Bus Inf Syst Eng 58(5):329–340 (2016)
DOI 10.1007/s12599-016-0447-z

RESEARCH PAPER

Provalets: Component-Based Mobile Agents as Microservices for Rule-Based Data Access, Processing and Analytics

Adrian Paschke

Received: 15 November 2015 / Accepted: 21 June 2016 / Published online: 18 August 2016

Abstract Provalets are mobile rule agents for rule-based data access, semantic processing, and inference analytics. They can be dynamically deployed as microservices from Maven repositories into standardized container environments such as OSGi, where they can be used via simple REST calls. The programming model supports rapid prototyping and reuse of Provalets components to build Linked Enterprise Data applications where the sensible corporate data is not transmitted outside the enterprise, but instead the Provalets providing data processing and knowledge inference capabilities are moved closer to the data.

Keywords Microservices · Linked enterprise data · Corporate semantic web · Corporate smart content · Rule-based agents · Inference service · Rule-based data analytics

1 Introduction

The Corporate Semantic Web (CSW) (Paschke and Schäfermeier 2015)[1] deals with applications of semantic technologies in a corporate context. The goal of the Linked Open Data (LOD) is to expose machine interpretable data in Resource Description Framework (RDF) on the web. The applied principles of Linked Data can also be used in a corporate environment, then named Linked Enterprise Data (LED). While interlinking of machine interpretable Linked Data has been adopted in the Public Semantic Web as LOD cloud (Schmachtenberg et al. 2014), corporate enterprises are still reluctant to use this approach to form their own private Linked Enterprise Web of Data for, e.g., semantic data processing, smart content enrichment[2] and inference-based analytics (Hu and Svensson 2010a; Wood 2010). Reasons for this are, e.g., the lack of an easy to use and understand engineering and service-oriented approach (e.g., Service Oriented Architecture – SOA). This has lead to, e.g., the W3C Linked Data Platform (LDP)[3], which standardizes a REST protocol for accessing (read/write) Linked Data. While this provides a basis for SOA-based Enterprise Application Integration (EAI) (Mihindukulasooriya et al. 2013) with a standardized, typically monolithic LDP orchestrating service and access to data and applications via service interfaces, it does not solve the problem of developing a single LED application composed of distributed LED microservices[4]. The microservice architectural style[5] is a modern SOA approach to developing a single application as a suite of small services (microservices) which are executed in their own container processes (e.g., Docker, OSGi) and communicating with other microservices via lightweight protocol mechanisms. Microservices are becoming the adopted standard for continuously deployed systems (Balalaie et al. 2016) with,

Accepted after two revisions by the editors of the special issue.

Prof. Dr. A. Paschke (✉)
Institute of Computer Science, AG Corporate Semantic Web, Freie Universität Berlin, Königin-Luise-Str. 24, 14195 Berlin, Germany
e-mail: paschke@inf.fu-berlin.de

[1] http://www.corporate-semantic-web.de, accessed 1 Jun 2016.

[2] http://sce.corporate-smart-content.de/, accessed 1 Jun 2016.

[3] http://www.w3.org/TR/ldp/, accessed 1 Jun 2016.

[4] Microservices is a software architecture style designing a software application as a suite of independently deployable small services providing e.g., (agent) intelligence in the endpoints, and decentralized control of languages and data.

[5] Martin Fowler: http://martinfowler.com/articles/microservices.html, accessed 1 Jun 2016.

e.g., mobile and distributed component-based application logic. They are increasingly used for the Platform-as-a-Service (PaaS) offerings of commercial provides (see e.g., IBM Bluemix, Microsoft Azure, Cloud Foundry). The microservice approach has major benefits for LED-based applications:

- Scaling of agile development processes becomes possible because microservice can be developed independently and distributed, e.g., development of microservices for different LED sources, involving different organisational units or different data and knowledge engineers.
- Continuous deployment in standardized container environments and composition becomes easier, making it possible to build LED processing and inference/analytics pipelines out of mobile microservices.
- Microservices can be easily enhanced, redeveloped and replaced because they are compact and introduce dependencies only via their API. This enables to develop a stable and robust application composed of independently scaled (including e.g., parallelization) and maintained microservices.

The microservice approach also addresses further LED adoption problems such as worries about data security and data privacy, as well as legal concerns. Enterprises are reluctant to move their data into the cloud and to allow external providers to store or process their sensible data. With greatly diverging security and judical standards (e.g., data sovereignty laws[6]) worldwide the risks of loosing control over sensible data appear too high for them.

To overcome these hurdles for the adoption of Linked Enterprise Data we propose an easy to use standardized component-based microservice model for mobile software agents providing rule-based semantic Linked Data access, data processing, and inference-based analytics functionalities. Instead of sending all data to a centralized external provider, the microservice-based agents can be moved closer to the data and their execution remains in the control of the data producer. Hence, privacy problems are avoided and additional permissions can be defined in the agents metadata and policies, so that the controlled container-based runtime environment can enforce them and reject agents that request permissions that cannot be granted. We call this novel concept *Provalets*[7] (Paschke 2015). For Provalets we adopt the Representational State Transfer (REST) architecture style which has become increasingly popular, e.g., for REST-based Services such as the LDP. REST uses URIs to address resources and transport protocols such as HTTP to transfer representations.

Provalets address core requirements in a Corporate Semantic Web by enabling the rule-based use of Linked Enterprise Data in new business models that avoid the core issues with data privacy. All sensitive data remains in a trusted and controlled container-based environment. Prototyping with Provalets requires no configuration of any hard- or software at all. Instead Provalets agents can be automatically managed and deployed from standard repositories, such as Git or SVN, in their runtime container environments as microservices. Developers can access and test Provalets with their data by simply calling the API with the right REST URL. Therefore, Provalets use ideas from software agents, microservices, automated DevOps build management, and trusted runtime container environments (including cloud and smart/edge infrastructures) to place data intensive routines near the actual data. Among many other benefits, this omits data privacy issues as well as bottlenecks in the network infrastructure and might also leverage in-memory architectures and streaming data pipelines. Application domains of Provalets are manifold, reaching from decentralized microservices for edge inference/analytics to multi agent systems for data pipelines and weakly-structured workflows (Zhao et al. 2016).

The implementation of Provalets encapsulates data intensive rule-based data processing and inference reasoning [e.g., for representing decision and reaction logic (Paschke 2011)] in rule-based Prova agents (Paschke and Boley 2011)[8], which are deployed as microservices in standardized containers such as Docker and trusted OSGi (OSGI Alliance 2009). For their engineering and automated build management they make use of aspect-oriented OntoMaven/RuleMaven (Paschke and Schäfermeier 2015; Paschke 2013a, b). For their composition an expressive Prova-based workflow language (Zhao and Paschke 2012, 2013a, b; Zhao et al. 2016) is used, which can be translated into Reaction RuleML[9] (Paschke 2014; Paschke and Boley 2014; Boley et al. 2010) for their standardized rule-based agent communication via Jetty WebSockets.

The paper follows a design science research methodology and addresses the development and evaluation of Provalet design artifacts. Our contributions are as follows:

1. We present an easy to use standardized component-based model for distributed microservices based on mobile software agents providing rule-based semantic Linked Data access, processing, and inference-based analytics functionalities.

[6] Data is subject to the laws of the country in which it is located, which becomes a big privacy and compliance issue if data is moved to cloud service provides which operate cross-countries.

[7] http://sce.corporate-smart-content.de/provalets, accessed 1 Jun 2016.

[8] http://prova.ws, accessed 1 Jun 2016.

[9] http://reaction.ruleml.org, accessed 1 Jun 2016.

2. The standardized (mobile) component model makes it a generic approach suitable for standard container runtime environments, in-memory architectures, cloud-based environments, as well as distributed smart IoT/IoE, and edge devices. The proof-of-concept implementation supports both OSGi containers and Docker containers.
3. The rule-based programming model supports rapid prototyping for both traditional ex-post data access to knowledge bases [rule-based data access + typed ontology-based data access (via order-sorted logic approach)] as well as real-time streaming data (RDF stream processing) with rule-based complex event processing and reasoning.
4. The existing repository architecture for knowledge base artifacts, the standardized API for Knowledge Bases (API4KB) (Athan et al. 2015a), and the tool-supported life cycle model makes it an easy to use solution for Linked Enterprise Data in the Corporate Semantic Web that can guarantee important properties such as policy-based/SLA-based quality of service (QoS), data security, and privacy.
5. The Provalet concept has potential for novel business models that allow to create new value chains based on Linked Enterprise Data microservices/agents (e.g., rule-based data analytic pipelines and reasoning workflows provided by platform-as-a-service/container-as-a-service models).

Provalets provide the basis for an analytics-as-a-service business model, in which data-driven inference analytics operations are provided as component-based Provalet agents. The dynamic processing pipelines and the expressive composition semantics with possible parallel executions make it a highly scaleable container approach. Furthermore, the standardized OSGi-based implementation makes Provalets useable not just for centralized cloud platforms, but also in edge and fog containers. The Provalet microservice proof-of-concept implementation with Apache Felix running in Apache Karaf can be also deployed in Docker containers. The use of controlled container resources allows Provalets to process the private data of a company, while restricting the transfer of data to an external user, e.g., by anonymization, search or aggregation of sensitive data without a loss of control.

This article details the Provalet concept previously introduced in Paschke (2015) by presenting application scenarios, a proof-of-concept implementation, and a concrete development example. The further paper is structured as follows: we begin with application scenarios in Sect. 2 and relate our work to previous efforts in Sect. 3. Then we introduce the underlying principles of Provalets in Sect. 4. In Sect. 5 our proof-of-concept implementation is described, and for better understanding we give a concrete development example in Sect. 6. An evaluation by comparison with related approaches and by looking at the representation expressiveness is give in Sect. 7. We conclude our work in Sect. 8.

2 Application Scenarios

In order to illustrate the distinctive features of Provalets, such as continuous deployment, mobile execution, and component-based composition, we describe several application scenarios in this section.

Corporate Smart Content Enrichment and Delivery An example for such a scenario are smart street lights equipped with a gateway capable of running Provalets, which deliver situation-aware personalized smart content (e.g., local information feeds collected from the surrounding Smart City IoT sensors, local news, or promotion offerings from local shops) to the location-based devices (e.g., mobile phones, connected cars) passing by the lamps. For instance, a deployed Provalet detects by analysing IoT event streams from Smart City street sensors that a known client attaches or will soon connect to a WiFi hot spot in a smart city environment. It calls and deploys a Provalet that pro-actively enriches the Wifi data point with the client's known history, metadata/background knowledge, and personalized state/profiling information. Relevant further Provalets that will contribute to the personalized smart content delivery service provided at the Wifi hot spot for the client are determined and selected from the many available ones. Decomposed processing tasks and analytical + semantic logic are automatically deployed as Provalets to the used edge device. Optimized streaming data pipelines are dynamically established by the Provalets, which transparently process the data streams and deliver the relevant smart content.

As a possible example consider a salesman on a business trip who wants to eat with one of the companies' clients to use the remaining working time. The location-aware Provalets can help to identify the nearest clients, automatically contact these clients to inquire if they want to meet, and arrange a meeting place by reserving a table at the next available restaurant. A related scenario exploiting Linked Enterprise Data has been studied in Hu and Svensson (2010b) for a centralized customer information portal as solution approach. A major benefit of the Provalet approach is that there is no need to collect and integrate all required data into one centralized information store. Instead the private data of the salesman and of the clients can remain locally in their phones, and the processing logic is continuously deployed to the data as mobile (trusted)

Provalet microservice via the nearest connected Smart Light gateway.

Another possible use case scenario is an online delivery service that can check the current location of an ordering customer via a Provalet and use its rule-based logic to decide where to deliver.

Corporate Smart Content Analytics-as-a-Service Although there is a number of third-party web services and platform-as-a-service offerings available for linked data management and text analytics (see, e.g., Dimitrov et al. 2015), companies are generally not able to use them because of privacy issues. Currently the services can only be used by sending the private text data, such as e-mails and working documents, to the service – which is of course a huge privacy issue. Additionally, this task also has scalability problems, as all the documents and even the results of the services have to be transferred over the network. That is the reason why most of the companies wishing to analyze company-internal data still rely on their own solutions, which is of course expensive and quite unnecessary. Provalets provide an excellent way to improve this situation. Instead of bringing the data to the service operation, the operation is transferred to the data and executed close to it. In this way, both problems can be solved at once – the privacy issue disappears as the data remains inside the company, and additionally neither the documents nor the results have to be transferred over the network in a time-consuming process. Text mining and other analytics operations are normally complex operations including links to additional data like knowledge bases and taxonomies. Such knowledge sources can be externally linked and accessed by the Provalet based on permissions granted. Of course it must be guaranteed that no private data leaves the company boundaries at any time. Access and usage of this knowledge resources can be provided by other Provalets, e.g., via Prova's query built-ins and via OMG API4KB compliant interfaces. This would also allow fine grained control of the Provalet information extraction from data sources such as enterprise documents. Instead of sending the whole document only the extracted (and maybe transformed) information is delivered to a location outside the container resource.

Smart Data Analytics Pipelines and Workflows Most LED processing and analytics scenarios – including the above text mining scenario - consist of multiple processing steps, reaching from data collection and access, pre-processing transformation and semantic enrichment, to knowledge-based processing, reasoning and inductive learning, as well triggering rule-based (re-)actions.

For instance, semantic Complex Entity Recognition (CER) combines a series of technologies from the Natural Language Processing (NLP), Text Mining and semantic knowledge representation field in order to go a step beyond recognizing simple named entities and to extract clusters of related information from text, such as unstructured enterprise information documents. The CER process is divided into three sub-processes:

1. In the first process the documents are processed in an NLP pipeline to recognize named entities, followed by relation extraction, where the explicit relationships between entities are recognized, extracted and matched to possible candidates in background LED knowledge bases. In a final reduction step the text documents are reduced to a set of entities and their relationships.
2. The second process is an enrichment activity, where additional knowledge, such as types, super-types or semantically related entities, from background KBs is added to the extracted entity/relationship sets. This enrichment is conducted by means of a series of rules which are used to conditionally infer which knowledge should be added.
3. In the third process the actual complex entities are mined through multiple machine learning steps, such as topic entities through (dynamic) LDAs, trend mining, and co-occurrences in enriched entity sets.

In Complex Event Processing (CEP) data streams are analyzed and monitored to detect sophisticated situations and phenomena in real-time based on event patterns (Teymourian and Paschke 2016). Semantic Event Pattern Mining aims at discovering and learning such complex event patterns from the real-time event instance sequences (EIS) by exploiting semantic knowledge. EIS are analyzed according to their (temporal) occurrence and structure to mine frequent patterns, association rules, time-series, episodes, and many more. By attaching semantic background knowledge to each event, the event content becomes more exploitable than the raw data is for learning complex patterns. This might involve multiple steps, such as semantic representation of streaming data (see e.g., W3C RDF Stream Processing), semantic enrichment of the real-time data streams, semantic semantic pattern mining and streaming analytics.

These use cases have in common that an event model is trained or an event pattern is learned and then applied to the productive streaming data for complex event recognition and streaming analytics in real-time. A further step are predictions of unknown events that are not part of the past EIS, i.e. an event that cannot be found in the historical data. For instance, this can be an industrial machine breakdown in predictive maintenance, an unforeseen pathological state in medicine, a suspicious financial transaction in fraud detection or a network intrusion in cybersecurity. Early discovery of anomalies, warning, prediction, and prevention of unwanted events is often of highest business value.

Provalet's microservice approach supports such applications by composing the needed agents with their LED processing and analytics/inference functionalities into data pipelines and more expressive rule-based workflows (Zhao et al. 2016).

3 Related Work

Different programming approaches, such as standard programming APIs to e.g., EJBs, CORBA, Web Services, REST services for data processing and data analytics, exist.

General program APIs can read and write Linked Data resources. Such programs can be in principle deployed to any device by downloading and executing them in an appropriate environment. However, the interface of typical program APIs does not offer a component model, network availability or a permission model, as it is provided by Provalets.

Enterprise Java Beans are comparable to Provalets with the main difference that EJBs cannot be installed dynamically at call-time and have no enhanced rights model.

CORBA supports remote procedure calls via an Object Request Broker. It is much more complex than REST based interaction and also does not provide a data access rights model.

Web Services and in particular REST-based Web Services share a lot of commonalities with Provalets. Web Services can run Prova agents (Paschke and Boley 2011) as rule-based inference services. The main difference lies in the difference between the traditional SOA approach and the new microservice architecture style. Traditionally, Web Services run in a monolithic style on their host server and provide standardized access to applications through loosely-coupled interfaces. In contrast, Provalet microservices are mobile components that can be composed to from an application, such as a LED analytics pipeline or a complete workflow.

Semantic Web Services, such as Semantic Annotations for WSDL (Verma and Sheth 2007), OWL-S (Martin et al. 2007), WISMO (Roman et al. 2005), support declarative semantic descriptions of their interfaces. They aim to disambiguating the description of Web services during automatic discovery and composition of the Web services. This is comparable to Provalets' description language. However, the security measures are focused on the protection of the communication channels, which requires trusted authorities and secure key exchange. In contrast to direct REST calls, as used in Provalets, the loosely-coupled communication protocols of Web Services additionally add an overhead to each service call. Web Services can use arbitrary transfer protocols, however, the predominant protocol used is HTTP.

Linked Data Services (Speiser and Harth 2011) follow the Web Service approach, acting as wrapper services for Linked Data. Linked Open Services (Krummenacher et al. 2010) combine Linked Data endpoints and REST services to a SPARQL-based[10] approach for the composition of services that communicate RDF. The main difference to Provalets is that these service approaches are not mobile. Provalets as mobile agents can be transferred to any device by downloading them as a as microservices via a REST request and executing them. Their runtime environment is not fixed at the time of publication, but only when they are requested.

Yahoo Pipes (Ankolekar et al. 2008) are used for the aggregation and processing of RSS-feeds for semantic web data. They follow the "pipes and filter" approach and allow an aggregation and manipulation of RSS-feed data via a simple user interface. Provalets provide a more expressive workflow composition language which in contrast to a simple pipes-and-filter approach also support parallel execution. The pipes approach is also not location independent, but runs on a server. Similar Semantic Web Pipes (Morbidoni et al. 2008) focus on Linked Data processing pipelines that specifically consume RDF data, but as Yahoo Pipes they do not allow the declarative implementation of own operators and are not location independent. Pipes also have the drawback that they read their input and write their output completely without maintaining any state. This prohibits parallelism in a sequential pipeline and needs the implementation of additional control structures for maintaining state, as needed e.g., for complex event processing over real-time data streams.

The W3C Linked Data Platform (LDP)[11] is a Linked Data specification defining a set of integration patterns for building REST HTTP services that are capable of reading and writing of RDF data. Like Provalets the LDP allows use of REST HTTP to consume, create, update and delete both RDF and non-RDF resources. As discussed already in the introduction, LDP follows a standard SOA approach and not a microservice style. For instance, the LDP4j framework (Gutiérrez et al. 2014) is a middleware implementation of LDP that facilitates the development of read-write Linked Data applications by using Enterprise Application Integration (EAI) techniques. Apache Marmotta[12] implements the LDP and falls into the category of linked data servers which provide data access SPARQL-based query interfaces. While the LDP and Marmotta follows a Service-Oriented Architecture (SOA), a major

[10] SPARQL Protocol and RDF Query Language is a W3C recommendation for an RDF query language.

[11] W3C LDP http://www.w3.org/TR/ldp/, accessed 1 Oct 2015.

[12] Apache Marmotta http://marmotta.apache.org/. Accessed 20 Aug 2013.

difference of Provalets is the adoption of a component-based microservice architecture (MSA) which runs mobile agents in container environments.

Apache Clerezza[13] is an OSGi-based modular application and a set of components for building REST Semantic Web applications and services. OSGi is used to provide a component model for the platform, but not for an agent based approach as in Provalets. Instead it is also a REST Web Service framework.

Data Mashup approaches, such as the one by IBM, address data extraction, data flow and data presentation. However, they are not specific to Linked Data access and have no specified composition language, rights models and declarative description language. Furthermore, they are not location independent.

Related are also so called multi agent systems (MAS) and distributed rule-based approaches, which have been surveyed in Badica et al. (2011) and Braubach et al. (2009). Provalets share underlying principles of such agent architectures. However, most of them only run in proprietary agent environments.

A feature which distinguishes Provalets from all these approaches is that the rule-base Prova agents (Paschke and Boley 2011) run as microservices in enterprise container platforms such as Java OSGi and Docker containers. Furthermore, Provalets provide support for rule-based compositions that is grounded in a formal model of concurrent transaction logic semantics (Zhao and Paschke 2013a).

4 Principles of Provalets

Provalets are location-independent mobile rule-based software agents (Paschke and Boley 2011) which are deployed as microservices in Provalet containers based on OSGi (OSGI Alliance 2009) or Docker. Using Prova[14], these Provalet microservices support rule-based linked (enterprise) data access[15], processing and inference reasoning, as well as composition and communication with messaging reaction rules. Prova (Prolog + Java) is both a declarative rule-based programming language and a Java-based rule engine. Provalets have a clear REST input and output interface, specifically an input URI and an output URI. They run in a controlled and secure Provalet container resource environment. Provalets describe their functionality in terms of pre- and post-conditions on the sets of input and output data. In the current reference implementation they are described as Maven artifacts and managed by OntoMaven (Paschke 2013a).

A Provalet container offers a service interface to grant network access to Provalets. It can manage one or more Provalets. The container resource describes itself with metadata via a REST API. A user or agent receives information about a container resource by sending an HTTP request to the container URI. For example, the container resource describes which permissions it can grant. To use a container resource for executing a Provalet the user sends a HTTP request adding three parameters to the container URI: the Provalet URI, the input URI and the output URI. In this way a Provalet can be easily executed by a standard Web client and the results can be looked up afterwards, by receiving the HTTP response of the output URI. The container configuration can restrict the access rights.

Provalets describe permissions they require as metadata that is read by the runtime environment during deployment. By default Provalets are solely allowed to see the data sources which are directly served by the configured input URI. Provalets may define additionally required permission to access other data sources, such as e.g., additional knowledge bases, for example to access additional static URIs or crawl URIs that are visible in the set of input triples. The sources of triples may be restricted by subnets, domains, protocols or even types of triples a Provalet is allowed to see. Provalet may provide HTTP access credentials to the input and output resources upon request. Provalets must request permission to use additional computing resources of the machine they are executed on. A Provalet may request harddisk space to store intermediate results. Other resources include memory, CPU time, account information, access to other web services. The latter can be used by a Provalet to enforce license models through trusted providers. It is the task of the runtime container of a Provalet to grant required permissions and allow access to requested resources.

The composition of Provalets consists of a component model, a composition technique and a composition language. Provalet components describe themselves by means of Linked Data principles: Each Provalet has a unique URI that is resolvable via HTTP. Each Provalet is configured with an input URI that it is allowed to read from and an output URI that it is allowed to write to. Furthermore, the Provalet artifact address and the executing container resource need to be defined. The runtime environment controls which data type formats and which data sources are accessible to the Provalet including the control of permissions. The Provalet description also contains semantic metadata about the Provalet including runtime dependencies and policies such as permissions required on

[13] Apache Clerezza https://clerezza.apache.org/, accessed 1 Oct 2015.

[14] http://www.prova.ws, accessed 1 Jun 2016.

[15] Prova has various built-ins for rule-based data access such as Java object access, file access, XML (DOM), SQL, RDF triples, XQuery, SPARQL.

the runtime platform as well as the description of the functionality it provides in the form of statements about pre- and post-conditions regarding the sets of input and output data, definition of types, side-effects, legal norms and policies, etc. This supports the automatic search of Provalets for their composition.

The composition of Provalets is executed by chaining (in a pipes-and-filter-style) the input and output connections via streaming connections, with the input stream acting as "pipe" and a Provalet as "filter". This supports sequential composition pipelines as well as data flow parallelization.

Compositor Provalets represent typical workflow control constructs, such as sequential execution, parallel execution, conditional alternatives and repetitions, which are used to compose Provalets to workflows. They are implemented using Prova as declarative rule-based composition language (Zhao et al. 2016). This provides users with maximum flexibility to develop new Provalet compositors and hence enrich the expressiveness of the composition language. Moreover, with Prova's support for mobile rule code, injection of functionalities into generic Provalets becomes possible.

In summary, Provalets have much in common with apps in modern application stores for mobile platforms. Further details about underlying Provalet concepts and the lifecycle of Provalets can be found in Paschke (2015).

5 Implementation

Our current proof-of-concept implementation[16] is provided as four OSGI bundles:

1. *provalet-container* A Provalet container resource is responsible for granting access to Provalets from the network via HTTP GET requests,
2. *provalet* implements the abstract classes, interfaces and dependencies for Provalets,
3. *provalet-maven-archetype* provides a Maven archetype to generate a Provalet project for the implementation of a new Provalet,
4. *provalet-test* is used to test and evaluate Provalets.

These bundles are installed and located in an OSGi target platform (using Apache Felix) which runs the OSGi core bundles and further necessary bundles, as Fig. 1 shows:

– *Prova Service* required by Provalets to use the Prova rule engine as published ProvaService,

[16] Gitlab: https://git.imp.fu-berlin.de/ag-csw/provalets.

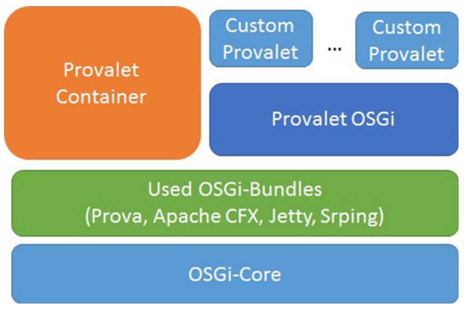

Fig. 1 Overview Provalet OSGi bundles

– *Apache CXF REST Webservice* required in the Provalet container to provide an HTTP access interface,
– *Jetty WebSockets* required by Provalets to establish and use WebSockets for the communication between Provalets,
– *Spring* supports automatic OSGi service registration and context resolution with Dynamic Modules.

A container resource is a special resource with an assigned OSGi environment for the execution of Provalets. A container resource environment, e.g., Apache Karaf (which can run in a Docker container), sets the required system parameters, starts the Java Security Manager and initializes and runs the Apache Felix OSGi framework. In the OSGi framework the Provalet container bundle is started. The Provalet container handles the Provalet call and answers the HTTP request with an HTTP response message. It can resolve the Provalet characteristics by calling the *ProvaletURI* and reading the Provalet description which also includes the necessary artifact characteristics (groupId, artifactId, version and optionally the repository). With OntoMaven's dependency resolution mechanism (Athan et al. 2014; Schäfermeier and Paschke 2014; Paschke 2013b) the Provalet artifact and all its dependencies are resolved and downloaded to a local repository using the integrated Aether library (Sonatype 2011). The downloaded artifacts are then deployed in the OSGi framework of the container resource and the *inputURI* is passed as an object to the working method of the Provalet. Finally the container resource starts the installed Provalet bundle. After executing the working method the Provalet returns its resulting data back to the container. The container checks the contents and enforces restrictions on the Provalet execution and the output and writes it into the `outputURI`.

A Provalet container consists of a HTTP REST interface (*ProvaletServiceREST*) which enables access to a particular Provalet by an OSGi service (*ProvaletService*) and contains a list of all registered Provalets in its implementation (*ProvaletServiceRESTImpl*). A Provalet (*ProvaletServiceImpl*) implements the *AbstractProvalet* with custom functionalities and properties. *AbstractProvalet* is the main class containing all important information about a Provalet. The OSGI-Activator (*AbstractProvaletActivator*) is responsible for registering the OSGi-Service implementation (*ProvaletServiceImpl*) in the OSGi-Service registry.

Once an instantiated Provalet exists, the verification of the Provalet constraints and rules can be performed using Prova's rule-based inference mechanisms. The OSGi bundle classloader is used to load the resources and instantiate a Provalet instance as OSGi component with the translated Provalet rules (and possible additional ontologies) describing the Provalet conditions and constraints. Furthermore, OSGi features (see RFC 125 OSGi) are used to make Provalet licensing information part of the machine readable component meta-data.

Provalets in distributed OSGi environments can communicate via the REST interface of the container. Therefore a Provalet can open a (streaming) socket by calling the *activeStream* function of the called Provalet. This will open a Web socket connection for the incoming and outgoing data communication between the two Provalets. The Provalet implementation uses Jetty WebSockets. The *AbtractProvalet* contains a socket manager (*SocketManager*) which is responsible for opening and connecting Web sockets. Provalets can create new socket servers (*ProvaletServer* by calling *newSocketServer*) in order to connect and work with another Provalet. Each server runs a socket thread (*SocketThread* implementing *Runnable*), which is responsible for sending and receiving messages. The streaming Web socket (StreamingSocket) will forward incoming messages (*onMessage*) and new connections (*newConnectionOccured*) to the SocketManager, which assigns it to the right Provalet for further processing.

A central goal of Provalets is to make the advantages of Linked (Enterprise) Data available for companies. Therefore, data privacy and security is a central aspect. The current version of the Provalet Container Resource is based on the Apache Felix OSGi framework. Its permission model makes use of the Java permission model. It follows a whitelist approach that consist of three elements. Standard java permissions allow the restriction of system resources. OSGi specific permissions allow the limitation of the OSGi functionality. Additionally introduced parameterized permissions, such HTTP depth access permissions, type permissions and entropy permissions, allow Provalet specific restrictions.

Finally, the container resource is responsible for uninstalling all bundles and optionally removing all the locally installed dependencies. The OSGi container continues to run. This allows hot deployment of Provalets and the start of more than one Provalet in a container at the same time.

Two working modes of container resources are defined. Asynchronously working containers immediately respond with an HTTP response code indicating that the Provalet working method was successfully started. The user of an asynchronously started Provalet has in principal two possibilities to work with the results: (1) an agent polls the output URI after a defined time, and (2) the agent uses a subscription mechanism to be informed about updates in the output URI. In the synchronous working mode of a Provalet container the agent is redirected to the output URI once the results have been successfully written into the output URI. In this working mode the user can read the result immediately after receiving the HTTP response. However, if the execution of the Provalet takes too long the server may return with a timeout.

6 Development Example

In this section we will illustrate the Provalet development with a concrete example.

To automatically generate a new Provalet a Maven archetype can be used. A Maven generated Provalet project provides all necessary dependencies and mechanism. Developers will obtain a Provalet project in which they can implement the Provalet's interfaces and abstract classes. Provalets are Maven OSGi artifacts managed in an OntoMaven artifact repository using OntoMaven (Paschke 2013a, b). The following example shows the use of the provalet-maven-archetyp:

```
mvn archetype:generate
  -DarchetypeRepository=http://www.corporate-semantic-web:8081/
   ontomaven
  -DarchetypeGroupId=de.fuBerlin
  -DarchetypeArtifactId=IfOperationProvalet
  -DarchetypeVersion=1.0
  -DgroupId=de.fuBerlin.provalet
  -DartifactId=<ProvaletArtifactID>
  -Ddeveloper=<developerName>
  -DdistributionRepositoryURL=<distrRepositoryURL>
  -DProvalet-repository-id=<distrRepositoryID>
```

The first four parameters declare the archetype's location and characteristics. Further parameters can be specified by the developer, e.g.:

- *provaletGroupID* and *provaletArtifactID* declare the artifact parameters of the Provalet that should be created.
- *distrRepositoryURL* and *distrRepositoryID* indicate the parameters of the distribution repository for the Provalet deployment.

When using this archetype Maven automatically creates the *IfOperationProvalet* with all dependencies and classes, namely an *Activator* as the entry point for the OSGi bundles and Provalet as extension of the *AbstractProvalet*. Four abstract methods from AbstractProvalet need to be implemented in the *IfOperationProvalet*:

1. *getRuleBase()* returns the Prova rule base of the Provalet.
2. *onMessage(String message)* receives the incoming messages and can optionally apply additional pre-processing and transformations before the message is forwarded to the Prova agent service.
3. *getProvaletDesription()* is used to request the Provalet's description.
4. *call()* is used for direct communication with the Provalet.

To execute the *IfOperationProvalet* the user needs to call the URI of a container resource (*container*) via an HTTP GET request providing the URI of the Provalet (*provalet*) with the input (*input*), the output URI (*output*), and three parameters for the if operation: a condition (*if*); a statement that will be executed in the case of a positive condition evaluation (*then*); and a statement that will be executed in the case of a negative condition evaluation (*else*):

```
http://de.fuBerlin.provalet/containerResource/start?provalet=http://de.fuBerlin.
provalet/repository/ifOperationProvalet&input=http://dbpedia.org/sparql/?query=
SELECT [...]&output=http://de.fuBerlin.provalet/output-pipe&if=[condition]&then=
http://de.fuBerlin.provalet/repository/ThenProvalet&else=http://de.fuBerlin.provalet
/repository/ElseProvalet
```

Figure 2 shows the control-flow of an if-then-else composition of Provalets.

When the above REST request is received, the container server activates and calls the *IfOperationProvalet*. It will call the server in order to start a *SPARQLProvalet* and open a socket connection to it. Programmatically, a socket server is create by the following code:

```
//create a new socket server
ProvaletSendMessage conn = createSocket(8123, new ProvaletOnMessage() {
    public void onMessage(ProvaletMessage message) { ... } }
)
```

After the server is started and the connection is open, a Provalet can connect to the created connection. The *IfOperationProvalet* will send the SPARQL query as a message to the *SPARQLProvalet* via the opened socket connection using Prova's send and received messaging reaction rules.

```
%Prova file: send a message
sender(Message, Answer) :-
    sendMsg(XID,osgi,"SPARQLProvalet",Message,
        {destination->"ws://127.0.0.1:8123"}),
    receiveMsg(XID,osgi,"SPARQLProvalet",Answer).
```

The *SPARQLProvalet* will receive the message and process the SPARQL query using Prova's SPARQL built-ins:

```
sparql(Connection,Input,Output) :-
 sparql_connect(ConnectionID, Connection),
 sparql_select(ConnectionID, Input, QueryID),
 sparql_results(QueryID, Output).
```

The query result is sent back to the *IfOperationProvalet* that will use it to evaluate the condition (*if*).

If the condition evaluation is positive, the *ThenProvalet* (*then*) will be started, or else the *ElseProvalet* will be started using the output stream (*output*) of the *IfOperationProvalet* as input data source.

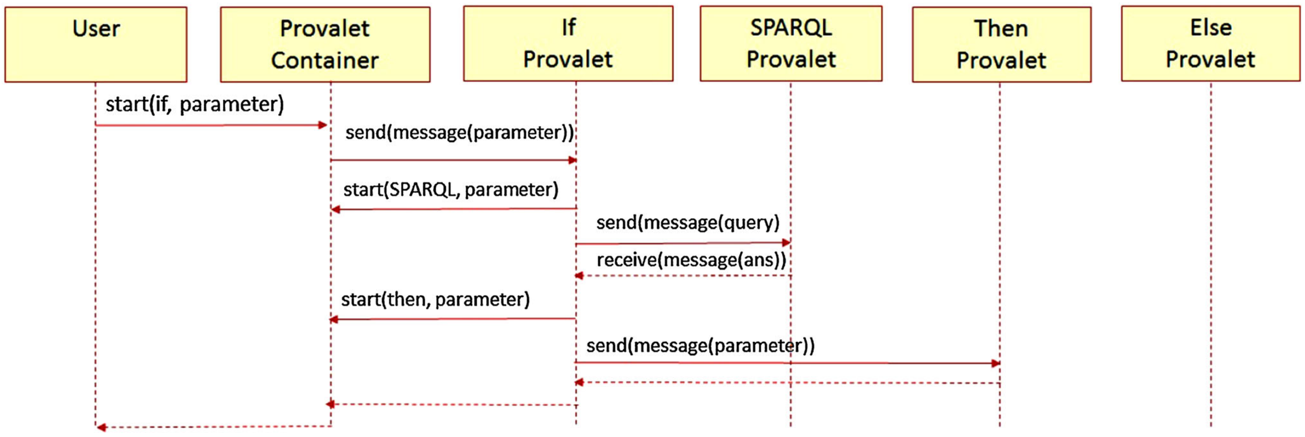

Fig. 2 Example with if-then-else composition of Provalets

By using Prova as rule language for the Provalets logic, it is possible to represent expressive conditions which go beyond simple arithmetic tests. Prova allows using Provalets for implementing expressive operators and interference logic. Moreover, for the composition of Provalets a Prova-based workflow composition technique (Zhao and Paschke 2012, 2013a, b; Zhao et al. 2016) is used and implemented by Compositor Provalets, such as in the above example IF-operator Provalet. There are three kinds of compositors: *and*, *xor* and *or*. They can be used as either *split* (one incoming, multiple outgoing branches) or *join* (multiple incoming, one outgoing branch) connectors. The task dependencies of a Provalet workflow is directly implemented by messaging reaction rules in the internal Prova agent logic. Therefore two types of Prova reaction rules are used: *inline* and *global* reaction rules. The *inline* reaction rules are usually locate in the body of a rule and act as its sub-goals. They can be restricted to accept just one message, a specified number of messages, or be limited by a timeout, which can be employed to implement a *non-local XOR* join connector, *local XOR* join connector and OR join connector, respectively. The semantics of *global* reaction rules are aligned with message (event)-driven reactive rules. A *global* reaction rule has a rule base lifetime scope, i.e., it is active while the rule base runs on a Prova engine (agent), and it is ready to receive any number of messages arriving at the Provalet.

7 Evaluation

As discussed in Sect. 3 the main distinguishing feature of Provalets, compared to other related approaches, is its component-based micro-service architecture which, by its use of rule-based Prova agents and a rule-based composition technique, allows combining Provalet agents to build applications and workflows for complex data processing and inference-based analytics tasks. Table 1 compares Provalets with related existing approaches according to the following requirements:

- *Possibility of (rule-based) data processing* Can the considered technology be used to consume and produce Linked Data resources?
- *Enhanced permission model* Does the technology offer a way to control the flow of sensitive data in a fine-grained way?
- *Network-availability* Is the considered technology useable over a network? Can standard protocols be used?
- *Location-independency* Can the corresponding code be executed on a server which the use can trust? How simple is the change of this location?
- *Composability* Does the technology offer a way to compose different elements after their compilation?
- *Scalability* Does the technology scale to large amounts of treated data?
- *Declarativity* Does it support a declarative description language which allows separation of concerns?

In summary, Provalets, with their mobile component-based and declarative rule-based approach, have a clear advantage in addressing the initially stated requirements in a Corporate Semantic Web for LED-based processing application. The existing disadvantage of Provalets (and microservices in general) is their additional complexity in terms of communication latencies. Overheads in interchange and network restrictions slow down the transfer of resources and (Onto-)Maven-based deployment of Provalet artifacts. However, by reusing instantiated Provalets in

Table 1 Provalet comparison

	Provalet	EJB	Corba	Web services	Linked data services	Yahoo pipes	Semantic web pipes	IBM mashups	Program APIs
Rule-based data access	x	x	x	x	x				x
Enhanced rights model	x								
Network availability	x	x		x	x	x	x	x	
Location independency	x		x						x
Composability	x	x	x	x	x	x	x		
Scalability	x	x	x	x	x	x		x	
Declarative language	x		x	x	x				

existing containers and by parallelization this disadvantage can be overcome.

The evaluation of the expressiveness of Provalets rule-based approach[17] is based on "Workflow Data Patterns (WDP), which capture the various ways in which data is represented and utilized in workflows" (Russell et al. 2005) and on "Workflow Control-Flow Patterns (WCP)" from the Workflow Patterns Initiative (2015).

In summary[18], of the 40 data-related WDPs, 33 patterns are supported by Provalets. The rule-based approach supports data transformations (WDP-32[19] and 33) and event-based conditional triggers and routing, as well as complex domain-specific preconditions and postconditions (WDP-34–37). As required for the security mechanisms of Provalets, data remains local in a Provalet task operation and no shared data between tasks is allowed, i.e. the data visibility patterns WDP-02, 03, 05, 06 and 07 are not supported. Based on Prova's messaging reaction rules, the composition language supports all external data interaction patterns, especially the ones which receive and respond to requests for data elements from the external environment (WDP-17–25).

From the WCPs, 26 patterns are supported, including basic workflow patterns (i.e., WCP-01–05), advanced branching and synchronization patterns (i.e., WCP-06–09, 28–33, 37–38, 41, 42), the state-based patterns (i.e., WCP-16–18, 39–40) and trigger patterns (i.e., WCP-23 and 24). This also includes state-based patterns in which decisions are made according to the currently processed data in a Provalet. Provalet's rule-based approach supports expressive decision logic, thereby supporting the patterns WCP-16 and 18 and reaction logic, supporting trigger patterns: *Transient Trigger* (WCP-23) and *Persisten Trigger* (WCP-24).

8 Conclusions

Provalets, as mobile component-based software agents, allow moving access, processing and inference-based analytics close to the data, which is essential to address typical requirements in a (partially) closed Corporate Semantic Web in which private Linked Enterprise Data is consumed. The application of the OSGi standard makes it a generic approach suitable for different environments from distributed smart Internet of Things (IoT) to centralized clouds and service platforms. The rule-based approach using the Prova rule language and agent architecture provides an expressive declarative programming model. In particular, by using the same language for the composition language, as for the rule-based data access, processing and analytics functionalities, has benefits, as this allows meta reasoning on the Provalet descriptions and makes defined operators first-class citizens in the rule-based programming logic. By using OntoMaven for the life cycle management of Provalet artifacts in standard repositories such as Git or SVN it becomes easy to use Provalets in Corporate Semantic Web applications. This makes Provalets a novel microservice programming model for LED-based applications providing solutions for various use cases, such as the ones described in Sect. 2. Future research will further generalize the description of Provalets with the upcoming OMG API4KB standard (Athan et al. 2015b; Paschke et al. 2015)[20] and will continue to optimize the Provalet architecture for streaming analytics and edge computing in resource constraint environments.

Acknowledgments This work has been partially supported by the "InnoProfile-Corporate Smart Content" project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder – Entrepreneurial Regions.

References

Ankolekar A, Krötzsch M, Tran T, Vrandecic D (2008) The two cultures: mashing up Web 2.0 and the Semantic Web. J Web Sem 6(1):70–75

Athan T, Bell R, Kendall E-F, Paschke A, Sottara D (2015a) API4KP Metamodel: a meta-API for heterogeneous knowledge platforms. In: Rule technologies: foundations, tools, and applications – Proceedings of the 9th International Symposium RuleML, Berlin, pp 144–160

Athan T, Bell R, Kendall E-F, Paschke A, Sottara D (2015b) API4KP Metamodel: A meta-API for heterogeneous knowledge platforms. In: Rule technologies: foundations, tools, and applications – Proceedings of the 9th International Symposium RuleML, Berlin, pp 144–160

Athan T, Schäfermeier R, Paschke A (2014) An algorithm for resolution of common logic (Edition 2) importation implemented in OntoMaven. In: Proceedings of the 8th International Workshop on Modular Ontologies, Rio de Janeiro

Badica C, Braubach L, Paschke A (2011) Rule-based distributed and agent systems. In: Rule-based reasoning, programming, and applications – Proceedings of the 5th International Symposium RuleML, Barcelona, pp 3–28

Balalaie A, Heydarnoori A, Jamshidi P (2016) Microservices architecture enables DevOps: migration to a cloud-native architecture. IEEE Softw 33(3):42–52

Boley H, Paschke A, Omair Shafiq M (2010) RuleML 1.0: the overarching specification of web rules. In: Semantic web rules – Proceedings of the International Symposium RuleML, Washington, DC, pp 162–178

[17] it should be noted that the evaluation of the full Prova rule language and the Prova agents, which are used within a Provalet, is out of the scope of this article and can be found elsewhere in the publications about Prova.

[18] For further details see Zhao et al. (2016).

[19] Our numbering of the patterns is according to ordering in release (Russell et al. 2006).

[20] http://www.omgwiki.org/API4KB/.

Braubach L, Pokahr A, Paschke A (2009) Using rule-based concepts as foundation for higher-level agent achitectures. In: Giurca A, Gasevic D, Taveter K (eds) Handbook of research on emerging rule-based languages and technologies: open solutions and approaches. IGI Global, Hershey, Pennsylvania (USA), pp 215–252

Dimitrov M, Simov A, Petkov Y (2015) Text analytics and linked data management as-a-service with S4. In: ESWC 2015 workshop on semantic web enterprise adoption and best practices

Gutiérrez M E, Mihindukulasooriya N, García-Castro R (2014) LDP4j: a framework for the development of interoperable read-write Linked Data applications. In: Proceedings of the ISWC Developers Workshop 2014, Riva del Garda, pp 61–66

Hu B, Svensson G (2010) A case study of linked enterprise data. In: The semantic web – 9th international semantic web conference, ISWC 2010, Shanghai, revised selected papers, part II. Springer, Heidelberg, pp 129–144

Hu B, Svensson G (2010) A case study of linked enterprise data. In: The semantic web—9th international semantic web conference, ISWC 2010, Shanghai, revised selected papers, part II. Springer, Heidelberg, pp 129–144

Krummenacher R, Norton B, Marte A (2010) Towards linked open services and processes. In: Berre A-J, Gómez-Pérez A, Tutschku K, Fensel D (eds) Future Internet – FIS 2010 – Proceedings of the 3rd Future Internet Symposium, Lecture Notes in Computer Science, vol 6369. Springer, Berlin, pp 68–77

Martin D-L, Burstein M-H, McDermott DV, McIlraith SA, Paolucci M, Sycara KP, McGuinness DL, Sirin E, Srinivasan N (2007) Bringing semantics to web services with OWL-S. World Wide Web 3:243–277

Mihindukulasooriya N, Garcia-Castro R, Gutiérrez M E (2013) Linked Data Platform as a novel approach for enterprise application integration. In: Proceedings of the 4th International Workshop on Consuming Linked Data, Sydney

Morbidoni C, Phuoc D L, Polleres A, Samwald M, Tummarello G (2008) Previewing semantic web pipes. In: The semantic web: research and applications. Proceedings of the 5th European Semantic Web Conference, Tenerife, pp 843–848

OSGI Alliance (2009) OSGi service platform, core specification, release 4, version 4.2. Technical report, OSGI Alliance

Paschke A (2011) Rules and logic programming for the web. In: Reasoning web. Semantic technologies for the web of data – 7th International Summer School 2011, Galway, Ireland, August 23–27, 2011, Tutorial Lectures, pp 326–381

Paschke A (2013a) OntoMaven API4KB – a Maven-based API for knowledge bases. In: Proceedings of the 6th international workshop on semantic web applications and tools for life sciences, Edinburgh

Paschke A (2013b) OntoMaven: Maven-based ontology development and management of distributed ontology repositories. CoRR, abs/1309.7341

Paschke A (2014) Reaction RuleML 1.0 for rules, events and actions in semantic complex event processing. In: Rules on the web. From theory to applications – Proceedings of the 8th international symposium RuleML 2014, Prague, pp 1–21

Paschke A (2015) Provalets – OSGi-based Prova agents for rule-based data access. In: On the move to meaningful internet systems – Proceedings of the confederated international conferences: CoopIS, ODBASE, and C&TC 2015, Rhodes, pp 519–526

Paschke A, Athan T, Sottara D, Kendall E-F, Bell R (2015) A representational analysis of the API4KP metamodel. In: Formal ontologies meet industry – Proceedings of the 7th international workshop FOMI 2015, Berlin, pp 1–12

Paschke A, Boley H (2011) Rule responder: rule-based agents for the semantic-pragmatic web. Int J Artif Intell Tools 20(6):1043–1081

Paschke A, Boley H (2014) Distributed rule-based agents with rule responder and reaction RuleML 1.0. In: Proceedings of the RuleML 2014 Challenge and the RuleML 2014 Doctoral Consortium hosted by the 8th International Web Rule Symposium, Challenge+DC@RuleML 2014, Prague

Paschke A, Schäfermeier R (2015) Aspect OntoMaven – aspect-oriented ontology development and configuration with OntoMaven. In: Abramowicz W (ed) Business information systems workshops – BIS 2015, Pozna, revised papers, vol 228

Paschke A, Schäfermeier R (2015) Einordnung und Abgrenzung des Corporate Semantic Webs. In: Ege B, Humm B, Reibold A (eds) Corporate semantic web. X.media.press, Springer, Heidelberg, pp 11–21

Roman D, Keller U, Lausen H, de Bruijn J, Lara R, Stollberg M, Polleres A, Feier C, Bussler C, Fensel D (2005) Web service modeling ontology. Appl Ontol 1(1):77–106

Russell N, van der Aalst W M P, Mulyar N (2006) Workflow control-flow patterns: a revised view. Technical report, BPMcenter.org

Russell N, ter Hofstede AHM, Edmond D, van der Aalst WMP (2005) Workflow data patterns: identification, representation and tool support. In: Proceedings of the 24th International conference on conceptual modeling ER'05. Springer, Heidelberg, pp 353–368

Schäfermeier R, Paschke A (2014) Aspect-oriented ontologies: dynamic modularization using ontological metamodeling. In: Formal ontology in information systems - Proceedings of the 8th International Conference FOIS Rio de Janeiro, pp 199–212

Schmachtenberg M, Bizer C, Paulheim H (2014) Adoption of the Linked Data best practices in different topical dDomains. In: The Semantic Web – ISWC 2014 – proceedings of the 13th international semantic web conference. Riva del Garda, Part I, pp 245–260

Sonatype (2011) Aether. http://aether.sonatype.org/. Accessed 23 Sep 2015

Speiser S, Harth A (2011) Integrating Linked Data and services with Linked Data Services. In: The semantic web: research and applications – Proceedings of the 8th extended semantic web conference ESWC 2011. Heraklion, Part I, pp 170–184

Teymourian K, Paschke A (2016) Semantic enrichment of event stream for semantic situation awareness. Springer, Cham, pp 185–212

Vermad K, Sheth A (2007) Semantically annotating a web service. IEEE Internet Comput 11:83–85

Wood D (2010) Linking enterprise data. Springer, New York

Workflow Patterns Initiative (2015) Workflow Patterns. http://www.workflowpatterns.com/. Accessed 23 Sep 2015

Zhao Z, Paschke A (2012) Event-driven scientific workflow execution. In: Business process management workshops – BPM 2012 international workshops. Tallinn, Revised Papers, pp 390–401

Zhao Z, Paschke A (2013a) A formal model for weakly-structured scientific workflows. In: Proceedings of the 6th international workshop on semantic web applications and tools for life sciences, Edinburgh

Zhao Z, Paschke A (2013b) Rule agent-oriented scientific workflow execution. In: Proceedings of the 5th international conference S-BPM ONE – running processes, Deggendorf, pp 109–122

Zhao Z, Paschke A, Ruisheng Z (2016) A rule-based agent-oriented approach for supporting weakly-structured scientific workflows. J Web Sem 37:36–52