Rule-based Programming of User Agents for Linked Data

Tobias Käfer
Karlsruhe Institute of Technology (KIT)
Institute AIFB
Karlsruhe, Germany
tobias.kaefer@kit.edu

Andreas Harth
Karlsruhe Institute of Technology (KIT)
Institute AIFB
Karlsruhe, Germany
harth@kit.edu

ABSTRACT

While current Semantic Web languages and technologies are well-suited for accessing and integrating static data, methods and technologies for the handling of dynamic aspects – required in many modern web environments – are largely missing. We propose to use Abstract State Machines (ASMs) as the formal basis for dealing with changes in Linked Data, which is the combination of the Resource Description Framework (RDF) with the Hypertext Transfer Protocol (HTTP). We provide a synthesis of ASMs and Linked Data and show how the combination aligns with the relevant specifications such as the Request/Response communication in HTTP, the guidelines for updating resource state in the Linked Data Platform (LDP) specification, and the formal grounding of RDF in model theory. Based on the formalisation of Linked Data resources that change state over time, we present the syntax and operational semantics of a small rule-based language to specify user agents that use HTTP to interact with Linked Data as the interface to the environment. We show the feasibility of the approach in an evaluation involving the specification of automation in a Smart Building scenario, where the presented approach serves as a theoretical foundation.

1 INTRODUCTION

Data in languages such as RDF (Resource Description Framework)[1], RDFS (RDF Schema)[2], and OWL (Web Ontology Language)[3] is widely deployed on the web[4] and allow for data representation and integration. The technologies around Linked Data [3] facilitate the publication of and access to such data: data publishers make RDF files available on web servers; user agents can then access the published RDF files using HTTP (Hypertext Transfer Protocol)[5]. On such read-only web data, developers can build data integration systems based on queries and ontologies, i. e. approaches grounded in mathematical logic, where most of the operations are carried out by query processors and reasoners. However, to integrate such data from read-only sources is only a first step; many scenarios, for example on the Web of Things, require to change data. With HTTP-based write access to Linked Data, we have a uniform interface on both the data and the interaction level, easing system integration. But while the W3C's LDP (Linked Data Platform) specification[6] combines read-only Linked Data (via the HTTP GET operation) with read-write capabilities (HTTP PUT, POST, and DELETE operations) for the interface to writeable resources, most systems that interact with LDP servers are still programmed in imperative code. We would like to specify the behaviour of user agents in a language grounded in mathematical logic.

A high-level language for the specification of the behaviour of user agents would allow expressing executable specifications, which are shorter and cheaper to create than imperative code, and which are easier to validate (e. g. with subject matter experts) and to analyse (e. g. using formal methods). The language could also serve as execution substrate for agent specifications created using planning approaches from Artificial Intelligence.

Several approaches exist for specifying user agents on read-only Linked Data. These user agents process queries and have the ability to follow links, which can lead to the discovery of additional data during query processing [5], [16], [12], [38], [17]. An extension to read and write capabilities in addition to the following of links, which is core to web architecture [9], [30], would allow for user agents that are able to not only query data but also to effect change. The choice of a formalism for dynamical aspects on the web should respect these standards, specifications, and practices as foundations. How to beneficially combine current technologies around Linked Data and REST into a unified formalism of dynamics is an open research problem [18], [43], [31], [13].

We present an approach for programming user agents for Linked Data. Following a hypermedia architectural style, the approach operates on a set of resources, with a resource's state described in RDF. The resource state descriptions contain references to other resources (links). Resource state can be manipulated via HTTP operations. To formalise the user agent specifications, which cause an evolution of Linked Data over time, we build on Abstract State Machines (ASMs) [11] rooted in mathematical logic. We believe that ASMs are the right starting point: ASMs encode state in first-order logic, which is a superset of the RDF-family of languages; on top, ASMs require only rules to represent state change, and rules have been proposed for RDF data processing[7],[8] In our operational semantics derived from ASM semantics, the execution of rule-based programs (user agents) leads to a series of HTTP requests.

Our contributions are as follows:

- We provide a formal account of the standards around Linked Data in combination with Abstract State Machines (ASMs). With the synthesis between Linked Data, which provides a way to represent and manipulate resource state, and ASMs, which provides operational semantics based on rules and stepwise execution of these rules, we are able to specify user

[1] http://www.w3.org/TR/rdf11-concepts/
[2] http://www.w3.org/TR/rdf-schema/
[3] http://www.w3.org/TR/owl2-overview/
[4] See e.g. the datasets in the Linking Open Data cloud at http://lod-cloud.net/
[5] http://www.ietf.org/rfc/rfc7230.txt
[6] http://www.w3.org/TR/ldp/

[7] http://www.w3.org/TeamSubmission/n3/
[8] http://www.w3.org/Submission/SWRL/

TheWebConf Workshop: Linked Data on the Web (LDOW), 2018, Lyon, France

agent behaviour with the entire user agent state maintained in RDF on web-accessible resources.
- We present a syntax for a rule language (based on Notation3[7], a superset of the RDF Turtle language) to define conditions that trigger updates based on HTTP state manipulation operations. We give an operational semantics of the language based on the ASM execution semantics.
- We conduct an evaluation of a prototype rule processing system in a Smart Buildings scenario and show the applicability of the approach.

The paper is structured as follows: we start in Section 2 with an introduction to a Smart Building scenario, in which we set the examples and the evaluation. We continue in Section 3 with basic definitions of Linked Data and ASMs. In Section 4, we propose a synthesis on the level of state representation (interpretations) in ASMs and RDF model theory. In Section 5, we present an evaluation using a benchmark and point to successful applications of our approach. In Section 6, we relate our work to the state of the art and in Section 7 we conclude with a summary and an outlook.

2 EXAMPLE SCENARIO: RULE-BASED CONTROL FOR BUILDING AUTOMATION

Throughout the paper, we use examples from the Smart Building domain, inspired by recent work in Building Management Systems: NIST identified interoperability as a major challenge for the building industry [10]. To raise interoperability in Building Management Systems, Balaji et al. developed Brick [2], an ontology to model buildings and corresponding building management systems.

We use a description of building 3 at IBM Research Dublin in Ireland. Balaji et al. provide a static description of this building using the Brick ontology[9]. The description covers the building and the building's parts (e. g. rooms) and the different parts of the building's systems (e. g. lights and switches). As there is nothing to automate in the static description, we introduce dynamics by adding information about the state of these system parts (e. g. on/off) in the form of properties from the SSN ontology[10]. We serve those properties as writeable Linked Data on localhost.

In the examples, we use our approach to program an user agent that controls one light in the building in a straight-forward fashion (turn the light off if the light is on). In the evaluation, we use more complex control schemes with up to 30 rules and control the lights of the entire building.

2.1 Intuition of the Syntax of a Rule Language

We express rule programs in a subset of the Notation3 (N3) syntax[7], for an example see Figure 1. N3 is a superset of the RDF Turtle syntax[11], hence we can express RDF in N3: In RDF, we use URIs (Uniform Resource Identifiers[12]) as names for resources, for instance we use the abbreviated[13] URI `IBM_B3:Lighting_1F_M59` to identify the lights in room 21 on the first floor in wing 42 of building 3. In RDF, we can link resources to each other, thus forming a graph. For instance, we link said lights to the `ssn:Property` `http://localhost/Lighting_1F_M59#it` (cf. (1) in Figure 1). To form rules, we use N3's graph quoting (curly brackets) and variables (terms starting with question marks). Note that in contrast to RESTdesc [43], which also uses the N3 syntax, we do not interpret the rules as input and output descriptions of HTTP interactions, but as executable specifications.

We distinguish derivation and request rules. In both types of rules, the body consists of a set of triple patterns (triples with variables, cf. basic graph patterns (BGPs) in SPARQL[14]). In a derivation rule, the head also consists of a set of triple patterns (cf. (2) in Figure 1, which defines the inverse to `ssn:hasProperty`). Derivation rules are not in the focus of the paper, but are important when layering higher-level entailment regimes on top of the work presented in this paper. Here, we use the Simple Interpretation. In a request rule, the head specifies an HTTP request. The request specification includes an HTTP method, a request URI, and optionally a set of triple patterns to form the HTTP body (cf. Figure 1, where (3) dereferences an end of `ssn:isPropertyOf` links, and (4) changes the state of `ssn:Properties` that fulfil a certain condition).

```
@prefix IBM_B3: <http://buildsys.org/ontologies/examples/IBM_B3#> .
@prefix brick:  <http://buildsys.org/ontologies/Brick#> .

# (1) Language feature: RDF:
IBM_B3:Lighting_1F_M59 a brick:Lighting ;
  ssn:hasProperty <http://localhost/Lighting_1F_M59#it> .

# (2) Language feature: Derivations:
{ ?thing ssn:hasProperty ?property . }
=> { ?property ssn:isPropertyOf ?thing . } .

# Language feature: Conditional requests:
## (3) Defining how to retrieve world state (GET requests):
{ ?y ssn:isPropertyOf ?x . }
=> { [] http:mthd httpm:GET ; http:requestURI ?y . } .

## (4) Defining the logic (PUT, POST, DELETE requests):
{ ?light a brick:Lighting .
  ?property a ssn:Property ; foaf:primaryTopicOf ?ir ;
    ssn:isPropertyOf ?light ; rdf:value "off" . }
=> { [] http:mthd httpm:PUT ; http:requestURI ?ir ;
       http:body
         { ?property a ssn:Property ; rdf:value "on" . } . } .
```

Figure 1: An example for a simple rule program that turns on the lights (in N3 syntax[13]).

2.2 Intuition of the Semantics of the Rule Language

We informally give the operational semantics for the rule language by describing how an interpreter would execute the program in Figure 1: An interpreter evaluates the triple patterns of the body of the rules on RDF data that is either given initially (1) or downloaded as mandated by request rules with GET requests (3). Meanwhile, the interpreter adds data as mandated by derivation rules (2). The interpreter executes derivations and GET requests until it calculated the fixpoint. Last, the interpreter executes the PUT, POST, DELETE requests of all request rules whose body holds in the data (4). The

[9] `https://github.com/BuildSysUniformMetadata/GroundTruth/blob/2e48662/building_instances/IBM_B3.ttl`
[10] `http://www.w3.org/TR/vocab-ssn/`
[11] `http://www.w3.org/TR/turtle/`
[12] `http://www.ietf.org/rfc/rfc3986.txt`
[13] We assume the usual definitions for the prefixes from standardised vocabularies such as RDF, HTTP, and SSN, which can be found on `http://prefix.cc/`
[14] `http://www.w3.org/TR/sparql11-query/`

interpreter operates in a looped fashion. The rationale behind this operational semantics is the subject of the paper.

3 PRELIMINARIES

In this section, we give foundational definitions of the technologies around Linked Data with a special focus on the aspects relevant to our proposed synthesis with Abstract State Machines, whose basic definitions form the rest of this section.

We assume a basic knowledge of the technologies Linked Data is built on: Uniform Resource Identifiers (URIs)[12] for identifying things, the Hypertext Transfer Protocol (HTTP)[5] for interacting with things, and the Resource Description Framework (RDF)[1] for static descriptions of things. As the technologies are about the transfer and the description of state information, we first discuss the relations of different kinds of state (e. g. world and resource state). Next, we go beyond the basics of HTTP and refresh the reader's knowledge of the semantics of different HTTP requests and responses. The HTTP semantics are an important building block in our formalisation. Next, we refresh the reader's knowledge of basic RDF model theory. We use model-theoretic semantics to describe static aspects in our formalisation. Subsequently, we outline a relation between Linked Data and the semantics of an RDF Dataset, which we need for our formalisation when it comes to the relation between a source of data and the data itself. Last, we introduce Abstract State Machines (ASMs), on which we base our formalisation of dynamic aspects. If two approaches use the same term, we introduce subscripts for the origin ($\cdot_{\text{RDF}}$ vs. $\cdot_{\text{ASM}}$).

3.1 State

We distinguish different kinds of state: When successfully retrieving state using HTTP GET from a URI, we obtain a description the *resource's state*, e. g. the light in a room. The corresponding HTTP *request's state* can e. g. be "successful". We call the union of the state information about all resources *world state*, i. e. all resources in the world. An application (e. g. implemented as rule program) can maintain state (e. g. in writeable resources), which then represents the *application's state*, e. g. a note that something just happened. The application state is a subset of the world state. ASM terminology uses the term *system state*, which can be translated to our terminology as the subset of the world state that is relevant for an application, e. g. the building description and the weather report.

3.2 HTTP Semantics

The Hypertext Transfer Protocol (HTTP) is a protocol to interact with a resource (called the target) in a request/response fashion[5]. In the spirit of Read-Write Linked Data, we consider the request methods that implement CRUD[15]: GET, PUT, POST, DELETE.

As our formalisation is based on world state, we only consider (of the responses to HTTP requests with different methods) the responses to successful HTTP GET requests:

- We assume the responses to *unsuccessful* HTTP requests to be empty: The response to an unsuccessful HTTP request against a target does not contain information about the target, but on the request[16].
- We also consider the responses to *successful HTTP PUT, POST, DELETE* requests as empty, as they may be empty or carry no information about the state of resources[6],[16].

We now turn to the individual HTTP request methods and summarise their semantics[16]. For legibility, we neglect redirects in this paper. We sketch in Section 4.2 how they can be re-introduced.

The *GET* request is a means to retrieve state information about the request target. For instance, we can find out whether the lights are on or not in said room 21 using a GET request to `http://localhost/Lighting_1F_M59#it`. By contract, the GET request is considered *safe*, i. e. a GET request must not change state.

The *PUT* request is a means to overwrite the state information at the request target. For instance, we can set the lights to be on in said room 21 using a PUT request to `http://localhost/Lighting_1F_M59#it`. For the formalisation in this paper, we assume all targets that we want to write to, to be writeable.

The *POST* request is a means to e. g. append a resource described in the body to a collection or to send data to a data handling process. In this paper, we restrict ourselves to the former.

The *DELETE* request is a means to delete the request target. For our considerations, we regard a DELETE request as a PUT request with an empty message body.

3.3 RDF Model-Theoretic Semantics

We use the graph-based data model Resource Description Framework (RDF) to describe things and their relations[1]. In an RDF Graph, typed connections between resources are stated in the form of triples, e. g. (1) in Figure 1, which relates a light to a `ssn:Property`. An RDF Graph is defined follows:

Definition 1 (RDF Graph, Triple). *Let $\mathcal{U}, \mathcal{B}, \mathcal{L}$ be the respective sets of all URIs, Blank Nodes and Literals. The set of all RDF triples T is then defined as follows: $T = \mathcal{U} \cup \mathcal{B} \times \mathcal{U} \times \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$. An RDF Graph G is a set of triples: $G = \{t | t \in T\}$.*

The RDF standard uses model theory to specify the meaning of an RDF Graph (e. g. that the URI for the light is interpreted as the resource that represents the light). We paraphrase the definitions as they are given in the corresponding W3C Recommendation[17].

Definition 2 (Interpretation$_{\text{RDF}}$, name, vocabulary$_{\text{RDF}}$, universe). *An* interpretation *is defined as a mapping from $\mathcal{U}$ and $\mathcal{L}$ into some set called the* universe. *Both the mapping and the set can be constrained. A* name *is an element from the union $\mathcal{U} \cup \mathcal{L}$. A set of names is also called* vocabulary.

The basic interpretation for an RDF Graph is the *simple interpretation*. The simple interpretation defines the following mappings and constraints:

Definition 3 (Simple interpretation, extension). *A* simple interpretation *is defined using the following sets and mappings: IR and IP denote the subsets of the universe that contain all resources and properties respectively. IEXT(p), called the* extension *of a property p, is a mapping $IP \to 2^{IR \times IR}$ that maps p to all pairs in IR that are connected by p. IS is a mapping $\mathcal{U} \to IR \cup IP$. IL is a partial mapping $\mathcal{L} \to IR$.*

[15] The basic operations of persistent storage: Create, Read, Update, Delete

[16] `http://www.ietf.org/rfc/rfc7231.txt`

[17] `http://www.w3.org/TR/rdf11-mt/`

To not to have to specifically address Blank Nodes, we assume ground RDF Graphs in this paper. The extension to Blank Nodes should be straight-forward.

3.4 RDF Dataset Semantics and Linked Data

To talk about RDF data from different sources, we use the notion of an RDF Dataset. An RDF Dataset[1] is a collection of RDF Graphs. Each RDF Graph in an RDF Dataset is identified using a name, which can be a URI, a Blank Node, or empty. The RDF Graph with the empty name is called the *default graph*. For legibility, we denote the default graph with `default` in this paper.

While the RDF specification does not impose restrictions on the relation between the graph name and the RDF Graph in the RDF Dataset, we use the semantics defined in Section 3.5 of[18] as abstraction of Linked Data: The graph names are URIs and the RDF Graph with a given name in the RDF Dataset is the graph that can be obtained using an HTTP GET request with the graph name as target. If the request to a graph name fails or returns no RDF data, the RDF Graph is empty. In the evaluation, we contrast two ways of constructing the RDF Dataset about the current system state: Using download of a dump (D1), or following links (D2).

3.5 Abstract State Machine (ASM)

For the dynamic aspects of our approach, we use Abstract State Machines (ASMs). Concretely, ASMs help us to define how to evolve Linked Data, which we interpret as RDF Dataset. Here, we give the standard definitions for ASMs to make the paper self-contained. The approach of ASMs has been developed towards the end of the 1980s with the aim to bridge between the specification and computation of programs [11]. ASMs are meant to improve on Turing's thesis. Turing Machines have proven too low-level to specify algorithms larger than toy examples. In ASMs however, the level of abstraction can be adapted to the requirements of the use-case. ASMs have been extensively used to give operational semantics to programming languages including C, Prolog, and Java[19].

An ASM consists of an algebraic first-order structure and a set of transition rules on how to modify the structure. In the following, we give the basic definitions for ASMs.

Definition 4 (Vocabulary$_{\text{ASM}}$, function name, relation name, characteristic function name, variable, term). *The* vocabulary Υ *be defined as a finite set of* function names *and their arity* $n \geq 0$. *The vocabulary also contains the nullary function names* `undef`, *and the boolean constants* `true` *and* `false`. *Moreover, the vocabulary contains the usual boolean operators as functions.* Relation names *and* characteristic function names *are special function names.* Terms *can be defined recursively:* Variables *are terms. If* f *is an* r*-ary function and* $t_1, \ldots, t_n$ *are terms, then* $f(t_1, \ldots, t_n)$ *is a term.*

For the presentation in this paper, we characterise a function name with arity > 0 with their correctly sized argument list, e. g. to talk about an RDF Dataset for Linked Data, we define $quad(\cdot, \cdot, \cdot, \cdot)$, which defines the function name *quad* with an arity of 4. We omit the argument list for nullary function names.

[18] `http://www.w3.org/TR/rdf11-datasets/`

[19] See the annotated ASM bibliography at `http://web.eecs.umich.edu/gasm/index.html`

Definition 5 (Super-universe, function, relation, characteristic function). *The* super-universe X *is a non-empty set.* X *contains e. g. all* functions $X^n \to X$, *true, false, and undef. The boolean functions behave the usual way on* $\{true, false\}$. Relations *are functions that map* $X^n \to \{true, false\}$. *A* characteristic function *for a set is a function* $X \to \{true, false\}$ *that maps to* `true` *iff the operand belongs to the set.*

Definition 6 (Interpretation$_{\text{ASM}}$, transition rule, function update, static and dynamic function). *An* interpretation $\mathcal{I} : \Upsilon \to X$ *maps the terms from an ASM vocabulary to a super-universe. A* transition rule *R is a function update, e. g.* $f(t_1, \ldots, t_n) := t_0$*, or a guarded function update, which has a boolean condition, e. g.* **if** *condition* **then** *function update(s). A* function update *changes the interpretation of a function name at given arguments. A* static function *is not subject to change by the function updates of an ASM, as opposed to a* dynamic function.

Definition 7 (Value and evaluated form). *The* value *of a term from the vocabulary* Υ *is the term's referent in the super-universe* X *under the current interpretation* $\mathcal{I}$. *We denote the referent of a term* t *under an interpretation* $\mathcal{I}$ *using* $eval(t, \mathcal{I})$. *If* t *is a tuple, we have:* $eval(t, \mathcal{I}) := (eval(t_1, \mathcal{I}), \ldots, eval(t_n, \mathcal{I}))$. *If* t *is a function, we have:* $eval(f(t_1, \ldots, t_n), \mathcal{I}) := eval(f, \mathcal{I})(eval(t_1, \mathcal{I}), \ldots, eval(t_n, \mathcal{I}))$.*A function update in its so-called* evaluated form *is* $eval(f(t_1, \ldots, t_n) := t_0, \mathcal{I}) = f(eval(t_1, \mathcal{I}), \ldots, eval(t_n, \mathcal{I}))) := eval(t_0, \mathcal{I})$.

Definition 8 (Algebra, state, run, step). *A (static)* algebra *or* state *is a triple of a vocabulary, a super-universe, and an interpretation. A* step *is a transition from one state to the next by the means of firing transition rules. A* run *is a sequence of steps.*

The transition rules to be fired in one step are composed in an update set.

Definition 9 (Update set). *The* update set *to be fired in an interpretation* $\mathcal{I}$ *under the transition rule set* T *can be defined as* $updates(\mathcal{I}, T) := \{eval(u, \mathcal{I}) | u \in T \wedge (\text{the condition of } u \text{ holds in } \mathcal{I} \vee u \text{ has no condition})\}$

In ASM, we assume discrete time. Time progresses from one state to the next, i. e. in a run, there is an ordered sequence of system states. There may be multiple transition rules that fire in one state. If multiple transition rules want to update the interpretation of a function name for the same arguments, then the system halts because of the conflict.

Definition 10 (Abstract State Machine). *An* Abstract State Machine *(ASM) can be defined as a quadruple of a vocabulary, a super-universe, an interpretation for time* t_0*, and a set of transition rules: ASM* $:= (\Upsilon, X, \mathcal{I}_0, T)$

We now introduce variables to be able to range over names in conditions and updates. In ASM, we have to state from which set of individuals those individuals are to take that are to be bound to variables. Therefore, a variable declaration is of the following form:

Variable(s) `?varname` **range(s) over** *set*
End variable scope

Formally, the variables can be covered by the help of an auxiliary vocabulary that contains the variables as nullary function names [11].

4 ABSTRACT STATE MACHINES AND LINKED DATA + RULES

This section is about our proposed synthesis of ASM, and Linked Data + Rules with the help of RDF model theory and HTTP semantics. We first describe on a high level the commonalities between ASM and RDF model theory, which motivate our synthesis, and then the different foci of both approaches. After that, we provide a synthesis of ASM and Linked Data + Rules. While the formalisation has the user agent in the focus, which regards Linked Data as external functions, we next sketch how a server fits into the picture. Then, we sketch how the synthesis can be used for specifying computation. Subsequently, we use the synthesis to give semantics to a rule language and discuss our findings. Last, we derive requirements for a Linked Data user agent specification language.

4.1 Overview

While both, ASMs and RDF are defined in terms of interpretations of vocabularies into a (super-)universe, the focus of both approaches is different: RDF model theory is about whether different RDF Graphs entail each other or whether models can exist for an RDF Graph. Therefore, conditions on the universe are in the focus. The interpretation is static in RDF model theory. ASMs are all about the evolution of system state, which manifests itself in the evolution of the interpretation of a vocabulary into a super-universe. Therefore, the transition function T is in the focus, in which updates to the interpretation are stated and the conditions under which the updates happen. Linked Data is about the publication and the updating of RDF data on the web using HTTP, where the RDF describes the state of a resource.

The high-level idea of the synthesis, which is described in the subsections of Section 4.2, is to (1) define RDF model theory for Linked Data using RDF Datasets, (2) define ASM functions $statement(\cdot,\cdot,\cdot)$ and $quad(\cdot,\cdot,\cdot,\cdot)$ for Linked Data/RDF Datasets and the functions' interpretations, (3) define an ASM transition function for Linked Data/RDF Datasets using rules with HTTP requests in the head (4) define how the ASM evaluation of the $statement(\cdot,\cdot,\cdot)$ function and the ASM updates to the $quad(\cdot,\cdot,\cdot,\cdot)$ function can be done in accordance with the semantics of HTTP requests, and (5) use above definitions in the definition of an ASM.

4.2 Synthesis

In this section, we take the definitions from the simple interpretation from RDF model theory and use them, slightly amended for RDF Datasets, in the definition of an ASM. The synthesis allows us to specify the evolution of an RDF Dataset using ASM semantics. Note that the terms *vocabulary* and *interpretation* used in ASM and RDF model theory mean the same. The term *universe* in RDF maps to the term *super-universe* in ASM.

4.2.1 Define RDF model theory for Linked Data using RDF Datasets. We base our considerations on two semantics for RDF Datasets that have been discussed in Sections 3.2 and 3.5 of a note of the W3C RDF Working Group[18].

We start out with definitions to simplify our presentation: Like the W3C Recommendation on model-theoretic semantics for RDF[17], we use as vocabulary the infinite sets of all URIs $\mathcal{U}$ and Literals $\mathcal{L}$ in our definitions, in contrast to the fixed vocabulary used by the Linked Data sources under consideration. Correspondingly, *IS* and *IL* interpret all $\mathcal{U}$ and $\mathcal{L}$, and *IR* has corresponding elements. We point to[17] for how to amend the definitions to finite vocabularies. To further ease the presentation, we assume ground graphs. Moreover, be $IP = IR$, such that we only need to address *IEXT* in our considerations. When working with RDF Datasets, we assume that the same elements of the vocabularies of the different graphs have the same referent when interpreted using *IS* and *IL*. We do not consider redirects in the definitions, although they can be layered on top by defining a function for the correspondence between a non-information resource and an information resource. In the definitions, we also assume all Linked Data sources to be in $\mathcal{N}$, the named graphs of our RDF Dataset, i. e. we assume web-completeness in the terminology of [15]. By restricting $\mathcal{N}$, we can implement other completeness classes.

Of the W3C RDF WG Note on RDF Dataset semantics[18], two sections are of particular use for our considerations:

3.5 "Named graph are in a particular relationship with what the graph name dereferences to" defines an interpretation for each graph in the RDF Dataset, hence we can define an extension function per dereferenceable source

$$IEXT_c := \textit{Extension function of the graph available at } \mathrm{c}$$

3.2 "Default graph as union or as merge" defines the default graph of an RDF Dataset as RDF union or merge of all named graphs in the RDF Dataset. As we assume ground graphs, there is no difference in RDF union or merge. Hence, we can define the extension function of the union as:

$$IEXT^{\text{UNION}}(p) := \bigcup_{c \in \mathcal{N}} IEXT_c(p)$$

The default graph can be regarded as knowledge of the client, e. g. knowledge that is not available at any $n \in \mathcal{N}$. The default graph can also be used in the case of higher-level entailment regimes for stating axiomatic triples in case they cannot be dereferenced from a Linked Data source.

4.2.2 Define ASM functions $quad(\cdot,\cdot,\cdot,\cdot)$ and $statement(\cdot,\cdot,\cdot)$ for Linked Data/RDF Datasets and the functions' interpretations. We define

$$quad(s,p,o,c) : IR \times IR \times IR \times \mathcal{N} \to \{\text{TRUE}, \text{UNDEF}\}$$

as the characteristic ASM function for $IEXT_c$. Moreover, be

$$statement(s,p,o) : IR \times IR \times IR \to \{\text{TRUE}, \text{UNDEF}\}$$

the characteristic ASM function for $IEXT^{\text{UNION}}$. The *statement* function has also been used in [36] to process RDF using Prolog.

So far, we have defined Linked Data as static ASM functions. Now, we proceed by introducing dynamics.

4.2.3 Define how the ASM transition function for Linked Data/RDF Datasets can be stated. The rules can be given as usual in ASM in rules of the form **if** *condition* **then** *update(s)* modifying the interpretations of function names. We use conjunctions of the $statement(\cdot,\cdot,\cdot)$ function for the conditions (reflecting BGP queries from SPARQL[14]) and the conjunctions of the $quad(\cdot,\cdot,\cdot,\cdot)$ function for the updates. Both the conditions and updates can contain variables.

4.2.4 Define how the ASM evaluation of the statement$(\cdot,\cdot,\cdot)$ function and the ASM updates to the quad$(\cdot,\cdot,\cdot,\cdot)$ function can be done in accordance with the semantics of HTTP requests. For the evaluation of the conditions, we consider the *statement*$(\cdot,\cdot,\cdot)$ function as making HTTP GET requests to all $n \in \mathcal{N}$ and to evaluate *statement*$(\cdot,\cdot,\cdot)$ like an external function in ASM. For the default graph, no request needs to be made. The updates are done using the *quad*$(\cdot,\cdot,\cdot,\cdot)$ function, which takes an additional c as parameter. The updates correspond to HTTP PUT, POST, DELETE requests.

We perform the requests ordered in accordance with ASM steps, i. e. we first evaluate all HTTP GET requests, and collect the updates that are mandated by the transition functions. After we collected the updates to be performed, we carry out the corresponding requests in bulk, i. e. multiple HTTP PUT, POST, DELETE requests at a time. After all updates in the form of HTTP PUT, POST, DELETE requests have been performed, we continue with the next ASM step. The result of the HTTP PUT, POST, DELETE requests is dependent on server implementations, see Section 4.3.

4.2.5 Define the ASM. We define the ASM for Linked Data as:

$$ASM := (\Upsilon, X, \mathcal{I}_0, T)$$

The vocabulary Υ contains all RDF names (all URIs and Literals), the boolean operator $\wedge$, and constant names. We omit (1) the boolean name `false` and the boolean operator $\neg$ because of the open world assumption, which is typically made on the Semantic Web[20], and (2) the operator $\vee$, because in a BGP, conditions are only connected using conjunctions. Disjunctions can be stated by using multiple rules. We add the function names *statement* and *quad*:

$$\Upsilon := \mathcal{U} \cup \mathcal{L} \cup \{\texttt{true}, \texttt{undef}\} \cup \{\wedge\} \cup \{quad, statement\}$$

The super-universe X is a set that contains elements for all RDF Graphs, and functions $X^n \to X$ including the boolean *and*. To be able to visually tell the elements of Υ and X from each other, we use typewriter font for the nullary function names and small caps for their counterparts in the super-universe.

$$X := IR \cup IP \cup \{\textsc{true}, \textsc{undef}\} \cup \{f | f : X^n \to X\}$$

The interpretation $\mathcal{I}_t$ of an element y of the vocabulary Υ at time t, consists of $IS(\cdot)$ and $IL(\cdot)$ from RDF model-theoretic semantics for the RDF names, mappings for the boolean `true`, and `undef`, and functions for *quad*$(\cdot,\cdot,\cdot,\cdot)$ and *statement*$(\cdot,\cdot,\cdot)$, and $\wedge$:

$$\mathcal{I}_t(y) := \begin{cases} IS(y) & \text{if } y \in \mathcal{U} \\ IL(y) & \text{if } y \in \mathcal{L} \\ \textsc{true} & \text{if } y = \texttt{true} \\ \textsc{undef} & \text{if } y = \texttt{undef} \\ \in \{f | f : X^n \to X\} & \text{if } y \in \{quad, statement, \wedge\} \end{cases}$$

4.3 Linked Data Servers

Although our formalisation is made for the user agent, we show for completeness the server behaviour, which reflects the update of the *quad*$(\cdot,\cdot,\cdot,\cdot)$ function for PUT, POST, and DELETE requests: A server processing a PUT request to the target information resource `http://localhost/Lighting_1F_M59` with the body

```
<http://localhost/Lighting_1F_M59#it> rdf:value "on".
```

[20] `http://www.w3.org/TR/owl2-primer/`

sets

$$quad(\cdot,\cdot,\cdot,\texttt{http://localhost/Lighting\_1F\_M59})$$

to `true` for the argument

$$(\texttt{<http://localhost/Lighting\_1F\_M59\#it>}, \texttt{rdf:value}, \texttt{"on"})$$

and `undef` for all other triples. A server processing a POST request with the target of a LDP container, e. g. `http://localhost/ldbbc/` and the RDF payload *payl* first determines a new URI *sub* that is subordinate to the container resource, then defines a new graph by relativising all URIs in the *payl* against *sub* and PUTs the new graph at *sub* into the RDF Dataset. Moreover, a membership triple for *sub* is added to the representation of the container resource.

4.4 Operational Semantics for the Condition-Action Rule Language

In this section, we give operational semantics to our condition-action rule language using the synthesis. We use the program in Figure 1 as example: The program consists of RDF (1), derivations (2), and requests (3+4), the latter to interact with the environment. Remember that in an ASM step, updates are executed after the update set has been constructed entirely. To construct the update set, all rules have to be evaluated on the current state.

In our language, information about the current system state is given by RDF, by evaluating all derivations, and by evaluating all GET requests on the Linked Data sources until the fixpoint has been calculated. In our example, we know from the RDF (1) that the light `IBM_B3:Lighting_1F_M59` has the `ssn:Property` `http://localhost/Lighting_1FM59#it`. The derivation rule (2) adds the inverse link to the knowledge. With the inverse information available, the conditional GET request is triggered (3). The data obtained using the GET request is added to the knowledge, e. g. that the light is off. No further knowledge can be derived using rules, so the fixpoint is calculated. The updates in our language are the unsafe requests. The conditional PUT, POST, DELETE requests (4) are collected during the information gathering about the current state, and are executed after the fixpoint calculation. Note that using conditional requests, we can do hypermedia-style link following. In our example, the condition for the last rule holds in the knowledge after the fixpoint calculation. Hence, the light is turned on.

To execute programs in our language along the ASM step semantics, an interpreter operates in nested loops (see Figure 2 for its algorithm). The algorithm takes as input a rule program and emits a sequence of sets of safe and unsafe requests, depending on the available state information. Note that ASM steps in combination with GET requests implement polling.

4.5 Discussion: Computation, ASMs, Simple Reflex Agents, and Linked Data

ASMs are a model of computation, i. e. one can specify arbitrary computation using the model, e. g. algorithms and programs. In the previous section, we defined an ASM view on Linked Data. Hence, we now can specify arbitrary computation with the state of the computation represented in Linked Data. Our ASM-based formalisation of Linked Data is based on the assumption that we have exclusive read and write access to all relevant URIs during

```
Require: assertions   ▷ Triples to be asserted in every ASM step
Require: rules                    ▷ Derivation and request rules
  var unsafeRequests: set<request>
  var data, oldData: set<triple>
  var fixpointReached: boolean
  while true do                              ▷ Loop of the ASM steps
    unsafeRequests.clear()
    data.clear()
    data.add(assertions)
    repeat                    ▷ Loop for determining the update set
      fixpointReached <- true
      for rule : rules do
        if rule.matches(data) then ▷ Also empty rule bodies
          oldData = data.copy()
          if rule.type==derivation then
            data.add(rule.match(data).data)
          else     ▷ So the rule must be an interaction rule
            if rule.match(data).request.type==GET then
              data.add(rule.match(data).request.execute())
            else
              unsafeRequests.add(rule.match(data).request)
            end if
          end if
          if ! data.copy().remove(oldData).isEmpty() then
            fixpointReached <- false
          end if
        end if
      end for
    until fixpointReached
    for request : unsafeRequests do   ▷ Enacting the update set
      request.execute()
    end for
  end while
```

Figure 2: The nested loops for the ASM-based operational semantics for the rule language.

an ASM step. When generalising to a setting with multiple agents, ASMs typically assume circulating exclusive access to the system state. As there is no central control in a web setting, we cannot circulate the access right. Thus, there is no general solution to the problem of concurrent access, but a number of remedies: (1) in a realistic setting, we do not have the entire Linked Data in our system state (i. e. web-completeness in the terminology of [15]), but a considerably smaller subset. (2) on the web, most access is typically reading, and only little is writing [32]. (3) if the servers use ETags[21], a client can send conditionally writing requests, which fail if a concurrent update has happened. (4) with sufficiently fast data processing, the problem can be mitigated.

Using our ASM view on Linked Data, we can also implement simple reflex agents for Linked Data, the simplest agent type in Russell and Norvig [34]. Simple reflex agents perceive the environment and choose their action by matching condition-action rules on the perceived environment. Then, the agent carries out the action and repeats. This sense-act cycle aligns with an ASM step of first, matching rules and second, acting in the form of updates.

[21] `http://www.ietf.org/rfc/rfc7232.txt`

4.6 Requirements for a Linked Data User Agent Specification Language

What current languages to specify queries and updates are missing is a way of expressing the transition function T. In ASM, T is given using rules in the form **if** *condition* **then** *function update(s)*. T is then executed in ASM steps. Thus, we need a language

(1) that allows to express conditions on state information
(2) that allows to define updates that send state information,
(3) whose semantics adhere to ASM steps, i. e. one repeatedly first evaluates all conditions of all rules and collects all updates, and then one evaluates all updates in bulk.

We chose N3 as rule language in the context of RDF data and gave ASM-based semantics to N3. Other approaches including SPARQL updates[22] or RUL [28] as very elaborate ways to address point 1. They do not implement server interaction using exchange of state representation, but RPC-style interaction (point 2). They also do not implement bulk updates, as required by ASM steps (point 3).

5 EVALUATION

To describe our evaluation, we start out with defining the data and different scenarios for data access. Next, we present different workloads. Then, we describe experimental setup. Next, we present and discuss our results. Last, we showcase applications of our approach.

5.1 Data

We evaluate our approach in the building automation setting of Section 2. We use the building's lighting subsystem, as lamps and switches can be regarded as having a discrete state. We use building 3[9], as the building's description includes a lighting subsystem.

The core classes of the Brick ontology are depicted in Figure 3. The first part of the ontology, `Location` and subclasses, allows for describing a building in terms of the subdivisions provided by the brickwork, e. g. floors and rooms. The second part of the ontology, `Equipment` and subclasses, allows for describing different systems that run the building, where we focus on the lighting system in this paper. Other systems include water, and HVAC (heating, ventilation, and air conditioning). The third part of the ontology, `Point` and subclasses, allow to talk about sensors and actuators in the building.

Building 3 is box-shaped with its long side oriented from north to south. In the RDF description, the building is subdivided into floors and wings. The rooms are assigned to their corresponding floor and wing (also has-part relationships). The lights are directly assigned to rooms or wings. A light system can consist in (1) an occupancy sensor that determines whether there are people in its vicinity, (2) a luminance command, in other words, a switch to control the lights, and (3) a luminance sensor that we consider to be triggered by daylight. We provide basic statistics in Table 1.

We bring the static building description "to life" by adding `ssn:hasProperty` links to dynamic Linked Data resources on localhost representing switches, occupancy sensors, luminance sensors, and lights. The state of each resource relevant for the evaluation is on/off for the lights, and a numeric value for the luminance sensors.

[22] `http://www.w3.org/TR/sparql11-update/`

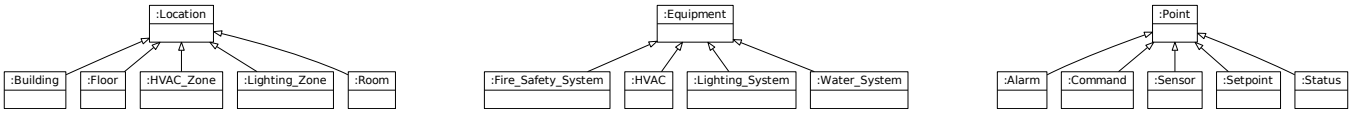

Figure 3: Excerpts from the Brick Ontology as UML Class Diagram. UML class inheritance denotes RDFS sub class relations.

5.2 Workload

In our evaluation, we implement different user agents that provide automation. We run the rules on different parts of the buildings to investigate how the approach scales. In terms of data access, i. e. the evaluation of the *quad* function, there are two extremes:

D1 The whole building description is available as one file, which is loaded from one single Linked Data source on the network

D2 The building description is available subdivided into one Linked Data source per resource in the building. The Linked Data sources can thus be accessed as required following links.

We scale the scenario from one room to the whole building.

In terms of automation workloads, we scale both the rule complexity, and energy efficiency and comfort of the building. According to UNEP [39], lighting is the second-highest energy consumer in commercial buildings, optimised control can yield high savings.

W1 Turn all light switches on (no conditions / baseline; 9 rules)

W2 Working hours (conditions and another source, a Linked Data clock / clock-based straight-forward control; 46 rules): The lights are on per default during working hours.

W3 Weather API (more complex rules, another source / raising energy efficiency; 20 rules): We turn the lights on only if a lack of sunlight indicates illumination.

W4 Luminance sensor (numerical computations / further raising efficiency; 34 rules): We consider luminance sensor values in the rooms to determine whether the lights should be on.

W5 Luminance sensor w/room-individual thresholds (more complex computation / raising individual comfort; 19 rules): We assign an individual light threshold per room.

5.3 Experimental Setup

We use the engine Linked Data-Fu version 0.9.12[23] to run the rule programs. We use LDBBC version 0.0.6[24] as LDP Container to serve the static building data. We implement the following dynamic sources: (1) an RDF weather API built from a sample file from OpenWeatherMap[25], a JSON-LD context, and a sunset/sunrise simulation for Dublin, and (2) luminance sensor readings according to time of day and season. We run the experiments on a laptop with an Intel Core i7-5600U CPU, 12 GB of RAM, and Ubuntu Linux 17.04. After the dynamic sources simulated one year, i. e. one minute in wall-clock time, we stop one experiment leading to repetition counts of up to 7'500 depending on the runtime of one ASM step. The evaluation system including data and rules can be found online[26].

5.4 Results and Discussion

We present the results for D1 in Table 2 and for D2 in Table 3. We report median values as due to firing many requests, there are high outliers in the 99 % percentile, which spoil mean and standard deviation. The median, however, is stable. The order of magnitude of the measurements in varies in Table 3, in contrast to Table 2. This is owed to the fact that each ASM step, regardless of the data that is actually needed, the whole building description has to be transferred in D1, which is 2.3 MB per transfer. Compared to that, the link following approach D2 can access the building data as required. This fine granularity comes at the price of additional requests, which make the processing of the whole building more expensive. The drop in time between W2 and W3 can be explained by the reasoning that is employed in W2 to check whether the current day and hour is a working hour, which is not needed in W3. The increase between W3 and W4 underestimates the real cost of the computations, as we go from all lights in W3 to only those lights with sensors in W4, cf. Table 1. The drop between W4 to W5 can be explained by the smaller number of lights with luminance sensors. The rise in time for the workloads in Table 2 is due to that different workloads need to access different amounts of data.

5.5 Applicability

In this section, we give examples of how we applied our approach.

Composition of RESTful services to VR systems We used our approach to connect different parts of Virtual Reality systems: In the i-VISION project, we connected a flight simulator to a workflow analysis software [22] (ca. 90 rules). In a demo [24], we connected a Microsoft Kinect sensor and APIs from the Web to a 3D engine (ca. 60 rules), where the interpreter running the rules performed about 30 ASM steps per second (i. e. the refresh rate of the Kinect sensor).

Specification of Operational Semantics Similar to previous applications of ASMs to define the operational semantics of programming languages, we currently work on defining the operational semantics of a workflow language using our approach (ca. 30 rules).

Turing-Completeness To show the expressiveness of our approach, we implemented a Turing Machine using Linked Data and four rules. The rules together with instructions for the set-up can be found online[27]. While the Turing-completeness is, of course, an inherent property of the ASM part of the approach, it persists despite the restrictions we imposed to make ASM fit Linked Data. All parts of our approach are necessary: Read-Write Linked Data for tape and machine state, and request rules executed in ASM steps.

6 RELATED WORK

In this section, we discuss related work subdivided by topic.

One-step updates of semantic data In Semantic Data Management, people studied updates to RDF Graphs, e. g. schema-preserving updates [28], and standardised updating the data in a triple store using SPARQL[22]. To update Linked Data using HTTP has been proposed in a Design Issues article [4] and detailed out in a

[23] http://linked-data-fu.github.io/
[24] http://github.com/kaefer3000/ldbbc/
[25] http://www.openweathermap.org/
[26] http://github.com/kaefer3000/rwld-brick-benchmark/
[27] http://github.com/kaefer3000/ldf-turingmachine

Table 1: Basic counts for building 3 and the benchmark.

Rooms	281
Floors	2
Wings	3
Lights w/occupancy sensors	156
Lights w/luminance commands	126
Lights w/luminance sensors	60
Triples in IBM_B3.ttl	24947
Resources in the LDP Container	3281
Dynamic resources	551

Table 2: Median time [ms] for one ASM step in scenario D1, data access from one single Linked Data source.

Rooms	W1	W2	W3	W4	W5
1	484	572	510	554	561
5	480	582	501	574	582
10	498	584	529	605	618
20	537	631	562	719	687
First Floor	563	629	590	750	728
Wing 42	527	595	550	651	604
Building 3	605	734	613	794	788

Table 3: Median time [ms] for one ASM step in scenario D2, data access via Linked Data + link following.

Rooms	W1	W2	W3	W4	W5
1	8	8	8	8	8
5	40	38	38	40	40
10	85	80	79	88	88
20	259	238	228	320	268
First Floor	938	1690	891	1063	1048
Wing 42	1435	1427	1371	1664	1408
Building 3	2442	2187	2192	2542	2497

W3C recommendation[6]. In previous work, we proposed a language and an interpreter for interacting with Read-Write Linked Data [37]. Those works lack the notion of ASM steps.

Automation on the web Unlike automation systems such as [42], IFTTT[28], and Arktik[29], we do not assume centralised information and event processing, but decentralised information and rule evaluation on state information. Ripple [35], and [27] allow for read-only scripting and programming with Linked Data. We also cover writing. Unlike the multi-agent approach of [8], we work without a central platform for discovery and for distributing notifications.

Descriptive works We monitored [23], analysed [20], and formally described [14] the dynamics of Linked Data. Those works are insufficient to specify computation.

Web Service descriptions A focus of Semantic Web Services are service descriptions and the processing of such descriptions, with some work on operational semantics [1], [29], [6], [33], mainly used for analysing composed services, instead of execution.

Descriptions/formalisations and HTTP Lately, descriptions in a Semantic Web Services fashion have gained traction again, especially on the Internet of Things [7], and even when working with HTTP [26], [31], [18]. The descriptions express in a standardised manner what you can do with a (server) API. Instead, we formally address user agent specifications. Our approach can be extended with planning (e. g. [43], [33]) if service descriptions are available. Thus, our Linked Data-based formalisation (which respects the HTTP message semantics) gives an alternative to BPEL-based approaches to service composition and execution.

Application of ASMs in Semantic Technologies Before the advent of RDF and Linked Data, the authors of [41] compared formalisms to specify dynamics for knowledge bases. Abstract State Machines (ASMs) stood out for their simplicity and ease of operationalisation. ASMs have also been used to describe the communication of services in a choreography in WSMO [33].

Combination of static and dynamic modelling The authors of [19] executed queries in linear temporal logic (LTL) over evolving ontologies, which is complementary to our approach. Similar to our work, the author of [25] combined approaches for static and dynamic modelling for object-oriented modelling. While Graph Rewriting could serve as an alternative to ASM for the specification of the evolution of RDF Graphs, we think ASM is more compatible to the Semantic Web stack, as both are based on first-order logic.

[28] http://ifttt.com/
[29] http://arktik.io/

7 CONCLUSION AND FUTURE WORK

We have presented a formal approach for capturing the dynamics of Linked Data with the aim of specifying user agents. To this end, we gave a synthesis of RDF model theory, ASMs, and HTTP, and described requirements for implementing the synthesis. We applied the approach to give semantics to a rule language such that we can specify Linked Data user agents. We presented application scenarios and showed our approach to be Turing complete (Section 5.5). We evaluated our approach in a Smart Building setting.

In the paper, we took the first step towards autonomous agents that operate on Linked Data: We presented a formal basis for the execution of agents. Still, there are more steps to take, a formal notion of (1) internal state in the agent, and (2) a notion of goals and capabilities, which we will address in future work.

Yet, we believe our work can already be applied today: Although experts built the applications in Section 5, the experiments of van Kleek et al. [42] or the success of IFTTT [40] show that rules are a way of programming that is highly relevant also for end-users.

ACKNOWLEDGMENTS

We thank Armin Haller for insights into ASMs in WSMO. This work is partially supported by the German federal ministry for education and research in AFAP, a Software Campus project (FKZ 01IS12051).

BIBLIOGRAPHY

[1] A Ankolekar, F Huch, and KP Sycara. 2002. Concurrent semantics for the web services specification language DAML-S. In *Proc. of the 5th Intl. Conf. on Coordination Models and Languages (Coordination)*.

[2] B Balaji et al. 2016. Brick: towards a unified metadata schema for buildings. In *Proc. of the 3rd Intl. Conf. on Systems for Energy-Efficient Built Environments (BuildSys)*. ACM.

[3] T Berners-Lee. 2006. Linked Data. Design Issues. http://www.w3.org/DesignIssues/LinkedData.

[4] T Berners-Lee. 2009. Read-write Linked Data. Design Issues. http://www.w3.org/DesignIssues/ReadWriteLinkedData.

[5] P Bouquet, C Ghidini, and L Serafini. 2009. Querying the Web of Data: A formal approach. In *Proc. of the 4th Asian Semantic Web Conf. (ASWC)*.

[6] S Chandrasekaran, JA Miller, GA Silver, IB Arpinar, and AP Sheth. 2003. Performance analysis and simulation of composite web services. *Electronic Markets*, 13, 2.

[7] V Charpenay, S Käbisch, and H Kosch. 2016. Introducing thing descriptions and interactions. In *Proc. of the 1st WS on SemanticWeb technologies for the Internet of Things (SWIT)*.

[8] A Ciortea, O Boissier, A Zimmermann, and AM Florea. 2016. Responsive decentralized composition of service mashups for the internet of things. In *Proceedings of the 6th International Conference on the Internet of Things (IoT)*.

[9] R Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis. University of California, Irvine, USA.

[10] MP Gallaher, AC O'Connor, JL Dettbarn Jr., and LT Gilday. 2004. Cost analysis of inadequate interoperability in the us capital facilities industry. NIST, (2004).

[11] Y Gurevich. 1995. Evolving algebras 1993: lipari guide. In *Specification and Validation Methods*. E Börger, editor. OUP.

[12] A Harth, K Hose, M Karnstedt, A Polleres, K Sattler, and J Umbrich. 2010. Data summaries for on-demand queries over Linked Data. In *Proc. of the 19th Intl. Conf. on World Wide Web (WWW)*.

[13] A Harth and T Käfer. 2017. Specifying and executing application behaviour with condition-request rules. In *Proc. of the WS on Decentralizing the Semantic Web at the 16th Intl. Semantic Web Conf. (ISWC)*.

[14] A Harth and T Käfer. 2016. Towards specification and execution of linked systems. In *Proc. of the 28th WS Grundlagen von Datenbanken (GvD)*. GI.

[15] A Harth and S Speiser. 2012. On completeness classes for query evaluation on linked data. In *Proc. of the 26th Conf. on Artificial Intelligence (AAAI)*.

[16] O Hartig, C Bizer, and JC Freytag. 2009. Executing SPARQL queries over the web of Linked Data. In *Proc. of the 8th Intl. Semantic Web Conf. (ISWC)*.

[17] O Hartig and J Pérez. 2016. LDQL: A query language for the web of Linked Data. *Web Semantics*, 41.

[18] AG Hernández and MNM García. 2010. A formal definition of RESTful semantic web services. In *Proc. of the First Intl. WS on RESTful Design (WS-REST)*.

[19] Z Huang and H Stuckenschmidt. 2005. Reasoning with multi-version ontologies: A temporal logic approach. In *Proc. of the 4th Intl. Semantic Web Conf. (ISWC)*.

[20] T Käfer, A Abdelrahman, J Umbrich, P O'Byrne, and A Hogan. 2013. Observing Linked Data dynamics. In *Proc. of the 10th European Semantic Web Conf. (ESWC)*.

[22] T Käfer, A Harth, and S Mamessier. 2016. Towards declarative programming and querying in a distributed CPS: the i-VISION case. In *Proc. of the 2nd Intl. WS on modelling, analysis, and control of complex CPS (CPSData)*.

[23] T Käfer, J Umbrich, A Hogan, and A Polleres. 2012. Towards a dynamic Linked Data observatory. In *Proc. of the 5th WS on Linked Data on the Web (LDOW)*.

[24] FL Keppmann, T Käfer, S Stadtmüller, R Schubotz, and A Harth. 2014. High performance Linked Data processing for Virtual Reality environments. In *Proc. of Posters & Demos at the 13th Intl. Semantic Web Conf. (ISWC)*.

[25] M Kifer. 1995. Deductive and object data languages: A quest for integration. In *Proc. of the 4th Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*.

[26] M Lanthaler and C Gütl. 2013. Hydra: A vocabulary for hypermedia-driven web APIs. In *Proc. of the 6th WS on Linked Data on the Web (LDOW)*.

[27] M Leinberger, R Lämmel, and S Staab. 2017. The essence of functional programming on semantic data. In *Proc. of the 26th European Symp. on Programming (ESOP)*.

[28] M Magiridou, S Sahtouris, V Christophides, and M Koubarakis. 2005. RUL: A declarative update language for RDF. In *Proc. of the 4th Intl. Semantic Web Conf. (ISWC)*.

[29] S Narayanan and SA McIlraith. 2002. Simulation, verification and automated composition of web services. In *Proc. of the 11th Intl. Conf. on World Wide Web (WWW)*.

[30] KR Page, D de Roure, and K Martinez. 2011. REST and Linked Data: a match made for domain driven development? In *Proc. of the 2nd Intl. WS on RESTful Design (WS-REST)*.

[31] C Pautasso, A Ivanchikj, and S Schreier. 2016. A pattern language for RESTful conversations. In *Proc. of the 21st European Conf. on Pattern Languages of Programs (EuroPLoP)*.

[32] C Pautasso and O Zimmermann. 2018. The web as a software connector. *IEEE Software*, 35, 1.

[33] D Roman et al. 2005. Web service modeling ontology. *Applied Ontology*, 1, 1.

[34] SJ Russell and P Norvig. 1995. *Artificial intelligence - a modern approach: the intelligent agent book*. Prentice Hall.

[35] J Shinavier. 2007. Functional programs as Linked Data. In *Proc. of the WS on Scripting for the Semantic Web (SFSW)*.

[36] M Sintek and S Decker. 2002. TRIPLE – A query, inference, and transformation language for the semantic web. In *Proc. of the 1st Intl. Semantic Web Conf. (ISWC)*.

[37] S Stadtmüller, S Speiser, A Harth, and R Studer. 2013. Data-fu: a language and an interpreter for interaction with Read/Write Linked Data. In *Proc. of the 22nd Intl. Conf. on World Wide Web (WWW)*.

[38] J Umbrich, A Hogan, A Polleres, and S Decker. 2015. Link traversal querying for a diverse web of data. *Semantic Web*, 6, 6.

[39] UNEP and SKANSKA. 2009. Energy efficiency in buildings.

[40] B Ur, MPY Ho, S Brawner, J Lee, S Mennicken, N Picard, D Schulze, and ML Littman. 2016. Trigger-action programming in the wild: an analysis of 200,000 IFTTT recipes. In *Proc. of the 34th Conf. on Human Factors in Computing Systems (CHI)*.

[41] P van Eck, J Engelfriet, D Fensel, F van Harmelen, Y Venema, and M Willems. 2001. A survey of languages for specifying dynamics: A knowledge engineering perspective. *Transactions on Knowledge and Data Engineering (TKDE)*, 13, 3.

[42] M van Kleek, B Moore, DR Karger, P André, and m. c. schraefel. 2010. Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web. In *Proc. of the 19th Intl. Conf. on World Wide Web (WWW)*.

[43] R Verborgh, T Steiner, D van Deursen, S Coppens, JG Vallés, and R van de Walle. 2012. Functional descriptions as the bridge between hypermedia APIs and the semantic web. In *Proc. of the 3rd Intl. WS on RESTful Design (WS-REST)*.