Data-Driven Workflows for Specifying and Executing Agents in an Environment of Reasoning and RESTful Systems

Benjamin Jochum, Leonard Nürnberg, Nico Aßfalg, and Tobias Käfer
uzebb@student.kit.edu, ujeng@student.kit.edu, uberq@student.kit.edu, tobias.kaefer@kit.edu

Institute AIFB, Karlsruhe Institute of Technology (KIT), Germany

Abstract. We present an approach to specify and execute agents on Read-Write Linked Data that are given as Guard-Stage-Milestone workflows. That is, we work in an environment of semantic knowledge representation, reasoning and RESTful systems. For specifying, we present a Guard-Stage-Milestone workflow and instance ontology. For executing, we present operational semantics for this ontology. We show 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.

The environment of the web is finally at a stage where hypermedia agents could be applied [3]: We see that dynamic, open, and long-lived systems are commonplace on the web forming a highly distributed system. For instance, microservices [13] 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 e. g. 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 [6], or HTTP[3], respectively as 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 agents in this environment called Read-Write Linked Data [1], 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 [10].

[1] https://www.w3.org/2016/07/wot-ig-charter.html
[2] https://solid.mit.edu/
[3] http://tools.ietf.org/rfc/rfc7230.txt
[4] http://www.w3.org/TR/rdf11-concepts/

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?*

As the environment determines why different workflow approaches are used in different circumstances [5], 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 to resort to polling to get informed about changes in the environment.

Reasoning and querying under the Open-World Assumption While in databases, typically the closed-world assumption is made, i. e. we conclude from the absence of information that it is false, RDF is based on the open-world assumption. 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, e. g., 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 [6][5]. **The open-world assumption** could, e. g., be addressed by introducing assumptions such as negation-as-failure once a certain completeness class [9] has been reached.

In previous work, we defined ASM4LD, a model of computation for the environment of Read-Write Linked Data [11], which allows for rule-based specification of agent behaviour. Based on this model of computation, we provided an approach to specify flow-driven workflows [12]. In contrast, we present a data-driven approach in this paper. Hull et al. present the Guard-Stage-Milestone (GSM) approach [10], which serves as basis for our work. 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 contrast to Pautasso, who presented an approach to retrofit REST into the BPEL approach to workflow modelling [14], our approach rather retrofits a workflow modelling approach into REST, here GSM.

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.

[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.

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 Section 1, we survey related work. Next, in Section 2, we give basic definitions on which we build our approach. Subsequently, in Section 3, we give an example, which we use in our explanations. In Section 4, we present our main contributions. Next, we evaluate our approach 5 by showing its correctness and performance. Last, in Section 6, we conclude.

1 Related Work

Workflow Management Previous Workflow languages and also workflow management systems rely on events using Event-Condition-Action (ECA) rules, e. g. [10, 2]. We base our approach on REST so there are no events to be processed. Instead we use state machines as an operational semantics to track the polled application state.

Web Services Web Services based on the WS-* standards, on which approaches such as BPEL are built, allow for arbitrary operations. In contrast, REST constrains this set [16, 19]. Thus, extensions for BPEL have been proposed to include RESTful services [15, 14]. Our approach however exploits the semantics of the constrained set of operations in REST.

Semantic Web Services OWL-S and WSMO mainly focus on descriptions of services and reasoning to allow agents to compose web services. WS-*-based semantic web services completely rely on events[8] while our approach is based on REST where there are no events.

Scientific Workflows Previous Scientific Workflow approaches often set their focus on the data flow between processes of a workflow[18, 7]. Our approach however uses a data-driven approach for control flow.

Ontologies for Workflows In previous works, ontologies for describing workflows and processes have been developed, e. g. in various research projects like Super, ASG, among others. Those ontologies require more expressive reasoning or do not allow for execution.

Workflows in Linked Data In previous work, we developed WiLD to specify and execute flow-based behaviour descriptions in Linked Data [12]. In contrast, in this paper, we investigate data-centric behavior descriptions.

2 Preliminaries

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

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

2.1 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 Uniform Resource Identifiers (URIs)[7] on the server. Requests are typed, and the type (i. e. the HTTP method) determines the semantics of 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.

2.2 The Resource Description Framework (RDF) and SPARQL

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. Things in RDF are identified globally using URIs[7], or document-locally using so-called blank nodes. Literals can be used to express values. In this paper, we use the following notation for RDF: As triples encode binary predicates, we write *rdf:type*(*:si*, *:StageInstance*) to mean the following triple in Turtle notation[8]: `<#si> rdf:type :StageInstance .` Moreover, for the special case of class assignments, we use unary predicates: That is, above triple becomes *:StageInstance*(*:si*). SPARQL is a query language for RDF data[9] and supports a so-called `ASK` query that returns `true` if a condition holds in a RDF graph.

2.2.1 Abstract State Machines for Linked Data (ASM4LD) ASM4LD is an Abstract State Machine based operational semantics given to Notation3, a rule language for the semantic web [11]. 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 [11], we derived the operational semantics based on the semantics of HTTP requests, first-order logic, and Abstact State Machines. The operational semantics can be summarized in four steps to be executed in a loop, thus implementing polling:

1. Initially, set the working memory be empty.
2. Add the assertions to the working memory.
3. Until the fixpoint, 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.

[7] `http://tools.ietf.org/rfc/rfc3986.txt`
[8] Turtle allows for abbreviating URIs, where a colon separates the abbreviating prefix from the local name. The example uses the empty prefix, which denotes `http://purl.org/gsm/vocab#`. We refer to `http://prefix.cc/` for other abbreviations.
[9] `http://www.w3.org/TR/sparql11-query/`

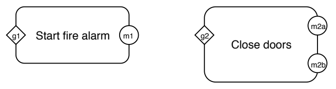

Fig. 1. A small example of two stages with associated guard and milestone(s).

   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.

2.3 The Guard-Stage-Milestone Approach

The Guard-Stage-Milestone approach is an artifact-centric workflow meta-model, presented by Hull et al. in [10]. 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** 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, and are formulated sentries. **Sentries** are expressions in a condition language, e. g. Event-Condition-Action rules (**on** <event> **if** <condition> **then** <action>). 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 Figure 1.

To specify the operational semantics, [10] provides a set of six PAC rules. PAC rules are a variation of Event-Condition-Action rules and are described by an 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.

3 Example

Figure 2 shows, using a fire alarm as an example, how a workflow execution proceeds. Every line represents a step. Line 2: when smoke has been detected, the start alarm stage is triggered. Line 3: the "close doors"-guard gets activated,

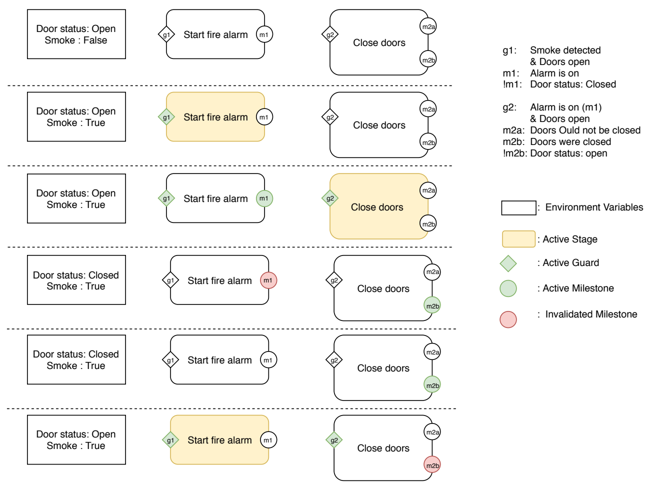

Fig. 2. Example of a workflow execution. Time progresses from top to bottom.

which triggers the closing of the doors. Line 4: the invalidating sentry of m1 deactivates g2. Line 5: after all doors had been closed, somebody re-opens a door. This triggers the invalidation of the corresponding milestone m2b (all doors are closed). As a consequence, the alarm stage is re-triggered.

4 Proposed Approach

Our approach consist in an ontology to model GSM workflows and instances and operational semantics in rules with ASM4LD semantics, which interpret those workflows and instances. The rules can be directly deployed on a corresponding interpreter. We first present the ontology and then the operational semantics. We use the syntax described in Section 2.

4.1 Ontology for Modelling Entities

We built an ontology[10] to describe the core modelling primitives from [10]. We stay as closely as possible to their definitions and divert only if demanded by the our environment of Read-Write Linked Data: **Tasks** are HTTP requests as atomic activities. **Sentries** contain a SPARQL `ASK` query in SPIN notation[11]. Correspondingly, we use SPIN's `true` boolean SPARQL query result (we cannot use false, as we excluded negation in the introduction) with sentries and guards. We introduce the class **State** of all states to e. g. model three states of a stage: active, inactive, and done. Stages are set to done after they have been executed once.

[10] `http://purl.org/gsm/vocab`
[11] `http://spinrdf.org/`

In our approach, the information model is not a database, but instead Read-Write Linked Data. Hence, we do not maintain the data attributes ourselves: We do not store about the system we control, but instead, we retrieve the system state live in RDF over HTTP from the system itself. We maintain the status attributes for uniform access and also describe and retrieve the workflow instance status in RDF over HTTP. Thus, from now on we call all information regarding the state of the workflow *status information*. All information about environmental conditions in form of external data providers like an external service, we call *environment information*.

4.2 Operational Semantics

The following operational semantics are based on the PAC-rule-based semantics from [4]. We distinguish between setup and flow conserving (FC) rules. We assume all status information in an collection resource at `http://ldpc.example/`.

Instance Set-up Rules The basic condition for all setup rules is:

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

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

$$CS \longrightarrow \mathrm{PUT}(i, \mathit{:SuperStageInstance}(i) \wedge \cdots \wedge \mathit{:hasState}(i, \mathit{:active}))$$

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

$$\begin{gathered} CS \wedge \mathit{:hasDescendantStage}(s, s_{child}) \\ \longrightarrow \mathrm{POST}(\texttt{http://ldpc.example/}, \mathit{:isInstanceOf}(\texttt{new}, s_{child}) \\ \wedge \mathit{:inSuperStageInstance}(\texttt{new}, i) \wedge \mathit{:hasState}(\texttt{new}, :\mathit{inactive})) \end{gathered}$$

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

$$\begin{gathered} CS \wedge \mathit{:hasDescendantStage}(s, s_{child}) \wedge \mathit{:hasMilestoneModel}(s_{child}, m) \\ \longrightarrow \mathrm{POST}(\texttt{http://ldpc.example/}, \mathit{:isInstanceOf}(\texttt{new}, m) \\ \wedge \mathit{:inSuperStageInstance}(\texttt{new}, i) \wedge \mathit{:hasState}(\texttt{new}, \mathit{:unachieved})) \end{gathered}$$

Flow-conserving Rules The following condition has to match in every FCR.

$$\begin{gathered} FC := \mathit{:SuperStageInstance}(S^I) \wedge \mathit{:StageModel}(s^M) \wedge \mathit{:StageInstance}(s^I) \\ \wedge \mathit{:isInstanceOf}(s^I, s^M) \wedge \mathit{:inSuperStageInstance}(s^I, S^I) \wedge \mathit{:hasState}(S^I, \mathit{:active}) \end{gathered}$$

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.

$$\begin{gathered}
FC \wedge \mathit{:hasGuard}(s^M, g) \wedge \mathit{:hasHttpRequest}(s^M, req) \wedge \mathit{:allAncestorsActive}(s^I, \texttt{true}) \\
\wedge \mathit{:hasState}(s^I, \mathit{:inactive}) \wedge \mathit{:hasCondition}(g, c) \wedge \mathit{sparql\text{-}results\text{:}boolean}(c, \texttt{true}) \\
\longrightarrow \mathrm{PUT}(s^I, \ldots \mathit{:hasState}(s^I, \mathit{:active})) \\
\longrightarrow \mathrm{req}(\cdot, \cdot)
\end{gathered}$$

FCR-2 Set an active stage done and its milestone achieved if the validating sentry holds.

$$\begin{gathered}
FC \wedge \mathit{:hasState}(s^I, \mathit{active}) \wedge \mathit{:hasMilestoneModel}(s^M, m^M) \\
\wedge \mathit{:hasValidatingSentry}(m^M, c^a) \wedge \mathit{:isInstanceOf}(m^I, m^M) \\
\wedge \mathit{sparql\text{-}results\text{:}boolean}(c^a, \texttt{true}) \\
\longrightarrow \mathrm{PUT}(m^I, \ldots \mathit{:isAchieved}(m^I, \texttt{true})) \\
\longrightarrow \mathrm{PUT}(s^I, \ldots \mathit{:hasState}(s^I, \mathit{:done}))
\end{gathered}$$

FCR-3 When an achieved milestone's invalidating sentry of a completed stage becomes true, three things are done by the following rule: (1) The invalidated milestone is set to unachieved (2) All other milestones of the stage are set to unachieved (3) The milestone's stage is set to inactive.

$$\begin{gathered}
FC \wedge \mathit{:hasMilestoneModel}(s^M, m^M) \wedge \mathit{:isInstanceOf}(m^I, m^M) \wedge \mathit{:isAchieved}(m^I, \texttt{true}) \\
\wedge \mathit{:hasInvalidatingSentry}(m^M, c) \wedge \mathit{sparql\text{-}results\text{:}boolean}(c, \texttt{true}) \\
\wedge \mathit{:hasMilestoneModel}(s^M, m_2^M) \wedge \mathit{:isInstanceOf}(m_2^I, m_2^M) \wedge \mathit{:isAchieved}(m_2^I, \texttt{true}) \\
\longrightarrow \mathrm{PUT}(m^I, \ldots \mathit{:isAchieved}(m^I, \texttt{false})) \\
\longrightarrow \mathrm{PUT}(m_2^I, \ldots \mathit{:isAchieved}(m_2^I, \texttt{false})) \\
\longrightarrow \mathrm{PUT}(s^I, \ldots \mathit{:hasState}(s^I, \mathit{:inactive}))
\end{gathered}$$

FCR-6 The invalidation of an active stage's milestone sets all descendant stages to inactive and all of the stage's and its descendants' milestones to unachieved.

$$\begin{gathered}
FC \wedge \mathit{:hasMilestoneModel}(s^M, m^M) \wedge \mathit{:isInstanceOf}(m^I, m^M) \wedge \mathit{:isAchieved}(m^I, \texttt{true}) \\
\wedge \mathit{:hasInvalidatingSentry}(m^M, c) \wedge \mathit{sparql\text{-}results\text{:}boolean}(c, \texttt{true}) \\
\wedge \mathit{:hasMilestoneModel}(s^M, m_2^M) \wedge \mathit{:isInstanceOf}(m_2^I, m_2^M) \\
\wedge \mathit{:hasDescendantStage}(s^M, s_d^M) \wedge \mathit{:isInstanceOf}(s_d^I, s_d^M) \\
\wedge \mathit{:hasMilestoneModel}(s_d^M, m_d^M) \wedge \mathit{:isInstanceOf}(m_d^I, m_d^M) \\
\longrightarrow \mathrm{PUT}(m_2^I, \ldots \mathit{:isAchieved}(m_2^I, \texttt{false})) \\
\longrightarrow \mathrm{PUT}(s_2^I, \ldots \mathit{:hasState}(s_2^I, \mathit{:inactive})) \\
\longrightarrow \mathrm{PUT}(m_d^I, \ldots \mathit{:isAchieved}(m_3^I, \texttt{false}))
\end{gathered}$$

5 Evaluation

In this section, we show the correctness of the proposed ruleset. We first show that the FCR rules cover the functionality of the PAC rules from [4]. Second we show that our approach does not violate the GSM invariants. This implies the correctness of the rules.

Discussion of our Rules PAC-1 activates a stage if its parent and one of its guards are active. FCR-1 does completely match this. PAC-2 sets a milestone to achieved if a milestone's achieving sentry changes to true and the stage the milestone is attached to is active. This also completely matches FCR-2. PAC-3 invalidates all milestones whose invalidating sentries are true. Again, FCR-3 covers this completely. PAC-4 resets Milestones of a stage, if the stages guards are triggered. As a result of missing change-events, we omitted this rule. Its functionality can be achieved by adding the guards' sentry as invalidating sentry to all attached milestones. PAC-5 deactivates a stage upon achievement of one of its milestones. We merged this into FCR-2, as otherwise, they would not happen in the same step. PAC-6 determines that all stages nested into another are deactivated, as soon as the latter is deactivated. FCR-6 covers this.

Checking the GSM Invariants Damaggio et al.[4] provide two invariants to ensure the consistency of their workflow model. To show that our approach does not violate the invariants, we first define the following:

- S is the set of stages
- G is the set of guards, $G_s \subseteq G$ all guards of a stage s
- M is the set of milestones, $M_s \subseteq M$ all milestones of a stage s,
- Φ is the set of Sentries, $\phi : M \rightarrow \{true, false\}$ represents the result of the sentry
- $d(s) = \{s_c \in S | s_c \text{ is child-stage of } s\}$,
- $\Sigma := \{active, inactive, done, achieved, unachieved\}$ is the set of possible states
- $\Sigma^* :=$ The set of all possible Snapshots,
- $f : \Sigma^* \rightarrow \Sigma^*$ representing the result after application of our ruleset,

For reasons of readability $f(\sigma)$ with $\sigma \in \Sigma^*$ will be abbreviated with σ'.
$\sigma(x)$ will be an abbreviation for the state of the instance (of a milestone or stage) x. Note that, as described, there are 3 possible states for stages: active, inactive and done.

Next, we present the two invariants and give a logical formulation:

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." [4]

$$\forall s \in S, m \in M_s : \sigma(s) = active \implies \sigma(m) \neq achieved$$
$$\wedge \sigma(m) = achieved \implies \sigma(s) \neq active$$

GSM-2 "If stage S becomes inactive, the executions of all substages of S also become inactive." [4]

$$\forall s \in S, s' \in d(s) : \sigma(s) = \mathit{inactive} \implies \sigma(s') = \mathit{inactive}$$

We now show that these invariants are not violated throughout the execution of one of our workflow models. $c : \Sigma^* \to \{true, false\}$ determines wheter a state σ is conform to GSM-1 and 2.

Theorem: GSM-1 and GSM-2 sustain true throughout the workflow execution.

Proof: We apply a set of rules to our distributed information model. We prove the invariants' correctness using mathematical induction. One snapshot σ contains all data concerning the workflow's state, as well as environment values at the beginning of the loop iteration. σ_0 will represent our initial workflow state after the initialization.

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

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

GSM-1 To investigate the consistency of the following state we will try to show of the contrary. Assume that there are rules that derive at least one triple so that $c(\sigma') = false$. To achieve that from a consistent state there must be a milestone m, and a stage with either (1) or (2) satisfied:

$$\sigma(m) = \mathit{unachieved} \wedge \sigma'(m) = \mathit{achieved} \wedge \sigma'(s) = \mathit{active} \quad (1)$$

$$(\sigma(s) = \mathit{done} \vee \sigma(s) = \mathit{inactive}) \wedge \sigma'(s) = \mathit{active} \wedge \sigma'(m) = \mathit{achieved} \quad (2)$$

Case (1): The only rule that sets a milestone to achieved (true) is FCR-2. The rule's condition requires an active stage. If then a corresponding milestone's achieving sentry is true, its milestone is set to achieved and FCR-5 is triggered which sets the stage to done. This leads to:

$$\phi^a_\sigma(m) = true \wedge \sigma(s) = \mathit{active} \implies \sigma'(m) = \mathit{achieved} \wedge \sigma'(s) = \mathit{done} \quad (3)$$

which contradicts to (1). ↯

Case (2): Assuming that (2) holds, there must be a rule that sets a stage's state to active, while the state of a milestone attached to it remains achieved. Only FCR-1 is able to change the state of a stage. Its condition requires the state to be inactive prior to being set active. If a stage's milestone is achieved it is set to done instead of inactive. Therefore a stage can not be set to active while its milestone is active. This contradicts (2). ↯

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

$$\exists s \in S, s_{child} \in d(s) : \sigma(s) = \mathit{inactive} \wedge \sigma'(s_{child}) = \mathit{active} \quad (4)$$

Due to FCR-1, if a stage has not been activated in the past, its children can not be active. This combination of states is only possible if a stage becomes inactive, while its children are still active. FCR-6 determines, that for all stages with an attached milestones' invalidating sentry triggered, all descendant stages are set to inactive. This leads to the following:

$$\sigma(s) = \mathit{inactive} \implies \forall s_{child}, s \in d(s) : \sigma'(s_{child}) = \mathit{inactive} \tag{5}$$

Again, we see a clear contradiction to (4). $\lightning \implies$ **Step case** ✓

These contradictions induce that the rules do not imply a transition from a consistent state σ to an inconsistent state σ', or $f(\sigma)$. □

5.1 Applicability

An implementation of our approach can be found online[12]. We use LDBBC[13] as Linked Data Platform Container implementation, and Linked Data-Fu[14] [17] as N3 rule interpreter with ASM4LD [11] operational semantics. We successfully applied the approach in an use-case with Internet of Things devices having Read-Write Linked Data interfaces.

6 Conclusion

We presented an approach to specify and execute agent specifications in the form of data-centric workflows in an environment of semantic knowledge representation and reasoning.

As with all approaches that work on the web architecture, our agent relies on polling to get informed about the world state instead of the environment reporting events. Thus, if the polling rate is not high enough, this sampling approach may miss important system states. For instance, in the case of the Internet of Things, we may miss short button presses. We showed in [11] that our interpreter can indeed achieve high update rates in the range of milliseconds.

References

1. Berners-Lee, T: Read-Write Linked Data. Design Issues, (2009). `http://www.w3.org/DesignIssues/ReadWriteLinkedData.html`
2. Casati, F, Ceri, S, Pernici, B, and Pozzi, G: Deriving Active Rules for Workflow Enactment. In: Proc. 7th DEXA (1996)
3. Ciortea, A, Mayer, S, Gandon, F, Boissier, O, Ricci, A, and Zimmermann, A: A Decade in Hindsight: The Missing Bridge Between Multi-Agent Systems and the World Wide Web. In: Proc. 18th AAMAS (2019)

[12] `http://github.com/nico1509/data-driven-workflows`
[13] `http://github.com/kaefer3000/ldbbbc`
[14] `http://linked-data-fu.github.io/`

4. Damaggio, E, Hull, R, and Vaculín, R: On the equivalence of incremental and fixpoint semantics for business artifacts with Guard–Stage–Milestone lifecycles. Information Systems 38(4) (2013)
5. Elmroth, E, Hernández-Rodriguez, F, and Tordsson, J: Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. Future Generation Computer Systems 26(2) (2010)
6. Fielding, R: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, USA (2000)
7. Gil, Y, Ratnakar, V, Deelman, E, Mehta, G, and Kim, J: Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In: Proc. 19th IAAI / 22th AAAI (2007)
8. Haller, A, Cimpian, E, Mocan, A, Oren, E, and Bussler, C: WSMX – a semantic SOA. In: Proc. ICWS (2005)
9. Harth, A and Speiser, S: On Completeness Classes for Query Evaluation on Linked Data. In: Proc. 26th AAAI (2012)
10. Hull, R, Damaggio, E, De Masellis, R, Fournier, F, Gupta, M, Heath, FFTI, Hobson, S, Linehan, MH, Maradugu, S, Nigam, A, Sukaviriya, PN, and Vaculín, R: Business artifacts with guard-stage-milestone lifecycles. In: Proc. 5th DEBS (2011)
11. Käfer, T and Harth, A: Rule-based Programming of User Agents for Linked Data. In: Proc. 11th LDOW (2018)
12. Käfer, T and Harth, A: Specifying, Monitoring, and Executing Workflows in Linked Data Environments. In: Proc. 17th ISWC (2018)
13. Newman, S: Building Microservices-Designing Fine-grained Systems. O'Reilly (2015)
14. Pautasso, C: RESTful Web Service Composition with BPEL for REST. Data and Knowledge Engineering 68(9) (2009)
15. Pautasso, C and Wilde, E: Push-Enabling RESTful Business Processes. In: Proc. 9th ICSOC (2011)
16. Pautasso, C, Zimmermann, O, and Leymann, F: Restful Web Services vs "Big" Web Services: Making the right architectural decision. In: Proc. 17th WWW (2008)
17. Stadtmüller, S, Speiser, S, Harth, A, and Studer, R: Data-Fu: A language and an interpreter for interaction with R/W Linked Data. In: Proc. 22nd WWW (2013)
18. Turi, D, Missier, P, Goble, C, Roure, DD, and Oinn, T: Taverna Workflows: Syntax and Semantics. In: Proc. 3rd e-Science (2007)
19. Zur Muehlen, M, Nickerson, JV, and Swenson, KD: Developing web services choreography standards—the case of REST vs. SOAP. Decision Support Systems 40(1) (2005)