Journal on Data Semantics
https://doi.org/10.1007/s13740-021-00123-0

ORIGINAL ARTICLE

Specifying and Executing User Agents in an Environment of Reasoning and RESTful Systems Using the Guard-Stage-Milestone Approach

Tobias Käfer[1] · **Benjamin Jochum**[1] · **Nico Aßfalg**[1] · **Leonard Nürnberg**[1]

Received: 3 February 2020 / Revised: 1 March 2021 / Accepted: 29 March 2021

Abstract
For Read-Write Linked Data, an environment of reasoning and RESTful interaction, we investigate the use of the Guard-Stage-Milestone approach for specifying and executing user agents. We present an ontology to specify user agents. Moreover, we give operational semantics to the ontology in a rule language that allows for executing user agents on Read-Write Linked Data. We evaluate our approach formally and regarding performance. Our work shows that despite different assumptions of this environment in contrast to the traditional environment of workflow management systems, the Guard-Stage-Milestone approach can be transferred and successfully applied on the web of Read-Write Linked Data.

1 Introduction

The environment of the web is finally at a stage where hypermedia agents could be applied [9]: We see that dynamic, open, and long-lived systems are commonplace on the web forming a highly distributed system. For example, microservices [31] build on the web architecture and provide fine-grained read-write access to business functions. Moreover, Internet of Things devices are increasingly equipped with web interfaces, see, for example, the W3C's Web of Things effort.[1] Furthermore, users' awareness for privacy issues leads to the decentralisation of social networks from monolithic silos to community- or user-hosted systems like SoLiD,[2] which builds on the web architecture. The web architecture offers REST [14], or its implementation HTTP,[3] as a uniform way for system interaction, and RDF[4] as uniform way for knowledge representation, where we can employ semantic reasoning to integrate data. To facilitate software agents in this environment called Read-Write Linked Data [4], we need to embrace the web architecture and find a suitable way to specify behaviour. As according to REST, the exchange of state information is in the focus on the web, we want to investigate a data-driven approach for specifying behaviour. Moreover, data-driven approaches to workflow modelling can be both intuitive and actionable, and hence are suited to a wide range of audiences with different experience with information technologies [20]. Hence, we want to tackle the research question of *how to specify and execute agent behaviour in the environment of Read-Write Linked Data in a data-driven fashion?*

Echoing the said application areas for web technologies, we envision our approach to be useful to define software agents that orchestrate services in microservice deployments, act as assistance systems in Internet of Things or Cyber-Physical Systems deployments, or manage the lifecycle of personal data. Deployments in which REST, semantic technologies, and some notion of behaviour (e.g. flow-driven workflows) play a role can be found in various industries, for academic descriptions see, for example, [6]—automotive,

[1] https://www.w3.org/2016/07/wot-ig-charter.html.

✉ Tobias Käfer
tobias.kaefer@kit.edu

Benjamin Jochum
uzebb@student.kit.edu

Nico Aßfalg
uberq@student.kit.edu

Leonard Nürnberg
ujeng@student.kit.edu

[1] Institute AIFB, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

[2] https://solid.mit.edu/.

[3] http://tools.ietf.org/rfc/rfc7230.txt.

[4] http://www.w3.org/TR/rdf11-concepts/.

Published online: 26 April 2021

[25]—aviation, [10]—manufacturing, and [26]—Internet of Things. Having been involved in the development of some of these deployments, we see the need for an approach to specify behaviour that embraces the data-driven nature of the environment.

As the environment determines why different workflow approaches are used in different circumstances [13], we need to look at the particularities of Read-Write Linked Data, whose basic assumptions are fundamentally different from traditional environments where workflow technologies are applied, e.g. databases:

The absence of events in HTTP Of the many HTTP methods, there is no method to subscribe to events. Hence, for our Read-Write-Linked-Data native approach, we rely on state data and resort to polling to get informed about changes in the environment.

Reasoning and querying under the OWA While in databases, typically the closed-world assumption is made, i.e. we conclude from the absence of information that it is false, reasoning in ontology languages for RDF is typically based on the open-world assumption (OWA). Hence, we have to explicitly model all options.

Mitigation strategies would introduce complexity or restrict the generality of our approach: **The absence of events** could, for example, be addressed by (1) generating events from differences between state snapshots and to process these events, which would add unnecessary complexity if we can do without; (2) assuming server implementations that implement change events using the WebSocket protocol, which would restrict the generality of our approach and, for uniform processing, would require clear message semantics, which, in contrast to HTTP, event-based systems do not have [14][5]. **The open-world assumption** could, for example, be addressed by introducing assumptions such as negation-as-failure once a certain completeness class [18] has been reached, which also would add complexity.

In contrast to previous work by Pautasso, who presented an approach to retrofit REST into the BPEL approach to workflow modelling in [32], our approach rather retrofits a workflow modelling approach into REST, here GSM. In previous work, we defined ASM4LD, a model of computation for the environment of Read-Write Linked Data [23], which allows for rule-based specification of agent behaviour. Based on this model of computation, we provided an approach to specify flow-driven workflows [24]. In contrast, we present a data-driven approach in this paper. The Guard-Stage-Milestone (GSM) approach, which serves as basis for our work, has first been presented in [20]. While GSM builds on events sent to a database, which holds the information model consisting in status and data attributes, in our approach, distributed components with web interfaces that supply state information hold the information model.

In [22], we described a previous version of this paper, which we presented at the 3rd International Workshop in Artificial Intelligence for Business Process Management (AI4BPM) at the 17th International Conference on Business Process Management (BPM). In this paper, we extended aforementioned work by providing a description on how we process queries with a theoretical insight into the required rule language, an extended formal evaluation from which we derive modelling requirements, a performance evaluation, updates to the operational semantics, and an extended discussion of related work.

Our approach consists in two main parts:

GSM Ontology We present an ontology to specify GSM workflows and instances in the ontology language RDFS. Using this ontology, we can specify, reason over, and query workflow models and instances at run-time.

Operational Semantics We present ASM4LD rules to execute workflow instances specified using our GSM ontology. To this end, we build on a Linked Data Platform container[6], i.e. a writable RESTful RDF data store, to store the status attributes, i.e. workflow instances in our ontology.

The paper is structured as follows: In Sect. 2, we survey related work. Next, in Sect. 3, we give basic definitions on which we build our approach with the help of examples. In Sect. 4, we present our main contributions: the ontology and the operational semantics, together with the modelling requirements. Then, in Sect. 5, we present how we process queries on the information model using rules. Next, we evaluate our approach in Sect. 6 regarding correctness and performance. Last, in Sect. 7, we conclude.

2 Related Work

As we work in the intersection of data-driven workflow management, knowledge representation using semantic technologies, and systems built using the web, we survey related work in those fields and intersections.

[5] Note that a client's (polling, state-based) application logic can, without changes, benefit from HTTP/2 server push (events): such specific events have been standardised to allow a server to update a client's cached state representations.

[6] http://www.w3.org/TR/ldp/.

2.1 Data in Workflow Management

Execution of workflows is typically driven by either flow or data. Flow-driven approaches include the popular BPMN language[7]. If we want to make use of data during workflow execution, we can either extend a flow-driven approach to make it data-aware, or we use an approach where data are a first-class citizen. In this paper, we want to investigate an entirely data-driven approach, but we give a brief overview on works in the former category first.

Semantics for the flow aspects in flow-based approaches are typically given using Petri nets [36] in an event-driven fashion. For instance, Dijkman et al. [12] provide such semantics for BPMN. Thus, approaches that make flow-based approaches more data-aware on a formal level are often based on Petri nets: Such approaches include [30,37]. They take an extension of Petri nets for the dynamic part of the system, and a formal view on data bases for the data aspect. The gap between flow-based approaches and data-based approaches becomes obvious in [38], which presents an approach for deriving GSM models from Petri nets. In the conversion, not all conditions in the GSM model can be filled automatically: Those sentries that build on data require additional sources for input, e.g. a human expert. We see that here, the data aspects need to be added to the model when talking about a flow-driven model from a data-centric perspective, which provides additional motivation for us to investigate entirely data-driven approaches.

Semantics for data-driven workflow languages are often specified using Event-Condition-Action (ECA) rules to be executed on databases, e.g. see GSM [20] for an artefact-centric approach, and [8] for a flow-centric approach. Another data-centric approach is RESEDA [39], whose semantics have been given using the event-driven reactive paradigm. Our approach is built for the environment of REST, where there are no events that could inform the enactment of the workflow instance. However, we make use of the GSM workflow language and transfer it to the environment of Read-Write Linked Data by specifying semantics in Condition-Action rules.

Other approaches assume processes to be given as Condition-Action rules, noting that workflow languages can be layered on top, i.e. the following three approaches do not talk about the semantics of such languages. For instance, the Daphne approach [7] provides actions as abstraction on a SQL query or an external service call, both of which change the contents of a relational data base. With its roots in the formal investigations of data-centric dynamic systems [2], Daphne is closely related to [3], where a similar approach is investigated that works on a Description Logic knowledge base instead of a data base, and also provides actions as abstractions for changes. In contrast, the knowledge base in our approach is the web of Read-Write Linked Data, i.e. our approach is built for a distributed hypermedia setting, where REST calls are the means to enact change. In terms of expressivity, we use an ontology language that is not based on description logics but rather on RDFS, which is typically less expressive than members of the Description Logic family of languages [19]. However, the transition systems in those condition-action rule-based approaches can be related to ASM4LD [23], the Abstract State Machine [16] based model of computation we build our work on.

2.2 Workflows and Web Services

Workflows applied to the web have been investigated under the headline Web Services, which produced a set of WS-* standards, most prominently SOAP[8] and WSDL[9], which allow for composition of services, i.e. arbitrary functions called via HTTP as transport protocol, using the flow-based BPEL[10] workflow language. Semantic descriptions, e.g. in OWL-S [28] and WSMO [17] of those functions, should allow for automated composition. In contrast, REST [14] constrains this set of arbitrary functions and emphasises the processing of state information instead of return values of function calls [35,42]. Thus, extensions for BPEL have been proposed to include RESTful services [32,33]. We built our approach with the same aim in mind, i.e. to make use of RESTful services in processes. However, as in REST, a web is built around resources and data about their state, we want to use a data-centric approach and exploit the semantics of the constrained set of REST operations.

2.3 Ontologies for Processes on the Web

Ontologies for processes have been proposed in numerous occasions. For instance, OWL-S [29] contains a process language, scientific workflows are often described using an ontology [15,41], and publicly funded research projects including Super[11] and Adaptive Services Grid[12] defined process ontologies. In contrast to our approach, those approaches have been built for flow-based processes, the function-call style web service interaction, cannot describe process instances, and do not come with operational semantics.

In previous work, we developed the WiLD ontology to describe flow-based workflow models and instances together

[7] http://www.omg.org/spec/BPMN/.

[8] http://www.w3.org/TR/soap12-part1/.

[9] http://www.w3.org/TR/wsdl/.

[10] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[11] http://ip-super.org/ is defunct, but the content is available at http://projects.kmi.open.ac.uk/super/.

[12] http://www.asg-platform.org/ is defunct, the page is available in the Web Archive.

with operational semantics in to specify and execute workflows on Read-Write Linked Data [24]. In contrast, in this paper, we investigate data-centric behaviour descriptions.

2.4 Information and Processes in Information Systems

In the broader scope of object-aware business process systems, [27] have developed 20 requirements around data, activities, processes, user integration and monitoring. Based on these requirements, they evaluate imperative, declarative, and data-driven approaches to workflow management. As we work with Read-Write Linked Data, our approach can contribute to filling the gaps left by other approaches: to fulfil requirement R1 (data integration) and R2 (access to data). As our approach is based on GSM, we of course inherit their requirement fulfilment in other areas including R10 (object behaviour), R13 (flexible process execution), R14 (re-execution of activities) and R15 (explicit user decisions).

3 Preliminaries

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

3.1 Uniform Resource Identifier (URI)

On the web, we identify things (called resources on the web) using URIs[13]. URIs are character strings consisting in a scheme, and a scheme-specific part, separated by the colon character. In this paper, we only consider the schemes of HTTP. For instance, consider the following URI that identifies the relation to assign a thing to a class:

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

In this URI, the URI scheme is "http", which refers to the HTTP protocol, which we discuss in the next section. Note that we use the term URIs when we actually mean their internationalised form, IRIs.[14] We do so, as the former is the more popular term and the former RFC provides more information about the meaning and the composition of URIs/IRIs than the RFC about the latter.

3.2 The Hypertext Transfer Protocol (HTTP)

HTTP[3] is a stateless application-level protocol, where clients and servers exchange request/response message pairs about resources that are identified using URIs on the server. Requests are typed, and the type (i.e. the HTTP method) determines the semantics of both the request and the optional message body. We make extensive use of the GET request to request a representation of a resource, the PUT request to overwrite the representation of a resource, and the POST request to append to an existing collection resource. Notably, the constrained set of HTTP methods does not include a way to subscribe to events. Therefore, polling, i.e. the repeated retrieval of state information, is the way to get informed about changes to resource state.

3.3 The Resource Description Framework (RDF)

RDF[4] is a graph-based data model for representing and exchanging data based on logical knowledge representation. In RDF, we represent data as triples that follow the form:

(*subject*, *predicate*, *object*)

Such a triple defines a relation of type *predicate* between graph nodes *subject* and *object*. Multiple triples form an RDF graph, and the triples can be regarded as defining the edges that connect the vertices of the graph. Things in RDF are identified globally using URIs, or document-locally using so-called blank nodes. Literals can be used to express values. RDF can be serialised in different formats. Those include Turtle[15], which is probably the easiest for humans to read, RDF/XML[16], which has been one of the first formats specified and works well in XML-based environments, and JSON-LD[17], which works well in JSON-based environments. In this paper, we use the following notation for RDF: As triples encode binary predicates, we write

$rdf{:}type(:active, :State)$

to mean the following triple in Turtle notation:

```
:active rdf:type :State .
```

Here, `:active` is the subject, `rdf:type` is the predicate, and `:State` is the object. Triples are asserted conjunctively. Note the use of URI abbreviations using the CURIE syntax[18], where a colon separates the abbreviating and possibly empty prefix from the local name, here `rdf` is short for `http://www.w3.org/1999/02/22-rdf-`

[13] http://tools.ietf.org/rfc/rfc3986.txt.

[14] http://tools.ietf.org/rfc/rfc3987.txt.

[15] http://www.w3.org/TR/turtle/.

[16] http://www.w3.org/TR/rdf-syntax-grammar/.

[17] http://www.w3.org/TR/json-ld/.

[18] http://www.w3.org/TR/curie/.

`syntax-ns#`[19]. For the special case of class assignments, we use unary predicates. That is, in our notation, above triple becomes:

$:State(:active)$

For boolean constants, we use typewriter font, e.g. `true`.

3.4 The SPARQL Protocol and Query Language

SPARQL[20] is a protocol and a query language for RDF. SPARQL queries can have different forms, most prominently `SELECT` queries. We use `ASK` queries in this paper. The basic construct is a SPARQL query is a so-called graph pattern, where variables are allowed in all positions of a triple. For instance, a query to select all states could look as follows (assuming appropriate prefix definitions):

```
SELECT ?s WHERE { ?s rdf:type :State . }
```

The `ASK` query

```
ASK WHERE { ?s rdf:type :State . }
```

returns `true` if the `SELECT` query above has a non-empty result.

3.5 Ontologies, Reasoning, and Rules

Besides encoding knowledge in a graph, RDF is the basis for a set of Knowledge Representation technologies including the languages RDFS[21] and OWL[22] to express ontologies. By and large, RDFS is less expressive than most members of the OWL family, which comes with computational benefits [19]. For instance, RDFS entailment can be implemented using monotonous deduction rules. A rule language in the context of RDF is Notation3[23]. For example, if we know that:

```
:active a :NonFailureState .
:NonFailureState rdfs:subClassOf :State .
```

or in the notation of our paper:

$$:NonFailureState(:active) \\ \wedge rdfs{:}subClassOf(:NonFailureState, :State)$$

the rule (with terms without colons being variables)

$$rdfs{:}subClassOf(x, y) \wedge x(z) \rightarrow y(z)$$

allows us to entail the triple from the RDF example. We call the part before the arrow antecedent or body and the part after the arrow consequent or head.

In this paper, we additionally use a special kind of rules: request rules, where the consequent is an HTTP request to be executed. For instance:

$$:hasState(s, :active) \rightarrow \mathrm{PUT}(s, :hasState(s, :done))$$

would mean that a `PUT` request should be sent to all s that have the state *:active* with the payload that this s shall have the state *:done*.

3.6 Abstract State Machines for Linked Data (ASM4LD)

ASM4LD is an Abstract State Machine-based operational semantics given to Notation3 [23]. In ASM4LD, we can encode two types of rules: derivation rules (to derive new knowledge) and request rules (which cause HTTP requests). Moreover, ASM4LD supports RDF assertions. In [23], we derived the operational semantics based on the semantics of HTTP requests, first-order logic, and Abstract State Machines. The operational semantics can be summarised in four steps to be executed in a loop, thus implementing polling:

1. Initially, set the working memory to be empty.
2. Add the assertions to the working memory.
3. Until no further data can be retrieved from URIs and no further data can be deducted, evaluate on the working memory:

(a) Request rules from which HTTP-GET requests follow. For the rules whose condition holds, make the HTTP requests add the data from the responses to the working memory.

(b) Derivation rules. Add the thus derived data to the working memory.
This way, we (a) obtain and (b) reason on data about the world state.

4. Evaluate all other request rules on the working memory, i.e. those rules from which PUT/POST/DELETE requests follow. Make the corresponding HTTP requests.
This way, we enact changes on the world's state.

In this paper, we use ASM4LD as formal basis to give operational semantics to our Guard-Stage-Milestone ontology.

[19] The example also uses the empty prefix, which shall denote http://purl.org/gsm/vocab# in this paper. We refer to http://prefix.cc/ for other abbreviations.

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

[21] http://www.w3.org/TR/rdf-schema/.

[22] http://www.w3.org/TR/owl2-overview/.

[23] http://www.w3.org/TeamSubmission/n3/.

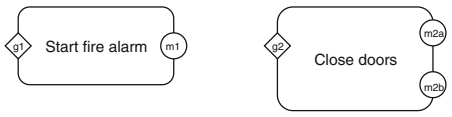


Fig. 1 A small example of two stages with associated guards and milestones

3.7 The Guard-Stage-Milestone Approach

The Guard-Stage-Milestone approach is an artefact-centric workflow meta-model, presented in [20]. The key modelling elements for Guard-Stage-Milestone workflows are the following: The **Information Model** contains all relevant information for a workflow instance: *data attributes* maintain information about the system controlled by the workflow instance, and *status attributes* maintain control information such as how far the execution has already progressed. **Stages** can contain a task (i.e. the actual activity, an unit of work to be done by a human or machine) and may be nested. **Guards** control whether a stage gets activated, i.e. the activity may execute. The conditions of a guard are given as sentries. **Sentries** are boolean expressions in a condition language. They come in the form Event-Condition (**on** <event> **if** <condition>). Here, events may be incoming from the system, or be changes to status attributes. **Milestones** are objectives that can be achieved during execution, and are represented using boolean values. Milestones have achieving and invalidating sentries associated: If an achieving sentry is evaluated to *true*, the milestone is set to *achieved*. An invalidating sentry can set a milestone back to *unachieved*.

An example can be found in Fig. 1.

The example is set in an Internet of Things scenario and shows two stages "Start Fire Alarm" and "Close doors", whose activation is controlled using guards. Those stages have milestones associated that can be validated and invalidated. The informal definition of those guards and milestones can be found in Fig. 2.

To specify the operational semantics, Hull et al. [20] provides a set of six PAC rules. PAC rules are a variation of Event-Condition-Action rules and are described by a prerequisite, antecedent, and consequent, respectively. Both prerequisite and antecedent range over the entire information model, and the consequent is an update to the status attributes. The rules can be subdivided into two categories: **explicit rules**, which accomplish the actual progress in a workflow instance, and **invariant preserving rules**, which perform "housekeeping" by, e.g. deactivating child stages if the parent has been deactivated. The operational semantics of GSM are defined in an event-triggered fashion. There are three types of **Events**: *Outgoing Events* for Task Invocation, sent from stages, *Incoming Events* sent from the environment, e.g. one-way messages and task terminations, and *Status*

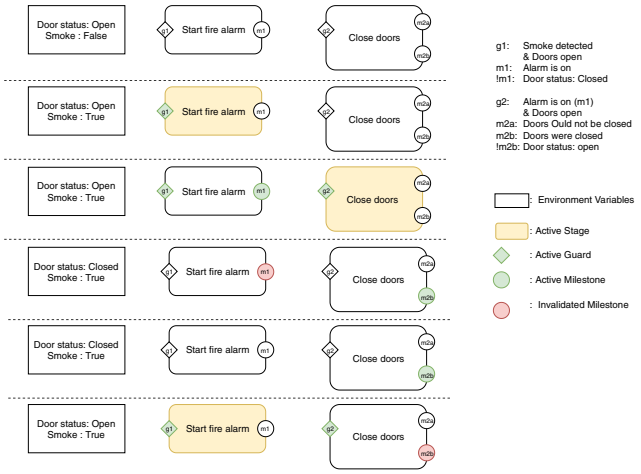


Fig. 2 Example of a workflow execution. Time progresses from top to bottom

changes who fire when a milestone or stage changes its state. If an event triggers the execution, all that follows from the rules gets incorporated into the information model. This full incorporation is called a Business Step (B-Step).

We illustrate the operational semantics using an example. It demonstrates a workflow of a fire alarm process in a public building. Figure 2 shows how the workflow execution proceeds triggered by changes in the information model.

Every line represents a B-Step.

- Line 2 Once smoke has been detected, the sentry of $g1$ becomes true, which leads to $g1$ being active and thus the execution of the left stage.
- Line 3 Guard $g1$ is still active. As the fire alarm has been started, the achieving sentry of milestone $m1$ is true and thus the milestone is set to achieved. In consequence, guard $g2$ becomes active, which triggers the execution of the right stage.
- Line 4 As the invalidating sentry of $m1$ becomes true, i.e. the fire alarm stops when all doors are closed, the dependent guard $g2$ becomes inactive as well. This does not affect the state of milestone $m2b$.
- Line 5 After $m1$ has been invalidated, the smoke cannot further circulate through open doors.
- Line 6 At some point after all doors had already been closed, somebody re-opens a door. This triggers the invalidation of the corresponding milestone $m2b$ (all doors are closed). As a consequence, the left stage is re-triggered.

We see that changes in the information model trigger in more changes in the information model.

4 Proposed Approach

To transfer the GSM approach to the environment of Read-Write Linked Data, we need to transfer both the modelling and the operational semantics to this environment. Our approach consists therefore in an ontology to model GSM workflows and instances (the latter corresponding to the status attributes), and in operational semantics in rules with ASM4LD semantics, which interpret those workflows and instances. The rules can be directly deployed on a corresponding interpreter.

To put our approach into practice, a workflow model has to be modelled using our ontology and to be made available as Linked Data. A workflow instance has to be defined using our ontology, with a link to a workflow model, and deployed as Read-Write Linked Data in a Linked Data Platform Container. Then, an interpreter—with the operational semantics rules deployed– has to be pointed to the workflow instance and can execute the instance according to the model. Necessary instances for the elements within the workflow model are created, also as Linked Data in the said container, by the interpreter using specialised set-up rules. The interpreter then changes the state of the instances according to the GSM lifecycle. As the rule language supports both, derivation rules and request rules, we can deploy derivation rules to implement the reasoning that is necessary to fulfil the semantics of ontology languages (e.g. RDFS) next to the rules required for the operational semantics. Thus, we can execute the workflow and perform semantic data integration at the same time.

We first present the ontology (Sect. 4.1) and then the operational semantics (Sect. 4.2). For the presentation of the operational semantics, we use the rule syntax described in Sect. 3. Last (Sect. 4.3), we present modelling requirements that need to be fulfilled for a correct workflow execution.

4.1 Ontology for Modelling Entities

We built an ontology[24] to describe the core modelling primitives from [20]. We depict the ontology in Fig. 3. We stay as closely as possible to their definitions and divert only if demanded by the environment of Read-Write Linked Data: **Tasks** are HTTP requests as atomic activities. **Sentries** contain a SPARQL `ASK` query in SPIN notation.[25] Correspondingly, we use SPARQL's `true` boolean query result (we cannot use false, see Sect. 1) with sentries and guards. We introduce the class **State** of all states to, e.g. model the states of a stage: active and inactive.

In our approach, the information model is not contained in a database, but is spread over Read-Write Linked Data resources. Hence, we do not maintain the data attributes ourselves: We do not store information about the system we control, but instead, we retrieve the system state live in RDF over HTTP from the system itself. However, we do maintain the status attributes, for uniform access in RDF over HTTP. Thus, from now on we call all information regarding the state of the workflow *status information*. All other information about the system under control, and other relevant information from the environment, e.g. external services, we call *environment information*.

4.2 Operational Semantics

The following operational semantics are based on the PAC-rule-based semantics from [11]. We distinguish between set-up and flow conserving (FC) rules. Our rules can be found online[26] in N3 notation. For the sake of the example, we assume all status information to reside in a fictitious collection resource at `http://ldpc.example/`. To improve legibility, we sometimes use "…" to denote when the values of other predicates stay the same.

Instance Set-Up Rules

The basic condition for all setup rules is:

$$CS := :StageInstance(i) \wedge :isInstanceOf(i, s) \\ \wedge :hasState(i, :uninitialized)$$

Workflow The setup can be requested by publishing an uninitialised instance:

$$CS \longrightarrow \text{PUT}(i, :ArtifactInstance(i) \\ \wedge \ldots \\ \wedge :hasState(i, :uninitialized))$$

Stage Then, resources are created for all the model's substages, linked to their counterpart in the model, and with state inactive.

$$CS \wedge :hasDescendantStage(s, s_{child}) \\ \longrightarrow \text{POST}(\texttt{http://ldpc.example/}, \\ :isInstanceOf(\texttt{new}, s_{child}) \wedge :inArtifactInstance(\texttt{new}, i) \\ \wedge :hasState(\texttt{new}, :inactive))$$

Milestone Also, resources are created for all the model's milestones, linked to their counterpart in the model, and with state unachieved.

[24] http://purl.org/gsm/vocab.

[25] http://spinrdf.org/.

[26] http://purl.org/gsm/semantics.

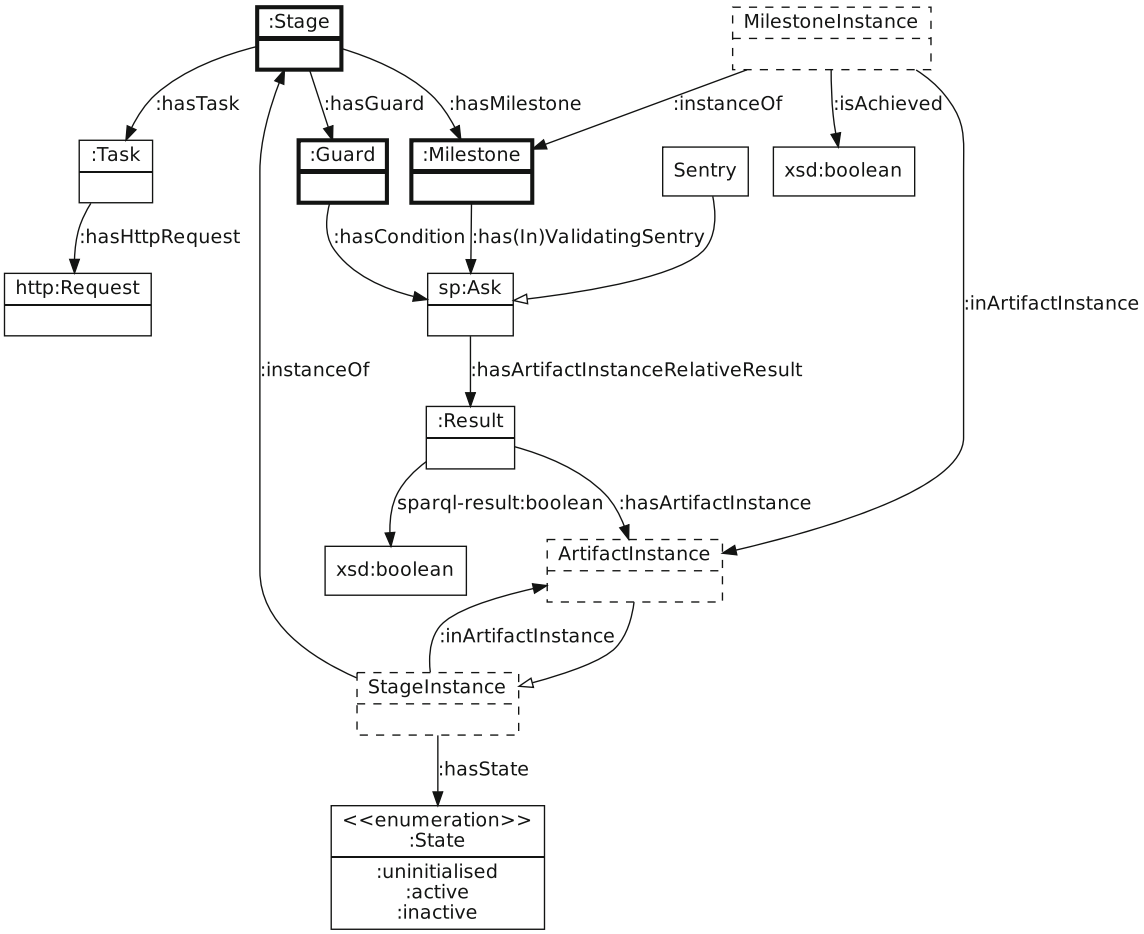


Fig. 3 Our ontology as UML class diagram with the following correspondence to RDF Schema: UML class depicts RDFS class; UML associations depict domain and range of RDF properties. UML inheritance depicts RDFS subclass. The core classes to describe models are depicted in bold; the core classes during execution are depicted dashed

$$
\begin{aligned}
&CS \wedge :hasDescendantStage(s, s_{child}) \\
&\wedge :hasMilestone(s_{child}, m) \\
&\longrightarrow \text{POST}(http://ldpc.example/, \\
&:isInstanceOf(\texttt{new}, m) \wedge :inArtifactInstance(\texttt{new}, i) \\
&\wedge :isAchieved(\texttt{new}, \texttt{false}))
\end{aligned}
$$

Flow-Conserving Rules

All FCRs share the following conditions, which we abbreviate with FC:

$$
\begin{aligned}
&FC := :ArtifactInstance(S^I) \wedge :Stage(s^M) \\
&\wedge :StageInstance(s^I) \wedge :isInstanceOf(s^I, s^M) \\
&\wedge :inArtifactInstance(s^I, S^I) \\
&\wedge :hasState(S^I, :active)
\end{aligned}
$$

For better readability when writing about sentries, we define the term *:hasBooleanResult* as abbreviation:

$$
\begin{aligned}
&:hasBooleanResult(c, \texttt{true}) \\
&:= :hasInstanceRelativeResult(c, r) \\
&\wedge :hasArtifactInstance(c, S^I) \\
&\wedge sparql\text{-}results{:}boolean(r, \texttt{true})
\end{aligned}
$$

FCR-1 For inactive stages, we check the guards. If a guard holds, the state of the stage is set to active and the task of the stage is executed.

$$
\begin{aligned}
&FC \wedge :hasGuard(s^M, g) \wedge :hasHttpRequest(s^M, \text{req}) \\
&\wedge :allAncestorsActive(s^I, \texttt{true}) \\
&\wedge :hasState(s^I, :inactive) \wedge :hasCondition(g, c) \\
&\wedge :hasBooleanResult(c, \texttt{true}) \\
&\longrightarrow \text{PUT}(s^I, \ldots :hasState(s^I, :active)) \\
&\longrightarrow \text{req}(\cdot, \cdot)
\end{aligned}
$$

FCR-2 If an active stage's validating sentry holds, set the stage and its children to inactive, and the stage's milestone to achieved.

$$
\begin{aligned}
&FC \wedge :hasState(s^I, active) \wedge :hasMilestone(s^M, m^M)\\
&\wedge :hasValidatingSentry(m^M, c^a) \wedge :isInstanceOf(m^I, m^M)\\
&\wedge :hasBooleanResult(c, \texttt{true})\\
&\wedge :hasDescendantStage(s^M, d^M) \wedge :isInstanceOf(d^I, d^M)\\
&\longrightarrow \text{PUT}(m^I, \ldots :isAchieved(m^I, \texttt{true}))\\
&\longrightarrow \text{PUT}(s^I, \ldots :hasState(s^I, :inactive))\\
&\longrightarrow \text{PUT}(d^I, \ldots :hasState(d^I, :inactive))
\end{aligned}
$$

FCR-3 When an achieved milestone's invalidating sentry of a completed stage becomes true, the milestone is set to unachieved.

$$
\begin{aligned}
&FC \wedge :hasMilestone(s^M, m^M) \wedge :isInstanceOf(m^I, m^M)\\
&\wedge :isAchieved(m^I, \texttt{true}) \wedge :hasInvalidatingSentry(m^M, c)\\
&\wedge :hasBooleanResult(c, \texttt{true})\\
&\longrightarrow \text{PUT}(m^I, \ldots :isAchieved(m^I, \texttt{false}))
\end{aligned}
$$

FCR-4 Upon activation of a stage, set all of the stage's and its descendant stages' milestones to unachieved.

$$
\begin{aligned}
&FC \wedge :hasGuard(s^M, g) \wedge :allAncestorsActive(s^I, \texttt{true})\\
&\wedge :hasState(s^I, :inactive)\\
&\wedge :hasCondition(g, c) \wedge :hasBooleanResult(c, \texttt{true})\\
&\wedge :hasMilestone(s^M, m^M) \wedge :isInstanceOf(m^I, m^M)\\
&\wedge :isAchieved(m^I, true) \wedge :hasDescendantStage(s^M, d^M)\\
&:hasMilestone(d^M, m_d^M) \wedge :isInstanceOf(m_d^I, m_d^M)\\
&\longrightarrow \text{PUT}(m^I, \ldots :isAchieved(m^I, \texttt{false}))\\
&\longrightarrow \text{PUT}(m_d^I, \ldots :isAchieved(m_d^I, \texttt{false}))
\end{aligned}
$$

4.3 Requirements to Workflow Modelling

We now add two conditions a workflow needs to fulfil. The disjointness conditions specify sets of sentries that need to be disjoint. If the workflow modeller violates one of them, a correct workflow execution cannot be guaranteed. In the following, "¬" denotes the logical negation. The reader can find the rationale behind those requirements in Section 6.2.1.5, where we evaluate our approach regarding correctness.

4.3.1 Disjointness of Achieving and Invalidating Sentry

For each milestone, the invalidating and achieving sentry must be disjoint. More formally:

$$
\begin{aligned}
&FC \wedge :hasMilestone(s^M, m^M)\\
&\wedge :hasInvalidatingSentry(m^M, c_I)\\
&\wedge :hasAchievingSentry(m^M, c_A)
\end{aligned}
$$

the workflow modeller must ensure that $\neg(c_A \wedge c_I)$ always holds.

4.3.2 Disjointness of Guards and Milestones

For each milestone, the guard of a stage and must be disjoint with the achieving sentries of all milestones of its parent stages. More formally:

$$
\begin{aligned}
&FC \wedge :hasDescendantStage(s^M, d^M)\\
&\wedge :hasMilestone(d^M, m_d^M)\\
&\wedge :hasAchievingSentry(m_d^M, c_A)\\
&\wedge :hasGuard(s^M, g)\\
&\wedge :hasCondition(g, c_g)
\end{aligned}
$$

the workflow modeller must ensure that $\neg(c_A \wedge c_g)$ always holds.

5 Rule-Based Querying of Data and Status Attributes

In our approach, we use SPARQL ASK queries for sentries. As the rule language we employ does not have special features to incorporate query results into conditions, we have to use the rule language to do query processing. We build on the SPIN[27] notation to describe SPARQL queries in RDF.

For queries on the data attributes, SPARQL features are sufficient. Consider the following query that could, e.g. be a part of the sentry of g1 in Fig. 1:

```
PREFIX smarthome: <http://iot.example/vocab#> .
ASK
WHERE {
<http://iot-house.example/front#door>
   a smarthome:Door ;
   smarthome:isOpen true .
}
```

[27] http://spinrdf.org/.

If we identify the query using q, we can write (imagine bx_y as blank nodes):

$$
\begin{aligned}
&\textit{sp:Ask}(q) \wedge \textit{sp:where}(q, bl_1) \wedge \textit{rdf:first}(bl_1, bt_1) \\
&\wedge \textit{sp:subject}(bt_1, \textit{http://iot-house.example/front\#door}) \\
&\wedge \textit{sp:predicate}(bt_1, \textit{rdf:type}) \\
&\wedge \textit{sp:object}(bt_1, \textit{smarthome:Door}) \\
&\wedge \textit{rdf:rest}(bl_1, bl_2) \wedge \textit{rdf:first}(bl_2, bt_2) \\
&\wedge \textit{sp:subject}(bt_2, \textit{http://iot-house.example/front\#door}) \\
&\wedge \textit{sp:predicate}(bt_3, \textit{smarthome:isOpen}) \\
&\wedge \textit{sp:object}(bt_3, \texttt{true}) \\
&\wedge \textit{rdf:rest}(bl_2, \textit{rdf:nil})
\end{aligned}
$$

SPIN makes use of reification, similar to RDF reification,[28] but with own RDF terms, to describe the triple patterns in the query. Using rules of the following form (with tp, s, p, o being variables), we can annotate the triple patterns using deductions that hold if a triple pattern is fulfilled:

$$
\begin{aligned}
&\textit{sp:subject}(tp, s) \wedge \textit{sp:predicate}(tp, p) \wedge \textit{sp:object}(tp, o) \\
&\wedge p(s, o) \rightarrow \textit{sparql-results:boolean}(tp, \texttt{true})
\end{aligned}
$$

Using further deductions, we can check whether all triple patterns in a query in SPIN notation are fulfilled, and eventually yield triples that serve in the conditions of the operational semantics, e.g. see the rule FCR-2 in Sect. 4.2.

For queries on the status attributes, this technique to annotate existing resources of triple patterns is not sufficient: when giving queries to be evaluated on the status attributes, we need a notion of the current artefact instance, and thus, we need new resources (and new terms in the vocabulary) using which we can annotate the triple patterns with the artefact instances in which they hold. For example, a guard g that requires a milestone m_1 to be achieved should, first, only look in the same artefact instance to check whether m_1 has been achieved, and should second, be defined at workflow design time, when the exact URI of the milestone instance for m_1 is not known yet. For the new resources, we need to extend the expressivity of the rule language: For the considerations so far, a rule language without existentials in the head was sufficient. As we need to create new resources in the deductions to annotate triple patterns relative to artefact instances, we need existentials in the head.

To annotate sentries relative to artefact instances, we thus deduce blank nodes for each sentry in each active artefact instance (bn as blank node):

$$
\begin{aligned}
&\textit{:ArtifactInstance}(a) \wedge \textit{:hasState}(a, \textit{:active}) \\
&\wedge \textit{:inArtifactInstance}(si, a) \wedge \textit{:isInstanceOf}(si, sm) \\
&\wedge \textit{:hasGuard}(sm, g) \wedge \textit{:hasCondition}(g, s) \\
&\rightarrow \textit{:hasArtifactInstanceRelativeResult}(s, bn) \\
&\wedge \textit{:hasArtifactInstance}(bn, a)
\end{aligned}
$$

Now, we can follow the same pattern when annotating triple patterns.

Last, we need a way to relate the SPARQL ASK query to the artefact instance. To this end, we introduce special triple patterns to be used in SPARQL ASK queries in SPIN notation. Those triple patterns have the form:

$$
\begin{aligned}
&\textit{:inArtifactInstance}(bt_1, \textit{:thisArtifactInstance}) \\
&\wedge \textit{:hasAchievedMilestone}(bt_1, m_1)
\end{aligned}
$$

where bt_1 is a triple pattern, similar as in the presented SPARQL query in SPIN notation, m_1 is the URI of the milestone that the sentry is supposed to consider, and *:thisArtifactInstance* is a magic term that represents the current artefact instance.

6 Evaluation

In this section, we evaluate our approach. We show the correctness of our approach using theoretical considerations and proofs in 6.2 after introducing some notation 6.1. Then, we briefly report on the applicability of our approach in Sect. 6.3. Last, we provide a performance evaluation in Sect. 6.4.

6.1 Notation

To improve readability, we introduce a new notation and simplify the concept of models and its instances. In the previous chapter, we made a difference between stage instances (s^I) and stage models (s^M). The FCRs, however, only consider one-stage instance of one-stage model; hence, we can simplify the notation by conflating the two. Therefore, in the new notation, we choose definitions that heavily build on the instances only. We also compact the notation by defining sets that contain all resources that fulfil certain conditions from the rules in Sect. 4. Table 1 contains the relevant definitions of sets and functions we use in the following.

On top of the sets and function that describe one instant in time, we also need to define the progression of time. To describe the progressing execution of a workflow, we look at it as a series of *snapshots* of the workflow. A snapshot contains the assignments of all status information variables of a workflow. The sequential of snapshots then describes the stepwise execution. To link a snapshot to its successor, we define a state transition function. The set of all possible

[28] https://www.w3.org/TR/rdf11-mt/#reification.

Table 1 Notation used in the proofs

Symbol	Formal definition	Explanation
S	$\{s^I \mid FC\}$	The set of stages
G	$\{g \mid FC \wedge :hasGuard(s^M, g)\}$	The set of guards
G_s	$\{g_s \in G \mid FC \wedge :hasGuard(s^M, g_s) \wedge :isInstanceOf(s, s^M)\}, s \in S$	The guards of stage s
$\mathcal{M}$	$\{m^I \mid FC \wedge :hasMilestone(s^M, m^M) \wedge :isInstanceOf(m^I, m^M)\}$	The set of milestones
$\mathcal{M}_s$	$\{m^I \in \mathcal{M} \mid FC \wedge :hasMilestone(s^M, m^M) \wedge :isInstanceOf(m^I, m^M) \wedge :isInstanceOf(s, s^M)\}, s \in S$	The milestones of stage s
D_s	$\{s_d \in S \mid FC \wedge :hasDescendantStage(s^M, s_d^M) \wedge :isInstanceOf(s_d, s_d^M) \wedge :isInstanceOf(s, s^M)\}, s \in S$	The descendants of stage s
A_s	$\{s_a \in S \mid FC \wedge :hasAncestorStage(s^M, s_a^M) \wedge :isInstanceOf(s_a, s_a^M) \wedge :isInstanceOf(s, s^M)\}, s \in S$	The ancestors of stage s
Φ	$\{c \mid (FC \wedge :hasGuard(s^M, g) \wedge :hasCondition(g, c)) \vee (FC \wedge :hasMilestoneModel(s^M, m^M) \wedge (:hasAchievingSentry(m^M, c) \vee :hasInvalidatingSentry(m^M, c)))\}$	The set of sentries
ϕ	$\phi : \mathcal{M} \cup G \rightarrow \{true, false\}$	The result of a guard's (ϕ) or milestone's achieving (ϕ^+) or invalidating (ϕ^-) sentry in snapshot σ (ϕ_σ)
$\mathcal{S}$	$\{active, inactive, achieved, unachieved\}$	The set of states

All definitions relate to an instance I of a model M

snapshots is Σ, and the state transition function is

$$f : \Sigma \rightarrow \Sigma$$

Given a snapshot $\sigma \in \Sigma$, $f(\sigma)$ determines the subsequent state of the workflow, derived by applying the proposed rules. We abbreviate $f(\sigma)$ with σ'. $\sigma(x)$ denotes the value of status information variables (i.e. for a milestone or stage) x in snapshot σ. Correspondingly, ϕ_σ determines the results of a sentry in snapshot σ.

We now apply this notation to the FCRs. Table 2 lists the FCRs using the new notation. Next, we shortly provide a recap of the intuition behind each rule:

FCR-1 activates stages. It requires that a stage is inactive, the stage's ancestors are active, and the stage's guard is true.

FCR-2 sets milestones achieved and inactivates the corresponding stages as well as its substages. It requires a stage to be active and the achieving sentry of a milestone of the stage to be true.

FCR-3 invalidates milestones. It requires a milestone to be achieved and the milestone's invalidating sentry to be true.

FCR-4 invalidates all achieved milestones when a stage gets activated again. It requires all parent stages to be active, the stage itself to be inactive and the guard of the stage to be true.

We also reformulate the two disjointness conditions using the new notation:

Disjointness of Achieving and Invalidating Sentries. A workflow must be modelled in a way that the guard of a child stage is never satisfied at the same time when the stage's milestones are satisfied.

$$\forall s \in S, m_s \in \mathcal{M}_s : \neg(\phi_\sigma^+(m_s) \wedge \phi_\sigma^-(m_s)) \tag{1}$$

Disjointness of Guards and Milestones. The sentry of a guard and the achieving milestones of its parent must be disjoint.

$$\forall s, s_d \in S, m_s \in \mathcal{M}_s, s_d \in D_s, g_{s_d} \in G_{s_d} : \neg(\phi_\sigma(g_{s_d}) \wedge \phi_\sigma^+(m_s) \tag{2}$$

Table 2 Flow-conserving rules: conditions and actions

Rule	Condition	Action
FCR-1	$\forall s \in S, s_a \in A_s, g_s \in G_s : \sigma(s) = inactive \wedge \sigma(s_a) = active \wedge \phi_\sigma(g_s)$	$\sigma'(s) = active$
FCR-2	$\forall s \in S, s_d \in D_s, m_s \in \mathcal{M}_s : \sigma(s) = active \wedge \phi^+_\sigma(m_s)$	$\sigma'(m_s) = achieved \wedge \sigma'(s) = inactive$ $\wedge \sigma'(s_d) = inactive$
FCR-3	$\forall s \in S, m_s \in \mathcal{M}_s : \sigma(m_s) = achieved \wedge \phi^-_\sigma(m_s)$	$\sigma'(m_s) = unachieved$
FCR-4	$\forall s \in S, s_d \in D_s, s_a \in A_s, m_s \in \mathcal{M}_s, m_{s_d} \in \mathcal{M}_{s_d}, g_s \in G_s :$ $\sigma(s_a) = active \wedge \sigma(s) = inactive \wedge \phi_\sigma(g_s) \wedge \phi^+_\sigma(m_s)$	$\sigma'(m_s) = unachieved \wedge \sigma'(m_{s_d}) = unachieved$

6.2 Correctness

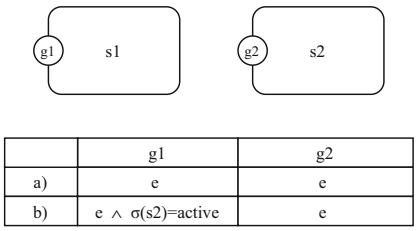
To show the correctness of our approach, we need to discuss two aspects in the light of the environment of Read-Write Linked Data: First, we need to have a look at the well-formedness criterion of [11] for workflows, which is a prerequisite for correct handling of workflows by their rules. Second, we need to investigate whether the invariants of [11] hold for our approach.

We therefore present in Sect. 6.2.1 the rationale behind the well-formedness criterion and show why the well-formedness criterion is not directly applicable to our approach. Then, we discuss the well-formedness criterion's implications and why they are valid in our approach nevertheless. Next, in Sect. 6.2.2, we present and formalise the invariants for our environment and show using mathematical induction that our FCRs do not violate those invariants.

6.2.1 Well-Formedness

In this section, we discuss the well-formedness criterion of [11]. The authors of said paper formulated this criterion to address non-intuitive behaviour resulting from different linearisations of the rules to be applied on incoming events. In contrast, our approach works without events and with parallel evaluation of rules. Therefore, we first explain the necessity for the well-formedness criterion in case of event data. Next, we show why we do not need to linearise the application of our rules on the condition part, as we only work with state data. However, new conflicts on the action part can arise from our parallel processing, which we discuss subsequently. We show that most conflicts cannot occur due to the nature of the conditions, and provide a rationale why the remaining conflicts are minor.

Well-formedness to Order M-Steps in B-Steps Remember that the execution in the GSM approach as described by [11] is event-triggered. A business step (B-Step) is the full incorporation of the implications of one event into the current state, i.e. the variable assignment on the data and status attributes. This incorporation includes the sequential evaluation of all sentries. If upon an event, a sentry's condition is met, its value is *true*. Corresponding updates to the variable assignment are enacted immediately. Thus, variables change over time and other sentry's conditions who would have held at the beginning of the incorporation, do not hold once it is this sentry's turn. This may lead to non-intuitive behaviour of the workflow, which [21] address using a well-formedness criterion.



	g1	g2
a)	e	e
b)	e ∧ σ(s2)=active	e

Fig. 4 Example for sensitivity to order of guard evaluation in a database environment. Case a) is not sensitive, while b) is sensitive

We illustrate the non-intuitive behaviour with an example in Fig. 4. Think of two guards g_1 and g_2 with corresponding stages s_1 and s_2. Guards g_1 and g_2 only depend on event e. Executing s_1 before s_2, or vice versa, leads to the same result. This is not always the case though. Now, g_1 depends on event e and on s_2 being active. Let event e fire. If g_1 is evaluated before g_2, s_1 does not become active, but s_2 becomes active. If g_2 is evaluated first, both become active. Intuitively, only s_2 should become active. Without a fixed order of execution, the incorporation of e is ambiguous.

To address this ambiguity, Hull et al. [21] introduce the micro-step (M-Step), into which a B-Step can be subdivided. Evaluating g_1 and g_2 each is M-Steps. A B-Step then consists of multiple M-steps. To handle the ordering of execution of M-steps [21] introduce the *dependency graph*. The dependency graph models dependencies between guards and milestones. A guard or a milestone, say g_1 or m_1, depends on another milestone m_2, if the truth of the sentry of g_1 or m_1 is dependent on m_2 being achieved or not. The dependency graph is a directed graph with sentries as nodes and dependencies between them as edges. The graph induces a topological sorting for execution that eliminates ambiguity and maintains an "intuitive" order of execution. This order requires the graph to be acyclic. Hence, Hull et al. [21] define

that a GSM model is well-formed iff its dependency graph is acyclic.

Parallel Processing of State Data In our approach, we use time-triggered operational semantics. Instead of using events, we query status of the workflow state and the environment periodically to determine the current state. Correspondingly, we apply updates in bulk instead of updating the status information variables as we go. Each state in the sequence is the result of the application of our entire ruleset. Above, we define the function f to describe this state transfer.

Therefore, the B-step concept is not directly applicable to our approach. A B-step includes every action which depends on an event e. If there are no events, we cannot associate any actions with them. Looked at from the perspective of the event-triggered GSM semantics, our approach rather performs M-steps periodically and in parallel. Still, we need to discuss the ramifications of non-intuitive behaviour and updates applied in bulk.

As an illustration, consider again the example from Fig. 4, which we treat using our notation. With the example's stages s_1 and s_2 and their guards g_1 and g_2, we define:

$\gamma \in \{true, false\}$ a boolean variable
γ_σ the value of γ in state σ
$g_1 = true$ if $s_2 = active$ and γ, else $false$
$g_2 = true$ if γ, else $false$

If we now assume $\gamma_\sigma = true$, then

$$\phi_\sigma(g_1) = false \wedge \phi_\sigma(g_2) = true$$

and

$$\sigma'(s_1) = inactive \wedge \sigma'(s_2) = active$$

We observe that our approach does not reproduce the "non-intuitive" behaviour. The reason is that each state σ is immutable and σ' is constructed independently, but based on σ. Therefore, there is no need to order the execution of sentry evaluations and their corresponding consequences. In other words, there are no conflicts between changing the state of a status information or environmental variables before or after observing it. There is also no need to order the M-Steps using a dependency graph because σ stays the same for all g.

Write Conflicts Above, we reason why an order for reading and changing values of status information variables in our approach is not necessary. We provide the immutability of states as main reason. Still, there is a possibility that we produce ambiguous state derivations. In this section, we illustrate the issue of such ambiguity, where and if it occurs, and its ramifications if it does occur.

Our approach consists of multiple rules, that, depending on the truth of a condition, derive new values for a set of status information variables as action. We do not know in which order the rules are applied. In the previous section, we show that the order of rules is not relevant, if one rule derives a value for a status information variables and another rules accesses it. If now two rules each derive a different value for the same status information variables in the subsequent state, the order becomes relevant. The value of this status information variables is ambiguous. We call it a *conflict* in the ambiguous variables. We identify all conflicts that may occur and discuss each one in detail.

Recall that the function f constructs for a state σ its subsequent state σ' and Table 2 specifies all rules and their corresponding actions in f. We now list all combination of rules that may derive ambiguous results. We skip derivations that are equal, since they are not ambiguous. In the following, we write $FCR\text{-}1 \rightarrow \sigma'(m) = achieved$, if from FCR-1 follows that milestone m is achieved in the next snapshot.

Conflicting Write by FCR-2 + FCR-3: We look at a possible conflict in σ' for a milestone's state, $\sigma'(m_s)$, with $FCR\text{-}2 \rightarrow \sigma'(m_s) = achieved$ and $FCR\text{-}3 \rightarrow \sigma'(m_s) = unachieved$. This requires that the conditions of both rules hold:

$$\forall s \in S, m_s \in \mathcal{M}_s : \sigma(s) = active \wedge \phi_\sigma^+(m_s) \\ \wedge \sigma(m_s) = achieved \wedge \phi_\sigma^-(m_s)$$

Besides the fact that an overlapping validating and invalidating sentry is semantically questionable, all logical operators are conjunctions. Therefore, all operands must hold individually and all their combination. This requires:

$$\forall s \in S, m_s \in \mathcal{M}_s : \phi_\sigma^+(m_s) \wedge \phi_\sigma^-(m_s)$$

Under the disjointness condition of achieving and invalidating sentries (Eq. 4.3.1), this is impossible. $\Longrightarrow$ There is no σ that leads to a conflict of FCR-2 and FCR-3 in m_s.

Conflicting Write by FCR-1 + FCR-2 FCR-1 and FCR-2 induce two possible conflicts in σ': The activation of a stage by $FCR\text{-}1 \rightarrow \sigma'(s) = active$ can conflict with both consequences of $FCR\text{-}2 \rightarrow \sigma'(s) = inactive \wedge \sigma'(s_d) = inactive$.

We discuss the first part of the conflict: $FCR\text{-}1 \rightarrow \sigma'(s) = active$ and $FCR\text{-}2 \rightarrow \sigma'(s) = inactive \wedge \sigma'(s_d) = inactive$. By combining the conditions of both rules, we get:

$$\forall s \in S, s_a \in A_s, g_s \in G_s, m_s \in \mathcal{M}_s : \sigma(s) = inactive \\ \wedge \sigma(s_a) = active \wedge \phi_\sigma(g_s) \wedge \sigma(s) = active \wedge \phi_\sigma^+(m_s)$$

Again, all operators are conjunctions and thus all combinations of operands must be satisfied. The following

combination contains a contradiction:

$$\forall s \in S : \sigma(s) = inactive \wedge \sigma(s) = active$$

$\Longrightarrow$ There is no σ that leads to a conflict of FCR-1 and FCR-2 in s

Next, we consider the second part of the conflict of *FCR-1* and *FCR-2*: The case of the descendant stage s_d, to which *FCR-2* intends to write. We thus assume that *FCR-1* applies to stage s_d. We start with *FCR-1* $\rightarrow \sigma'(s_d) = active$ and *FCR-2* $\rightarrow \sigma'(s) = inactive \wedge \sigma'(s_d) = inactive$ and construct the conjunction. By combining the conditions of *FCR-1* and *FCR-2*, we get:

$$\begin{aligned}&\forall s \in S, s_d \in D_s, s_{d_a} \in A_{s_d}, g_{s_d} \in G_{s_d}, m_s \in \mathcal{M}_s : \\ &\sigma(s_d) = inactive \wedge \sigma(s_{d_a}) = active \\ &\wedge \phi_\sigma(g_{s_d}) \wedge \sigma(s) = active \wedge \phi^+_\sigma(m_s)\end{aligned}$$

Note that A_{s_d} is the set of all ancestors of s_d, and therefore, $A_{s_d} \supset A_s$ holds. The contradiction we observe when applying *FCR-1* to the first part of the consequent of *FCR-2* does not occur anymore. Given this, we cannot assure that there are no write conflicts just by contradiction of conditions. Instead, we have to avoid this conflict when modelling the workflow. The disjointness condition of guards and milestones (Eq. 2) prescribes that a guard and its parent's milestones have disjoint sentries. That is, $\phi^+_\sigma(m_s)$ and $\phi_\sigma(g_{s_d})$ cannot be *true* the same time. Hence, the equation above is never satisfied if we impose the disjointness condition.

$\Longrightarrow$ There is no σ that leads to a conflict of *FCR-1* and *FCR-2* in s_d

We remark that the effects of the described ambiguity may be considered minor. The consequence of the ambiguity is that a child stage s_d is either triggered once again or not. To put it in context, the stage s_d is supposed to be deactivated because its parent is deactivated. The consequence is that s_d may perform its task and it does not abort. After finishing, its state changes to *inactive* and its milestone is set to achieved. As soon as the parent stage becomes active again, the milestone is set back to *unachieved*. Of course, this requires the task of s_d to finish. Other assumptions would require the ability to abort tasks, which is beyond the scope of this work. For the sake of completeness, we provide the disjointness condition as a modelling constraint.

Conflicting Write by FCR-2 + FCR-4: Lastly, we discuss a possible conflict in FCR-2 and FCR-4. There is a conflict in the state of the milestone by *FCR-2* $\rightarrow \sigma'(m_s) = achieved$ and *FCR-4* $\rightarrow \sigma'(m_s) = unachieved \wedge \sigma'(m_{s_d}) = unachieved$. Again, we combine the conditions:

$$\begin{aligned}&\forall s \in S, s_a \in A_s, s_d \in D_s, m_s \in \mathcal{M}_s, g_s \in G_s : \\ &\sigma(s) = active \wedge \phi^+_\sigma(m_s) \wedge \sigma(s_a) = active \\ &\wedge \sigma(s) = inactive \wedge \phi_\sigma(g_s) \wedge \phi^+_\sigma(m_s)\end{aligned}$$

Since $\sigma(s) = active$ contradicts to $\sigma(s) = inactive$, this conflict cannot occur for the same stage s.

$\Longrightarrow$ There is no σ that leads to a conflict of *FCR-2* and *FCR-4* in s

As in the previous rule combination with *FCR-2*, we also examine the case where *FCR-2* applies to the descendant stage s_d. Then,

$$\begin{aligned}&\forall s \in S, s_d \in D_s, s_a \in A_s, m_s \in \mathcal{M}_s, m_{s_d} \in \mathcal{M}_{s_d}, \\ &g_s \in G_s : \\ &\sigma(s_d) = active \wedge \phi^+_\sigma(m_{s_d}) \wedge \sigma(s_a) = active \\ &\wedge \sigma(s) = inactive \wedge \phi_\sigma(g_s) \wedge \phi^+_\sigma(m_s)\end{aligned}$$

Now we observe that $\sigma(s) = inactive$ and $\sigma(s_d) = active$ do not contradict. So, if a stage s is inactive, its descendants s_d need to be *active*. We did discuss a situation where this may happen in the previous paragraph when we considered the combination *FCR-1 + FCR-2*, but under the disjointness condition this inconsistency is eliminated. Hence, we can conclude that given a workflow that aligns with the disjointness condition, no conflicts occur.

$\Longrightarrow$ There is no σ that leads to a conflict of *FCR-2* and *FCR-4* in s and/or s_d

As result, we know that under the disjointness condition, for each state σ, σ' is unambiguously determined. Formally, we can say:

$$\sigma \implies \sigma'$$

In addition to that, we even can conclude that if any rule derives a new value for a status information variables, there exists no other rule that derives a different value, even if we do not know the order of the rule execution. This implies that if there is any status information variables $\sigma(x) = y$, and any *individual* FCR r deriving a value $\sigma_r(x)' = y'$, then $\sigma'(x) = y'$ (where $\sigma_r(x) = y$ is the value y of x if only r is applied). Thus, we can write:

$$\sigma(x) = y \wedge \sigma_r(x) = y' \implies \sigma'(x) = y' \tag{3}$$

6.2.2 GSM Invariants

The invariants *GSM-1* and *GSM-2* define combinations of status attributes that are inconsistent. If one of the invariant is violated, the workflow state is inconsistent.

GSM-1 "If a stage S owns a milestone m, then it cannot happen that both S is *active* and m has status *true*. In particular, if S becomes active, then m must change status to *false*, and if m changes status to *true*, then S must become *inactive*." [11]

$$\forall s \in S, m \in \mathcal{M}_s :$$
$$\sigma(s) = active \implies \sigma(m) \neq achieved$$
$$\wedge \sigma(m) = achieved \implies \sigma(s) \neq active$$

GSM-2 "If stage S becomes *inactive*, the executions of all substages of S also become *inactive*." [11]

$$\forall s \in S, s_d \in D_s :$$
$$\sigma(s) = inactive \implies \sigma(s_d) = inactive$$

Additionally we define a function $c : \Sigma \rightarrow \{true, false\}$, that determines whether both GSM invariants hold on a snapshot σ.

None of these invariants are allowed to be violated at any time. As the first step of our proof, we show that all possible states induced by the FCRs do not violate the GSM invariants using mathematical induction.

Theorem: *GSM-1* and *GSM-2* are not violated throughout the workflow execution.

Proof: We apply the set of FCRs to the information model. One snapshot σ contains all data concerning the workflow's state, as well as environment values at the beginning of each iteration. The state of the workflow and the environment values correspond to status and data attributes, respectively. σ_0 represents our initial workflow state after the initialisation. We now proof the theorem using mathematical induction.

Base case: σ_0 : No stage is activated yet $\implies c(\sigma_0) = true$. ✓

Step case: $\sigma \rightarrow \sigma'$: In order to get into an inconsistent state one of the invariants must be violated. We distinguish two cases—a violation of *GSM-1* and a violation of *GSM-2*:

$GSM-1$ To infer the consistency of σ' we assume the contrary. We assume a set of rules that derive at least one triple so that $c(\sigma') = false$. To achieve that given $c(\sigma) = true$, there must be a milestone m and a stage s which either satisfy (C1) or (C2):

$$(C1) : \sigma(m) = unachieved \wedge \sigma'(m) = achieved$$
$$\wedge \sigma'(s) = active$$
$$(C2) : \sigma(s) = inactive \wedge \sigma'(s) = active$$
$$\wedge \sigma'(m) = achieved$$

In words, case (C1) describes the case where a milestone becomes achieved and its stage stays active. Case (C2) corresponds to the case, where a stage becomes active although one of its milestones is still achieved.

$Case(1)$: We assume that condition (C1) holds. Since only *FCR-2* leads to the achievement of a milestone, we know that $\sigma(m) = unachieved$ and $\sigma'(m) = achieved$ is true, if and only if the condition of *FCR-2* is satisfied. Hence, we conclude that the action of *FCR-2* holds as well. From equation 3, we know that $\sigma'(m) = unachieved$. This is a contradiction to our condition (C1). ↯

$Case(2)$: We assume condition (C2) holds. Because (C2) requires a stage becoming active in σ', we conclude that the condition *FCR-1* must be satisfied. Considering equation 3, we know that

$$\sigma(s) = inactive \wedge \sigma(s_a) = active \wedge \phi_\sigma(g_s)$$
$$\wedge \sigma'(m_s) = active$$

If we consider the assumption that there exists a milestone $m_s \in \mathcal{M}_s$ and $\sigma'(m_s) = achieved$, either the milestone must have been already achieved in σ or it became achieved in σ'. *FCR-1* and *FCR-4* share the almost same condition. The difference is that *FCR-4* sets all corresponding *achieved* milestones to *unachieved*. Hence, in the first case, if $\sigma(m_s) = achieved$, *FCR-4* implies $\sigma'(m_s) =$*unachieved*. By Eq. 3, we know that $\sigma'(m_s) =$*unachieved*. This contradicts assumption (C2). In the latter case, *FCR-2* must hold, since it is the only rule that implies an achieved milestone. *FCR-2* also implies $\sigma'(s) = inactive$ though, which also contradicts our assumption. ↯

$GSM-2$ Similar to *GSM-1*, we show the contrary by assuming there are rules that derive at least one triple in such a way, that $c(\sigma') = false$. This requires

$$\exists s \in S, s_d \in D_s :$$
$$\sigma(s) = inactive \wedge \sigma'(s_d) = active$$

Because FCR-1 requires all ancestors to be active, a violation of this condition is only possible if a parent stage is set to inactive while its parent stays active. Only FCR-2 leads to a stage being set to *inactive*. While it implies $\sigma(s) = inactive$, it also implies $\sigma(s_d) = inactive$ for all $s_d \in D_s$. Again, by Eq. 3, we know that $\sigma(s_d) = inactive$ holds for all $s_d \in D_s$. ↯

As result we conclude that, under the disjointness conditions, f does not imply a transition from a state σ with $c(\sigma) = true$ to a state σ' with $c(\sigma) = false$. $\implies$ **Step case** ✓

By the principle of mathematical induction, we have shown that the invariants *GSM-1* and *GSM-2* are not violated. □

6.3 Applicability

In Sect. 1, we described different scenarios and deployments from automotive, avionics, manufacturing, and Internet of Things, where our approach could be applied. To publicly showcase the approach presented in this paper, we build an small conference demonstrator [1] based on Internet of Things devices with Read-Write Linked Data interfaces. The implementation this demonstrator can be found online.[29] For the status attributes and rule interpreter, the demonstrator uses LDBBC[30] as Linked Data Platform Container implementation, and Linked Data-Fu[31] [40] as N3 rule interpreter with ASM4LD [23] operational semantics.

6.4 Performance

Our approach takes the GSM approach and brings it to Read-Write Linked Data. Therefore, we contrast our approach with the converse, namely to bring Read-Write Linked Data to GSM. We compare: our operational semantics deployed on Linked Data-Fu and status attributes maintained in LDBBC, with the CMMN implementation in Camunda,[32] a commercial process and case management suite (CMMN,[33] the Case Management Model and Notation, is a standard that is closely related to GSM [11]).

Both engines therefore need to make HTTP requests to sources that provide RDF and reason over the data using RDF using rules that implement the RDFS semantics given to the engines for data integration.

With Camunda of course not built for that workload, we now describe our assumptions and implementation: For data processing, Camunda has SPIN,[34] a library to extend the process languages they support with data processing of XML[35] and JSON in a scripting fashion, such that we can use a classless, declarative approach to define the processing steps, in line with the declarative input of rules to Linked Data-Fu. We chose the tree-based XML as data format, as one can serialise RDF graphs in XML trees using the RDF/XML[16] serialisation format, and the querying support of the XML stack is better supported in the extensions of Camunda. We then specified the processing steps as BPMN[36] process, which we plugged into our CMMN diagram as process task. Our BPMN process mimics Linked Data-Fu's operational semantics (cf. Sect. 3.6) by parallel downloading RDF/XML, and subsequent[37] parallel rule evaluation of the RDFS semantics. As the processing of arbitrary RDF/XML using the XML querying stack is a research endeavour of its own, where one reason is that different XML trees can represent the same RDF graph [5], we assume a deterministic mapping from triples to XML trees, where each triple gets a dedicated `rdf:Description` node, as employed by many popular RDF/XML serialisers.[38] Furthermore, we assume ground triples without literals and lists, as blank nodes, datatypes and lists would require specialised treatment in RDFS and RDF/XML. Then, we can give the rules to implement RDFS using BPMN Script Tasks implemented declaratively in XQuery 3.1.[39] Those parallel tasks with rules are placed in a loop that runs until no further deductions are derived (and thus, the fixpoint is calculated).

For our tests, we crafted a workflow, which we designed to include different constellations of stages, guards, and milestones. Specifically, the workflow includes nested stages and milestones whose sentries check whether other milestones have been reached. The workflow contains 8 stages, starts with two sequentially arranged stages, and leads into a combination of stages with 2 as the maximum level of nesting. Thereby occur some interesting situations like parallel execution of s2 and s4 where s4 is nested in another stage s3. For a deterministic evaluation, the sentries are designed to be always fulfilled, but their queries need to be evaluated. We designed the data part of the sentries to check for data that needs to be derived using reasoning. In case of Linked Data-Fu, those are SPARQL ASK queries in SPIN notation, in case of Camunda, those are XPath 1.0[40] queries within an EL[41] expression. We did not include invalidating sentries for the milestones in order not to have to build the equivalence of dynamic data attributes, which could also reduce the determinism and the repeatability of the evaluation. Although there is no specific story to that workflow you can think of it as a simplification of the Order-to-Design workflow example from [11]. The workflow model in both implementations, the BPMN for Linked Data processing, and the data to be retrieved can be found online.[42]

[29] http://github.com/nico1509/data-driven-workflows.

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

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

[32] https://camunda.com/.

[33] https://www.omg.org/spec/CMMN/1.1/.

[34] https://github.com/camunda/camunda-spin, not to be confused with the SPIN notation for SPARQL queries, also used by our approach.

[35] http://www.w3.org/TR/xml/.

[36] https://www.omg.org/spec/BPMN/2.0/About-BPMN/.

[37] For simplicity sake, we left out writing and link following, and provided the URIs to be read from upfront.

[38] Including librdf http://librdf.org/, and OWLAPI if not in pretty printing mode https://github.com/owlcs/owlapi.

[39] http://www.w3.org/TR/xquery-31/.

[40] http://www.w3.org/TR/1999/REC-xpath-19991116/.

[41] https://jcp.org/en/jsr/detail?id=341.

[42] http://people.aifb.kit.edu/co1683/2019/gsm/jods/.

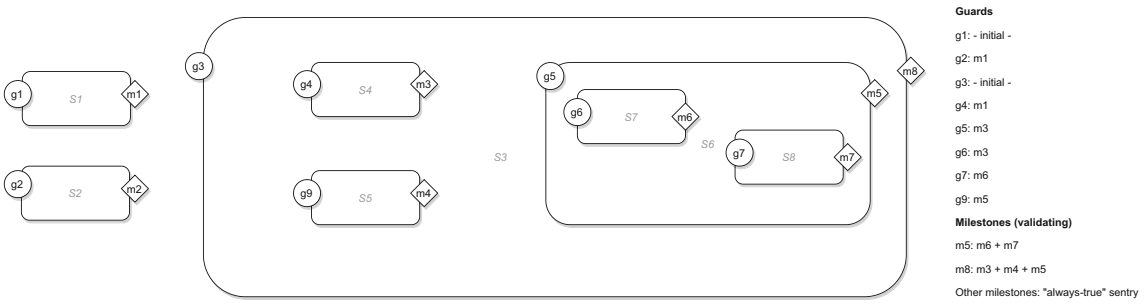


Fig. 5 The workflow we use in our evaluation

We ran our experiments on a MacBook Pro with an Intel Core i5-5257U processor with 2.7 GHz, 4 threads, and 8 GB RAM running OpenJDK 1.8.0_131 on Mac OSX Version 10.14.6 and allow for a warm-up time of 10s. We repeated our experiments 5 times. For our set-up, we used Linked Data-Fu 0.9.13pr1 and LDBBC 0.0.5. For the Camunda set-up, we used Camunda 7.14 Community Edition, where we needed to remove the limit of 4000 characters in certain columns of the database Camunda runs on.

We show our results in Fig. 6. We see that the declarative Read-Write Linked Data processing approach in Linked Data-Fu outperforms the approach with a declarative implementation of data retrieval and reasoning from within the Camunda process engine. On top, the Linked Data-Fu approach retrieves and reasons with every execution cycle such that the queries run on fresh data, where the Camunda-based approach only retrieves and reasons in 3 of the stages. This limitation of our implementation is due to difficulties we faced when connecting the retrieval and reasoning part to the CMMN workflow lifecycle. Therefore, our results for Camunda only serve as a lower bound. On the other hand, we see for Linked Data-Fu that the handling of workflow instances in data structures and code not optimised for that purpose puts considerable load on the system: Already between 1 and 9 instances, the runtime doubled.

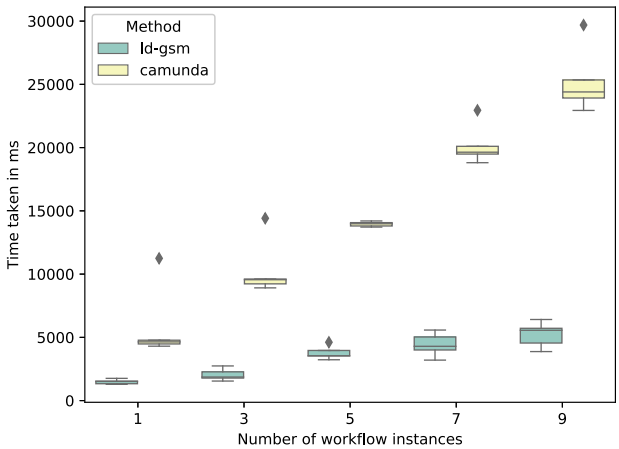


Fig. 6 Results from 5 runs of our evaluation: Time in ms until the termination of the last workflow when starting n workflows in parallel

7 Conclusion and Discussion

We presented an approach to specify and execute agent specifications in the form of data-centric workflows in Read-Write Linked Data, i.e. an environment of semantic knowledge representation and reasoning. To this end, our approach consists of an ontology, and operational semantics. We gave the operational semantics in a rule language for Read-Write Linked Data, derived requirements for modelling and discussed the rule expressivity required for querying data and status attributes. We showed the correctness of our rules for the operational semantics and provided a performance evaluation.

While we envision our approach to be particularly useful in small-scale scenarios where agents interact with a handful of resources with only few workflow instances, we evaluated our approach against a workflow engine, which is built for scenarios with many instances. In the evaluation, our general-purpose Read-Write Linked Data processor made to maintain workflow instance state outperformed a workflow engine made to perform Read-Write Linked Data processing with data retrieval and reasoning. This is despite the overhead people often fear when considering polling-based approaches. Viewed from a broader perspective, we remark that in essence, our agent behaviour specification encodes the behaviour part missing in the integration standards from the web architecture such as HTTP for interaction and symbolic reasoning in RDFS for data integration. If we take BPM solutions as way of doing integration in practice by specifying the behaviour that orchestrates system components, our evaluation agrees with [34] that there is still a way to go until the integration capabilities of the web architecture can be fully exploited in practice.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Aßfalg N, Nürnberg L, Jochum B, Käfer T (2019) Controlling internet of things devices with read-write linked data interfaces using data-driven workflows. In: Proceedings of posters and demos at the 15th international conference on semantic systems (SEMANTiCS)
2. Bagheri Hariri B, Calvanese D, De Giacomo G, Deutsch A, Montali M (2013) Verification of relational data-centric dynamic systems with external services. In: Proceedings of the 32nd symposium on principles of database systems (PODS). ACM
3. Bagheri Hariri B, Calvanese D, Montali M, De Gia-como G, De Masellis R, Felli P (2013) Description logic knowledge and action bases. J Artif Intell Res 46:651–686
4. Berners-Lee T (2009) Read-write linked data [Design issues]. http://www.w3.org/DesignIssues/ReadWriteLinkedData.html
5. Bischof S, Decker S, Krennwallner T, Lopes N, Polleres A (2012) Mapping between RDF and XML with XSPARQL. J Data Semantics 1(3):147–185
6. Brauns S, Kafer T, Koriath D, Harth A (2016) Individualisiertes Gruppentraining mit Datenbrillen fur die Produktion. In: Proceedings of the 46th annual convention of the German informatics society (INFORMATIK) (in German)
7. Calvanese D, Montali M, Patrizi F, Rivkin A (2019) Modeling and in-database management of relational, data-aware processes. In: Proceedings of the 31st international conference on advanced information systems engineering (CAiSE)
8. Casati F, Ceri S, Pernici B, Pozzi G (1996) Deriving active rules for workflow enactment. In: Proceedings of the 7th international conference on database and expert systems applications (DEXA)
9. Ciortea A, Mayer S, Gandon F, Boissier O, Ricci A, Zimmermann A (2019) A decade in hindsight: The missing bridge between multi-agent systems and the world wide web. In: Proceedings of the 18th international conference on autonomous agents and multi agent systems (AAMAS)
10. Ciortea A, Mayer S, Michahelles F (2018) Repur-posing manufacturing lines on the fly with multi-agent systems for the web of things. In: Proceedings of the 17th international conference on autonomous agents and multi-agent systems (AA-MAS)
11. Damaggio E, Hull R, Vaculin R (2013) On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. Inf Syst 38(4):561–584
12. Dijkman RM, Dumas M, Ouyang C (2008) Semantics and analysis of business process models in BPMN. Inf Softw Technol 50(12):1281–1294
13. Elmroth E, Hernandez-Rodriguez F, Tordsson J (2010) Three fundamental dimensions of scientific workflow interoperability: model of computation, language, and execution environment. Future Gener Comput Syst 26(2):245–256
14. Fielding R (2000) Architectural styles and the design of network-based software architectures. Doctoral dissertation. University of California, Irvine, USA
15. Gil Y, Ratnakar V, Deelman E, Mehta G, Kim J (2007) Wings for pegasus: creating large-scale scientific applications using semantic representations of computational workflows. In: Proceedings of the 19th conference on innovative applications of artificial intelligence (IAAI)/22th conference on artificial intelligence (AAAI)
16. Gurevich Y (1995) Evolving algebras 1993: Lipari guide. In Specification and validation methods, Oxford University Press
17. Haller A, Cimpian E, Mocan A, Oren E, Bus-sler C (2005) Wsmx—a semantic service-oriented architecture. In: Proceedings of the 3rd international conference on web services (ICWS)
18. Harth A, Speiser S (2012) On completeness classes for query evaluation on linked data. In: Proceedings of the 26th conference on artificial intelligence (AAAI)
19. Hogan A (2014) Linked data and the semantic web standards. In: Harth A, Hose K, Schenkel R (eds) Linked data management. Chapman and Hall/CRC, Boca Raton
20. Hull R, Damaggio E, De Masellis R, Fournier F, Gupta M, Heath FFTI, Vaculin R (2011) Business artifacts with guard-stage-milestone lifecycles. In: Proceedings of the 5th international conference on distributed event-based systems (DEBS)
21. Hull R, Damaggio E, Fournier F, Gupta M, Heath FFT, III, Hobson S, Vaculin R (2011) 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)
22. Jochum B, Niirnberg L, Afifalg N, Kafer T (2019) Data-driven workflows for specifying and executing agents in an environment of reasoning and restful systems. In: Proceedings ofthe 3rd international workshop in artificial intelligence for busi-ness process management (AI4BPM) at the 17th international conference on business process managment (BPM)
23. Kaifer T, Harth A (2018) Rule-based programming of user agents for linked data. In: Proceedings of the 11th workshop on linked data on the web (LDOW) at the 27th webconf (WWW)
24. Kaifer T, Harth A (2018). Specifying, monitoring, and executing workflows in linked data environments. In: Proceedings of the 17th international semantic web conference (ISWC)
25. Kafer T, Harth A, Mamessier S (2016) Towards declarative programming and querying in a distributed cyber-physical system: The i-VISION case. In: Proceedings of the 2nd international workshop on modelling, analysis, and control ofcomplex CPS (CPSData) at the 9th CPS week
26. Korkan E, Kabisch S, Kovatsch M, Steinhorst S (2018) Sequential behavioral modeling for scalable iot devices and systems. In: Proceedings of the 21st forum on specification and design languages (FDL)
27. Kiinzle V, Weber B, Reichert M (2011) Object-aware business processes: fundamental requirements and their support in existing approaches. Int J Inf Syst Model Des 2(2):1–29
28. Martin DL, Paolucci M, McIlraith SA, Burstein MH, McDermott DV, McGuinness DL, Sycara KP (2004) Bringing semantics to web services: the OWL-S approach. In: Proceedings of the 1st international workshop on semantic web services and web process composition (SWSWPC)
29. Martin D, Burstein M, Hobbs J, Lassila O, Mc-Dermott D, McIlraith S, Sycara K (2004) Owl-s: semantic markup for web services [Member submission]. https://www.w3.org/Submission/OWL-S/
30. Montali M, Rivkin A (2017) Db-nets: on the marriage of colored petri nets and relational databases. In: Transactions on Petri Nets and Other, Models of Concurrency, p 12
31. Newman S (2015) Building microservices—designing fine-grained systems. O'Reilly, Sevastopol

32. Pautasso C (2009) Restful web service composition with BPEL for REST. Data Knowl Eng 68(9):851–866
33. Pautasso C, Wilde E (2011) Push-enabling restful business processes. In: Proceedings of the 9th international conference on service-oriented computing (ICSOC)
34. Pautasso C, Zimmermann O (2018) The web as a software connector: integration resting on linked resources. IEEE Softw 35(1):93–98
35. Pautasso C, Zimmermann O, Leymann F (2008) Restful web services vs Big web services: making the right architectural decision. In: Proceedings of the 17th international conference on world wide web (WWW)
36. Petri CA (1962) Kommunikation mit Automaten. Doctoral dissertation, Technische Hochschule Darmstadt, Germany
37. Polyvyanyy A, van der Werf JMEM, Overbeek S, Brouwers R (2019) Information systems modeling: Language, verification, and tool support. In: Proceedings of the 31st international conference on advanced information systems engineering (CAiSE)
38. Popova V, Dumas M (2012) From petri nets to guard-stage-milestone models. In: Proceedings of the 1st international workshop on data- and artifact-centric bpm (DAB) at the 10th international conference on business process management (BPM)
39. Seco JC, Debois S, Hildebrandt TT, Slaats T (2018) RESEDA: declaring live event-driven computations as reactive semi-structured data. In: Proceedings of the 22nd international enterprise distributed object computing conference (EDOC)
40. Stadtmiiller S, Speiser S, Harth A, Studer R (2013) Data-fu: a language and an interpreter for interaction with R/W Linked Data. In: Proceedings of the 22th international conference on world wide web (WWW)
41. Turi D, Missier P, Goble CA, Roure DD, Oinn T (2007) Taverna workflows: syntax and semantics. In: Proceedings of the 3rd international conference on e-science and grid computing (e-Science)
42. zur Muehlen M, Nickerson JV, Swenson KD (2005) Developing web services choreography standards-the case of REST vs. SOAP. Decis Support Syst 40(1):9–29