Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data

Steffen Stadtmüller
Institutes AIFB, KSRI
Karlsruhe Institute of
Technology (KIT), Germany
steffen.stadtmueller@kit.edu

Sebastian Speiser
Institutes AIFB, KSRI
Karlsruhe Institute of
Technology (KIT), Germany
speiser@kit.edu

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

Rudi Studer
Institutes AIFB, KSRI
Karlsruhe Institute of
Technology (KIT), Germany
studer@kit.edu

ABSTRACT

An increasing amount of applications build their functionality on the utilisation and manipulation of web resources. Consequently REST gains popularity with a resource-centric interaction architecture that draws its flexibility from links between resources. Linked Data offers a uniform data model for REST with self-descriptive resources that can be leveraged to avoid a manual ad-hoc development of web-based applications. For declaratively specifying interactions between web resources we introduce *Data-Fu*, a lightweight declarative rule language with state transition systems as formal grounding. Data-Fu enables the development of data-driven applications that facilitate the RESTful manipulation of read/write Linked Data resources. Furthermore, we describe an interpreter for Data-Fu as a general purpose engine that allows to perform described interactions with web resources by orders of magnitude faster than a comparable Linked Data processor.

Categories and Subject Descriptors

H.5.4 [**Hypertext/Hypermedia**]: Architectures

General Terms

Languages, Performance

Keywords

REST; Linked Data; Web Interaction; Rule Language; Interpreter

1. INTRODUCTION

There is a growing offer of functionality via web APIs[1]. Increased value comes from combining data from multiple sources and functionality from multiple providers. The importance of such compositions is reflected in the constant growth of mashups – small programs that combine multiple web APIs [33]. There is a strong movement in the web community toward a resource-oriented model of services based on Representational State Transfer (REST [11]). Flexibility, adaptivity and robustness are the major objectives of REST and are particularly useful for software architectures in distributed data-driven environments such as the web [22]. However, data sources and APIs are published according to different interaction models and with interfaces using non-aligned vocabularies, which makes writing programs that integrate offers from multiple providers a tedious task.

The goal of our work is to provide a declarative means to specify interactions between data and functionality from multiple providers. Such declarative specifications provide a modular way of composing the functionality of multiple APIs. Also, declarative methods allow for automatically optimising a program and parallelising the execution.

In a REST architecture, client and server are supposed to form a contract with content negotiation, not only on the data format but implicitly also on the semantics of the communicated data, i.e., an agreement on how the data have to be interpreted [32]. Since the agreement on the semantics is only implicit, programmers developing client applications have to manually gain a deep understanding of the provided data, often based on natural text descriptions. The combination of RESTful resources originating from different providers suffers particularly from the necessary manual effort to use and combine them. The reliance on natural language descriptions of APIs has led to mashup designs in which programmers are forced to write glue code with little or no automation and to manually consolidate and integrate the exchanged data.

Linked Data unifies a standardised interaction model with the possibility to align vocabularies using RDF, RDFS and OWL. However, the interactions are currently constrained to simple data retrieval. Following the motivation to look beyond the exposure of fixed datasets, the extension of Linked Data with REST technologies has been explored [5, 34] and

[1] Alone `http://programmableweb.com/` lists 7,991 APIs on November 24th 2012, which is almost twice the number from one year earlier.

WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2035-1/13/05.

led recently to the establishment of the *Linked Data Platform*[2] W3C working group.

Several existing approaches recognise the value of combining RESTful services and Linked Data [17, 26, 30]. In this paper, we go one step further and propose *Data-Fu*, a data- and resource-driven programming approach leveraging the combination of REST with Linked Data. Data-Fu enables the development of applications built on semantic web resources with a declarative rule language. The main goal of Data-Fu is to minimise the manual effort to develop web-based applications and the preservation of loose coupling by

- leveraging links between resources provided by Linked Data, and
- specifying desired interactions dependent on resource states, which is enabled by a uniform state description format, i.e., RDF.

A further requirement for our programming approach in a web-based environment is a fast and scalable execution of the applications. While there has been recent work on extending the Map/Reduce model for data-driven processing [15, 4], these approaches are geared towards deployment in data centers. In contrast, our approach operates on the networked open web.

This paper is based on a previous publication on a data-driven programming model for the web [27] and describes

- how self-descriptive resources can be designed to enable loosely coupled clients (Section 4.1);
- a service model for REST based on state transition systems as formal grounding (Section 4.2);
- the Data-Fu language, a declarative rule-based execution language to allow an intuitive specification of the interaction with resources from different providers (Section 5);
- an execution engine as an artefact to perform the defined interactions in a scalable manner (Section 6).

We provide a motivating scenario in Section 2. We evaluate our approach in two ways: (i) we describe throughout the paper how our motivating scenario can be realised with Data-Fu; and (ii) we conduct performance experiments with the Data-Fu interpreter in Section 7. Section 8 covers existing work. We conclude in Section 9.

2. MOTIVATING SCENARIO

In our scenario, we consider the Acme corporation, a consumer goods producer, that aims at extending their social media activities to a broader range of dissemination channels (for more on multi-channel communication see [7]). Acme's marketing department observes that while the number of potential channels is constantly increasing, the channels can be broadly categorised into micro blog services and social networks. Information about new products, special offers, and other news should be disseminated in the following ways: (i) posts on the company's micro blogs; and (ii) messages to social network users who are followers of the company.

We assume that the dissemination channels offer Linked APIs, i.e., resources are exposed that offer read/write Linked Data functionality.[3]

Table 1: URI prefixes used throughout this paper

Prefix	IRI
`acme:`	`http://acme.example.org/company/`
`p:`	`http://acme.example.org/vocabulary/`
`sna:`	`http://sna.example.org/lapi/`
`snb:`	`http://snb.example.org/rest/`
`mb:`	`http://mb.example.org/interface/`

The marketing department orders a system from Acme's IT that manages the dissemination channels and automatically disseminates a post to all available channels either as a micro blog entry or as a personal message. Initially the micro blog service MB and the social network SNA have to be supported. Marketing will supply their posts in an Acme-specific vocabulary as so-called InfoItems.

After a while, the marketing department decides to add the new social network SNB as a dissemination channel, which requires two steps: (i) the IT department extends the dissemination system to support the interface of SNB; and (ii) the marketing department adds Acme's identity in SNB to the dissemination channels.

Throughout the paper, we will illustrate our technical contributions by realising bits and pieces of the proposed scenario. When modeling services and interactions, we will use a number of URI prefixes for brevity that are either common[4] or listed in Table 1.

3. BACKGROUND

According to the Richardson maturity model [24] REST is identified as the interaction between a client and a server based on three principles:

- The use of URI-identified resources.
- The use of a constrained set of operations, i.e., the HTTP methods, to access and manipulate resource states.
- The application of hypermedia controls, i.e., the data representing a resource contains links to other resources. Links allow a client to navigate from one resource to another during his interaction.

The idea behind REST is that applications, i.e., clients, using functionalities provided on the web, i.e., APIs, are not based on the call of API-specific operations or procedures but rather on the direct manipulation of exposed resource representations or the creation of new resource representations. A resource can be a real world object or a data object on the web. The representation of a resource details the current state of the resource. A manipulation of the state representation implies that the represented resource is manipulated accordingly. For brevity in this paper we often talk about "the manipulation of a resource", when we actually mean "the manipulation of the state representation of a resource and the subsequent change of the resource itself".

The flexibility of REST results from the idea that client applications do not have to know about all necessary resources. The retrievable representations of some known resources contain links to other resources, that the client can

[2] `http://www.w3.org/2012/ldp/charter`

[3] If there is no Linked API available, the conventional APIs can be easily wrapped to consume and produce RDF, see, e.g., [29, 17]. Wrapping APIs is out of scope of this paper.

[4] See `http://prefix.cc/` for their full URIs, accessed on November 22nd 2012.

discover during runtime. Clients can use such discovered resources to perform further interaction steps.

The Linked Data design principles[5] also address the use of URI-identified resources and their interlinkage. However Linked Data is so far only concerned with the provisioning and retrieval of data. In contrast to REST, Linked Data does distinguish explicitly between URI-identified objects (i.e., non-information resources) and their data representation (information resources). An extension of Linked Data with REST to allow for resource manipulation leads to read/write Linked Data, i.e., information resources can be accessed and manipulated. REST furthermore implies that a change of an information resource implies a change in the corresponding non-information resource.

The development of applications in a REST framework is especially challenging, since the links between resources and the resource states can only be determined during runtime, however, programmers have to specify their desired interactions at design time.

Traditional service composition approaches that aim to decrease the manual effort to use web-offered functionality lead to a tight coupling between client and server, i.e., they sacrifice flexibility and are prone to failures due to server-side changes. Traditional composition approaches often fail to leverage links between resources and do not provide straightforward mechanisms to dynamically react to state changes of resources. The reaction on state changes becomes especially important in a distributed programming environment, since a client cannot ex ante predict the influence of other clients on the resources, i.e., REST does not allow a client to make assumptions on resource states.

4. READ/WRITE LINKED DATA

In this section, we describe our approach for modelling of RESTful services based on Linked Data. Our approach has two layers:

- Individual *Read/Write Linked Data Resources* with descriptions that allow predicting the effect of the execution of a functionality before invocation (Section 4.1);
- A formal *REST Service Model*. A single REST service can consist of several resources, potentially spread over different servers. The *service model* is the grounding for describing the interactions that are offered by the individual RESTful Linked Data resources and the overall service (Section 4.2).

4.1 Read/Write Linked Data Resources

In a RESTful interaction with Linked Data resources only the HTTP methods can be applied to the resources. The semantics of the HTTP methods itself is defined by the IETF[6] and do not need to be explicitly described.

Table 2 shows an overview of the most important HTTP methods. We can distinguish between safe and non-safe methods, where safe methods guarantee not to affect the current states of resources. Further, some of the methods require additional input data to be provided for their invocation. The communicated input data can be subject to requirements that need to be described to allow an automated interaction, e.g., the input data can be required to use a specific vocabulary. Furthermore, the effect of a non-safe method on the state of an addressed resource can depend on the input data. The dependency between communicated input and the resulting state of resources also needs to be described. Therefore, only the non-safe HTTP methods that require input data need further description mechanisms. Note, the POST method can also influence the states of not directly addressed resources. The precise effect of a POST depends on the resource, since POST allows to send input data to a data-handling process of a resource.

Table 2: Overview of HTTP methods

Method	Safe	Input required	Intuition
GET	x		Retrieve the current state of a resource.
OPTIONS	x		Retrieve a description of possible interactions.
DELETE			Delete a resource
PUT		x	Create or overwrite a resource with the submitted input.
POST		x	Send input as subordinate to a resource or submit input to a data-handling process.

The state of a Linked Data resource is expressed with RDF. It is sensible to serialise the input data in RDF as well, i.e., data that is submitted to resources to manipulate their state. To convey the resulting state change after application of a HTTP method we use RDF output messages. In previous work [20] we analysed the potential of graph patterns, based on the syntax of SPARQL[7], to describe required input as well as their relation to output messages. The resulting graph pattern descriptions are attached to the resource and can be retrieved via the *OPTIONS* method on the respective resource. Therefore the resources stay self-descriptive, i.e., their current state can be retrieved with *GET*, the possibilities to influence their state with *OPTIONS*.

Example. Acme's IT creates the resource `acme:Acme` representing Acme. A GET on `acme:Acme` returns the following initial description: `acme:Acme rdf:type p:Company .` The marketing department updates the `acme:Acme` resource with the dissemination channels SNA and MB by performing a PUT with the following input data:

```
acme:Acme  rdf:type        p:Company .
acme:Acme  p:dissChannel   sna:Acme, mb:Acme .
sna:Acme   rdf:type        p:SocialNetworkID .
mb:Acme    rdf:type        p:MicroBlogTimeline .
```

A subsequent GET on `acme:Acme` would result in exactly the description that marketing supplied with their PUT request.

A GET on `sna:Acme`, Acme's identifier in the social network SNA, would result in a description of Acme in SNA's vocabulary including its fans:

```
sna:Acme  rdf:type     sna:CommercialOrganisation .
sna:Acme  sna:founded  "11/20/2012" .
sna:Acme  sna:hasFan   sna:User1, sna:User2, . . . .
```

The resources representing users in the SNA network provide

[5] http://www.w3.org/DesignIssues/LinkedData.html

[6] http://www.ietf.org/rfc/rfc2616.txt

[7] http://www.w3.org/TR/rdf-sparql-query/#GraphPattern

functionality to send messages to the corresponding users. A POST can be employed to send a message to a user resource (e.g., to sna:User1). The input data for the POST contains its sna:sender and its sna:content, according to the description of the user resource that can be retrieved with an OPTIONS request:

```
INPUT:   ?m  rdf:type      sna:Message .
         ?m  sna:sender    ?s .
         ?m  sioc:content  ?c .
OUTPUT:  ?m  sna:sender    ?s .
         ?m  sioc:content  ?c .
         ?m  sna:receiver  sna:User1 .
```

Acme's timeline mb:Acme on the micro blogging service MB also supports the POST operation. Figure 1 illustrates the timeline resource mb:Acme of our example, with a set of entries in the current state and the graph pattern that describe how a new entry can be POSTed.

Applying a DELETE on a blog post, e.g., one that advertises an expired sale, does not require input; its effect is inherently defined by the method: the entry is erased.

4.2 REST Service Model

A REST service can be identified with the resources it exposes. An interaction within a REST architecture is based on the manipulation of the states of the exposed resources.

We develop a model, that allows to formalise the functionalities exposed by a REST API based on read/write Linked Data resources. A formal service model serves as rigorous specification of how the use of individual HTTP methods influences resource states and how these state changes are conveyed to interacting clients.

We model a Linked Data-based RESTful service as a REST state transition system (RSTS) similar to a state machine as defined by Lee and Varaiya [18]. The behavior of the clients themselves is not in the scope of this model, it rather formalises all possible interaction paths of a client with the resources.

Definition 1. *A REST state transition system (RSTS) is defined as a 5-tuple $RSTS = \{R, \Sigma, I, O, \delta\}$ with:*

- *A set of resources $R = \{r_1, r_2, ...\}$.*
- *A set of states $\Sigma = \{\sigma_1, ..., \sigma_m\}$. Each state $\sigma_k \in \Sigma$ of the RSTS is defined as the union of the states of all resources: $\sigma_k = \bigcup_{r_i \in R} \overline{r_i^k}$. The state of a single resource $r_i \in R$ in a state σ_k is given by its RDF representation $\overline{r_i^k} \in G$, where G is the set of all possible RDF graphs.*
- *An input alphabet $I = \{(r, \mu, g) : R \times M \times G\}$, where $M = \{GET, DELETE, PUT, POST\}$ is the set of the supported HTTP methods[8].*
- *An output alphabet $O = \{(c, o) : C \times G\}$, where C is the set of all HTTP status codes.*
- *An update function $\delta : \Sigma \times I \rightarrow \Sigma \times O$ that returns for a given state and input the resulting state and the output. We decompose δ into a state change function $\delta^s : \Sigma \times I \rightarrow \Sigma$ and an output function $\delta^o : \Sigma \times I \rightarrow O$, such that $\delta(\sigma, i) = (\delta^s(\sigma, i), \delta^o(\sigma, i))$. We define the state change function as*

$$\delta^s(\sigma_k, (r_i, \mu, g)) = \begin{cases} \sigma_k, & \text{if } \mu = GET \\ \sigma_k \setminus \{\overline{r_i^k}\}, & \text{if } \mu = DELETE \\ (\sigma_k \setminus \{\overline{r_i^k}\}) \cup g, & \text{if } \mu = PUT \\ \mathsf{post}_i(\sigma_k, g), & \text{if } \mu = POST, \end{cases}$$

where the function post_i encapsulates the resource specific behaviour of a POST request, as described by its INPUT/OUTPUT patterns, which can be obtained via an OPTIONS request on the resource. Let σ_l be the new state as defined by δ^s, we define the output function as

$$\delta^o(\sigma_k, (r_i, \mu, g)) = \begin{cases} (c\,, \overline{r_i^k}), & \text{if } \mu = GET \\ (c\,, \emptyset), & \text{if } \mu = DELETE \\ (c\,, \sigma_l \backslash \sigma_k), & \text{if } \mu = PUT \\ (c\,, \sigma_l \backslash \sigma_k), & \text{if } \mu = POST. \end{cases}$$

A client interacting with a service modelled by an $RSTS = \{R, \Sigma, I, O, \delta\}$ creates an input $i = (r_i, \mu, g)$ for $RSTS$ by invoking the HTTP method μ on the resource r_i and passing the potentially empty RDF graph g in the request body. Depending on the current state σ_k of the service the following happens:

1. The service transitions into the state $\delta^s(\sigma_k, (r_i, \mu, g))$.
2. The client gets an HTTP response with the HTTP code c and the RDF graph g' in the body, where $(c, g') = \delta^o(\sigma_k, (r_i, \mu, g))$.

Safe methods that do not change any resource states, describe self-transitions, i.e., transitions that start and end in the same state.

The output function in the case of PUT and POST report to the client the effect the invocation of the method had on the state of the RSTS (i.e. $\sigma_l \backslash \sigma_k$).

Resources do not necessarily allow the use of all HTTP methods. Note that all state change functions are defined for every resource, i.e., every resource can be addressed with all methods: If a resource does not allow for the application of a specific method, the state change function describes a self-transition.

The defined service model serves as formal grounding of the execution language described in Section 5. However, the self-descriptive resources provide sufficient information for the interaction with the exposed resources.

- The current state of Linked Data resources – and therefore the state of the RSTS – can be accessed as RDF.
- The possible transitions and the state they result in are independent of the specific resource, except for POST transitions. The effect of POST transitions is declared with graph pattern descriptions (see Section 4.1).

Example. Figure 2 illustrates a state transition in RSTS where an entry is POSTed to mb:Acme. Note, that a client could derive the input for the POST method from the states of other resources (e.g., from Acme InfoItems).

5. THE DATA-FU LANGUAGE

In this section, we present Data-Fu[9], an *execution language* to instantiate a concrete interaction between a client

[8] For brevity we focus here on the four most important methods. Other methods can be added analougously.

[9] We use the name *Data-Fu* in adaption of the term *google-fu*, which adopts the suffi x*Fu* of from Kung Fu, implying great skill or mastery. Thus Data-Fu hints at the mastery of data interaction that can be achieved with the language.

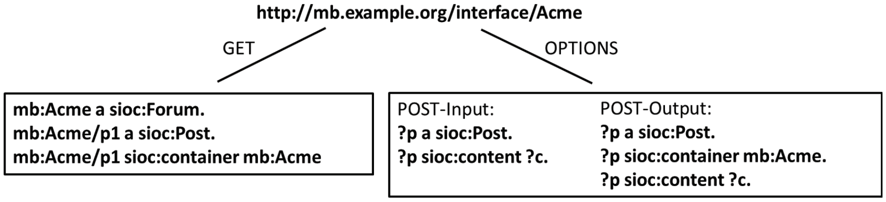

Figure 1: Self-descriptive resource: current state can be accessed with GET, input/output description with OPTIONS

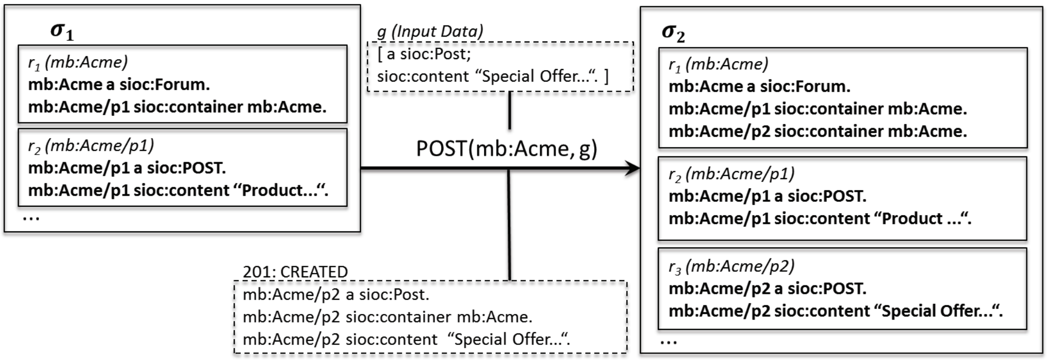
σ_1
r_1 (mb:Acme)
mb:Acme a sioc:Forum.
mb:Acme/p1 sioc:container mb:Acme.
r_2 (mb:Acme/p1)
mb:Acme/p1 a sioc:POST.
mb:Acme/p1 sioc:content "Product...".
...
g (Input Data)
[a sioc:Post;
sioc:content "Special Offer...".]
POST(mb:Acme, g)
201: CREATED
mb:Acme/p2 a sioc:Post.
mb:Acme/p2 sioc:container mb:Acme.
mb:Acme/p2 sioc:content "Special Offer...".
σ_2
r_1 (mb:Acme)
mb:Acme a sioc:Forum.
mb:Acme/p1 sioc:container mb:Acme.
mb:Acme/p2 sioc:container mb:Acme.
r_2 (mb:Acme/p1)
mb:Acme/p1 a sioc:POST.
mb:Acme/p1 sioc:content "Product ...".
r_3 (mb:Acme/p2)
mb:Acme/p2 a sioc:POST.
mb:Acme/p2 sioc:content "Special Offer...".
...

Figure 2: State transition of a RSTS, with excerpts of two states.

and resources, which preserves the adaptability, robustness and flexibility of REST.

In a resource-driven environment, applications retrieve and manipulate resources exposed on the Web. Since the resources can potentially be accessed by a multitude of clients, applications have to react dynamically on the state of the resources. Therefore, an important factor in the development of resource-driven applications is the dependency between the invoked transitions and resource states. The dependency between the invoked state transitions (i.e., applied HTTP methods) and the states of resources is that

1. input data for the transition is derived from RDF detailing the states of resources and/or
2. the transition is only invoked, if resources are in a specified state.

Data-Fu, a declarative rule-based execution language, enables programmers to define their desired state transitions. Data-Fu rules specify the interaction of a client with RESTful Linked Data resources and congruously a path through the RSTS. Further Data-Fu allows to specify the conditions under which a specific transition is to be invoked as subject to the states of resources.

Definition 2. *A rule ρ is of the form $\mu(r, g) \leftarrow q$, where $\mu \in M$ is an HTTP method, $r \in R \cup V$ is a resource or a variable with V the set of all variables, $g \in G \cup P$ is a (potentially empty) RDF graph or graph pattern, and $q \in P$ is a conjunctive query with P the set of all possible RDF graph patterns. If r is a variable, it must be bound in q. If g is a graph pattern, all its variables must be bound in q.*

The head of a rule corresponds to an update function of the RSTS in that it describes an HTTP method that is to be applied to a resource. The rule bodies are conjunctive queries that allow programmers to express their intention under which condition a method is to be applied. Thus, programmers can define an interaction pattern with a set of rules for their client applications.

The use of conjunctive queries is motivated by the idea that clients have to maintain a knowledge space (KS) in which they store their knowledge about the states of the resources they interact with [17, 25]. KS is filled with the RDF data the client receives after applying an HTTP method, as defined by the output functions of the RSTS. The output always informs the client about the current state after the application of the method.

Concretely N3 graph patterns are employed as queries q, which are evaluated over KS. If the evaluation of q is successful, i.e., matches are found in KS, the defined HTTP method μ is applied to r with input g. The query q can also be used to dynamically (i.e., during runtime)

1. derive input data from the states of other resources, as stored in KS and
2. identify the resource to which an HTTP method has to be applied, i.e., leveraging hypermedia controls.

Regarding 1: Instead of specifying the input data g explicitly as RDF graph, a graph pattern can be used. If a match is found for q in KS, the identified bindings for q are used to replace the variables in g to establish the input data for the interaction (with HTTP method μ at resource r). g as graph pattern and q act together similar to a SPARQL *construct* query over KS, where the result of the query is used as input data for the invocation of the method μ.

Regarding 2: To preserve the flexibility provided by REST our execution language has to be able to make use of links in the resource states to other resources. Rather than specifying the addressed resource r of a rule explicitly as URI, a variable can be used. If a match is found for q in KS, an identified binding for a variable q is used for the variable r. r as variable and q act together as a SPARQL *select* query to identify the targeted resources of method μ.

A Data-Fu program terminates when there are no active transitions and no rules can be activated that could trigger new transitions. In general, termination of a program cannot be guaranteed, as every transition can result in data that triggers new transitions. However, the termination of a program is not necessarily intended by a programmer, in the case of applications that are supposed to continuously interact with resources. Furthermore, the deletion and change of resources can lead to applications with a non-deterministic execution behavior. For discussions about properties of rule sets in related languages that guarantee termination and determinism, we refer the reader to [2].

Example. The IT department of Acme creates the dissemination system with four Data-Fu rules. The marketing department has simply to create new InfoItems and the system automatically distributes the information over the dissemination channels of Acme. The rules are defined as follows:

1. Whenever a InfoItem is found, retrieve the resource `acme:Acme` to get an up-to-date list of the current dissemination channels.

```
GET (acme:Acme, {}) ← { ?x rdf:type p:InfoItem }
```

2. If a `p:MicroBlogTimeline` is found (from the retrieved dissemination channels), post a new entry to the timeline using the content from the InfoItem.

```
POST (?mb, { []  rdf:type       sioc:Post ;
                 sioc:content   ?c .           } )
 ← {  ?x   rdf:type    p:InfoItem .
      ?x   p:content   ?c .
      ?mb  rdf:type    p:MicroBlogTimeline   } .
```

3. If a social network ID of Acme is found (from the retrieved dissemination channels), retrieve the representation of Acme from the social network to get a list of Acme's followers.

```
GET (?sid, {})
      ← { ?sid rdf:type p:SocialNetworkID } .
```

4. Post to every found follower of Acme on SNA a message with the content of the InfoItem.

```
POST (?f, { []  rdf:type      sna:Message ;
                sna:sender    sna:Acme ;
                sna:content   ?c .            })
 ← {  sna:Acme   sna:hasFan   ?f .
      ?x         rdf:type     p:InfoItem .
      ?x         p:content    ?c            }.
```

The described rules disseminate new information items automatically to social network SNA and the micro blog MB. IT deploys the dissemination system itself as a read/write Linked Data resource under `acme:Dissemination`. Marketing uses the dissemination service by POSTing a graph to the dissemination resource that corresponds to the following input pattern:

{ `?x rdf:type p:InfoItem. ?x p:content ?c` } .

Other dissemination channels can easily be added to the system, simply by adding corresponding rules in the system. For example, we consider that IT adds support for social network SNB by adding a rule that uses SNB's vocabulary for retrieving followers and sending a message:

```
POST (?f, {  []  rdf:type     snb:PrivateMsg ;
                 snb:origin   snb:ACME ;
                 snb:text     ?c .                 })
 ← {  snb:ACME   snb:followedBy   ?f .
      ?x         rdf:type         p:InfoItem .
      ?x         p:content        ?c              }.
```

The new dissemination channel is active when marketing PUTs Acme's identifier in SNB's network to `acme:Acme`.

6. THE DATA-FU INTERPRETER

The Data-Fu interpreter is an execution engine for service interactions specified as a set of Data-Fu rules. The engine implements the KS as well as the functionality to invoke interactions with resources as defined in the rules. In practice, we translate a Data-Fu program into a logical dataflow network, which is then optimised (e.g., re-using triple patterns and joins). The optimised logical network is then transformed into an evaluator plan that actually implements the dataflow network.

We realise the evaluator plan for the Data-Fu engine as a streaming processor that can process several queries in parallel. We implement the processor as a multi-threaded component with one thread evaluating individual triple patterns, and separate threads for each join operator and for each rule head, i.e., the component that performs the state transitions by invoking the corresponding HTTP methods on resources. The joins are implemented as symmetric hash join operators [35]. The implemented dataflow network is similar to a parallel version of the Rete algorithm [12].

To enable a wide variety of applications the engine can include an extension to support the interaction with REST resources that are not based on Linked Data. The engine can store data entities (e.g., binaries, JSON documents) received from such services separately. A triple pointing to a received non-RDF entity can be included in KS, thus the entities can be used in the logic of the execution rules. However, an interaction with such non-RDF entities requires to fall back to a more mashup-like programming approach.

Example. The dataflow network shown in Figure 3 evaluates the plan generated for the Data-Fu program for Acme's dissemination system. We can see that joins (e.g., the join on `?x`) are re-used, i.e., have multiple outgoing edges. The triple stream is initialised by the service input, which is sent by the client via a POST request. If the input data contains a description of an information item, it will trigger the rule retrieving Acme's description containing links to its dissemination channels. The social networks will fire a rule, which then retrieves the social network id's of Acme and thus retrieve the corresponding followers. Both social network followers and micro blog timelines will then trigger the corresponding POST actions that will sent the information item in the appropriate vocabulary to the dissemination channels, i.e., as micro blog posts or personal messages to the followers.

7. EVALUATION

To evaluate the scalability of the Data-Fu engine we compared execution times for different numbers of interactions and rules with Cwm[10], a data-processor for the Semantic Web. Cwm uses a local triple store that supports the full N3 language to save data and intermediate results. The local

[10] `http://www.w3.org/2001/sw/wiki/CWM`

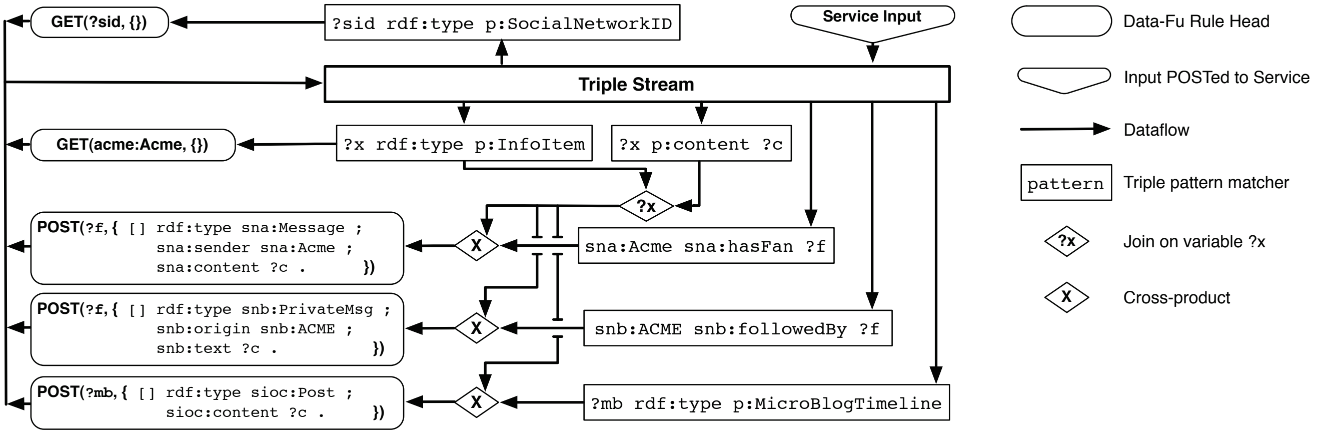

Figure 3: Dataflow network of Acme's dissemination system

triple store of Cwm uses seven indices to allow for a rapid readout of the local data with almost every combination of subject, predicate and object patterns. For inferencing Cwm uses a forward chain reasoner for N3 rules. The pattern matching for the rules is done by recursive search with optimisations, such as identifying an optimal ordering for the evaluation of the rules and patterns.

Cwm is built as a general purpose tool to query, process, filter and manipulate data from the Semantic Web. As such, the motivation behind Cwm is closest to the Data-Fu engine, compared with any other rule engines or reasoning systems, to the best of our knowledge. However, Cwm is not targeted on the direct RESTful manipulation of web resources, but their retrieval and the local manipulation of the data. Therefore to make the systems comparable we limit the evaluated interactions to GET transitions, i.e., we use only rules that retrieve resources, if a match for the rule body is found. Please note that the limitation to GET transitions does not influence the validity of the evaluation: Since additional execution time when using non-safe interactions (e.g., PUT, POST) only results from time required to transmit data to resources and the subsequent time necessary to process this data by the server, where the resource resides. This time overhead caused by non-safe transitions is neither influenced by the Data-Fu engine, nor could it be avoided by any other system that we could use as comparison.

We conducted the experiments on a 2.4 GHz Intel Core 2 Duo with 4 GB of memory (2 GB assigned to Java virtual machine on which the experiments run). Thus we evaluate the Data-Fu engine on commodity hardware with the intent to show the parallelisation-based scalability of the Data-Fu engine not only on high-end industrial machines.

We deploy Linked Data resources used for the interactions locally on an Apache Tomcat[11] server to further minimise execution time variations caused by establishing HTTP connections and retrieving data over the web. In the rules used by the Data-Fu engine and Cwm the resources are addressed with their localhost address. Every deployed resource represents a number. Every number resource is typed as `number` and contains its value as literal and a link to the successor of the number:

```
local:1  rdf:type         local:number.
local:1  local:value      "1".
local:1  local:successor  local:2.
```

We chose this design to easily keep track of the number of performed interactions.

For the evaluation we start with the resource number 0, which we manually inject into the Data-Fu engine and Cwm. We identify and retrieve the successor of the number. The successor of a number yields a new successor to retrieve, and so on. The interactions of this set-up are illustrated in Figure 4.

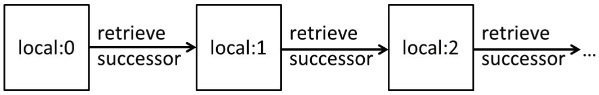

Figure 4: Interactions of evaluation set-up with one rule

We realise the interactions with the Data-Fu Engine and Cwm (for the latter in two different ways) as follows:

- *Data-Fu:* For the Data-Fu engine we use a rule:

```
GET (?suc, {})  ←  {?n rdf:type local:number
                    ?n local:value ?v
                    ?n local:successor ?suc}
```

  The rule body queries for a resource (variable `?n`) that is typed as number, has a value and a successor. If a match is found, a GET transition is triggered at whatever URI is identified to be the successor of the matched number. The Data-Fu engine adds the retrieved representation of the successor to the data flow network, which results in the identification of the next successor to retrieve. Thus, all numbers are iteratively found and retrieved.
- *Cwm direct:* Cwm offers built-in functions to perform web-aware queries in rules. The keyword `log:semantics` in a query of a rule allows to resolve a URI and bind the retrieved RDF data to a variable as formula. The formula bound to a variable can then be used to construct triples in the rule head. We used the following rule to perform the desired interaction:

[11] http://tomcat.apache.org/

```
{{ :n rdf:type local:number.
   :n local:value :v.
   :n local:successor :suc }
        local:is local:known. }
:suc log:semantics :sem.
⇒
  { :sem local:is local:known. }
```

Like in the approach for the Data-Fu engine we query for the successor of a number. The successor is retrieved and bound as formula in subject position to a new triple that is written to the triple store. Since the retrieved representation of the number appears only as formula in triples we have to extend the query in the rule body to search for the successor of a number in a formula in subject position of a triple, thus making the query slightly more complicated than in the case of the Data-Fu engine. Cwm repeatedly applies the rule to the triple store, thus retrieving all numbers.

- *Cwm import:* To compare the performance of Cwm with the Data-Fu engine, where the queries of the rules are equally complex, we implemented the desired retrieval with another approach, with the following rule:

```
{ :n rdf:type local:number.
  :n local:value :v.
  :n local:successor :suc }
⇒
  { :n owl:imports :suc. }
```

We use the same query to identify the successor of a number as for the Data-Fu engine. For every found match we write a triple to the Cwm store, that marks the identified successor with `owl:imports`. Cwm offers a command to retrieve all resources marked with owl:imports. This allows us to programmatically instruct Cwm to apply the rule and retrieve the successor, as many times as needed. Note, that this implementation of the interaction does not deliver the same functionality as with the Data-Fu engine: We have to manually define how often the rule followed by the retrieve command is to be applied (once for every number), rather then having the engine automatically retrieve all the numbers.

We evaluate the execution time of the interaction with all three setups for sets of 20, 40, 60, 80 and 100 numbers. With the approaches *Data-Fu* and *Cwm direct* the interaction ends when the last number in a set does not refer to a next successor to retrieve. For *Cwm import* we had to decide manually how often the rule is applied and thus how many numbers are retrieved and when the interaction stops. The results are shown in Table 3 and Figure 5. We provide the average execution times from ten runs to reduce variations.

Table 3: Average execution time from ten runs for different evaluation set-ups with one rule

number set size	Data-Fu	Cwm direct	Cwm import
20	342 ms	1549 ms	468 ms
40	371 ms	5144 ms	976 ms
60	500 ms	11272 ms	1595 ms
80	555 ms	21005 ms	2309 ms
100	594 ms	32213 ms	3688 ms

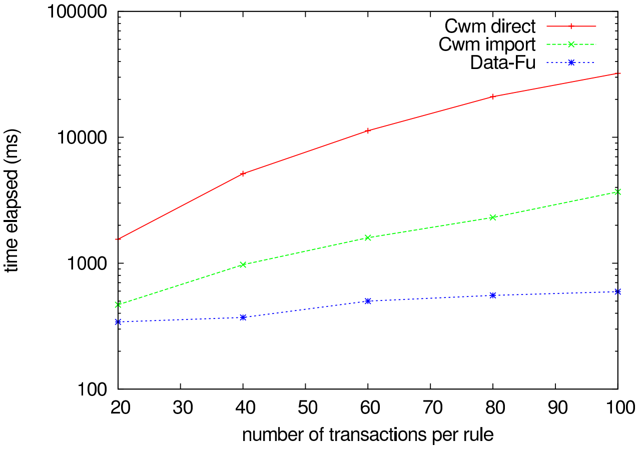

Figure 5: Average execution time from ten runs for different evaluation set-ups with one rule

The Data-Fu engine is able to execute the interaction by orders of magnitude faster than the other two approaches with Cwm. Also the growth-rate of the execution time with the increasing size of number sets is much lower with Data-Fu compared to the Cwm approaches (note the log scale in Figure 5). The Data-Fu engine achieves this time saving by leveraging the data flow network: Data-Fu has just to put the new results after an interaction through the data flow network to find new bindings. Cwm on the other hand has to apply the rules repeatedly over the increasing dataset in its triple store.

To evaluate the capabilities of the Data-Fu engine with regard to parallelisation we run the same interaction of retrieving successors of numbers again, with ten different "kinds" of numbers (A-J) in parallel. The numbers are distinguished by different namespaces. Each of the three evaluation set-ups requires ten rules for the interaction (each addressing another namespace), analog to the previously shown rules. Figure 6 illustrates this evaluation set-up.

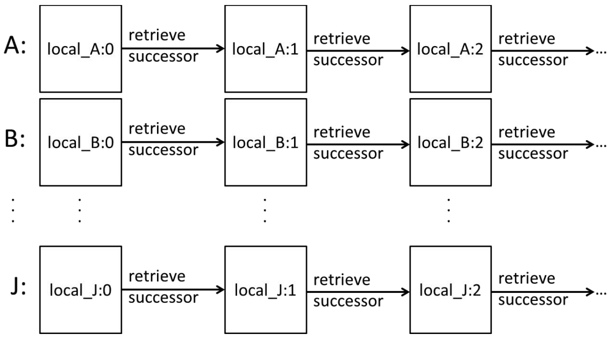

Figure 6: Interactions of evaluation set-up with one rule

The results for the different evaluation set-ups are shown in Table 4 and Figure 7 as average from ten runs. Again Data-Fu executes the interaction significantly faster with a lower growth rate than Cwm in the other set-ups: In the case of the most interactions (10 x 100) *Cwm direct* requires over 17 minutes and *Cwm import* over 32 seconds, the Data-Fu engine handles the same interactions in under 4 seconds.

Table 4: Average execution time from ten runs for different evaluation set-ups with ten rules in parallel

number set size	Data-Fu	Cwm direct	Cwm import
20	1833 ms	22513 ms	2836 ms
40	2421 ms	108421 ms	7067 ms
60	2916 ms	310498 ms	13518 ms
80	3889 ms	621798 ms	21729 ms
100	3944 ms	1038524 ms	32983 ms

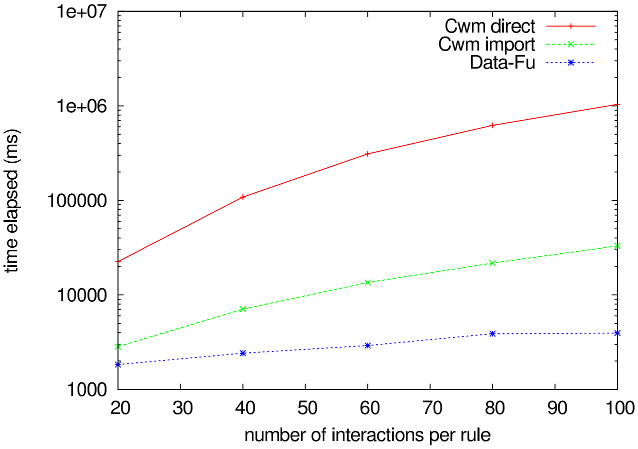

Figure 7: Average execution time from ten runs for different evaluation set-ups with ten rules in parallel

Comparing the results of the interactions with a single rule and the interactions with ten rules in parallel we note, that the Data-Fu engine suffers less than Cwm from the ten times increased workload when executing ten rules in parallel. On average for the individual sizes of number sets

- *Data-Fu* requires 6.2 times longer,
- *Cwm direct* requires 25 times longer,
- *Cwm import* requires 8 times longer,

when running with ten rules compared to one single rule.

The reason for this time advantage is the capability of the Data-Fu engine to execute several components of the interaction in parallel, e.g., the evaluation of the triple patterns of the queries and the communication with several web resources. Note, that the theoretically possible speedup due to parallelisation on a dual core system implies that a 10 times increased workload results in a 5 times longer execution time. However, the Data-Fu engine cannot quite reach this optimal speedup, since not all parts in the interaction can be completely parallelised, e.g., the management of the individual threads. These parts of an interaction that cannot be completely parallelised result in a slightly diminished speedup, as stated by Amdahl's Law [3].

Following the results of the evaluation in comparison with Cwm, we devise a final evaluation setting to test the scalability of the Data-Fu engine when performing large amounts of interactions. Similar to the previous evaluation setting we retrieve number resources that are identified during runtime as successor of an already found number. We fix the size of the number sets to 100, i.e., we deploy sets of 100 consecutive number resources that are distinguished with their namespace. Then we retrieve the numbers of every set with a respective rule. We evaluate the runtime of the Data-Fu engine with 20, 40, 60, 80 and 100 rules/number sets, thus performing between 2 000 and 10 000 interactions. Additionally we measure the time needed to calculate the evaluation plan separately to compare it with the total execution time. The results are shown in Table 5 and Figure 8.

Table 5: Average execution time from ten runs of Data-Fu engine with number sets of size 100

rules/number sets	execution time	evaluation plan
20	8357 ms	4 ms
40	17195 ms	6 ms
60	30767 ms	7 ms
80	49430 ms	8 ms
100	75764 ms	9 ms

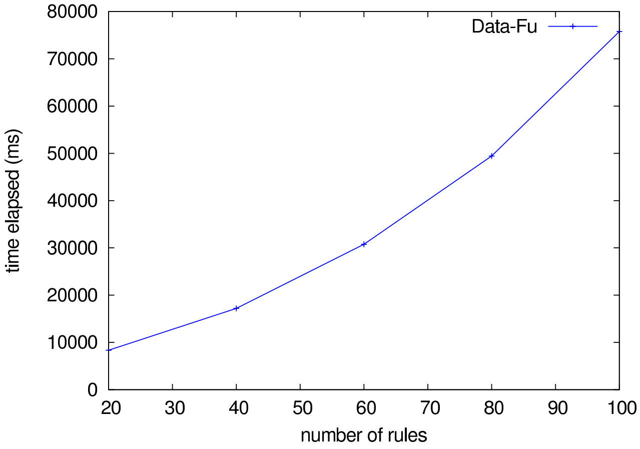

Figure 8: Average execution time from ten runs of Data-Fu engine with number sets of size 100

The results of the evaluation for large amounts of interactions show that the Data-Fu engine scales well up to thousands of interactions even on commodity hardware. The Data-Fu engine is capable of interacting with 10 000 web resources in about 1:15 min. The necessary time required to establish the evaluation plan increases with the number of rules, but remains a very small fraction of the overall execution time and is therefore negligible.

The evaluation shows the advantages of the parallel processing of queries and interactions and provides evidence that the Data-Fu engine is capable of performing rapid interactions with web resources as desired. We did not consider the necessary time to establish HTTP connections on the web and the response time of the servers, where resources are deployed, since these additional time requirements would be the same for any employed interaction system. Note however, that due to its parallel processing nature, the Data-Fu engine could further benefit from longer response times of servers compared to other systems: At the same time as the Data-Fu engine performs the manipulations and retrieval of resources other rules can be evaluated, thus the overall execution time can be minimised.

We provide the data used for the evaluation and an executable jar online[12] to re-run the experiments.

[12] `http://people.aifb.kit.edu/sts/datafu/evaluation/`

8. RELATED WORK

Pautasso introduces an extension to BPEL [21] for a composition of REST and traditional web services. REST services are wrapped in WSDL descriptions to allow for a BPEL composition. Our approach focuses on a native composition of REST services, rather than relying on technologies of traditional web services. For a comparison between RESTful services and "big" services see [23].

There exist several approaches that extend the WS-* stack with semantic capabilities by leveraging ontologies and rule-based descriptions (e.g., [28, 10, 8]) to achieve an increased degree of automation in high level tasks, such as service discovery, composition and mediation. Those approaches extending WS-* became known as Semantic Web Services (SWS). An Approach to combine RESTful services with SWS technologies in particular WSMO-Lite [31] was investigated by Kopecky et al. [16]. In contrast to SWS, REST architectures do not allow to define arbitrary functions, but are constrained to a defined set of methods and are built around another kind of abstraction: the resource. Therefore our approach is more focused on resource/data centric scenarios in distributed environments (e.g., in the Web).

Active XML introduces service calls as XML nodes that are placeholders for new XML documents that can be retrieved from the service [1]. The service calls are comparable to hypermedia links in resource descriptions and the active XML document corresponds to the knowledge space. In contrast to Active XML, our work discovers links to new resources instead of links to function calls. The resource model provides more flexibility, e.g., a Data-Fu program could perform a DELETE on a discovered resource, whereas the Active XML equivalent would be constrained to the predefined operations in the original link.

The scripting language S [6] allows to develop Web resources with a focus on performance due to parallelisation of calculations. Resources can make use of other resources in descriptions, thus also enabling a way of composing REST services. S does not explicitly address the flexibility of REST and has no explicit facilities to leverage hypermedia controls or to infer required operations from resource states.

RESTdesc [30] is an approach in which RESTful Linked Data resources are described in N3-Notation. The composition of resources is based on an N3 reasoner and stipulates manual interventions of users to decide which hypermedia controls should be followed.

Hernandez et al. [14] proposes a model for semantically enabled REST services as a combination of pi-calculus [19] and approaches to triple space computing [9] pioneered by the Linda system [13]. They argue, that the resource states can be seen as triple spaces, where during an interaction triple spaces can be created and destroyed as proposed in an extension of triple space computing by Simperl et al. [25]. Our service model is in contrast to this approach more focused on the composition of data driven interactions.

Similar to the idea of triple spaces is the composition of RESTful resources in a process space, proposed by Krummenacher et al. [17] based on resources described using graph patterns. Speiser and Harth [26] propose similar descriptions for RESTful Linked Data Services. Our approach shares the idea that graph pattern described resources read input from and write output to a shared space. We improve on this approach by providing a service model and a more explicit way of defining the interaction with resources.

9. CONCLUSION

In this paper, we addressed the problem of creating value-added compositions of data and functionalities. As a unifying model for both static data sources and dynamic services, we described how Linked Data Resources can be extended with descriptions for RESTful manipulation. The natural extension of Linked Data with RESTful manipulation of resources enables a framework with uniform semantic resource representations for REST architectures. We have proposed to exploit the advantages resulting from the combination of REST and Linked Data in a programming framework for the Semantic Web. We have introduced Data-Fu, a declarative rule-based execution language with a state transition system as formal grounding, and the challenges we address with this language, i.e., achieving scalability and performance while preserving the flexibility and robustness of REST. Furthermore, we described our implementation of an execution engine for the Data-Fu language.

For future work, we plan to extend our approach in the following directions. First, we will add capabilities to improve handling of failures of resource interactions. Second, we will extend our formal model of Data-Fu to provide clearly defined semantics in the presence of non-deterministic rules. Third, we will integrate support for rule-based reasoning into the execution engine. The rules bring useful expressivity for aligning different vocabularies and can be easily supported in the engine by introducing triple-producing rule heads in addition to the current state transition handlers.

Acknowledgments

This work was partially supported by the PlanetData NoE (FP7:ICT-2009.3.4, #257641) and by the German Ministry of Education and Research (BMBF) within the Software-Campus project framework.

10. REFERENCES

[1] S. Abiteboul, O. Benjelloun, and T. Milo. Positive Active XML. In *Proceedings of the 23rd Symposium on Principles of Database Systems (PODS'04)*, pages 35–45. ACM, 2004.

[2] A. Aiken, J. Widom, and J. M. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. *SIGMOD Record*, 21(2):59–68, 1992.

[3] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the 1967 Spring Joint Computer Conference (AFIPS'67)*, pages 483–485, Atlantic City, New Jersey, 1967. ACM.

[4] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/PACTs: A programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*, pages 119–130, Indianapolis, Indiana, USA, 2010. ACM.

[5] T. Berners-Lee. *Read-Write Linked Data*. August 2009. Avaiable at `http://www.w3.org/DesignIssues/ReadWriteLinkedData.html`, accessed 26th November 2012.

[6] D. Bonetta, A. Peternier, C. Pautasso, and W. Binder. S: A scripting language for high-performance RESTful

web services. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'12)*, 2012.

[7] C. Brenner, A. Fensel, D. Fensel, A. Gagiu, I. Larizgoitia, B. Leiter, I. Stavrakantonakis, and A. Thalhammer. How to domesticate the multi-channel communication monster. Available at `http://oc.sti2.at/sites/default/files/oc_short_handouts.pdf`.

[8] J. Cardoso and A. Sheth. *Semantic Web Services, Processes and Applications*. Springer, 2006.

[9] D. Fensel. Triple-space computing: Semantic web services based on persistent publication of information. In *Proceedings of the IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM'04)*, number 3283 in Lecture Notes in Computer Science, pages 43–53, Bangkok, Thailand, 2004. Springer.

[10] D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer, 2006.

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

[12] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[13] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7:80–112, 1985.

[14] A. G. Hernández and M. N. M. García. A formal definition of RESTful semantic web services. In *Proceedings of the First International Workshop on RESTful Design (WS-REST'10)*, pages 39–45, 2010.

[15] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys'07)*, pages 59–72, Lisbon, Portugal, 2007. ACM.

[16] J. Kopecky, T. Vitvar, and D. Fensel. MicroWSMO: Semantic description of RESTful services. Technical report, WSMO Working Group, 2008.

[17] R. Krummenacher, B. Norton, and A. Marte. Towards Linked Open Services. In *Proceedings of the 3rd Future Internet Symposium (FIS'10)*, volume 6369 of *Lecture Notes in Computer Science*, Berlin, Germany, 2010. Springer.

[18] E. A. Lee and P. Varaiya. *Structure and Interpretation of Signals and Systems*. Addison-Wesley, 2011.

[19] R. Milner. *Communicating and Mobile Systems: π-calculus*. Cambridge University Press, Cambridge, UK, 1999.

[20] B. Norton and S. Stadtmüller. Scalable discovery of linked services. In *Proceedings of the 4th International Workshop on REsource Discovery (RED'11)*, 2011.

[21] C. Pautasso. RESTful web service composition with BPEL for REST. *Journal of Data and Knowledge Engineering*, 68(9):851–866, 2009.

[22] C. Pautasso and E. Wilde. Why is the web loosely coupled?: A multi-faceted metric for service design. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 911–920, Madrid, Spain, 2009. ACM.

[23] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[24] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2007.

[25] E. Simperl, R. Krummenacher, and L. Nixon. A coordination model for triplespace computing. In *Proceedings of the 9th International Conference on Coordination Models and Languages (COORDINATION'07)*, 2007.

[26] S. Speiser and A. Harth. Integrating Linked Data and services with Linked Data Services. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC'11) Part I*, volume 6643 of *Lecture Notes in Computer Science*, pages 170–184, Heraklion, Crete, Greece, 2011. Springer.

[27] S. Stadtmüller and A. Harth. Towards data-driven programming for RESTful Linked Data. In *Workshop on Programming the Semantic Web (ISWC'12)*, 2012.

[28] R. Studer, S. Grimm, and Abecker, A. (eds.). *Semantic Web Services: Concepts, Technologies, and Applications*. Springer, 2007.

[29] M. Taheriyan, C. A. Knoblock, P. A. Szekely, and J. L. Ambite. Rapidly integrating services into the Linked Data cloud. In *Proceedings of the 11th International Semantic Web Conference (ISWC'12)*, volume 7649 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 2012.

[30] R. Verborgh, T. Steiner, D. V. Deursen, R. V. de Walle, and J. G. Valls. Efficient runtime service discovery and consumption with hyperlinked RESTdesc. In *Proceedings of the 7th International Conference on Next Generation Web Services Practices (NWeSP'11)*, Salamanca, Spain, 2011.

[31] T. Vitvar, J. Kopecky, M. Zaremba, and D. Fensel. WSMO-Lite: Lightweight semantic descriptions for services on the web. In *Proceedings on the 5th European Conference on Web Services (ECOWS'07)*, pages 77–86, 2007.

[32] J. Webber. *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly, 2010.

[33] M. Weiss and G. R. Gangadharan. Modeling the mashup ecosystem: Structure and growth. *R&D Management*, 40(1):40–49, 2010.

[34] E. Wilde. REST and RDF granularity, 2009. Available at `http://dret.typepad.com/dretblog/2009/05/rest-and-rdf-granularity.html`.

[35] A. N. Wilschut and P. M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems (PDIS'91)*, pages 68–77, Miami Beach, FL, USA, 1991. IEEE Computer Society Press.