Monitoring and Executing Workflows in Linked Data Environments

Tobias Käfer and Andreas Harth

Institute AIFB, Karlsruhe Institute of Technology (KIT), Germany
{tobias.kaefer|harth}@kit.edu

Abstract. The W3C's Web of Things working group is aimed at addressing the interoperability problem on the Internet of Things using Linked Data as uniform interface. While Linked Data paves the way towards combining such devices into integrated applications, traditional solutions for specifying the control flow of applications do not work seamlessly with Linked Data. We therefore tackle the problem of the specification, execution, and monitoring of applications in the context of Linked Data. We present a novel approach that combines workflows, semantic reasoning, and RESTful interaction into one integrated solution. We contribute to the state of the art by (1) defining an ontology for describing workflow models and instances, (2) providing operational semantics for the ontology that allows for the execution and monitoring of workflow instances, (3) presenting a benchmark to evaluate our solution. Moreover, we showcase how we used the ontology and the operational semantics to monitor pilots executing workflows in virtual aircraft cockpits.

1 Introduction

Information systems become increasingly distributed. Consider the growing deployment of sensors and actuators, the modularisation of monolithic software into microservices, and the movement of data from centralised silos into user-owned data pods. The drivers for the 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 [15],
- Fast development cycles require independent evolution of components [15],
- Privacy-aware users demand to retain ownership of their data[2]

In such modern information systems, technologies around REST and Linked Data can be used to provide uniform interfaces to distributed components. Con-

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

[2] "Putting Data back into the Hands of Owners", http://tcrn.ch/2i8h7gp

sider, for example, 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, while a variant of RDF provides a uniform data model for representing component state. RESTful APIs and Linked Data, in combination with semantic reasoning, can in principle facilitate the aggregation of distributed components into applications. While techniques and architectures for (read-only) data integration systems based on Linked Data are relatively mature [11], techniques for the creation of applications that combine components with read and write access are still an active area of research [25, 4, 2]. In this paper, we tackle the problem of specifying, executing, and monitoring application behaviour in the context of REST and Linked Data.

The playing field for applications in the context of REST and 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 specification[11] specifies interaction with Read-Write Linked Data sources. Next to projects such as Solid, these technologies can offer read-write access to sensors and actuators on the Web of Things[3] or even a car entertainment system[6]. Other non-RDF REST APIs provide access to weather reports[7] or building management systems (e. g. Project Haystack[8]). In case these components do not already use RDF, they can be wrapped to support RDF.

To build applications that combine components under diverse ownership is difficult due to heterogeneity of the network protocol and the syntax and semantics of the component's data. Uniform interfaces provide a path towards simplifying the integration, however, making all components adhere to a uniform interface is costly. Therefore, the current way to combine different APIs is via mashups that are often written using imperative languages. Also, imperative code is used to resolve heterogeneities, but doing so is costly and inflexible. If we assume that component providers increasingly use Read-Write Linked Data as interface, we can attempt to use higher-level programming abstractions such as workflows as notion for realising the vision of programming-in-the-large [5].

Our approach is to combine RESTful interaction (to resolve the heterogeneity on the protocol level), semantic reasoning (to resolve the heterogeneity on the data level) and workflow-based application control flow specification for a flexible high-level description of application behaviour that operates over distributed components. In the combination of the three – RESTful interaction, semantic reasoning, and workflows –, there exist solutions for the combination of two: RESTful interaction and semantic reasoning, e. g. by Stadtmüller et al. [21]. But for an integrated solution, we have to consider the other two combinations left:

[3] http://www.w3.org/WoT/
[4] https://solid.mit.edu/
[5] http://lod-cloud.net/
[6] http://www.w3.org/Submission/2016/01/
[7] http://openweathermap.org/
[8] http://www.project-haystack.org/

workflows and REST, and workflows and reasoning. The latter two combinations pose challenges, which we address in our approach:

The absence of events in REST HTTP implements the CRUD operations (create, retrieve, update, delete), but not the subscriptions to events. However, traditional approaches from workflow management build on event data.

Reasoning under the open-world assumption Ontology languages such as RDFS and OWL make the open-world assumption (OWA). Approaches from workflow management typically operate on relational databases, where the closed-world assumption is made.

While both challenges could be mitigated by extending the technologies (e. g. implement events using Web Sockets[9], or Linked Data Notifications [2]) or by introducing additional assumptions (e. g. negation-as-failure once we reach a certain completeness class [12]), those mitigation strategies would restrict the generality of the approach. In contrast, our approach can include arbitrary web resources with more liberal completeness classes, and builds on fast data processing using only monotonic operators.

Previous work from Workflow Management and Business Process Management (BPM), Semantic Web Services (SWS), and the REST community is not sufficient: Traditional work from Web Services builds on arbitrary method calls as interaction model. Correspondingly, assignments of process variables, which typically contain results of method calls, are basis of decisions. The result of such an execution model is that data from different components is stored and processed separately, and not in an integrated fashion. In REST however, resources and state are first-class citizens. Correspondingly, we would like the combined state of resources (e. g. integrated by reasoning) to be basis of decisions. Ideas from SWS, where elaborate service description languages such as WSMO and OWL-S have been developed, have recently been picked up again: RESTdesc allows for describing REST calls for automated composition using proofs [25]. Another related approach is smartAPI, which is about describing REST calls semantically to support developers of compositions [26]. However, elaborate and correct descriptions, which we do not yet see available at web scale, represent a major obstacle to achieve automated composition. We thus want to begin with manually specified applications based on workflows as a step towards automated composition. We provide a discussion of more related work in Section 2.

The paper is structured as follows: In Section 2, we discuss related work. In Section 3, we introduce the scenario we use for examples throughout the paper and for our benchmark. Next, in Section 4, we provide an overview of the technologies underlying our approach. Subsequently, we present our contributions:

– An ontology to describe workflows and workflow instances modelled in RDFS[10] (Section 5), which allows for reasoning over workflows and workflow instances under the OWA. RDFS has a low expressivity but allows for fast rule-based data processing using only monotonic operators.

[9] `http://www.ietf.org/rfc/rfc6455.txt`
[10] `http://www.w3.org/TR/rdf-schema/`

- Operational semantics for our workflow ontology. We use a condition-action rule language for data processing and API interaction in the context of Linked Data (Section 6). The rule language does not require event data. We maintain workflow state in a Linked Data Platform[11] Container. Thus, we can access the workflow state and the world state in a uniform manner, and rely on reasoning for data integration.
- A formal and an empirical evaluation of our approach, which includes the definition of a benchmark from the Smart Buildings domain and a showcase around the design of aircraft cockpits in Virtual Reality (Section 7).

We conclude and outline future work in Section 8.

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. This ECA rule-based approach has been used to give operational semantics to workflow languages [13], and to implement workflow management systems [3]. Similar to the case handling paradigm [23], we employ state machines for the activities of a workflow instance.

Web Services WS-*-based approaches assume arbitrary operations, whereas our approach works with REST resources, where set of operations is constrained. The BPM community has compared WS-*- and REST-based approaches [18, 27]. Pautasso et al. proposed extensions to BPEL such that e. g. a BPEL process [16] can invoke REST services, and that REST resources representing processes push events [17]. 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 We can only present a selection of the large body of reasearch conducted in the area of SWS. Approaches like OWL-S, WSMO and semantic approaches to scientific workflows like [8] are mainly concerned with service descriptions and corresponding reasoning for composition and provenance tracking. For the execution of compositions, SWS build on Web Service technology instead of REST, e. g. the execution in the context of WSMO, WSMX [10], is entirely event-based. European projects such as "Super" and "Adaptive Services Grid" build on WSMO.

Ontologies for Workflows Similar to workflows in our ontology, processes in OWL-S are also tree-structured (see Section 4) and make use of lists in RDF. Unlike OWL-S, our ontology also covers workflow instances. Rospocher et al. [20] and the project "Super" developed ontologies that describe process meta models such as BPMN, BPEL, and EPC[12]. Their ontologies require more expressive reasoning or do not allow for execution under the OWA.

[11] `http://www.w3.org/TR/ldp/`

[12] `http://www.ip-super.org/content/view/129/136/`, available in the Web Archive

3 Scenario

Similar to the IoT domain, fragmentation of technologies is also an issue in the area of building automation: NIST identified interoperability as major challenge for the building industry [7]. Balaji et al. aim to raise interoperability in Building Management Systems (BMS) using Semantic Technologies: Using the Brick ontology [1], people can model buildings and corresponding BMSs. Read-Write Linked Data interfaces to a building's BMSs allow for executing automation tasks for the building. Such tasks can include: (1) Control schemes based on time, sensor data and web data, (2) Automated supervision of cleaning personnel, (3) Presence simulation, (4) Evacuation support. Those tasks go beyond simple rule-based automation tasks as typically found in home automation (e. g. Eclipse SmartHome[13]) and on the web (e. g. IFTTT[14]), as the tasks require a notion of task instance state. We therefore model the tasks as workflow and run them as workflow instances, which access the building management systems in an integrated fashion using the Read-Write Linked Data interfaces.

As building, we use a description of building 3 at IBM Research Dublin in Ireland. Balaji et al. provide a static description of the building using the Brick ontology[15]. The description covers the building's parts (e. g. rooms) and the parts of the building's systems (e. g. lights and switches). We serve the description on a Linked Data interface. To add state information to the systems, we add writeable properties from the SSN ontology[16] to the Linked Data interface.

4 Preliminaries

In this section, we introduce the technologies on which we build our approach.

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[17] should be used for identifying things. HTTP GET[18] requests to those URIs should be answered using descriptive data, e. g. in RDF[19]. Hyperlinks in the data should enable the discovery of more information[20]. Read-Write Linked Data[21] introduces write access to Linked Data (later standardised in the Linked Data Platform specification[11]). The HTTP PUT, POST, DELETE methods can serve as means to write to Linked Data and thus may be used to change state.

[13] `https://www.eclipse.org/smarthome/`
[14] `https://ifttt.com/`
[15] `https://github.com/BuildSysUniformMetadata/GroundTruth/blob/2e48662/building_instances/IBM_B3.ttl`
[16] `https://www.w3.org/TR/vocab-ssn/`
[17] `http://www.ietf.org/rfc/rfc3986.txt`
[18] `http://www.ietf.org/rfc/rfc7230.txt`
[19] `http://www.w3.org/TR/rdf11-concepts/`
[20] `https://www.w3.org/DesignIssues/LinkedData.html`
[21] `https://www.w3.org/DesignIssues/ReadWriteLinkedData.html`

In the paper, we denote RDF triples using binary predicates[22]. For instance, we write for the Triple in Turtle notation "`<http://example.org/ldpc/1#it> rdf:type :WorkflowModel .`" the following:

$$\textit{rdf:type}(\texttt{http://example.org/ldpc/1\#it}, \texttt{:WorkflowModel})$$

We may abbreviate a class assignment using a unary predicate with the class as predicate name, e. g. $\textit{:WorkflowModel}(\texttt{http://example.org/ldpc/1\#it})$.

Condition-Action Rule Language We use a monotonic production rule language to specify both forward-chaining reasoning on RDF data and interaction with REST resources [21]. We distinguish two types of rules: (1) derivation rules, which contain the productions, and (2) request rules, which have a specification of an HTTP request in the rule head. We assume safe rules, i. e. all universally quantified variables in the head are also contained in the body. Moreover, we exclude derivation rules with existential quantifiers in the head. We can implement hypermedia-style link following using request rules. Rule programs consist in initial assertions and rules, where the initial assertions cannot get modified.

The operational semantics of the rule language is built on an Abstract State Machine (ASM) [9] abstraction of Read-Write Linked Data. The data processing is done in repeating steps, which we subdivide into the following phases:

(1) Initially, the working memory of the rule interpreter is empty.
(2) The interpreter adds the initial assertions to its working memory.
(3) The interpreter evaluates on the working memory until the fixpoint:
    (a) Request rules that contain GET requests. The interpreter makes the requests, and adds the data from the responses to the working memory.
    (b) Derivation rules, adding the produced data to the working memory.
    Using those GET requests and derivations, the interpreter acquires knowledge about the current world state (from the responses to the GET requests) and reasons on this knowledge (using the productions).
(4) The interpreter evaluates all request rules with HTTP methods other than GET on the working memory and makes the corresponding HTTP requests. Using those requests, the interpreter changes the world state.

To implement polling to get information about changes in a RESTful environment, the interpreter loops all of those phases (emptying the working memory each time step 1 is reached). Hyperlink following (to discover new information) can be implemented using request rules.

In this paper, we use the following syntax for the rules: In the binary predicates for RDF, we allow for variables. Constants are printed in typewriter font. Head and body of derivation rules are connected using $\Longrightarrow$. Head and body of request rules are connected using $\rightarrow$. The head of a request rule contains one HTTP request with the method as function name, the target as first parameter, and the RDF payload as second parameter.

[22] We assume URI prefix definitions as provided by `http://prefix.cc/`. For the ontology that we present in this paper, we use the empty prefix, which shall denote `http://purl.org/wild/vocab#`.

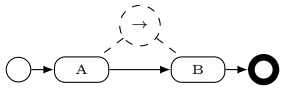


Fig. 1: Workflow with two sequential activities (A, B) in BPMN notation (solid). Dashed: the tree representation with the parent node marked as sequential.

Activities, Workflows On the level of the workflow specification, we regard an atomic activity as a basic unit of work. We characterise the activity by a postcondition, which states what holds in the world state after the activity has been executed. We give the postcondition as a SPARQL ASK query[23]. For the execution of an atomic activity, the activity description needs an HTTP request.

A workflow is a set of activities put into a defined order. To express a workflow, BPMN widely-used notation. In the simplest form, the course of action (i. e. control flow) in a BPMN workflow is denoted using arrows that connect activities and gateways (e. g. decisions, branches). In this paper, we assume a different view on the control flow of a workflow: A tree, as investigated by Vanhatalo et al. [24]. Here, the activities are leaf nodes in the tree. The non-leaf nodes are typed, and the type determines the control flow of the children. The family of tree-structured workflow languages includes BPEL, a language to describe executable workflows. We show a workflow in Figure 1. Flow-based workflows can be losslessly translated to tree-structured workflows and vice versa [19]. We use the tree view, as checks for completion of workflow parts are easier in a tree.

5 Ontology for Workflow Models and Instances

We now present our ontology for workflow models and instances[24]. Figure 2 illustrates the classes from the ontology in diagrams for models and instances.

A workflow model is constructed as follows: As we assume tree-structured workflows, each workflow model has a root activity. If the activity is composite, i. e. a control flow element, then the activity has a RDF list of child activities. Those can again be composite, thus forming a tree. The leaves in the tree are atomic activities. Note that the RDF list implements an explicitly terminated linked list, which allows us for processing the workflows under the OWA.

6 Operational Semantics

In this section, we give operational semantics in rules to our workflow language. Before we define the rules, we give a high-level overview on what the rules do.

6.1 Overview

The rules fulfil the following purposes:

[23] `http://www.w3.org/TR/sparql11-query/`
[24] The ontology can be accessed at `http://purl.org/wild/vocab`

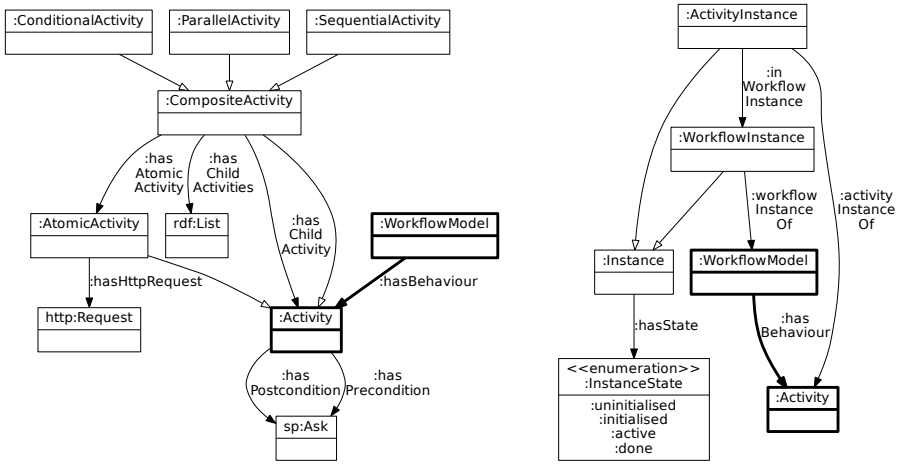


Fig. 2: The ontology to express workflow models and instances as UML Class Diagram. Shared classes between the diagrams are depicted in bold. We use the UML Class Diagram's class, inheritance, association, and enumeration to denote the RDFS ontology language's `rdfs:Class`, `rdfs:subClassOf`, `rdf:Property` with `rdfs:domain` and `rdfs:range`, and instances.

I. Retrieve state
   a) Retrieve the state of the writeable Linked Data resources, which are used to maintain program state
   b) Retrieve the relevant world state
II. Initialise workflow instances
   a) Set the root activity's instance active
   b) Set the workflow instance initialised
   c) Creating 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 Activities
   a) Execution: when setting an atomic activity active, fire the HTTP request
   b) If the postcondition of an active activity is fulfilled, set it done
V. Advance Composite Activities according to basic control workflow patterns [22], which includes:
   a) Set a composite activitiy's children active
   b) Advance between children
   c) Finalise a composite activity

The activity instances' and the workflow instances' state evolves according to the state machine depicted in Figure 3.

6.2 Condition-Action Rules

In this section, we give the rules for the listed purposes. To shorten our presentation, we note that to do the actual execution of activities, to all rules that set

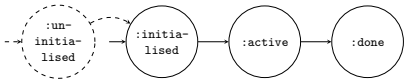


Fig. 3: State machine for the instance resources for the workflow and activity instance resources. The dashed only concerns workflow instance resources.

an atomic activity active, a rule along the following lines has to be added such that the HTTP request is performed (note the variable *method* determines the request type to be made):

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

I. Retrieve State The following rules specify the retrieval of data where the rule interpreter locally maintains state. Analogously, the world state can get retrieved: 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:

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

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

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

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

$$\begin{gathered} \mathit{WorkflowInstance}(i) \wedge \mathit{hasState}(i,\texttt{uninitialised}) \wedge \mathit{workflowInstanceOf}(i,m) \\ \rightarrow \textsc{put}(i,\ \mathit{WorkflowInstance}(i) \wedge \mathit{hasState}(i,\texttt{initialised}) \\ \wedge \mathit{workflowInstanceOf}(i,m)) \end{gathered}$$

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

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

IV. Execute and Observe Atomic Activities In the following, we give a rule in its entirety, which marks an activity as done if its postcondition is fulfilled. Afterwards, we introduce simplifications that make the presentation easier to understand while still conveying the core elements.

$$\begin{gathered}\mathit{WorkflowInstance}(i) \wedge \mathit{hasState}(i, \texttt{active}) \wedge \mathit{workflowInstanceOf}(i, m) \\ \wedge \mathit{hasDescendantActivity}(i, a) \wedge \mathit{AtomicActivity}(a) \wedge \mathit{hasPostcondition}(a, p) \\ \wedge \mathit{ActivityInstance}(j) \wedge \mathit{activityInstanceOf}(j, a) \wedge \mathit{hasState}(j, \texttt{active}) \\ \wedge \mathit{sp{:}hasBooleanResult}(p, \texttt{true}) \\ \rightarrow \text{PUT}(j, \mathit{activityInstanceOf}(j, a) \wedge \mathit{inWorkflowInstance}(j, i) \wedge \mathit{hasState}(j, \texttt{done}))\end{gathered}$$

To shorten the presentation of the rules, 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 triple.

V. Advance According to Control Flow In this section, we give the rules for advancing a workflow instance according to workflow patterns.

Workflow Pattern 1: Sequence If there is an active sequential activity with the first activity initialised, we set this first activity to active:

$$\begin{gathered}\mathit{SequentialActivity}(s) \wedge \mathit{hasState}(s, \texttt{active}) \wedge \mathit{hasChildActivities}(s, c) \\ \wedge \mathit{rdf{:}first}(c, a) \wedge \mathit{hasState}(a, \texttt{initialised}) \rightarrow \text{PUT}(a, \mathit{hasState}(a, \texttt{active}))\end{gathered}$$

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

$$\begin{gathered}\mathit{SequentialActivity}(s) \wedge \mathit{hasState}(s, \texttt{active}) \wedge \mathit{hasChildActivity}(s, c) \\ \wedge \mathit{hasState}(c, \texttt{done}) \wedge \mathit{hasState}(n, \texttt{initialised}) \\ \wedge \mathit{rdf{:}first}(l, c) \wedge \mathit{rdf{:}rest}(l, i) \wedge \mathit{rdf{:}first}(i, n) \rightarrow \text{PUT}(n, \mathit{hasState}(n, \texttt{active}))\end{gathered}$$

If we have reached the end of the list of children of a sequence, we regard the sequence as done (Here, we exploit the explicit termination of the RDF list to address the open world assumption):

$$\begin{gathered}\mathit{SequentialActivity}(s) \wedge \mathit{hasState}(s, \texttt{active}) \wedge \mathit{hasChildActivity}(s, c) \\ \wedge \mathit{hasState}(c, \texttt{done}) \wedge \mathit{rdf{:}first}(l, c) \wedge \mathit{rdf{:}rest}(l, \texttt{rdf:nil}) \rightarrow \text{PUT}(s, \mathit{hasState}(s, \texttt{done}))\end{gathered}$$

Workflow Pattern 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:

$$\begin{gathered}\mathit{ParallelActivity}(p) \wedge \mathit{hasState}(p, \texttt{active}) \wedge \mathit{hasChildActivity}(p, c) \\ \wedge \mathit{hasState}(c, \texttt{initialised}) \rightarrow \text{PUT}(c, \mathit{hasState}(c, \texttt{active}))\end{gathered}$$

Workflow Pattern 3: Synchronisation If all the components of a parallel are done, the whole parallel can be considered done. To find out whether all components of a parallel are done, we have to mark instances the following way to deal with the RDF list. First, we check whether the first element of the children of the parallel activity is done:

$$\begin{gathered} \mathit{ParallelActivity}(p) \wedge \mathit{hasState}(p, \texttt{active}) \wedge \mathit{hasChildActivities}(p, l) \\ \wedge \mathit{rdf{:}first}(l, c) \wedge \mathit{hasState}(c, \texttt{done}) \implies \mathit{hasState}(c, \texttt{doneFromListItemOne}) \end{gathered}$$

Then, starting from the first, we one by one check the activities in the list of child activities whether they are done: If the check proceeded to the last list element, the whole parallel activity is done:

$$\begin{gathered} \mathit{ParallelActivity}(p) \wedge \mathit{hasState}(p, \texttt{active}) \wedge \mathit{hasChildActivity}(p, c) \\ \wedge \mathit{rdf{:}first}(l, c) \wedge \mathit{rdf{:}rest}(l, \texttt{rdf:nil}) \wedge \mathit{hasState}(c, \texttt{doneFromListItemOne}) \\ \rightarrow \textsc{put}(p, \mathit{hasState}(p, \texttt{done})) \end{gathered}$$

Workflow Pattern 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 Workflow Pattern 3:

$$\begin{gathered} \mathit{ConditionalActivity}(a) \wedge \mathit{hasState}(a, \texttt{active}) \wedge \mathit{hasChildActivities}(a, l) \\ \wedge \mathit{rdf{:}first}(l, c) \wedge \mathit{hasState}(c, \texttt{initialised}) \\ \implies \mathit{hasState}(c, \texttt{initialisedFromListItemOne}) \\ \mathit{ConditionalActivity}(a) \wedge \mathit{hasState}(a, \texttt{active}) \wedge \mathit{hasChildActivities}(a, l) \\ \wedge \mathit{rdf{:}first}(l, c) \wedge \mathit{hasState}(c, \texttt{initialisedFromListItemOne}) \\ \wedge \mathit{rdf{:}rest}(l, m) \wedge \mathit{rdf{:}first}(m, d) \wedge \mathit{hasState}(d, \texttt{initialised}) \\ \implies \mathit{hasState}(d, \texttt{initialisedFromListItemOne}) \end{gathered}$$

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

$$\begin{gathered} \mathit{ConditionalActivity}(a) \wedge \mathit{hasState}(a, \texttt{active}) \wedge \mathit{hasChildActivitiy}(a, c) \\ \wedge \mathit{hasState}(c, \texttt{initialisedFromListItemOne}) \wedge \mathit{hasPrecondition}(c, p) \\ \wedge \mathit{rdf{:}first}(l, c) \wedge \mathit{rdf{:}rest}(l, \texttt{rdf:nil}) \wedge \mathit{sp{:}hasBooleanResult}(p, \texttt{true}) \\ \rightarrow \textsc{put}(c, \mathit{hasState}(c, \texttt{active})) \end{gathered}$$

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

Workflow Pattern 5: Simple Merge If one of the children of a conditional activity is done, the whole conditional activity is done:

$$\begin{gathered} \mathit{ConditionalActivity}(a) \wedge \mathit{hasState}(a, \texttt{active}) \wedge \mathit{hasChildActivitiy}(a, c) \\ \wedge \mathit{hasState}(c, \texttt{done}) \rightarrow \textsc{put}(a, \mathit{hasState}(a, \texttt{done})) \end{gathered}$$

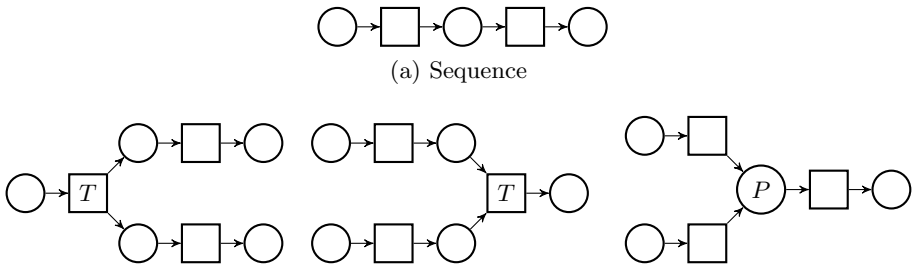
7 Evaluation

In this section, we provide two evaluations: We formally showing the correctness of our approach by presenting the relation of our operational semantics to the Petri-Net-based formal specification of the workflow patterns [22]. We then empirically evaluate our approach in the scenario presented in Section 3. We also report on a showcase implemented as part of an EU project.

7.1 Mapping 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:

- 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 Figure 4a.
- 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 Figure 4b.
- 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. Figure 4c).
- In the `ConditionalActivity`, one child activity is chosen by the rule according to mutually exclusive conditions. Similarly, exclusive conditions determine the continuation of the flow after transition T in the Petri Net for the Exclusive Choice, see Figure 4b.
- 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 (Figure 4d) may fire iff there is a token in P.



(a) Sequence

(b) Parallel Split and Exclusive Choice

(c) Synchronisation

(d) Simple Merge

Fig. 4: Petri Nets for the Basic Workflow Patterns.

Table 1: Average runtime [s] for workflows Wn 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

7.2 Empirical Evaluation

We provide an empirical evaluation in the Smart Building scenario of Section 3:

As environment for our evaluation, we built a Linked Data representation of building 3 of IBM Research Dublin from the Brick ontology [1]: We subdivided the ontology into one-hop RDF graphs around each URI from the building and provide each graph for dereferencing at the corresponding URI. As there are no blank nodes in the ontology, we did not lose data during the subdivision. Writeable Linked Data resources allow for actuation in the building. To evaluate at different scales, we can run multiple editions of the building.

As workload, we use the control flow of the five representative workflow models proposed by Ferme et al. [6] for evaluating workflow engines, determined via clustering workflows collected from literature, the web, and industry. We interpreted the workflow models according to the automation tasks presented in Section 3: For task 1, we use the first two workflow models from [6]; for the subsequent tasks, the subsequent workflow models follow in the same order. We distinguish two types of activities in the tasks: activities that are mere checks i. e. have only a postcondition (e. g. an hour of the day to build time-based control), and tasks that enact change (e. g. turn on a light), where we attach an HTTP request. We assigned the types to the activities in the workflows, and made sure for reproducibility that the postconditions are always fulfilled and the requests do not interfere with the workflow.

We run both the buildings and the workflow management on an 32-core Intel Xeon CPU E5-2670 with 256 GB of RAM running Debian Jessie. We deploy the workflow management with rules for the operational semantics, and required RDFS reasoning on Linked Data-Fu 0.9.12[25]. Moreover, we add rules for inverse properties as presented in the examples of the Brick ontology. Both the buildings and the workflow state are maintained in individual Eclipse Jetty servers running LDBBC 0.0.6[26] LDP implementations, based on JAX-RS, and NxParser. We allow for a warm-up time of 20 s before we add workflow instances each 0.2 s. The workflow models, together with additional information, can be found online[27].

[25] `http://linked-data-fu.github.io/`
[26] `http://github.com/kaefer3000/ldbbc`
[27] `http://people.aifb.kit.edu/co1683/2018/eswc-wild/`

7.3 Application in Virtual Aircraft Cockpit Design

We successfully applied the approach of this paper in workflow-aware aircraft cockpit design [14]. Workflow monitoring allows to evaluate cockpit designs regarding Standard Operating Procedures. The monitoring is typically done by a Human Factors expert using pen, paper, and stopwatch. Early in the design process, cockpits are only available in Virtual Reality built by CAD experts. We integrated the CAD and the Human Factors view on the cockpit using reasoning, built Linked Data interfaces to the subsystems of the Virtual Reality to get access to live state information, and used our approach to digitise the process of monitoring pilots execute workflows in Virtual Reality.

8 Conclusion and Future Work

We presented an ontology to describe workflows and operational semantics for the workflow language to run workflows on the web architecture. We formally aligned the ontology to the Basic Workflow Patterns, provided an empirical evaluation and reported on the applicability of the approach in Virtual Reality.

The assumptions of the web architecture present a peculiar environment for a workflow system: There are no notifications and we work under the open-world assumption. Thus, we resort to polling, which comes at the price of all sampling-based approaches: to miss system states.

In the future, we want to look into data-centric workflow languages.

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, AV, and Berners-Lee, T: Linked Data Notifications: A Resource-Centric Communication Protocol. In: Proceedings of the 14th European Semantic Web Conference (ESWC) (2017)
3. Casati, F, Ceri, S, Pernici, B, and Pozzi, G: Deriving Active Rules for Workflow Enactment. In: Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA) (1996)
4. Ciortea, A, Boissier, O, Zimmermann, A, and Florea, AM: Give Agents Some REST. In: Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) (2017)
5. DeRemer, F and Kron, HH: Programming-in-the-Large Versus Programming-in-the-Small. IEEE Transactions on Software Engineering 2(2) (1976)
6. Ferme, V, Skouradaki, M, Ivanchikj, A, Pautasso, C, and Leymann, F: Performance Comparison Between BPMN 2.0 Workflow Management Systems Versions. In: Proceedings of the 18th International Conference on Business Process Modeling, Development, and Support (BPMDS) (2017)
7. Gallaher, MP, O'Connor, AC, Dettbarn Jr., JL, and Gilday, LT: 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, and 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, and 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, and Schenkel, R: Linked Data Management. CRC (2014)
12. Harth, A and 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: Proceedings of the 7th International Workshop on Web Services and Formal Methods (WS-FM) (2011)
14. Käfer, T, Harth, A, and 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)
15. Newman, S: Building microservices - designing fine-grained systems. O'Reilly (2015)
16. Pautasso, C: RESTful Web Service Composition with BPEL for REST. Data and Knowledge Engineering 68(9) (2009)
17. Pautasso, C and Wilde, E: Push-Enabling RESTful Business Processes. In: Proceedings of the 9th International Conference on Service-Oriented Computing (ICSOC) (2011)
18. Pautasso, C, Zimmermann, O, and Leymann, F: RESTful Web Services vs. "Big" Web Services. In: Proceedings of the 17th International Conference on World Wide Web (WWW) (2008)
19. Polyvyanyy, A, García-Bañuelos, L, and Dumas, M: Structuring Acyclic Process Models. In: Proceedings of the 8th International Conference on Business Process Management (BPM) (2010)
20. Rospocher, M, Ghidini, C, and 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)
21. Stadtmüller, S, Speiser, S, Harth, A, and 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)
22. Van der Aalst, WMP, ter Hofstede, AHM, Kiepuszewski, B, and Barros, AP: Workflow Patterns. Distributed and Parallel Databases 14(1) (2003)
23. Van der Aalst, WMP, Weske, M, and Grünbauer, D: Case handling: a new paradigm for business process support. Data and Knowledge Engineering 53(2) (2005)
24. Vanhatalo, J, Völzer, H, and Koehler, J: The Refined Process Structure Tree. In: Proceedings of the 6th International Conference on Business Process Management (BPM) (2008)
25. Verborgh, R, Steiner, T, van Deursen, D, Coppens, S, Vallés, JG, and 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)
26. Zaveri, A et al.: smartAPI: Towards a More Intelligent Network of Web APIs. In: Proceedings of the 14th European Semantic Web Conference (ESWC) (2017)
27. Zur Muehlen, M, Nickerson, JV, and Swenson, KD: Developing Web Services Choreography Standards. Decision Support Systems 40(1) (2005)