# Specifying, Monitoring, and Executing Workflows in Linked Data Environments

Tobias Käfer[1(✉)] and Andreas Harth[2]

[1] Institute AIFB, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
`tobias.kaefer@kit.edu`
[2] University of Erlangen-Nuremberg (FAU), Nuremberg, Germany
`andreas.harth@fau.de`

**Abstract.** We present an ontology for representing workflows over components with Read-Write Linked Data interfaces and give an operational semantics to the ontology via a rule language. Workflow languages have been successfully applied for modelling behaviour in enterprise information systems, in which the data is often managed in a relational database. Linked Data interfaces have been widely deployed on the web to support data integration in very diverse domains, increasingly also in scenarios involving the Internet of Things, in which application behaviour is often specified using imperative programming languages. With our work we aim to combine workflow languages, which allow for the high-level specification of application behaviour by non-expert users, with Linked Data, which allows for decentralised data publication and integrated data access. We show that our ontology is expressive enough to cover the basic workflow patterns and demonstrate the applicability of our approach with a prototype system that observes pilots carrying out tasks in a virtual reality aircraft cockpit. On a synthetic benchmark from the building automation domain, the runtime scales linearly with the size of the number of Internet of Things devices.

## 1 Introduction

Information systems are increasingly distributed. Consider the growing deployment of sensors and actuators, the modularisation of monolithic software into microservices, and the movement to decentralise data from company-owned data silos into user-owned data pods. The drivers of increasing distribution include:

- Cheaper, smaller, and more energy-efficient networked hardware makes widespread deployment feasible[1].
- Rapidly changing business environments require flexible re-use of components in new business offerings [17].

---

[1] http://www.forbes.com/sites/oreillymedia/2015/06/07/how-the-new-hardware-movement-is-even-bigger-than-the-iot/.

– Fast development cycles require independent evolution of components [17].
– Privacy-aware users demand to retain ownership of their data[2].

The distribution into components raises the opportunity to create new integrated applications out of the components, given sufficient interoperability. One way to make components interoperable is to equip the components with uniform interfaces using technologies around Linked Data. Consider, e.g. the W3C's Web of Things[3] initiative and the MIT's Solid ("social linked data") project[4], where REST provides an uniform interface to access and manipulate the state of components, and RDF provides an uniform data model for representing component state that allows for using reasoning to resolve schema heterogeneity. While the paradigms for (read-only) data integration systems based on Linked Data are relatively agreed upon [11], techniques for the creation of applications that integrate components with Read-Write Linked Data interfaces are an active area of research [2,4,15,28]. Workflows are a way to create applications, according to Jablonski and Bussler [14], that is highly suitable for integration scenarios, easy to understand (for validation and specification by humans), and formal (for execution and verification by machines). E.g., consider an evacuation support workflow for a smart building (cf. task 4 in our evaluation, Sect. 6), which integrates multiple systems of the building, should be validated by the building management and the fire brigade, verified to be deadlock-free, and executable. Hence, we tackle the research question: *How to specify, monitor, and execute applications given as workflows in the environment of Read-Write Linked Data?*

The playing field for applications in the context of Read-Write Linked Data is big and diverse: As of today, the Linking Open Data cloud diagram[5] lists 1'163 data sets from various domains for read access. The Linked Data Platform (LDP)[6] specifies interaction with Read-Write Linked Data sources. Besides Solid for social networks, a showcase for Read-Write Linked Data is the Web of Things, which is built on sensors and actuators on the Internet of Things. Using such sensors and actuators, we can build applications such as integrated Cyber-Physical Systems, where sensors and actuators provide the interface to Virtual Reality systems (cf. the showcase in the evaluation, Sect. 6.2). Other non-RDF REST APIs provide access to weather reports[7] or building management systems (e.g. Project Haystack[8]) and can be wrapped to support RDF. Using such APIs, we can build applications such as integrated building automation systems (cf. the scenario of the synthetic benchmark in the evaluation, Sect. 6.3).

Traditional environments for workflows are fundamentally different from Read-Write Linked Data. Elmroth et al. argue that the properties of the environment determine the model of computation, which serves as the basis of a

---

workflow language [5]. Consequently, we have developed ASM4LD [15], a model of computation for the environment of Read-Write Linked Data. In this paper, we investigate an approach for a workflow language consisting of an ontology and operational semantics in ASM4LD. The differences between traditional environments of workflow languages and the environment of Read-Write Linked Data (i.e. RDF and REST) pose the following particular challenges:

**Querying and reasoning under the open-world assumption.** Ontology languages around RDF such as RDFS and OWL make the open-world assumption (OWA). However, approaches from workflow management operate on relational databases, which make the closed-world assumption (CWA). Closedness allows for testing if something holds for *all* parts of a workflow.

**The absence of events in REST.** HTTP implements CRUD (the operations create, read, update, delete), but not the subscriptions to events. However, approaches from workflow management use events as change notifications.

While both challenges could be mitigated by introducing assumptions (e.g. negation-as-failure once we reach a certain completeness class [12]) or by extending the technologies (e.g. implement events using Web Sockets[9] or Linked Data Notifications [2]), those mitigation strategies would restrict the generality of the approach, i.e. we would have to exclude components that provide Linked Data, but do not share the assumptions or extensions of the mitigation strategy.

Previous works from Business Process Management, Semantic Web Services, Linked Data, and REST operate on a different model of computation or are complementary: [10,13,19] assume event-based data processing, decision making based on process variables, and data residing in databases under the CWA, whereas our approach relies on integrated state information from the web under the OWA. [28,29] provide descriptions for automated composition or for assisting developers. Currently, we do not see elaborate and correct descriptions available at web scale, which hinders automated composition. We see our approach, which allows for manual composition, as the first step towards automated composition.

The paper is structured as follows: In Sect. 2, we discuss related work. In Sect. 3, we present the technologies on which we build our approach. Next, we present our approach, which consists of two main contributions:

– An ontology to specify workflows models and workflow instances modelled in OWL LD[10] (Sect. 4) that allows for monitoring and execution using querying and reasoning under the OWA. The ontology is strongly related to BPMN, a graphical workflow notation, via the workflow patterns [25].
– An operational semantics for our workflow ontology. We use ASM4LD, a model of computation for Read-Write Linked Data in the form of a condition-action rule language (Sect. 5), which does not require event data and is directly executable. We maintain workflow state in an LDP container.

---

[9] http://www.ietf.org/rfc/rfc6455.txt.
[10] http://semanticweb.org/OWLLD/.

Fast data processing thanks to OWL LD and the executability of ASM4LD allow for the direct application of our approach in practice. In the evaluation (Sect. 6), we present a Virtual Reality showcase, and a benchmark in an Internet of Things setting, specifically in the building automation domain. Moreover, we show correctness and completeness of our approach. We conclude in Sect. 7.

## 2   Related Work

We now survey related work grouped by field of research.

**Workflow Management.** Previous work in the context of workflow languages and workflow management systems is based on event-condition-action (ECA) rules, whereas our approach is built for REST, and thus works without events. ECA rules have been used to give operational semantics to workflow languages [13], and to implement workflow management systems [3]. Similar to the case handling paradigm [26], we employ state machines to track the state of activities in a workflow instance.

**Web Services.** WS-*-based approaches assume arbitrary operations, whereas our approach works with REST resources, where the set of operations is constrained [20,30]. Pautasso et al. proposed extensions to BPEL such that e.g. a BPEL process can invoke REST services [18], and that REST resources representing processes push events [19]. While those extensions make isolated REST calls fit the Web Services processing model of process variable assignments, we propose a processing model based on integrated polled state.

**Semantic Web Services.** OWL-S and WSMO are mainly concerned with service descriptions and corresponding reasoning for composition. Semantic Web Services build on WS-* technology for workflow execution, e.g. the execution in the context of WSMO, WSMX [10], is entirely event-based. In contrast, our work is based on REST.

**Scientific Workflows.** Approaches like Taverna [24] and Wings [8] focus on representing the data flow between processing steps. Our approach applies control flow techniques from Workflow Management to REST.

**Ontologies for Workflows.** Similar to workflows in our ontology, processes in OWL-S are also tree-structured (see Sect. 4) and use lists in RDF. Unlike OWL-S, our ontology also covers workflow instances. Rospocher et al. [22] and the project "Super" developed ontologies that describe process metamodels such as BPMN, BPEL, and EPC. In contrast to our work, their ontologies either require more expressive (OWL) reasoning or do not allow for execution under the OWA.

## 3   Preliminaries

We next introduce the environment, Read-Write Linked Data, and the model of computation, ASM4LD [15].

**Read-Write Linked Data.** Linked Data is a collection of practices for data publishing on the web that advocates the use of web standards: HTTP URIs[11] should be used for identifying things. HTTP GET[12] requests to those URIs should be answered using descriptive data, e.g. in RDF[13]. Hyperlinks in the data should enable the discovery of more information[14]. Read-Write Linked Data[15] introduces RESTful write access to Linked Data (later standardised in the LDP specification (see Footnote 6)). Hence, we can access the world's state using multiple HTTP GET requests and enact change using HTTP PUT, POST, DELETE requests.

In the paper, we denote RDF triples using binary predicates[16], e.g. we write for the triple in Turtle notation "`<#wfm> rdf:type :WorkflowModel`.":

$$rdf\!:\!type(\texttt{<\#wfm>}, \texttt{:WorkflowModel})$$

We abbreviate a class assignment using a unary predicate with the class as predicate name, e.g. $:WorkflowModel(\texttt{<\#wfm>})$. The term $rdf\!:\!List(\dots)$ is a shortcut, similar to the RDF list shortcut with `()` brackets in Turtle, and can be regarded as a procedure that (1) takes as argument list elements, (2) adds the corresponding RDF list triples, i.e. with terms $rdf\!:\!first$, $rdf\!:\!rest$, and `rdf:nil`, to the current data, and (3) returns the blank node for the RDF list's head.

**ASM4LD, A Condition-Action Rule Language.** We use a monotonic production rule language to specify both reasoning on RDF data and interaction with Read-Write Linked Data resources [23]. Rule programs in the language consist of initial assertions and rules. The body of all rules is a basic graph pattern query (see Footnote 18) (BGP). We distinguish two types of rules: (1) a derivation rule specifies productions using a BGP in the rule head, and (2) a request rule specifies an interaction using an HTTP request description in the rule head. We assume safe rules and exclude existential variables in rule heads.

As operational semantics for the rule language, we use ASM4LD, an Abstract State Machine-based [9] model of computation for Read-Write Linked Data [15]. In the following, we sketch the operational semantics, where data processing is done in repeated steps, subdivided into the following phases (cf. [15] for details):

(1) The working memory be empty.
(2) Add the initial assertions to the working memory.
(3) Evaluate on the working memory until the fixpoint:
  (a) Request rules that contain GET requests, making the requests and adding the data from the responses to the working memory.

---

[11] http://www.ietf.org/rfc/rfc3986.txt.
[12] http://www.ietf.org/rfc/rfc7230.txt.
[13] http://www.w3.org/TR/rdf11-concepts/.
[14] http://www.w3.org/DesignIssues/LinkedData.html.
[15] http://www.w3.org/DesignIssues/ReadWriteLinkedData.html.
[16] We assume the URI prefix definitions of http://prefix.cc/ The empty prefix denotes http://purl.org/wild/vocab. The base URIs be http://example.org/.
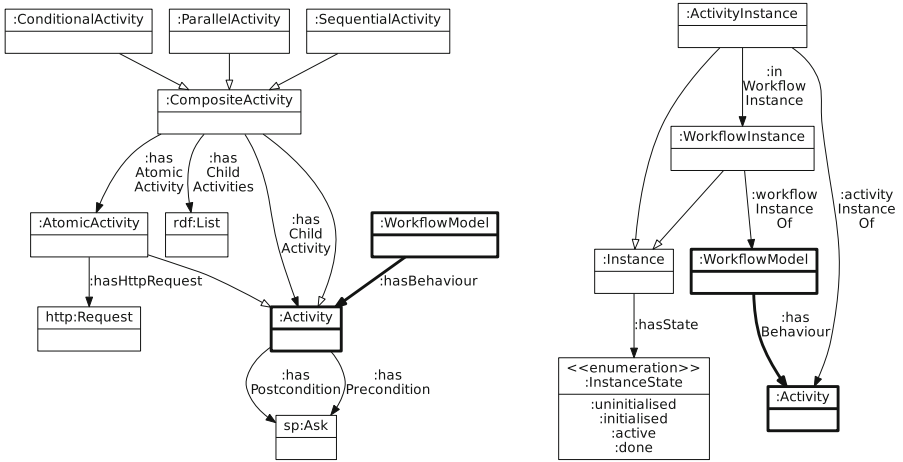
**Fig. 1.** The ontology to express workflow models and instances as UML Class Diagram. Shared classes between the diagrams are depicted in bold. We use the UML Class Diagram's class, inheritance, association, and enumeration to denote the RDFS ontology language's `rdfs:Class`, `rdfs:subClassOf`, `rdf:Property` with `rdfs:domain` and `rdfs:range`, and instances.

(b) Derivation rules, adding the produced data to the working memory.
   We thus acquire data about the world's current state (from the responses to the GET requests) and reason on this data (using the productions).

(4) Evaluate all request rules that contain PUT/POST/DELETE requests on the working memory and make the corresponding HTTP requests. We thus enact changes on the world's state.

A loop over the phases (1) to (4) implements polling, the way to get information about changes in a RESTful environment. Hypermedia-style link following (to discover new information) can be implemented using request rules, e.g. in the example below.

We use the following rule syntax: In the arguments of the binary predicates, we allow for variables (printed in italics). We print constants in typewriter font. We connect rule head and body using $\rightarrow$. The head of a request rule contains one HTTP request with the method as the function name, the target as the first argument, and the RDF payload as the second argument (if applicable). E.g. consider the following rule to retrieve all elements $e$ of a given LDP container:

$$ldp : contains(\texttt{http://example.org/ldpc}, e) \rightarrow \text{GET}(e)$$

## 4 Activity, Workflow Model and Instance Ontology

To describe workflow models and instances as well as activities, we propose an ontology. We developed the ontology, see Fig. 1[17], with execution based on

---

[17] The ontology can be accessed at http://purl.org/wild/vocab.

**Fig. 2.** Workflow (solid: BPMN notation) with sequential activities (`<#A>`, `<#B>`). Dashed: the tree representation with the parent node marked as sequential.



**Fig. 3.** State machine for the instance resources for the workflow and activity instance resources. The dashed part only concerns workflow instance resources.

querying and reasoning under the OWA in mind. In this section, we define activities, workflows, and instances using the workflow in Fig. 2 as example.

*Activities.* We regard an atomic activity as a basic unit of work. We characterise an activity by a postcondition represented as a SPARQL ASK query[18], which has to hold in the world's state after the activity has been executed. We use the postcondition (cf. `:hasPostcondition` in Fig. 1) to monitor the execution of activities in workflows. For the execution of an atomic activity, the activity description needs an HTTP request (cf. `:hasHttpRequest` in Fig. 1).

*Workflow Models.* A workflow model is a set of activities put into a defined order. As notation to describe workflow models, BPMN is a popular choice. The course of action (i.e. control flow) in a BPMN workflow model is denoted using arrows that connect activities and gateways (e.g. decisions and branches). For instance, the middle arrow in the workflow model in Fig. 2 orders activities `<#A>` and `<#B>` sequentially. We call this view on the course of action *flow-based*.

In this paper, instead of a flow-based view on the course of action, we consider a *tree-based* view, as investigated by Vanhatalo et al. [27]. Tree-structured workflow languages include BPEL, a popular language to describe executable workflows. In the tree, activities are leaf nodes. The non-leaf nodes are typed, and the type determines the control flow of the children. The connection between the tree-based (dashed) and the flow-based (solid) workflow representation is depicted in Fig. 2. Flow-based workflows can be losslessly translated to tree-structured workflows and vice versa [21]. We use the tree structure, as checks for completion of workflow parts are easier in a tree. Of the multitude of control flow features of different workflow languages, we support the most basic and common, which have been compiled to the *basic workflow patterns* [25].

---

[18] http://www.w3.org/TR/sparql11-query/.

We now show how to specify workflow models in RDF using Fig. 2's model:

$$:WorkflowModel(\texttt{<\#wfm>}) \wedge :SequentialActivity(\texttt{<\#root>})$$
$$\wedge :AtomicActivity(\texttt{<\#A>}) \wedge :AtomicActivity(\texttt{<\#B>})$$
$$\wedge :hasBehaviour(\texttt{<\#wfm>}, \texttt{<\#root>})$$
$$\wedge :hasChildActivities(\texttt{<\#root>}, rdf:List(\texttt{<\#A>}, \texttt{<\#B>}))$$

As we assume tree-structured workflows, each workflow model (`<#wfm>`) has a root activity (`<#root>`). If an activity is composite, i.e. a control flow element, then the activity has an RDF list of child activities. Here, `<#root>` is sequential, with the child activities `<#A>`, `<#B>`. The child activities could again be composite, thus forming a tree. Leaves in the tree (here `<#A>` and `<#B>`) are atomic activities. We require child activities to be given in an RDF list, which is explicitly terminated. This termination closes the set of list elements and thus allows for executing workflows under the OWA, which e.g. includes querying whether *all* child activities of a parent activity are `:done`. Yet, for the operational semantics we also need a direct connection between a parent activity and a child activity, which we derive from an RDF list using monotonic reasoning, here:

$$:hasChildActivity(\texttt{<\#root>}, \texttt{<\#A>}) \wedge :hasChildActivity(\texttt{<\#root>}, \texttt{<\#B>})$$

*Instances.* Using workflow instances, we can run multiple copies of a workflow model. A workflow instance consequently consists of instances of the model's activities. We model the relation of the instances to their counterparts as shown in Fig. 1. During and after workflow monitoring/execution, the operational semantics maintain the states of instances in an LDP container. At runtime, the instances' states evolve according to the state machine depicted in Fig. 3 (terms from Fig. 1). Section 5 is about the operationalisation of the evolution.

## 5    Operational Semantics

In this section, we give operational semantics to our workflow language[19] in rules[20]. Before we define the rules, we give an overview of what the rules do.

### 5.1    Overview

The rules fulfil the following purposes (the numbers are only to guide the reader):

   I. Retrieve state[21]
   (1) Retrieve the state of the writeable resources in the LDP container, which maintain the workflow/activity instances' state

---

[19] In a production environment, access control to the instances' LDP container needs to be in place to keep third parties from interfering with the monitoring/execution.

[20] A corresponding Notation3 file can be found at http://purl.org/wild/semantics.

[21] A benefit of using Linked Data throughout is that we can access the workflow/activity instances' state and the world's state in a uniform manner.

(2)  Retrieve the relevant world state

II.  Initialise workflow instances if applicable

    (1)  Set the root activity's instance `:active`

    (2)  Set the workflow instance `:initialised`

    (3)  Create instance resources for all activities in the corresponding workflow model and set them `:initialised`

III.  Finalise workflow instances if their root node is `:done`

IV.  Execute and observe `:active` activities

    (1)  Execution: if an atomic activity turns `:active`, fire the HTTP request

    (2)  If the postcondition of an `:active` activity is fulfilled, set it `:done`

V.  Advance composite activities according to control flow, which includes:

    (1)  Set a composite activity's children `:active`

    (2)  Advance between children

    (3)  Finalise a composite activity

## 5.2  Condition-Action Rules

We next give the rules for the listed purposes. To shorten the presentation, we factor out those rules that, for workflow execution, fire an activity's HTTP request if the activity becomes `:active`. Those rules are not needed when monitoring. The rules are of the form (the variable *method* holds the request type):

$$AtomicActivity(a) \wedge hasHttpRequest(a, h) \wedge http{:}mthd(h, method)$$
$$\wedge http{:}requestURI(h, u) \wedge \cdots \rightarrow \textsc{method}(u, \dots)$$

**I. Retrieve State**. The following rules specify the retrieval of data where the rule interpreter locally maintains state. Analogously, other rules retrieve the world's state, either by explicitly stating URIs to be retrieved:

$$\texttt{true} \rightarrow \textsc{get}(\texttt{http://example.org/ldpc})$$

or by following links from data that is already known:

$$ldp{:}contains(\texttt{http://example.org/ldpc}, e) \rightarrow \textsc{get}(e)$$

**II. Initialise Workflow Instances.** If there is an uninitialised workflow instance (e.g. injected by a third party using a POST request into the polled LDP container), the following rules create corresponding resources for the activity instances and set the workflow instance initialised:

$$WorkflowInstance(i) \wedge hasState(i, \texttt{:uninitialised}) \wedge workflowInstanceOf(i, m)$$
$$\wedge hasBehaviour(m, a) \rightarrow \textsc{post}(\texttt{server:ldpc}, activityInstanceOf(\texttt{<\#it>}, a)$$
$$\wedge inWorkflowInstance(\texttt{<\#it>}, i) \wedge hasState(\texttt{<\#it>}, \texttt{:active}))$$

Also, the workflow instance is set initialised (analogously, we initialise instances for the activities in the workflow model):

$$WorkflowInstance(i) \wedge hasState(i, \texttt{:uninitialised}) \wedge workflowInstanceOf(i, m)$$
$$\rightarrow \textsc{put}(i, WorkflowInstance(i) \wedge hasState(i, \texttt{:initialised})$$
$$\wedge workflowInstanceOf(i, m))$$

**III. Finalise Workflow Instances.** The done state of the root activity gets propagated to the workflow instance:

$$WorkflowInstance(i) \wedge hasState(i, \texttt{:active}) \wedge workflowInstanceOf(i, m)$$
$$\wedge hasBehaviour(m, a) \wedge hasState(m, \texttt{:done})$$
$$\rightarrow \textsc{put}(i, WorkflowInstance(i) \wedge hasState(i, \texttt{:done}) \wedge workflowInstanceOf(i, m))$$

**IV. Monitor Atomic Activities.** An activity is done if its postcondition holds.

$$WorkflowInstance(i) \wedge hasState(i, \texttt{:active}) \wedge workflowInstanceOf(i, m)$$
$$\wedge hasDescendantActivity(i, a) \wedge AtomicActivity(a) \wedge hasPostcondition(a, p)$$
$$\wedge ActivityInstance(j) \wedge activityInstanceOf(j, a) \wedge hasState(j, \texttt{:active})$$
$$\wedge sp{:}hasBooleanResult(p, \texttt{true})$$
$$\rightarrow \textsc{put}(j, activityInstanceOf(j, a) \wedge inWorkflowInstance(j, i) \wedge hasState(j, \texttt{:done}))$$

To shorten the presentation of the rules in the following, we introduce the following simplifications: We assume that (1) we are talking about an active workflow instance, and (2) that the resource representing an instance coincides with its corresponding activity in the workflow model. (3), the PUT requests in the text do not actually overwrite the whole resource representation but patch the resources by ceteris paribus overwriting the corresponding $hasState(\cdot, \cdot)$ triple.

**V. Advance According to Control Flow.** We now give the rules for advancing a workflow instance according to the basic workflow patterns (WFPs) [25].

*WFP 1: Sequence.* If there is an active sequential activity with the first activity initialised, we set this first activity to active:

$$SequentialActivity(s) \wedge hasState(s, \texttt{:active}) \wedge hasChildActivities(s, c)$$
$$\wedge rdf{:}first(c, a) \wedge hasState(a, \texttt{:initialised}) \rightarrow \textsc{put}(a, hasState(a, \texttt{:active}))$$

We advance between activities in a sequence using the following rule:

$$SequentialActivity(s) \wedge hasState(s, \texttt{active}) \wedge hasChildActivity(s, c)$$
$$\wedge hasState(c, \texttt{done}) \wedge hasState(n, \texttt{initialised})$$
$$\wedge rdf{:}first(l, c) \wedge rdf{:}rest(l, i) \wedge rdf{:}first(i, n) \rightarrow \textsc{put}(n, hasState(n, \texttt{active}))$$

If we have reached the end of the list of children of a sequence, we regard the sequence as done (the rule is an example of the exploitation of the explicit termination of the RDF list to address the OWA):

$$SequentialActivity(s) \wedge hasState(s, \texttt{:active}) \wedge hasChildActivity(s, c)$$
$$\wedge hasState(c, \texttt{:done}) \wedge rdf : first(l, c) \wedge rdf : rest(l, \texttt{rdf:nil})$$
$$\rightarrow \textsc{put}(s, hasState(s, \texttt{:done}))$$

*WFP 2: Parallel Split.* A parallel activity consists of several activities executed simultaneously. If a parallel activity becomes active, all of its components are set to active:

$$ParallelActivity(p) \wedge hasState(p, \texttt{active}) \wedge hasChildActivity(p, c)$$
$$\wedge hasState(c, \texttt{:initialised}) \rightarrow \textsc{put}(c, hasState(c, \texttt{:active}))$$

*WFP 3: Synchronisation.* If all the components of a parallel activity are done, the whole parallel activity can be considered done. To find out whether all components of a parallel are done, we mark instances as follows. First, we check whether the first child element of the parallel activity is done and mark the element using the state `:doneFromListItemOne`:

$$ParallelActivity(p) \land hasState(p, \texttt{:active}) \land hasChildActivities(p, l)$$
$$\land rdf\!:\!first(l, c) \land hasState(c, \texttt{:done}) \rightarrow hasState(c, \texttt{:doneFromListItemOne})$$

Then, starting from the first activity, we go through the list of child activities and propagate the mark between the activities in the list if the activities are done. If the mark reaches the last list element, the whole parallel activity is done:

$$ParallelActivity(p) \land hasState(p, \texttt{:active}) \land hasChildActivity(p, c)$$
$$\land rdf\!:\!first(l, c) \land rdf\!:\!rest(l, \texttt{rdf:nil}) \land hasState(c, \texttt{:doneFromListItemOne})$$
$$\rightarrow \text{PUT}(p, hasState(p, \texttt{:done}))$$

*WFP 4: Exclusive Choice.* The control flow element choice implements a choice between different alternatives, for which conditions are specified. For the evaluation of the condition, we first have to check whether all child activities are in initialised state, similarly to the rules for WFP 3:

$$ConditionalActivity(a) \land hasState(a, \texttt{:active}) \land hasChildActivities(a, l)$$
$$\land rdf\!:\!first(l, c) \land hasState(c, \texttt{:initialised})$$
$$\rightarrow hasState(c, \texttt{:initialisedFromListItemOne})$$
$$ConditionalActivity(a) \land hasState(a, \texttt{:active}) \land hasChildActivities(a, l)$$
$$\land rdf\!:\!first(l, c) \land hasState(c, \texttt{:initialisedFromListItemOne})$$
$$\land rdf\!:\!rest(l, m) \land rdf\!:\!first(m, d) \land hasState(d, \texttt{:initialised})$$
$$\rightarrow hasState(d, \texttt{:initialisedFromListItemOne})$$

If the check succeeded, we can evaluate the conditions and set an activity active:

$$ConditionalActivity(a) \land hasState(a, \texttt{:active}) \land hasChildActivitiy(a, c)$$
$$\land hasState(c, \texttt{:initialisedFromListItemOne}) \land hasPrecondition(c, p)$$
$$\land rdf\!:\!first(l, c) \land rdf\!:\!rest(l, \texttt{rdf:nil}) \land sp\!:\!hasBooleanResult(p, \texttt{true})$$
$$\rightarrow \text{PUT}(c, hasState(c, \texttt{:active}))$$

We leave it to the modeller to make sure that the preconditions of the children of a conditional activity are mutually exclusive.

*WFP 5: Simple Merge.* If one of the children of a conditional activity is done, the whole conditional activity is done:

$$ConditionalActivity(a) \land hasState(a, \texttt{:active}) \land hasChildActivitiy(a, c)$$
$$\land hasState(c, \texttt{:done}) \rightarrow \text{PUT}(a, hasState(a, \texttt{:done}))$$

# 6   Evaluation

First, we formally show the correctness of our approach to *specifying* workflows by presenting the relationship of our operational semantics to the formal specification of the basic workflow patterns, which we support completely. Second, to show the applicability of our approach in a real-world setting, we report on how we used the approach to do *monitoring* of workflows for human-in-the-loop aircraft cockpit evaluation in Virtual Reality. Third, we empirically evaluate our approach to *executing* workflows in a building simulator.

## 6.1   Mapping to Petri Nets

Van der Aalst et al. use Petri Nets to precisely specify the semantics of the basic workflow patterns [25]. We now show correctness by giving a mapping of our operational semantics to Petri Nets. Similar to *tokens* in a Petri Net that pass between *transitions*, our operational semantics passes the *active* state between *activities* using rules (linking to the WFP rules from Sect. 5.2(V)):

– The rule to advance between activities within a `:SequentialActivity` may only set an activity active if its preceding activity has terminated. In the Petri Net for the Sequence, a transition may only fire if the preceding transition has put a token into the preceding place, see Fig. 4a and the WFP 1 rules.
– Only after the activity before a `:ParallelActivity` has terminated, the rule to advance in a parallel activity sets all child activities active. In the Petri Net for the Parallel Split, all places following transition $T$ get a token iff transition $T$ has fired, see Fig. 4b and the WFP 2 rules.
– Only if all activities in a `:ParallelActivity` have terminated, the rules pass on the active state. In the Petri Net for the Synchronisation, transition $T$ may only fire if there is a place with a token in all incoming arcs (cf. Fig. 4c and the WFP 3 rules).
– In the `ConditionalActivity`, one child activity is chosen by the rule according to mutually exclusive conditions. Similarly, exclusive conditions determine



(a) Sequence

(b) Parallel Split and Exclusive Choice      (c) Synchronisation      (d) Simple Merge
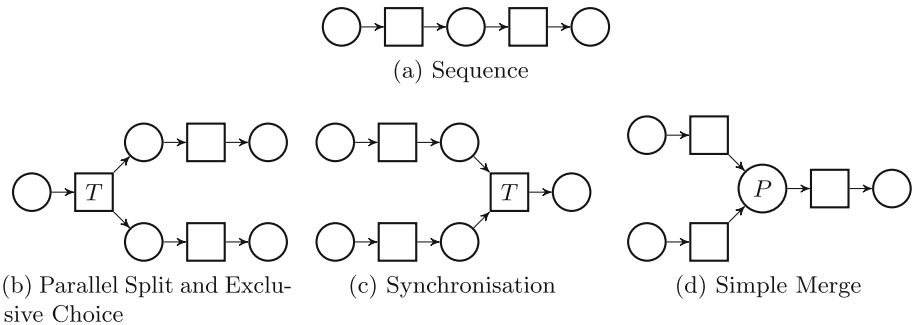
**Fig. 4.** Petri Nets for the basic workflow patterns.

the continuation of the flow after transition $T$ in the Petri Net for the Exclusive Choice, see Fig. 4b and the WFP 4 rules.
– If one child activity of a `:ConditionalActivity` switches from active to done, the control flow may proceed according to the rule. Likewise, the transition following place $P$ in the Petri Net for the Simple Merge (Fig. 4d) may fire iff there is a token in $P$, cf. the WFP 5 rules.

Hence, our approach correctly and completely covers the basic workflow patterns.

## 6.2    Applicability: The Case of Virtual Aircraft Cockpit Design

Together with industry, we successfully applied our approach in aircraft cockpit design [16], where workflow monitoring is used to evaluate cockpit designs regarding Standard Operating Procedures. The monitoring is traditionally done by Human Factors experts using stopwatches in physical cockpits. We built an integrated Cyber-Physical System of Virtual Reality, flight simulation, sensors, and workflows to digitise the monitoring. The challenge was to integrate the different components on both the system interaction and the data level. We built Linked Data interfaces to the components for the interaction integration, and integrated the data using reasoning. Our approach allows for workflow monitoring in the Linked Data setting during runtime. The system's user interface to model workflows has been evaluated by Human Factors experts highly efficient.

## 6.3    Empirical Evaluation Using a Synthetic Benchmark

The scenario for our benchmark is from the Internet of Things domain, where buildings are equipped with sensors and actuators from different vendors. The devices may be not interoperable, which has been identified by NIST as a major challenge for the building industry [7]. Balaji et al. aim to raise interoperability in Building Management Systems by proposing the Brick ontology [1] to model buildings and Building Management Systems. We thus assume Read-Write Linked Data interfaces to a building's management systems and want to execute building automation tasks. We consider tasks that go beyond rule-based automation typically found in home automation (e.g. Eclipse SmartHome[22]) or on the web (e.g. IFTTT[23]). Such tasks require task instance state, e.g.: (1) flow-based control schemes, (2) automated supervision of cleaning personnel, (3) presence simulation, (4) evacuation support. We thus model the tasks as workflows and access the Building Management Systems integrated via Read-Write Linked Data interfaces.

The environment for our benchmark is a Linked Data representation of building 3 of IBM Research Dublin. We built the representation from a static description of building 3 in the Brick ontology[24], which covers the building's parts (e.g.

---

[22] http://www.eclipse.org/smarthome/.
[23] http://ifttt.com/.
[24] http://github.com/BuildSysUniformMetadata/GroundTruth/blob/2e48662/building_instances/IBM_B3.ttl.

**Table 1.** Average runtime [s] for workflows W$n$ in different numbers of buildings.

|  | W1 | W2 | W3 | W4 | W5 |
|---|---|---|---|---|---|
| 1 Building | 2 | 2 | 6 | 12 | 18 |
| 10 Buildings | 8 | 9 | 26 | 61 | 75 |
| 20 Buildings | 12 | 13 | 38 | 80 | 109 |
| 50 Buildings | 19 | 21 | 61 | 156 | 218 |

rooms) and the building's systems (e.g. lights and switches). We subdivided the description into one-hop RDF graphs around each URI from the building and provide each graph at a corresponding URI. To add state information to the systems, we add writeable SSN[25] properties to the Linked Data interface. To evaluate at different scales, we run multiple copies of the building.

The workload for our benchmark is the control flow of the five representative workflow models proposed by Ferme et al. [6] for evaluating workflow engines, determined by clustering workflows from literature, the web, and industry. We interpreted the five workflow models using the four automation tasks presented above: task 1 corresponds to the first two workflow models; the subsequent tasks to the subsequent workflow models. We assigned the activities in the tasks to two classes: monitoring activities that are checks (e.g. a sensor value), where we attached a postcondition, and execution activities that enact change (e.g. turn on a light), where we attached an HTTP request. For repeatability, the postconditions always hold and the requests do not interfere with the workflow.

The set-up for our evaluation consists of a server with a 32-core Intel Xeon E5-2670 CPU and 256 GB of RAM running Debian Jessie. We deploy the operational semantics and required OWL LD reasoning on Linked Data-Fu 0.9.12[26]. We include reasoning as indicated by the Brick ontology. We maintain building and workflow state in LDP containers, LDBBC 0.0.6[27]. We add workflow instances each 0.2 s after 20 s of warm-up time. The workflow models can be found online[28].

The results of our evaluation can be found in Table 1. Varying the number of activities (W1–W5), and varying the number of devices (proportional to buildings), we observe linear behaviour. The linear behaviour stems from the number of requests to be made, which depends on the number of activities and workflow instances. With no reusable data between buildings, there is no benefit in running the workflows for all buildings on one engine. Instead, we could run one engine per building, thus mirroring the decentralisation of data.

---

[25] http://www.w3.org/TR/vocab-ssn/.

[26] http://linked-data-fu.github.io/.

[27] http://github.com/kaefer3000/ldbbc.

[28] http://people.aifb.kit.edu/co1683/2018/iswc-wild/.

## 7    Conclusion

We presented an approach to specify, monitor, and execute applications that builds on distributed data and functionality provided as Read-Write Linked Data. We use workflows to specify applications, and thus defined a workflow ontology and operational semantics. We aligned our approach to the basic workflow patterns, reported on an application in Virtual Reality, and evaluated using a synthetic benchmark in an Internet of Things scenario.

The assumptions of the environment of Read-Write Linked Data present peculiar challenges for a workflow system: We work under the OWA and without events as change notifications. Our approach addresses the challenges without adding assumptions to the architecture of the environment, but by modelling a closed world where necessary and by using polling to access the world's state.

We believe that our approach, which brings workflows in a language that is closely related to the popular BPMN notation to Read-Write Linked Data, enables non-experts to engage in the development of applications that can be verified, validated, and executed.

## References

1. Balaji, B., et al.: Brick: towards a unified metadata schema for buildings. In: Proceedings of the 3rd International Conference on Systems for Energy-Efficient Built Environments (BuildSys) (2016)
2. Capadisli, S., Guy, A., Lange, C., Auer, S., Sambra, A., Berners-Lee, T.: Linked data notifications: a resource-centric communication protocol. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) ESWC 2017. LNCS, vol. 10249, pp. 537–553. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58068-5_33
3. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Deriving active rules for workflow enactment. In: Wagner, R.R., Thoma, H. (eds.) DEXA 1996. LNCS, vol. 1134, pp. 94–115. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0034673
4. Ciortea, A., Boissier, O., Zimmermann, A., Florea, A.M.: Give agents some REST: hypermedia-driven agent environments. In: El Fallah-Seghrouchni, A., Ricci, A., Son, T.C. (eds.) EMAS 2017. LNCS (LNAI), vol. 10738, pp. 125–141. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91899-0_8
5. Elmroth, E., Hernández-Rodriguez, F., Tordsson, J.: Three fundamental dimensions of scientific workflow interoperability: model of computation, language, and execution environment. Future Gener. Comput. Syst. **26**(2), 245 (2010)
6. Ferme, V., Skouradaki, M., Ivanchikj, A., Pautasso, C., Leymann, F.: Performance comparison between BPMN 2.0 workflow management systems versions. In: Reinhartz-Berger, I., Gulden, J., Nurcan, S., Guédria, W., Bera, P. (eds.) BPMDS/EMMSAD -2017. LNBIP, vol. 287, pp. 103–118. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59466-8_7

7. Gallaher, M.P., O'Connor, A.C., Dettbarn Jr., J.L., Gilday, L.T.: Cost analysis of inadequate interoperability in the US capital facilities industry. NIST GCR 04–867 (2004)
8. Gil, Y., Ratnakar, V., Deelman, E., Mehta, G., Kim, J.: Wings for Pegasus. In: Proceedings of the 19th Conference on Innovative Applications of Artificial Intelligence (IAAI) (2007)
9. Gurevich, Y.: Evolving algebras 1993: lipari guide. In: Specification and Validation Methods. Oxford University Press (1995)
10. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - a semantic service-oriented architecture. In: Proceedings of the 3rd International Conference on Web Services (ICWS) (2005)
11. Harth, A., Hose, K., Schenkel, R.: Linked Data Management. CRC, Boca Raton (2014)
12. Harth, A., Speiser, S.: On completeness classes for query evaluation on linked data. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence (2012)
13. Hull, R., et al.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 1–24. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19589-1_1
14. Jablonski, S., Bussler, C.: Workflow Management. International Thomson, London (1996)
15. Käfer, T., Harth, A.: Rule-based programming of user agents for linked data. In: Proceedings of the 11th International Workshop on Linked Data on the Web (LDOW) (2018)
16. Käfer, T., Harth, A., Mamessier, S.: Towards declarative programming and querying in a distributed cyber-physical system: the i-VISION case. In: Proceedings of the 2nd CPSData Workshop (2016)
17. Newman, S.: Building Microservices - Designing Fine-Grained Systems. O'Reilly, Sebastopol (2015)
18. Pautasso, C.: RESTful web service composition with BPEL for REST. Data Knowl. Eng. **68**(9), 851 (2009)
19. Pautasso, C., Wilde, E.: Push-enabling RESTful business processes. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 32–46. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25535-9_3
20. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful web services vs. "Big" web services. In: Proceedings of the 17th International Conference on World Wide Web (WWW) (2008)
21. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 276–293. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15618-2_20
22. Rospocher, M., Ghidini, C., Serafini, L.: An ontology for the business process modelling notation. In: Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS) (2014)
23. Stadtmüller, S., Speiser, S., Harth, A., Studer, R.: Data-Fu: a language and an interpreter for interaction with read/write linked data. In: Proceedings of the 22nd International Conference on World Wide Web (WWW) (2013)
24. Turi, D., Missier, P., Goble, C.A., De Roure, D., Oinn, T.: Taverna workflows: syntax and semantics. In: Proceedings of the 3rd International Conference on e-Science and Grid Computing (e-Science) (2007)
25. Van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1), 5 (2003)

26. Van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data Knowl. Eng. **53**(2), 129 (2005)
27. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 100–115. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_10
28. Verborgh, R., Steiner, T., van Deursen, D., Coppens, S., Vallés, J.G., van de Walle, R.: Functional descriptions as the bridge between hypermedia APIs and the Semantic Web. In: Proceedings of the 3rd International Workshop on RESTful Design (WS-REST) (2012)
29. Zaveri, A.: smartAPI: towards a more intelligent network of web APIs. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) ESWC 2017. LNCS, vol. 10250, pp. 154–169. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58451-5_11
30. Zur Muehlen, M., Nickerson, J.V., Swenson, K.D.: Developing web services choreography standards. Decis. Supp. Syst. **40**(1), 9 (2005)