JSON SPARQL Application Profile for Linked Data

Andrea Ferrari*, Elisa Riforgiato*, Luca Roffia‡
* VAIMEE srl, Italy
‡University of Bologna, Italy
Email:andrea.ferrari@vaimee.it,elisa.riforgiato@vaimee.it,luca.roffia@unibo.it

***Abstract*—This paper presents the JSON SPARQL Application Profile for Linked Data which is a JSON-LD compliant file that can be used to describe applications powered by the SPARQL Event Processing Architecture. Thanks to a publish-subscribe broker built on top of a generic SPARQL endpoint, the architecture allows the development of distributed and context-aware applications based on interoperable microservices. The SPARQL Application Profile includes all the information needed to describe an application, like the SPARQL queries and updates used by each microservice as well as the parameters needed to connect to a broker instance. The paper presents the ontology that has been designed to represent the SPARQL Application Profile from which the JSON-LD context has been eventually extracted.**

I. Introduction

Services and application are usually designed in a static way: they are connected each other in a fixed schema and their interaction is usually described using a set of configuration files. With the spread of the Internet of Things (IoT), more and more flexible methods and tools are required to build, deploy and manage services as well as applications. This is because IoT applications are heavily based on the interaction with the physical environment which imposes to deal with heterogeneous and dynamic data sources. Here a critical aspect is to ensure interoperability to services and clients. Thereby, the possibility to reuse in part, or total, other services to deploy new ones would represent a very important asset. In this paper, we propose an approach based on Semantic Web ontologies described in OWL [1] which can provide a formal and shared representation of the application and its components. The representation includes all the details, starting from the microservice configuration up to its own setup for composing and deploying. We consider as reference architecture the SPARQL Event Processing Architecture (SEPA) [2] which implements a publish-subscribe mechanism over a generic SPARQL endpoint [3]. SEPA provides developers with an application design pattern named PAC (Producer-Aggregator-Consumer) which assumes a software component to be: a producer (i.e., linked to a specific SPARQL update), a consumer (i.e., linked to a specific SPARQL query used as subscription request) or an aggregator (i.e., on receiving events related to a specific SPARQL query it is subscribed to, it performs a specific SPARQL update) [4]. As shown in 1, from the JSAP Ontology can be derived a JSON-LD context that can be used by developers to compose the JSON-LD file of their applications. This file includes in general also references to contexts of specific vertical ontology (e.g., Schema.org [5], SOSA [6]). The JSAP Ontology, and as consequence the JSON-LD context, is composed by three logical parts: *CORE* that contains all the basic information of the services, *PAC* that includes the information on the business logic and *DEPLOY* that targets the deployment of all services. This paper aims to present the JSAP-LD and the process we followed to build it starting from a previous version which was just in plain JSON [7]. As a first work in this direction, the current implementation focuses on the CORE part of the JSAP-LD. In the Fig. 1 is possible to have a look at the JSAP-LD ultimate overview. Since the JSAP-LD is based on an ontology it is suitable to be converted into RDF triples and uploaded inside SEPA. The rest of the paper is structured as follows. As first we present the background about SEPA and the JSAP configuration file, as plain JSON. Then an introduction about JSAP-LD follows, along with the construction process of the JSAP Ontology. A section is dedicated to the extracted JSAP-LD context, and a real use case is presented including a comparison with the plain JSON format. Eventually, discussions and future works are drawn.

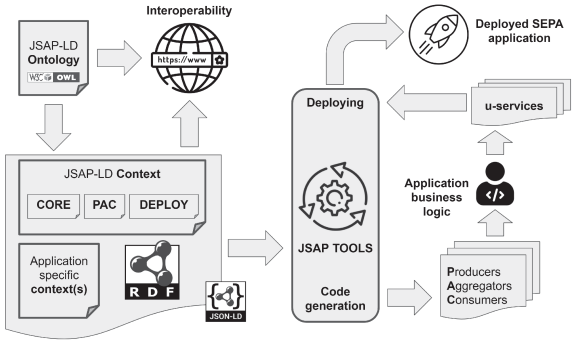

Fig. 1. Overview of JSAP-LD environment and usage

II. Reference architecture and technologies

SEPA is a decentralized Web-based architecture designed to support the development of distributed, dynamic, context-aware, and interoperable services and applications. A full description of an application based on SEPA is already available as a JSON file named JSAP [8] (*JSON SPARQL Application Profile*).

SEPA enables the so called Dynamic Linked Data [9], a layer of distributed publish-subscribe brokers built on top of Linked Data where publishers and subscribers use respectively W3C SPARQL Updates and Queries. A graphical representation is provided by the Fig. 2. Notifications include the added and removed binding results in the SPARQL Query due to an incoming SPARQL update. SEPA implements the PAC (*Producer-Aggregator-Consumer*) design pattern with a reusable and extensible set of components. A generic component can be one of the following: *Producer* which corresponds

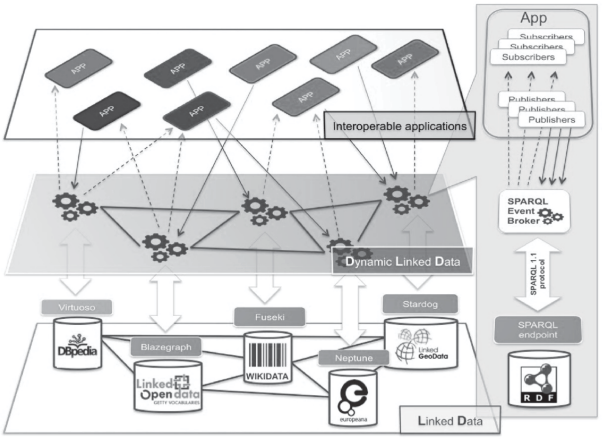

Fig. 2. Dynamic Linked Data, JSAP-LD content

to a publisher, a *Consumer* which corresponds to a subscriber and an *Aggregator* that plays both the roles. For further details please refer to IEEE Internet of Things Journal[10].

III. On the design of a semantic description of a SEPA application

The main scope of the JSAP-LD - which stands for an enriched version of JSAP in JSON-LD [11] - is to represent in a semantic way what a SEPA application can do. Developers are recommended to follow the PAC design pattern and to use JSAP-LD in order to describe the SPARQL queries and updates used by their applications. This would be the key to enable interoperability and drive the reuse of application's components. JSAP-LD allows to modify the microservices' interactions, without have to depend on the implementation of the microservices themselves. Starting from the plain JSON JSAP configuration file [9] a JSAP Ontology has been designed and then the corresponding JSON-LD context extracted.

A. JSAP Ontology

The first step of this work has been the design of the JSAP Ontology through a middle-out approach [12] with the interest in answering to the following competency questions [13]:

1) What is a JSAP?
2) Which are the main component parts?
3) At what level of specification do we need to represent these parts?
4) How this ontology may be also suitable to extrapolate the context for a JSON-LD format of JSAP?

Following the guidelines given by *Mcguinness* [14], the explication process [15] of the JSAP Ontology takes also into account the conceptualization models, the mental model creator and the mental model user. The ontology, as a particular model of explicit conceptualization, provides the alignment between the two models and the creation of a common knowledge base for the creator and the user. The major benefit of this choice is given by an active user participation since this decrease both the possibility of making conceptualization mistakes and the effort for correcting them on a long term project [16]. Thereby, people within a multi-agent interaction system are not just users, but full-fledged actors [17]. A sketch of the construction process of the JSAP Ontology based on conceptualization models approach [15] [16] is shown in Fig. 3.

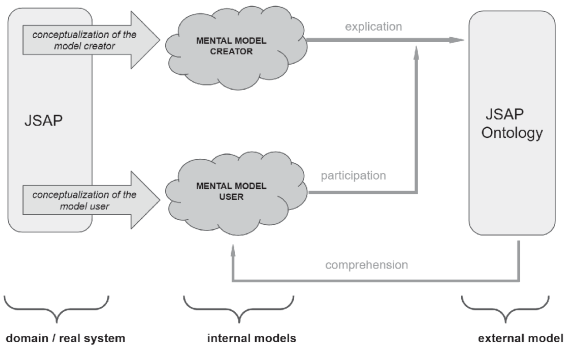

Fig. 3. JSAP Ontology conceptual modeling process

During the ontology creation process, as a result of this dual cooperation between the user and the creator, we realized that the current JSAP version wasn't optimized for the applications that it would be possible implemented, mostly looking into the future of SEPA. This was a real turning point of our work, that had to take into account what the JSAP was and what the JSAP, and even more JSAP-LD, would become. In this new view, JSAP is more versatile and open to new extensions. A practical example could be the use of the *host* member inside the JSAP: originally the JSAP configuration let to set just one *host* for each client but the JSAP Ontology and the JSAP-LD are actually able to provide more hosts per client, using the *host* member related to the SPARQL protocols. A concrete example will be give in the Section IV.

Once scope, models and main parts were setup, the JSAP Ontology was developed in OWL format [1] inside the Protégé editor [18]. Fig. 4 gives an overview of the main ontology classes and their relationships. As already mentioned in the introduction, the JSAP-LD is supposed to be composed of three main parts, namely *CORE*, *PAC* and *DEPLOY*. The current version of the ontology, and as consequence the derived JSON-LD context, is focused on the CORE part. The ontology specification level doesn't involve instances, that is because the ontology has to provide a context that makes possible to express the JSAP-LD instance: the instances are expressed inside the JSAP-LD and supported by the ontology semantic. Moreover, although the first released version of JSAP Ontology is enough expressive for the given purpose, it could be considered as a *simple* ontology which is quite convenient given the largely endorsed idea that a less complex application-domain ontology provides more re-usability - making it easy to be extend for more specific applications - and dynamism [19]. The semantic introduced with the JSAP Ontology is completely suitable for generating a JSON-LD context to be included in a JSAP-LD file through the entities and the properties, semantically and syntactically defined inside the ontology. As soon as the extrapolation process into JSON-LD context occurred, syntax provide by ontology was no longer available inside the JSAP-LD. The aim of this work was not

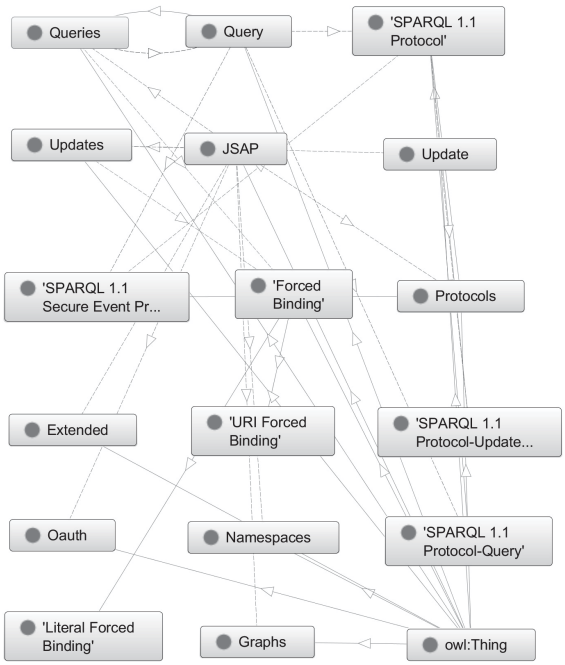

Fig. 4. JSAP Ontology classes and related object properties

developing a JSON-LD format of the JSAP Ontology itself but we wanted to enrich the JSAP - in JSON format- with a semantic enable capacity using the semantic features and a controlled vocabulary expressed by an ontology. By providing a formal specification and a structured semantic model to the JSAP and turning it into a JSAP-LD, we enhanced its potential via interoperability of services and applications.

B. JSON-LD Context

Starting from the JSAP Ontology, the extracted JSON-LD context contains all the mapping of terms to URIs which can than be used within a JSAP-LD document to define classes, data properties, and objects properties. In addition, we have defined a *@vocab* equal to the general URI of the JSAP Ontology and with the slash symbol instead of the sharp symbol as last character. The purpose of that *@vocab* is to allow the existence of possible terms which are not strictly defined in the context and it is mainly used as default prefix. In the listing 1 there is an extract of the JSON-LD context of JSAP.

Listing 1. JSAP context.

```
"@context" : {
    "@vocab":"http://www.vaimee.it/ontology/jsap/",
    "queries" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#queries",
      "@container": "@set"
    },
    "versionInfo" : {
      "@id" : "http://www.w3.org/2002/07/owl#versionInfo"
    },
    "sparql" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#sparql",
      "@type" : "http://www.w3.org/2001/XMLSchema#string"
    },
    "reconnect" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#reconnect",
      "@type" : "http://www.w3.org/2001/XMLSchema#boolean"
    },
    "protocol" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#protocol",
      "@type" : "http://www.w3.org/2001/XMLSchema#string"
    },
    "graphs" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#graphs",
      "@container": "@set"
    },
    "host" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#host",
      "@type" : "http://www.w3.org/2000/01/rdf-schema#Literal"
    },
    "updates" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#updates",
      "@container": "@set"
    },
    "path" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#path",
      "@type" : "http://www.w3.org/2001/XMLSchema#string"
    },
    "enable" : {
      "@id" : "http://www.vaimee.it/ontology/jsap#enable",
      "@type" : "http://www.w3.org/2001/XMLSchema#boolean"
    },
    ...
}
```

IV. Use case: a simple but complete example of a SEPA chat application

The use case simulates a chat system in which the users have a private chat with each other. A user can register herself in the chat system (i.e., by means of the *UserRegistration* producer), and then she can send and receive messages (i.e., by means of the *ChatClient*). As shown in Fig. 5, the chat system also includes a consumer which maintains the list of available users. A message is stored into the RDF store until the sender will be notified of a successful reception. At this point, the sender will remove the message from the RDF store.

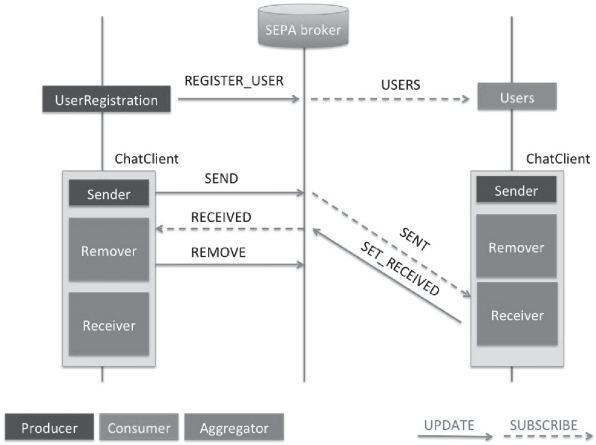

Fig. 5. Components of the chat systems and their interactions through the publish-subscribe mechanism of SEPA. Labels are symbolic identifiers of updates and queries used by components

The JSAP-LD file describing the chat system has been designed starting from the already available JSAP file in plain JSON [20]. A fundamental part of the JSAP is to describe how to connect to a specific broker, using the SPARQL 1.1 Protocol [3] and the extension provided by SEPA to implement subscriptions [21] [22]. Listing 2 and listing 3 show this part respectively as it was in the JSAP and as it is now in JSAP-LD. The advantage of using JSAP-LD is twofold: on the

one hand it enables interoperability, on the other hand it is open to extensions. For example, if new protocols are defined in the JSAP Ontology, these can be added to the JSAP-LD protocols section just including a new "@type". The queries and the updates will be referring to the right protocol using the properly "@id".

Listing 2. JSAP protocols section in the Chat example.

```
"sparql11protocol": {
        "protocol": "http",
        "port": 8000,
        "query": {
                "path": "/query",
                "method": "POST",
                "format": "JSON"
        },
        "update": {
                "path": "/update",
                "method": "POST",
                "format": "JSON"
        }
},
"sparql11seprotocol": {
        "protocol": "ws",
        "reconnect" : true,
        "availableProtocols": {
        "ws": {
                "port": 9000,
                "path": "/subscribe"
        },
        "wss": {
                "port": 9443,
                "path": "/secure/subscribe"
        }
        }
}
```

Listing 3. JSAP-LD protocols section in the Chat example.

```
"protocols":[
  { "@id":"http://chat/protocol_01",
    "@type":"sparql11protocol",
    "protocol": "http",
                "port": 8000,
                "path": "/updateAndQuery",
                "method": "POST",
                "format": "JSON",
        "host": "localhost"},
  { "@id":"http://chat/protocol_02",
    "@type":"sparql11protocol-query",
    "protocol": "http",
                "port": 8000,
                "path": "/query",
                "method": "POST",
                "format": "JSON",
        "host": "localhost"},
  { "@id":"http://chat/protocol_03",
    "@type":"sparql11protocol-update",
    "protocol": "http",
                "port": 8000,
                "path": "/update",
                "method": "POST",
                "format": "JSON",
        "host": "localhost"  },
  { "@id":"http://chat/protocol_04",
    "@type":"sparql11seprotocol",
    "protocol": "ws",
                "port": 9000,
                "path": "/subscribe",
                "reconnect" : true,
        "host": "localhost"  },
  { "@id":"http://chat/protocol_05",
    "@type":"sparql11seprotocol",
    "protocol": "wss",
                "port": 9443,
                "path": "/secure/subscribe",
                "reconnect" : true,
        "host": "localhost"  }
]
```

Listing 4 shows the semantic description of the "SEND" SPARQL update. It is used by a chat user, the sender, to send a message to another one, the receiver. That listing is an example of how to define force bindings, which is an important feature provided by the SEPA APIs. The force binding allows to replace a SPARQL variable at run-time. If we consider the following three forced bindings *http://chat/fb/msg*, *http://chat/fb/sender* and *http://chat/fb/receiver*, they are semantically described by the type, name, and if needed the dataType. At any time the application can use them to replace the corresponding SPARQL variable. All the semantic entities in a JSAP-LD, such as the force bindings, the updates, and the protocols, are referred by an "@id", which means they are URIs. This allows a JSAP-LD to be split in different files or locations. For example, a file may contain all the updates, while another all the available protocols. A JSAP-LD or part of it may also be stored as RDF into a SPARQL endpoint. In such a way, the application can be modified at run-time which could be a very interesting feature in some application contexts.

Listing 4. An example of SPARQL Update in JSON-LD.

```
"updates": [
{
"@id":"http://chat/update/send",
"@type":"update",
"label":"visible name of the update",
"description":"description ...",
"sparql": "INSERT{ graph <http://chat> {
        _:message rdf:type schema:Message;
        schema:text ?text;
        schema:sender ?sender;
        schema:toRecipient ?receiver;
        schema:dateSent ?time
    }} WHERE {
        ?sender rdf:type schema:Person.
        ?receiver rdf:type schema:Person.
        BIND(STR(now()) AS ?time)
    }",
"hasProtocolUpdate":"http://chat/protocol_06",
"forcedBindings":[
    {
      "@id":"http://chat/fb/msg",
      "@type": "LiteralForcedBinding",
      "name": "text",
      "default": "Ciao!",
      "dataType" : "xs:string"
    },
    {
      "@id":"http://chat/fb/sender",
      "@type": "UriForcedBinding",
      "name": "sender",
      "default": "chat:IamASender"
    },
    {
      "@id":"http://chat/fb/receiver",
      "@type": "UriForcedBinding",
      "name": "receiver",
      "default": "chat:IamAReceiver"
    }
]} ... ,
```

V. Conclusion and future work

In this paper, we presented a project supporting the development of SEPA based applications. Thanks to JSAP-LD it is possible to give a semantic description of an application at different levels, starting from connection details to a set of brokers, the SPARQL primitives used by the application components, up to the description of how to compose and deploy the application. The semantic is provided by the JSAP Ontology from which the JSAP-LD context has been extracted and that can be used by developers to design well-formed JSAP-LD descriptions. At the current stage, the CORE part

of the JSAP-LD has been implemented and the next step is to start to rewrite the current parser based on the plain JSON format, including also a validation of the semantic content. The JSAP Ontology, the JSAP JSON-LD context as well the JSAP-LD files of the use case are all available on Github [20]. We are at an early stage of this work but we are getting one step closer to allow the SEPA environment to be more and more interoperable, modular, flexible and extensible as supplier of services and applications in the IoT world.

REFERENCES

[1] W3C, "OWL Web Ontology Language Overview." Web:https://www.w3.org/TR/owl- features/.

[2] C. Aguzzi, F. Antoniazzi, L. Roffia, and F. V. Viola, "SPARQL Event Processing Architecture (SEPA)." http://mml.arces.unibo.it/TR/ sepa.html, 2018.

[3] L. Feigenbaum, G. T. Williams, K. G. Clark, and E. Torres, "SPARQL 1.1 Protocol." https://www.w3.org/TR/sparql11-protocol/, 2013.

[4] C. Aguzzi, F. Antoniazzi, L. Roffia, and F. Viola, "Producer-Aggregator-Consumer design pattern." http://mml.arces.unibo.it/TR/jsap.htmlproducer-aggregator-consumer-design-pattern, 2018.

[5] "Schema.org." https://schema.org/docs/developers.html.

[6] A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, and M. Lefranois, "Semantic Sensor Network Ontology, SOSA (Sensor, Observation, Sample, and Actuator)." https://www.w3.org/TR/vocab-ssn/, 2017.

[7] "JSON (JavaScript Object Notation)." https://www.json.org/json- en.html.

[8] F. Viola, C. Aguzzi. F. Antoniazzi, L. Roffia, "JSON SPARQL Application Profile (JSAP)." http://mml.arces.unibo.it/TR/jsap.html, 2018.

[9] L. Roffia, P. Azzoni, C. Aguzzi, F. Viola, F. Antoniazzi, and T. Salmon Cinotti, "Dynamic Linked Data: A SPARQL Event Processing Architecture," Future Internet, vol. 10, no. 4, 2018.

[10] L. Roffia, F. Morandi, J. Kiljander, A. DElia, F. Vergari, F. Viola, L. Bononi, and T. Salmon Cinotti, "A Semantic Publish-Subscribe Architecture for the Internet of Things," IEEE Internet of Things Journal, vol. 3, no. 6, pp. 1274–1296, 2016.

[11] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindstrm, "JSON-LD, A JSON-based Serialization for Linked Data, W3C Recommendation 16 July 2020." https://www.w3.org/TR/json-ld11/, 2019.

[12] M. Uschold and M. Grninger, "Ontologies: Principles, methods and applications," The Knowledge Engineering Review, vol. 11, 1996.

[13] M. Gru ninger and M. S. Fox, The Role of Competency Questions in Enterprise Engineering, pp. 22–31. Springer US, 1995.

[14] N. F. Noy and D. L. Mcguinness, "Ontology development 101: A guide to creating your first ontology," Knowledge Systems Laboratory, vol. 32, 2001.

[15] R. Schtte and S. Zelewski, "Epistemological problems in working with ontologies.," The 6th World Multiconference on Systemics,Cybernetics,and Informatics, 2002.

[16] R. Ferrario, N. Guarino, C. Janiesch, T. Kiemes, D. Oberle, and F. Probst, "Towards an ontological foundation of services science: The general service model," 10th International Conference on Wirtschaftsinformatik, 2011.

[17] N. Guarino, E. Bottazzi, R. Ferrario, and G. Sartor, "Open ontology-driven sociotechnical systems: Transparency as a key for business resiliency," in Information Systems: Crossroads for Organization, Management, Accounting and Engineering (M. De Marco, D. Te'eni, V. Albano, and S. Za, eds.), (Heidelberg), pp. 535–542, Physica-Verlag HD, 2012.

[18] Stanford University, "Protegè." https://protege.stanford.edu, 2016.

[19] D. Mcguinness, "Ontologies come of age," Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential, pp. 171–194, 01 2003.

[20] A. Ferrari, E. Riforgiato, and L. Roffia, "JSAP-LD Github repository." https://github.com/arces- wot/JSAP- LD.

[21] C. Aguzzi, F. Antoniazzi, L. Roffia, and F. Viola, "SPARQL 1.1 Subscribe Language," 2018.

[22] C. Aguzzi, F. Antoniazzi, L. Roffia, and F. Viola, "SPARQL 1.1 Secure Event Protocol, Unofficial Draft 12 October 2018," 2018.