction is always necessary for the signature *TaskEvent*. This scope restriction represents the maximum process execution trace length and, consequently, the number of events that the trace may consist of. Additionally, it is necessary to restrict the scope of all signatures in the RALph's metamodel provided in Listing 3 if no other signature extends them having an explicit multiplicity. However, if those signatures are extended the corresponding extending signatures

[8] A formal definition of these analysis operations is omitted since they are designed for illustrating our approach's extensibility.

[9] The statements for roles and capabilities are analogous.

should have the multiplicity one, since capabilities, person names, etcetera, can be assumed to be clearly distinguishable.

In order to give an example (cf. Listing 10 for its Alloy representation), we assume a RALph-aware MP-Declare model consisting of two activities *A* and *B*, two resources *John* (who occupies position *CEO*) and *Sarah* (who occupies position *CIO*) and two assignment rules:

- The two activities have to be executed by different resources (*separationOfDuties[A,B]*).
- The resource that is responsible for *A* must occupy the position *CEO* (*posBasedAssignment[A,CEO]*).

A possible analysis operation that might be executed for the given model could be the retrieval of potential participants for activity *A*. This can be achieved by creating an assertion which states that the predicate *PP* (cf. Listing 7) holds for the given argument *A*. Applying the check command to this assertion causes Alloy's analysis engine to create counter examples that falsify this assertion if possible—for the given scope.

```
one sig A extends Task{}
one sig B extends Task{}

one sig John extends Person{}{occupies = CEO }
one sig Sarah extends Person{}{occupies = CIO }

one sig CEO extends Position{}
one sig CIO extends Position{}

fact {
  separationOfDuties[A,B]
  posBasedAssignment[A,CEO]
}

assert opPP {
  PP[A]
}
check opPP for 2 TaskEvent, 0 Role, 0
    Capability, 1 OrganisationalUnit
```

Listing 10 Exemplary scope restriction

Alloy's check command can be configured regarding the number of times an instance of a particular signature may occur in potential results of executing the analysis operations. This is what is already mentioned as *scope restriction*. Listing 10 shows an exemplary scope restriction to two for the metamodel signature *TaskEvent* (cf. Listing 4). This means that, independently from any other constraint or restriction, all analysis operations are performed for process execution traces consisting of two events at most.

Listing 10 contains extensions for the metamodel signatures *Person* and *Position* (cf. Listing 3). For the remaining signatures of RALph's metamodel a scope restriction is necessary. Thus, the call of Alloy's *check* command is accompanied by scope restrictions for all those signatures. Since the exemplary RALph-aware MP-Declare model does not mention anything about roles or capabilities, we can set the expected number of instances of those signatures to zero. In contrast, *OrganisationalUnit* must be set to at least one since RALph's metamodel (Listing 3) requires that a *Position* is member of exactly one *OrganisationalUnit*. Though this necessary background knowledge seems to be an uncomfortable limitation of the proposed approach, we rather suggest an automation procedure as future work. This is valid since this internal knowledge about the metamodel is static and therefore, can be reflected by likewise static rules.

Figure 6 shows a partial result for the analysis operation for retrieving potential participants for activity *A*. It is a partial result because it contains only one potential participant while the full set of potential participants can be retrieved by iterating over all examples. The visualisation in Fig. 6 shows a process execution trace consisting of two events. However, the analysis operation asks for specific *event details*, namely, the performers of activity *A*. This specific detail is marked with *$PP_e*; the performer in the marked event is one potential participant for activity *A*. Since Alloy searches *exhaustively* for examples (counter examples in this case) with respect to the given scope, the resulting set of potential participants is complete with respect to the given scope, too. This dependency to the scope and further implications are discussed in the subsequent paragraph.

Implications of the necessity of a scope restriction. There are two major implications that result from the necessity to artificially configure the size of the solution space in terms of a scope restriction: (i) scope-boundedness of results and (ii) necessity of background knowledge.

Scope-boundedness of results means that the result of each analysis operation is only valid for the given scope restriction and, consequently, it is not possible to answer any analysis question in general. This requires the follow-up conclusion that the results of the analysis operations might differ depending on the configured scope. If, for instance, the scope for *TaskEvent* is too low, running the analysis operation *PP* for activity *A* might produce an empty set—just because the *TaskEvent* scope does not allow an event where *A* is executed (e.g., because of control-flow restrictions) and not because there is no resource that could execute this activity. However, always configuring a "huge" scope significantly lowers the performance [3,4] and thus, leads to an issue from a pragmatic point of view. Hence, an *efficient* scope—denoting a scope that is both able to produce valuable analysis results while being still performant—heavily depends on the respective model and the analysis question.

A second implication—the *necessity of background knowledge*—results from the scope-boundedness. More concretely, the approach currently requires background knowledge about the given process model in terms of the required size of analysis results. This is necessary in order to choose an efficient scope restriction. For the current state of the proposed approach this is a manual task. Thus, the current state of

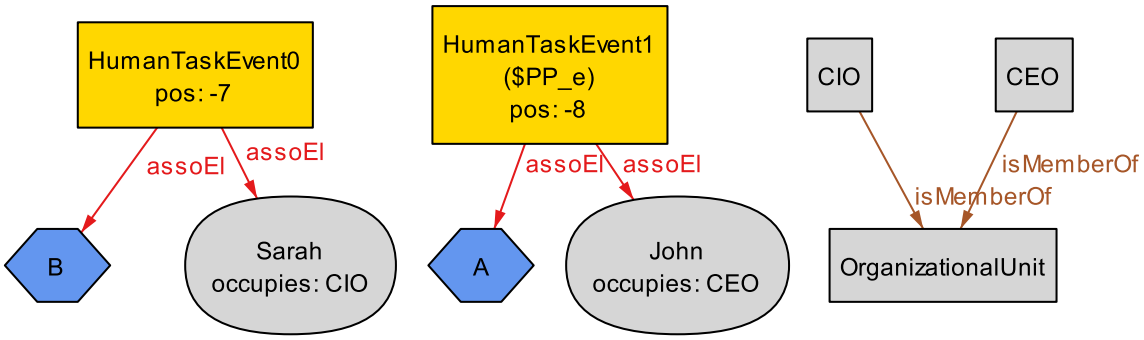


Fig. 6 Example result after invoking the check command from Listing 10

the approach assumes that sufficient background knowledge about the process model is available. However, since the proposed approach focuses on design-time checking the target user group are modelling experts that are in charge of developing the process model. Consequently, the assumption of background knowledge is consistent with the intended application environment.

7 Implementation and evaluation

In this section, we provide details and examples of the RALph Miner implementation and evaluate its components. In particular, we show the feasibility of the resource mining and resource-aware process verification approaches (cf. Sect. 7.1) as well as the performance of the latter (cf. Sect. 7.2). The performance of the SQL-based resource mining approach with RALph was already assessed in [20]. We also briefly report how the system requirements defined in Sect. 3.1 are supported (cf. Sect. 7.3).

7.1 RALph miner

The RALph mining approach has been implemented as a web-based process mining tool. The implemented architecture and used toolset are illustrated in Fig. 7. We aim at discovering RALph-aware process models and hence, two main elements must be discovered, namely, the definition of the process itself (i.e., the functional and behavioural perspectives) and the resource assignment rules for the process activities (i.e., the organisational perspective). As mentioned in Sect. 1, there is a number of approaches for discovering a business process. Implementations for many of them are available as plug-ins in the ProM framework.[10] We use the *BPMN Miner* tool with a XES event log to extract a resource-unaware BPMN model. Afterwards, the resulting BPMN model is exported as an XML file according to the *BPMN-XML* specification [39]. We use the SQL mining approach described in Sect. 5.3 for extracting RALph assignment rules. Since this approach builds on the relational RXES event log representation, we first have to import the XES event log to relational database tables in RXES format as well as make the organisational information available as tables as explained in Sect. 5.3. We can then run the set of SQL queries required to extract RALph resource assignment rules. The resulting assignment rules are attached in the previous BPMN-XML file to the respective activity as specific resource tags. Here, we match activities from the given BPMN model and the extracted assignment rules based on activity identifiers given in the event log. The RALph-aware BPMN model is then visualised in the graphical BPMN diagram editor *bpmn.io*,[11] which has been extended with the RALph symbols. For automatically arranging and layouting the RALph assignment symbols in the process diagram, we used a Java Script based implementation of the Sugiyama graph layout algorithm [25]. Additionally, the underlying formal RAL expressions can be imported and edited in BPMN editors like *Signavio*.[12] A plug-in is available to automatically analyse such RAL assignments so that the RAL-aware process model can be automatically executed [19].

As a proof of concept, we applied the described toolset to an event log of a university business trip management system. The log contains 2104 events of eight different activities related to the application and the approval of university business trips as well as the management of accommodations and transfers (e.g., booking accommodations and transport tickets). The system has been used for six months by eleven employees of a research institute. The organisational model of the institute comprises two organisational units: Administration, with two employees; and Research Group, divided into three positions that include one professor, six researchers and two secretaries as depicted in Fig. 1. On the given event log, we were able to execute all RALph resource assignment queries (cf. Table 2) in less than one second. The resulting BPMN model with the extracted RALph assignment rules

[10] http://www.promtools.org/doku.php.

[11] BPMN Viewer and Editor, https://bpmn.io.

[12] https://www.signavio.com.

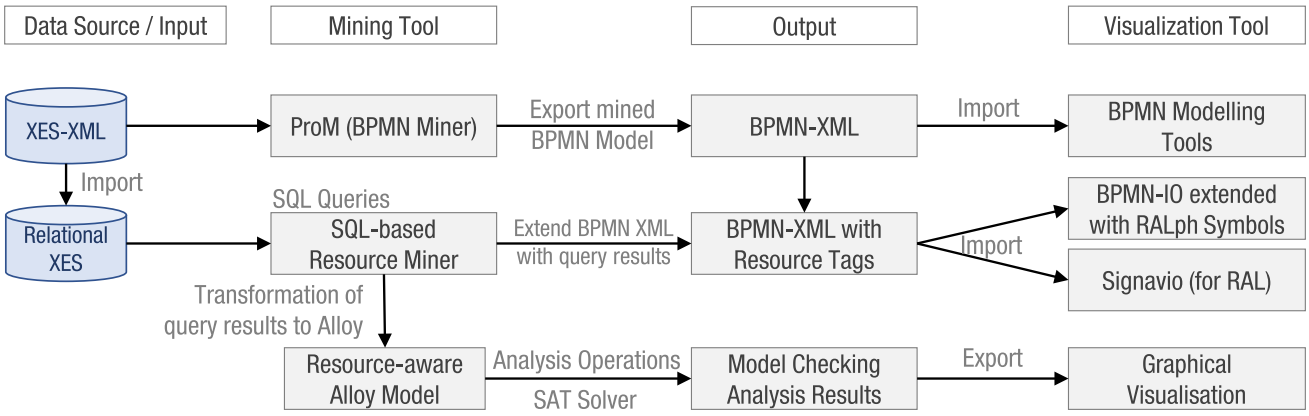


Fig. 7 Implementation architecture and mining procedure

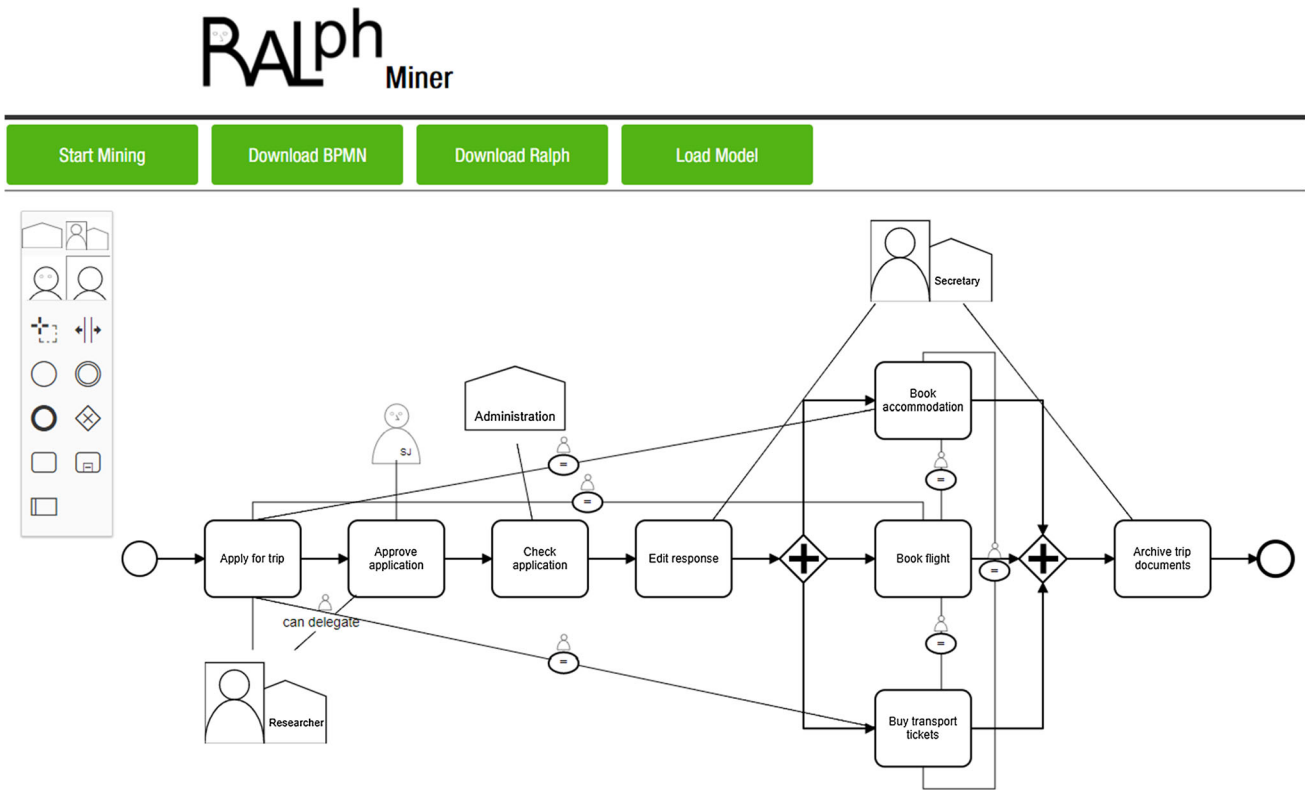


Fig. 8 User interface of RALphMiner with extracted assignment rules

is shown in Fig. 8. The screenshot also shows the extended *bpmn.io* modelling toolbox on the left-hand side. Note that the model has not been pruned as described in Sect. 5.4 since the implementation of that post-processing feature is still pending work. Therefore, the model contains some assignment rules that are irrelevant (e.g., the direct assignment of entity *SJ* to *Approve Application* or the binding of duties rule between *Book accommodation* and *Buy transport tickets*).

As a second part of our proof of concept the discovered process model (Fig. 8) was translated into Alloy.[13] Since RALph is independent from a particular modelling language for the definition of the control flow, the subsequent explanations focus on the resource perspective and skip the behavioural aspect.

[13] The full example is available online: https://github.com/stefanschoenig/mpdeclaremining.

```
one sig Admin1 extends Person{}{ occupies =
    AdminPosition  }
one sig Admin2 extends Person{}{ occupies =
    AdminPosition }
one sig Secretary1 extends Person{}{ occupies =
     Secretary }
one sig Secretary2 extends Person{}{ occupies =
     Secretary }
one sig Researcher1 extends Person{}{ occupies
    = Researcher }
one sig Researcher2 extends Person{}{ occupies
    = Researcher }
one sig Researcher3 extends Person{}{ occupies
    = Researcher }
one sig Researcher4 extends Person{}{ occupies
    = Researcher }
one sig Researcher5 extends Person{}{ occupies
    = Researcher }
one sig Researcher6 extends Person{}{ occupies
    = Researcher }
one sig SJ extends Person{}{ occupies =
    Professor }

one sig Researcher extends Position{}{
    #participatesIn = 0
    isMemberOf = ResearchGroup
    #canDelegateWorkTo = 0
    #reportsTo = 0
}
one sig Secretary extends Position{}{
    #participatesIn = 0
    isMemberOf = ResearchGroup
    #canDelegateWorkTo = 0
    #reportsTo = 0
}
one sig Professor extends Position{}{
    #participatesIn = 0
    isMemberOf = ResearchGroup
    canDelegateWorkTo = Researcher
    #reportsTo = 0
}

one sig AdminPosition extends Position{}{
    #participatesIn = 0
    isMemberOf = Administration
    #canDelegateWorkTo = 0
    #reportsTo = 0
}

one sig Administration extends
    OrganisationalUnit{}
one sig ResearchGroup extends
    OrganisationalUnit{}
```

Listing 11 Exemplary organisational model in Alloy

Listing 11 shows the representation of the organisational model. The first block of signatures represents the eleven employees that are distributed over three explicitly mentioned positions. An additional position was introduced (*AdminPosition*) since RALph's metamodel given in Fig. 3 prescribes that resources are assigned to organisational units via their positions (cf. relation *isMemberOf* in *Position*). The last two signatures describe the two organisational units.

```
fact {
    // RALph resource assignment rules
    bindingOfDuties[ApplyForTrip, BookFlight]
    bindingOfDuties[ApplyForTrip,
        BookAccommodation]
    bindingOfDuties[ApplyForTrip,
        BuyTransportTickets]
    bindingOfDuties[BookAccommodation,
        BookFlight]
    bindingOfDuties[BookAccommodation,
        BuyTransportTickets]
    bindingOfDuties[BookFlight,
        BuyTransportTickets]
    posBasedAssignment[ApplyForTrip, Researcher
        ]
    posBasedAssignment[EditResponse, Secretary]
    posBasedAssignment[ArchiveTripDocuments,
        Secretary]
    directAssignment[ApproveApplication, SJ]
    hierarchyBasedAssignmentDelegate[
        ApproveApplication, Researcher]
    unitBasedAssignment[CheckApplication,
        Administration]

    assert testPP {
        PP[BookFlight]
    }

    check testPP for 8 TaskEvent, 0 Role, 0
        Capability, 2 OrganisationalUnit, 4
        Position
}
```

Listing 12 Exemplary assignment rules Alloy

Listing 12 shows the Alloy representation of the assignment rules depicted in Fig. 8. The potential participants of an activity can be retrieved by means of the *PP* predicate. In the example it is applied to the activity *BookFlight*. The corresponding *check* command is configured for the maximum number of events that may occur within one process instance—which is equal to the number of activities since each activity has to be executed exactly once. The remaining scope parameters are depending on the organisational model which contains no roles and no capabilities but two organisational units and four positions.

Alloy's analysis engine interprets the check command and generates examples of potential participants. Collecting all of them shows that this comprises all resources occupying the position *Researcher*. This is because a rule *bindingOfDuties* was applied for the two tasks *ApplyForTrip* and *BookFlight*. However, if we encode a partial trace with a single event that associates *Researcher2* with *ApplyForTrip*, the retrieved potential participants for *BookFlight* are reduced to *Researcher2* which conforms to the sequence of *bindingOfDuties* constraints.

7.2 Performance of RALph-aware MP-declare model checking

In this section, we describe the evaluation of our resource-aware model checking approach regarding performance measurements. In particular, we measure the time that Alloy's analysis engine needs to determine *one single* solution for a given maximum number of events n. Within a RALph model checking setting, this represents the time Alloy needs for calculating one specific execution instance of the given RALph-aware MP-Declare process model of trace length n.

For measuring the execution time of the approach, we use a RALph-aware MP-Declare process model that consists of the following entities: (i) three activities *A, B* and *C*; (ii) three MP-Declare constraints covering the control-flow per-

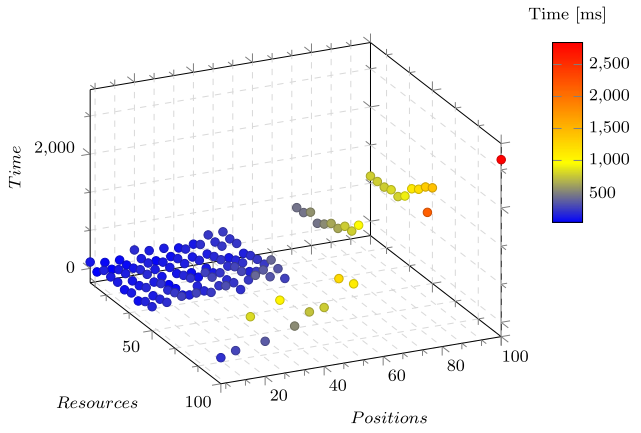


Fig. 9 Performance evaluation of RALph-aware model checking

spective; and (iii) *BindingOfDuties(A,B)* and *SeparationofDuties(B,C)* representing RALph assignment constraints.

The organisational model consists of two different roles, capabilities and organisational units each. For the given model we measured the runtime performance for a maximum trace length of $n = 20$. Already existing performance measurements showed the trace length's strong influence on the computation time [3,4]. Since the current paper focuses on resource assignment, we keep the trace length fixed and pick the constant arbitrarily. Consequently, the experimental setup varies the number of positions and resources in the range from 0 to 100. The measurements visualised in Fig. 9 have been performed on a Dell Latitude E6430 (Core i7-3720QM, 8 x 2.6GHz, 16 GB RAM, SSD drive and Win 8 64 Bit).

Our experimental setup applies the analysis operation for retrieving potential participants for activity *B*. This means that the collected performance data represents the computation times for one potential participant for the corresponding variation of the number of resources and positions. Consequently, the computation time for retrieving all potential participants can be calculated by multiplying a particular computation time from Fig. 9 with the number of resources at its worst. We say "at most" since the actual computation time will usually be smaller because only a subset of all resources will be potential participants of *B*.

The results (i.e. the time measured for retrieving one potential participant) show that the calculation of one single solution of the given RALph-aware MP-Declare process model is performed in less than 1 second in most of the cases. The figure additionally shows that the execution time is mainly influenced by the number of positions and not by the number of resources. This is caused by the fact that Alloy needs to check all variations of possible resource-to-position mappings, which results in a potentially huge number of possibilities. The calculation run time scales linearly (i.e., calculating two possible solutions will double the given runtime values).

7.3 Discussion

Table 1 includes the support provided by our integrated solution, the RALph Miner, for the functional and non-functional requirements that frame this work. Almost all the creation patterns can be mined and represented with the output resource assignment notation, RALph. The only exception is History-based Assignment (RM8). History-based assignments can be defined with RALph constructs. However, our mining approach currently does not consider data from past process instances to infer resource assignment constraints. RM9 and RM10 are supported thanks to RALph's design.

The resource-aware process verification component of the RALph Miner provides automated support for the execution of seven of the eight analysis operations behind the model-checking requirements. MC6 (Consistency Checking) cannot be implemented in a comprehensive and smart way. A brute-force solution could be defined but this requires to implement the brute force strategy outside of Alloy. The reason for that is the necessity to encode all potential execution states of the process by means of a trace in Alloy. However, since the current metamodel is limited to a representation of the execution state of exactly one process instance it is necessary to generate *a set* instead of one Alloy model, one for each potential execution state.

As depicted in Fig. 2, the RALph Miner constitutes an integrated solution for resource mining and resource-aware process verification, satisfying non-functional requirement NF1. Note that three different modelling languages are used: BPMN as a graphical process modelling language, MP-Declare with its strong logical foundation, and RALph, which is the connection point of the proposed tools and thus, serves as an integration base. RALph is independent from any particular process modelling language but in order to make it usable with existing process modelling languages and tools, it is integrated with BPMN for the process mining task and with MP-Declare for model verification. Both BPMN and MP-Declare here serve as *host languages* for enabling the control flow perspective of processes, too. This shows RALph's flexibility on the one hand side and adapts existing approaches on the other hand side, allowing us to concentrate on the integration based on RALph's logical foundation instead of reinventing process mining and verification techniques. Furthermore, since RALph's formal semantics is defined independently from any process modelling language but is implemented in both BPMN and MP-Declare in an additive way, it is straightforward to map a RALph assignment rule discovered by an BPMN-based process mining technique to MP-Declare.

Lastly, the performance evaluations of the resource mining and the resource-aware process verification components (cf. [20] and Sect. 7.2, respectively) conclude that NF2 is also supported.

8 Conclusions and future work

In this work we have focused on the organisational perspective of business processes and we have developed the RALph Miner, an integrated solution that allows for the automated discovery of expressive graphical resource-aware process models and their automated verification. We have used the RALph notation for defining the resource assignments in the discovered resource-aware process models. RALph's semantics has been defined using LTL_f as a common semantic foundation to give support to the resource mining and model-checking approaches. We have shown how the RALph Miner provides full support for eighteen of the twenty system requirements identified from studies of the literature involved with resource management in different domains.

Nevertheless, our work also has some limitations that we aim to tackle in future efforts. Regarding the resource mining approach, the implementation of the pruning step after the discovery of the RALph-aware process models is the next task to be performed. As for resource-aware process model checking, the transformation of RALph templates to Alloy code needs to be done manually at the moment. In addition to the described usability issues, we will also try to further improve the performance of the introduced model-checking approach with a smarter configuration of the Alloy's analysis engine. As far as the scope is concerned, the two requirements not supported or partly supported at the moment will be further addressed. Finally, since use cases might differ in terms of the resource specialisation, we plan to investigate the impact that that may have on the RALph Miner by testing it on more use cases.

References

1. Abrial, J.R.: Specification language. On the Construction of Programs (1980)
2. Abrial, J.R., Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (2005)
3. Ackermann, L., Schönig, S., Jablonski, S.: Simulation of multi-perspective declarative process models. In: Business Process Management Workshops, pp. 61–73. Springer, Berlin (2016)
4. Ackermann, L., Schönig, S., Petter, S., Schützenmeier, N., Jablonski, S.: Execution of multi-perspective declarative process models, pp. 154–172 (2018). https://doi.org/10.1007/978-3-030-02671-4_9
5. American National Standards Institute, I.: Role-based access control. ANSI INCITS 359-2004 (2004). http://csrc.nist.gov/rbac
6. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logics Handbook: Theory, Implementations, and Applications. Cambridge University Press, Cambridge (2003)
7. Baumgrass, A.: Deriving current state RBAC models from event logs. In: Int. Conf. on Availability, Reliability and Security, pp. 667–672 (2011)
8. Bertino, E., Ferrari, E.: Data security. In: COMPSAC, pp. 228–239 (1998)
9. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Trans. Inf. Syst. Secur. **2**, 65–104 (1999)
10. Bjorner, D., Jones, CB, et al.: The Vienna development method: the meta-language (1978)
11. Bose, J.C., Maggi, F.M., van der Aalst, W.: Enhancing declare maps based on event correlations. BPM **8094**, 97–112 (2013)
12. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
13. Burattin, A., Sperduti, A., Veluscek, M.: Business models enhancement through discovery of roles. In: IEEE CIDM, pp. 103–110 (2013)
14. Caballero, H.S.G., Westenberg, M.A., Verbeek, H.M.W., van der Aalst, W.M.P.: Visual analytics for soundness verification of process models. Bus. Process Manag. Workshops **308**, 744–756 (2017). https://doi.org/10.1007/978-3-319-74030-0_59
15. Cabanillas, C.: Process- and resource-aware information systems. In: Int. Conf. on Enterprise Distributed Object Computing (EDOC), pp. 1–10 (2016)
16. Cabanillas, C., Knuplesch, D., Resinas, M., Reichert, M., Mendling, J., Ruiz-Cortés, A.: RALph: A graphical notation for resource assignments in business processes. In: CAiSE, vol. 9097, pp. 53–68. Springer, Berlin (2015)
17. Cabanillas, C., Resinas, M., del Río-Ortega, A., Ruiz-Cortés, A.: Specification and automated design-time analysis of the business process human resource perspective. Inf. Syst. **52**, 55–82 (2015)
18. Cabanillas, C., Resinas, M., Ruiz-Cortés, A.: A template-based approach for responsibility management in executable business processes. Enterp. Inf. Syst. **12**(5), 550–586 (2018). https://doi.org/10.1080/17517575.2017.1390166
19. Cabanillas, C., del Río-Ortega, A., Resinas, M., Cortés, A.R.: CRISTAL: collection of resource-centrIc supporting tools and languages. In: BPM (Demos), pp. 51–56. CEUR-WS.org (2012)
20. Cabanillas, C., Schönig, S., Sturm, C., Mendling, J.: Mining expressive and executable resource-aware imperative process models. In: Int. Conf. on Enterprise, Business-Process and Information Systems Modeling (BPMDS), vol. 318, pp. 3–18 (2018)
21. Ciccio, C.D., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. **64**, 425–446 (2017). https://doi.org/10.1016/j.is.2016.09.005
22. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Berlin (2000)
23. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for Artful processes. ACM Trans. Manag. Inf. Syst. **5**(4), 241–2437 (2015)
24. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 2nd edn. Springer, Berlin (2018)

25. Eiglsperger, M., Siebenhaller, M., Kaufmann, M.: An efficient implementation of Sugiyama's algorithm for layered graph drawing. In: Graph Drawing, pp. 155–166 (2005)
26. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. Electron. Proc. Theor. Comput. Sci. **69**, 59–73 (2011)
27. Hompes, B.F.A., Maaradji, A., Rosa, M.L., Dumas, M., Buijs, J.C.A.M., van der Aalst, W.M.P.: Discovering causal factors explaining business process performance variation. In: CAiSE, pp. 177–192 (2017)
28. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. Knowl. Eng. Rev. **19**(4), 281–316 (2004). https://doi.org/10.1017/S0269888905000317
29. Jackson, D.: Software Abstractions: Logic, Language, and analysis. MIT Press, London (2012)
30. Jin, T., Wang, J., Wen, L.: Organizational modeling from event logs. In: Int. Conf. on Grid and Cooperative Computing (GCC), pp. 670–675 (2007)
31. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. Int. J. Softw. Tools Technol. Transf. **4**(2), 224–233 (2003)
32. Larkin, J.H., Simon, H.A.: Why a diagram is (sometimes) worth ten thousand words. Cogn. Sci. **11**(1), 65–100 (1987). https://doi.org/10.1016/S0364-0213(87)80026-5
33. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of LTL-based declarative process models. In: Int. Conf. on Runtime Verification (RV)—Revised Selected Papers, vol. 7186, pp. 131–146 (2011). https://doi.org/10.1007/978-3-642-29860-8_11
34. Mendling, J.: Empirical studies in process model verification. Trans. Petri Nets Other Models Concurr. **2**, 208–224 (2009). https://doi.org/10.1007/978-3-642-00899-3_12
35. Mendling, J., Neumann, G., Nüttgens, M.: Yet another event-driven process chain—modeling workflow patterns with yEPCs. Enterp. Model. Inf. Syst. Archit. (EMISA) **1**, 3–13 (2005)
36. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographies. TWEB **4**(1), 3:1–3:62 (2010)
37. Mora, B., Garcia, F., Ruiz, F., Piattini, M.: Graphical versus textual software measurement modelling: an empirical study. Softw. Qual. J. **19**, 201–233 (2011). https://doi.org/10.1007/s11219-010-9111-x
38. Nakatumba, J., van der Aalst, W.: Analyzing resource behavior using process mining. In: Business Process Management Workshops, pp. 69–80 (2010)
39. OMG: BPMN 2.0. Recommendation, OMG (2011)
40. Peffers, K., et al.: A design science research methodology for information systems research. J. Manag. Inf. Syst. **24**(3), 45–77 (2007)
41. Pika, A., Leyer, M., Wynn, M.T., Fidge, C.J., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Mining resource profiles from event logs. ACM Trans. Manag. Inf. Syst. **8**(1), 1:1–1:30 (2017)
42. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. Trans. Petri Nets Other Mod. Concurr. **2**, 115–135 (2009). https://doi.org/10.1007/978-3-642-00899-3_7
43. Rinderle-Ma, S., van der Aalst, W.M.: Life-cycle support for staff assignment rules in process-aware information systems. Technical Report TU/e (2007)
44. Roth, W.M., Bowen, G.M.: When are graphs worth ten thousand words? An expert–expert study. Cogn. Instr. **21**(4), 429–473 (2003). https://doi.org/10.1207/s1532690xci2104_3
45. Rovani, M., Maggi, F.M., de Leoni, M., van der Aalst, W.M.: Declarative process mining in healthcare. Expert Syst. Appl. **42**(23), 9236–9251 (2015)
46. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: identification, representation and tool support. In: CAiSE, pp. 216–232 (2005)
47. Schönig, S., Ackermann, L., Jablonski, S.: DPIL navigator 2.0: multi-perspective declarative process execution. In: Online Proceedings of the BPM Demo Track. CEUR-WS.org (2017)
48. Schönig, S., Cabanillas, C., Ciccio, C.D., Jablonski, S., Mendling, J.: Mining team compositions for collaborative work in business processes. J. Softw. Syst. Model. (SoSyM) **1**, 19 (2015). https://doi.org/10.1007/s10270-016-0567-4
49. Schönig, S., Cabanillas, C., Jablonski, S., Mendling, J.: A framework for efficiently mining the organisational perspective of business processes. Decis. Support Syst. **89**, 87–97 (2016)
50. Schönig, S., Ciccio, C.D., Maggi, F.M., Mendling, J.: Discovery of multi-perspective declarative process models. In: Service-Oriented Computing—14th International Conference, ICSOC 2016, Banff, AB, Canada, October 10–13, 2016, Proceedings, pp. 87–103 (2016)
51. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and customisable declarative process mining with SQL. In: CAiSE, pp. 290–305 (2016)
52. Skydanienko, V., Di Francescomarino, C., Ghidini, C., Maggi, F.M.: A tool for generating event logs from multi-perspective declare models. In: BPM (Dissertation/Demos/Industry), pp. 111–115 (2018)
53. Song, M., van der Aalst, W.M.: Towards comprehensive support for organizational mining. Decis. Support Syst. **46**(1), 300–317 (2008)
54. Strembeck, M., Mendling, J.: Modeling process-related RBAC models with extended UML activity models. Inf. Softw. Technol. **53**, 456–483 (2011)
55. Tan, K., Crampton, J., Gunter, C.A.: The consistency of task-based authorization constraints in workflow systems. In: IEEE Workshop on Computer Security Foundations, pp. 155–169 (2004)
56. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, London (2004)
57. van Dongen, B.F., Shabani, S.: Relational XES: data management for process mining. CAiSE Forum **2015**, 169–176 (2015)
58. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: balancing between flexibility and support. Comput. Sci. R&D **23**(2), 99–113 (2009)
59. van der Aalst, W.M.P.: Process Mining—Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
60. Verbeek, E., Buijs, J., van Dongen, B., van der Aalst, W.: XES, xESame, and ProM 6. In: Information Systems Evolution, pp. 60–75 (2011)
61. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems
62. Wynn, M.T., Poppe, E., Xu, J., ter Hofstede, A.H.M., Brown, R., Pini, A., van der Aalst, W.M.P.: ProcessProfiler3D: a visualisation framework for log-based process performance comparison. Decis. Support Syst. **100**, 93–108 (2017)
63. Wynn, M.T., Verbeek, H.M.W., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Business process verification—finally a reality!. Bus. Proc. Manag. J. **15**(1), 74–92 (2009). https://doi.org/10.1108/14637150910931479
64. Zeising, M., Schönig, S., Jablonski, S.: Towards a common platform for the support of routine and agile business processes. In: IEEE Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing, pp. 94–103 (2014)
65. Zhao, W., Zhao, X.: Process mining from the organizational perspective. Adv. Intell. Syst. Comput **277**, 701–708 (2014)

Dr. Cristina Cabanillas is a post-doctoral researcher with the ISA Research Group at the University of Seville (Spain). She has taken part in many R&D projects, has experience as a reviewer in top international conferences and journals, and has chaired a number of workshops and conference tracks. She is currently coordinating the CONFLEX project on the integration of context-aware resource management into flexible process-oriented organisations. Her main research interests relate to business process management with a special focus on their organisational perspective.

Dr. Lars Ackermann is an Assistant Professor of Computer Science with the Institute for Computer Science at University of Bayreuth (Germany). He received the master's degree (with honours) in Computer Science and the doctoral degree from University of Bayreuth. He has an established background in BPM/Process Mining and has been working in this field for several years. He published extensively in the research area of business process management and information systems, both in international conferences and journals.

Dr. Stefan Schönig is a Professor for Information Systems with the Institute of Management Information Systems at the University of Regensburg in Germany. He received both the master's degree (with honours) in Applied Computer Science (Engineering/Computer Science) and the doctoral degree from University of Bayreuth. Before, he held a position as a tenured assistant professor at University of Bayreuth. He was a post-doctoral researcher with the Institute for Information Business at WU Vienna (Vienna University of Economics and Business). He has an established background in BPM/Process Mining and IoT research and has been working in this field for over 9 years. He published extensively in the research area of BPM and information systems, both in international conferences and journals.

Christian Sturm is a researcher and lecture assistant with the Institute for Computer Science at University of Bayreuth (Germany). He graduated as a B.Sc. and as an M.Sc. in Applied Computer Science at University of Bayreuth in Germany. His research is focused on the implementation and execution of cross-organisational business processes on blockchain technology. He has participated in several projects that addressed process mining and process execution. Based on his work, he has published scientific papers in international conferences and journals, among them Future Generation Computer Systems. He participated in several conferences and workshops as organiser or in program committees.

Prof. Jan Mendling is a Full Professor with the Institute for Information Business at Wirtschaftsuniversität Wien (WU Vienna), Austria. His research interests include business process management and information systems. He is co-author of the textbooks Fundamentals of Business Process Management and Wirtschaftsinformatik. He has published more than 400 research papers and articles, among others in ACM Transactions on Software Engineering and Methodology, IEEE Transaction on Software Engineering, Information Systems, Data and Knowledge Engineering, and Decision Support Systems. He is member of several international journals, member of the board of the Austrian Society for Process Management, a co-founder of the Berlin BPM Community of Practice, organiser of several academic events on process management, and member of the IEEE Task Force on Process Mining.