Complex Environment Evolution

Challenges with Semantic Service Infrastructures

Andrej Eisfeld
Furtwangen University, Germany

David A. McMeekin
Curtin University, Australia

Achim P. Karduck
Furtwangen University, Germany
Curtin University, Australia

***Abstract*—Service infrastructures are often key for the efficient operation of complex environments, such as a large Smart Home infrastructure for a mining camp. The evolution of such service infrastructures, in response to an increased workload on the system or a changing resource allocation, is often tedious and expensive, due to weak evolvability support of its service portfolio. This underpins the need for services to be designed with high evolvability characteristics. Semantic Web technologies have been anticipated as a basis for the required Web service evolution. JSON-LD is a prominent Semantic Web technology used in combination with ontologies, developed with the Web Ontology Language (OWL), to face the evolutionary challenges. Our applied research investigates common Web service tasks, automated using above technologies. It is explained, that this enables the creation of evolutionary building blocks. These building blocks are incrementally combined here for the overall service portfolio, culminating in a model of a semantic agent which excels in its capacity to evolve. Our model is then adopted and assessed for the system infrastructure of the Smart Camp project, as a use case for an agile, complex logistic environment.**

***Keywords*—Semantic Service Infrastructures, System Evolvability and Sclability, Semantic Agents, Smart Camp.**

I. Introduction

WEB services, which communicate with each other in a network, form a service infrastructure that is often the key for the efficient operation of a complex environment. The resulting service infrastructures have to face huge challenges over time. One of them is described by the following quotation from Lehman [1]:

> "*Continuing Change – E-type systems must be continually adapted or they become progressively less satisfactory*"

Above quote shows one law of the series of **Laws of Software Evolution** formulated by Lehman and Belady starting in 1974. The term "E-Type" hereby stands for programs which are embedded in the real world. This law of software evolution indicates that software systems have to be changed over time. Consequently, Web services also have to be evolved continually to maintain service and infrastructure performance. However, the actual process of evolution is costly due to the involved effort to adapt to changes. This effort is even bigger in a service infrastructure due to the inherent dependencies between its Web services.

Our work applies Semantic Web technologies to support the evolution of Web services. It fills the applied research gap of Semantic Web by applying them in a large mining camp infrastructure, the Smart Camp project, which is briefly described in the next section.

II. Smart Camp

A high demand for Australian mineral resources has led to a large increase in mining operations within the remote regions of Australia where most of these resources are located. Providing accommodations, including domestic needs, for the labor force is an expensive task. A substantial proportion of these costs are due to both high cost and high usage of electricity that must be produced locally. The Smart Camp project [2] arose from the imperative to save energy within these camps. An initial prototype of the Smart Camp system utilized two kinds of software solutions: (SHC) a control unit which interacts with its environment to regulate the consumption of electricity and (SCMU) a central management unit for the coordination of the SHCs. These components utilise Web services which form the service infrastructure in a Smart Camp:

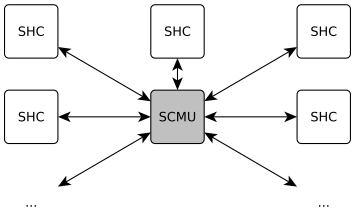


Fig. 1. Service Infrastructure in a Smart Camp

These Web services change over time in functionality and location, since they have to be adapted to new requirements. This is typical for an agile, complex service infrastructure. Our use case focuses not on the service bundling in agile, complex environments, but on the underlying agile service invocation. Possible changes in this context are the addition of new sensors or actuators or the removal of old ones. This growth/reduction of information can lead to the need for a rearrangement of information in resources offered by Web services. If the system archi-

978-1-4673-0311-8/12/$31.00

tecture is based on a tight URI coupling, these changes involve to change depending service consumers.

III. Related Work

There are numerous approaches to enhance existing Web services with a semantic layer. The most popular are SAWSDL [4], SA-REST [5] and hRESTs [6] which is used in combination with MicroWSMO [7]. SAWSDL is a semantic annotation mechanism for existing WSDL [8] descriptions of a service's interface. Hereby the semantic layer is given by annotating WSDL documents with links to RDF descriptions. SA-REST is another semantic annotation mechanism inspired by SAWSDL. Instead of using existing WSDL descriptions, SA-REST annotates HTML documentations of RESTful Web services. hRESTs in combination with MicroWSMO uses the same approach as SA-REST to add semantics to the interface of a service. The problem with all of the above approaches is that they are based on a RPC style and are not resource-oriented and therefore "do not align well with clear RESTful service design"[10]. They also all describe the service's interface and thereby introduce a tight coupling of the Web service to its interface description.

There are also approaches which are resource-oriented such as EXPRESS [11] and ReLL [10]. EXPRESS is not further considered, since it is not suitable to build upon existing service infrastructures [9]. The Resource Linking Language (ReLL) describes resources and transforms their data to RDF, which then can be used for further processing. The problem with ReLL is, that it currently only supports HTTP GET operations [9] and therefore is also not further investigated.

Further, promising solutions are JSON-LD [12] and SEREDASj [9]. Both are based on the lightweight data serialization format JSON [13] and add semantics to existing JSON documents by linking data in the document to their description. Therefore both approaches are data-oriented. Differences between them are, that SEREDASj additionally provides a description of the data structure in a JSON document and by this allows clients to automate processing data without caring about the data hierarchy.

However, JSON-LD is a more sophisticated solution with implementations in almost any programming language. Also from a Smart Camp perspective JSON-LD is the best choice since the current implementations use JSON as a data serialization format and these can be easily extended with a semantic layer by adding some meta data to existing JSON documents. Therefore JSON-LD is the choice of technology for our work.

IV. Automation of Common Web Service Tasks

Resource discovery, composition and invocation are highly desirable processes for Web services. Approaches to automate these processes are given in this section. A set of requirements was also derived as a solid base for this purpose.

A. Requirements

The first important requirement is about the applied Semantic Web technologies. As stated in the last sections, OWL is chosen for the creation of ontologies and JSON-LD is chosen for interlinking resources provided by a RESTful Web service and thus add a semantic layer to the service. This work defines the result of the latter process as a **Linked Resources Graph (LRG)**.

Another important requirement is, that the RESTful Web service is designed guided by the approach, how human stakeholders generally offer services to their customers. Human stakeholders are expected to offer services if they have a priori knowledge in the service domain (ontology). For the development of Web services this means that there is an existing ontology which is used to design the offered service. This includes marking data and links with ontological concepts such that they can be understood by service consumer. This process results in the fact, that the LRG is a part of the ontology graph. Noteworthy is here, that concepts which are not included in the ontology can still be used in the LRG if this concept can be reasoned with the help of the ontology.

There is also the goal for a smooth communication. All parties involved in a communication should share a common ontology in their domain. However, slight differences are allowed if the unknown words can be reasoned. This common ontology can be stored in a central unit such as the SCMU in a Smart Camp.

B. Resource Discovery

Resource or service discovery is about finding a concrete resource offered by a RESTful Web service which totally or partially matches the client's need. It is therefore the most important task for a service consumer since everything relies on finding the best fitting resource for the client. It is the foundation of resource composition and invocation. The proposed approach in this section focuses on the discovery of resources of known Web services, and thus focuses on the semantic issues. This means, that the actual machine where the Web service is running on has to be discovered a priori.

The here adopted approach for resource discovery relies on a Web service which is build under the requirements introduced in the previous section. If this is the case, one can use ontology paths to constraint the traversal of the Linked Resources Graph (LRG) starting from the root resource in the graph. Figure 2 shows a general ontology graph and the resulting ontology paths to the ontological concept "12" as an example. This work defines the resulting green paths from the root of the ontology graph to the chosen concept as the **Not Permitted Ontology Paths (POPs)** and the red paths which are not leading to this concept as **Permitted Ontology Paths (NPOPs)**. The actual traversal of the LRG is then done by checking if a given semantic link in the LRG is an element of the POPs.

If this link is on a NPOP then the client is not allowed to follow this link.

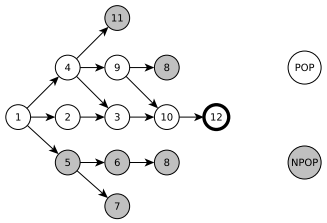


Fig. 2. POPs and NPOPs for the goal "12"

C. Resource Composition

Resource composition is the process of combining existing resources in a way such that they provide enriched information which can help clients to perform new functions. The result of this process is called a composite resource. The actual composition can simply be the combination of multiple resources, or the combination of parts of multiple resources. This section shows one approach which is limited to compose resources which include interrelated data in their representations.

As stated above, the proposed resource composition approach is limited to compose resources, in which data is somehow related to each other. One possible relationship between data in the representations is, that they link to ontological concepts, which describe the same object in an abstract sense. In ontologies, this is especially the case for super-classes of two or more OWL classes. Thus, the service consumer can define a concept, which is at a higher layer in the ontology graph as the goal of a request. Consequently, this triggers a possible composition of multiple resources. Figure 3 illustrates this case. Picking the concept "5" as the goal of a request results in the fact, that all paths in the ontology graph which are underneath "5" are permitted. All concepts underneath "5" then can be seen as sub-goals, and this sub-goals can actually lead to multiple resources during the discovery process. Another possibility to achieve the composition of multiple resources is picking a concept which is used in several nodes in the ontology graph (see goal "8" in Figure 3).

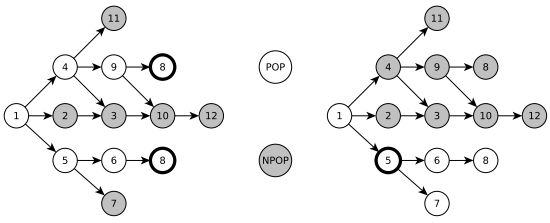


Fig. 3. POPs and NPOPs for the goals "8" and "5"

D. Resource Invocation

Resource invocation is the actual task of performing a CRUD (Create, Retrieve, Update, Delete) operation on a resource to achieve an outcome. The actual invocation of HTTP GET requests on a resource is trivial since normally no additional information is needed to retrieve a resource's representation. However, thinking of other operations there might cause problems, such as the need of information given in the message body or parameters in the URI (e.g. in the URI "http://www.example.org/test.php?parm=123", the parameter "parm" may be needed for the actual invocation). Another challenge is the fact, that the client does not know which of the provided resources to invoke (since the resource is discovered automatically). This makes resource invocation even more difficult. The following approach deals only with the expected message body to invoke a resource and does not cover parameters in the URI.

As a first step of the resource invocation process, the resource has to be discovered using the proposed resource discovery approach. From this point on, there are two ways to invoke the resource: the first is "asking" the resource how it should be invoked and the second is to pass a self-descriptive message and assume that the service understands it. The first approach represents the way service invocation is done using a service interface description document such as WSDL. Since in this work we are using JSON-LD, the second approach is chosen for resource invocation. This approach assumes for example, that if the representation of a resource is about the term "AccessCard", then one can pass a new representation to the resource to perform an operation on "AccessCard". Below listings show how a value is updated in general by performing a HTTP PUT request. HTTP DELETE requests can be invoked similarly.

HTTP GET

```
{
   "@context":{
      "a":"ontoA"
   },
   "@type":"Z",
   "a":"valueA",
}
```

HTTP PUT

```
{
   "@context":{
      "a":"ontoA"
   },
   "@type":"Z",
   "a":"valueB"
}
```

V. Putting it all Together: A Semantic Agent

Our work derives a model of a semantic agent by defining two components which are responsible for performing service requests and handling responses using Linked Resources Graphs and ontologies.

A. Request Handler

The tasks resource discovery, composition and invocation can be seen as building blocks which can be combined to automate a request handling. This automation results in a great flexibility for the service provider to make changes in his offered Web services. The figure below depicts an approach to combine these tasks with the result that giving a goal **g** as input, the client receives a representation **r** of a (maybe composed) resource. In the following this process

combining resource composition, discovery and invocation is called a **Semantic Request Handler** process.

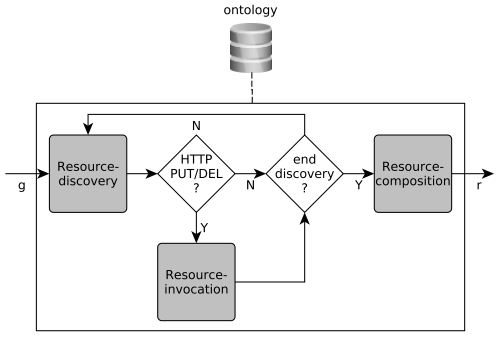


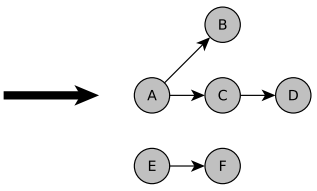
Fig. 4. Semantic Request Handler

After receiving a goal **g** (left side of above figure), the Semantic Request Handler process starts traversing linked resources of a service provider. On each match of a resource to a goal, the next action is executed depending on the type of the request. In case of a HTTP GET or PUT request the resource invocation process is executed. In all cases the resources' representation is stored (in a set or list) during each iteration. After the resource discovery process has found all possible matches, all representations are composed (in case if there is only one representation, the result of the composition is the representation itself) and returned to the requester as one representation **r** (right side of above figure).

B. Response Handler

After the agent retrieved a representation of the resource(s) which match his goals, he has to interpret the retrieved data. This is necessary since the Semantic Request Handler only takes care of finding all the desired information, but the retrieved representation can still have additional data which is not useful for the agent and the useful data has to be matched to the agents' goals. thus, the agent has to filter and match data in the representation. This can also be done using an ontology to constraint the discovery of data in a representation since the data with its hierarchy also can be seen as a graph:

```
{
  "A":{
    "B":{},
    "C":{
      "D":{}
    }
  },
  "E":{
    "F":{}
  }
}
```



This approach for **data discovery** is basically the same as the one proposed for resource discovery. Another beneficial task the Semantic Response Handler can fulfill is the mapping of generic functions to ontological concepts, and by this allows a dynamic code reuse. Please refer to [3] for more details.

C. Model of a Semantic Agent

Two thinkable ways of how semantic agents can use their building blocks are **Request Handler** and **Response Handler** to establish a semantic communication. These communication approaches differ in whether the Request Handler process traverses his own local LRG or a remote LRG.

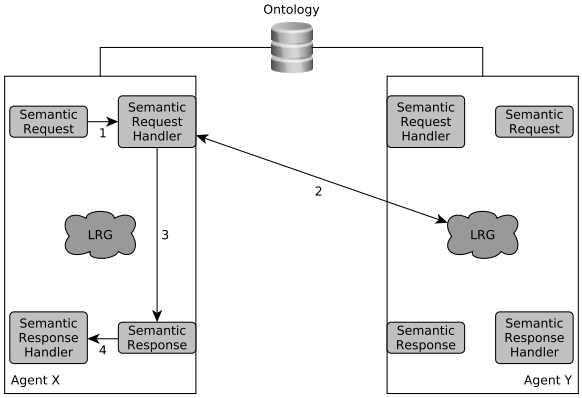


Fig. 5. Communication between Semantic Agents

The agents in figure 5 use their request handler to traverse a remote LRG. The resulting flow of the communication is as follows:

1. Agent X passes a semantic request to his own Semantic Request Handler.
2. The Semantic Request Handler of Agent X receives the request and starts the traversal of the Linked Resources Graph (LRG) of Agent Y.
3. After Agent X has processed the request and traversed the LRG of Agent Y, he generates a semantically enhanced representation of a resource(s) and passes it as a semantic response to his Semantic Response Handler.
4. The Semantic Response Handler of Agent X receives the representation and interprets the included data.

VI. Use Case

A. Setting

In the context of this research, a Smart Camp ontology was developed with the Web Ontology Language (OWL) [14]. This ontology acts as the base for the proposed approaches. A small extract of the complete ontology represented as a graph is depicted in figure 6. This ontology model consists of several classes (for example "Sensor"), with object properties (for example "knows") to declare relations between these classes and some data type properties (for example "sensorValue"). Please refer to [3] for the complete ontology.

This ontology can be used to add semantics to resources and provide semantic links between them. One possible

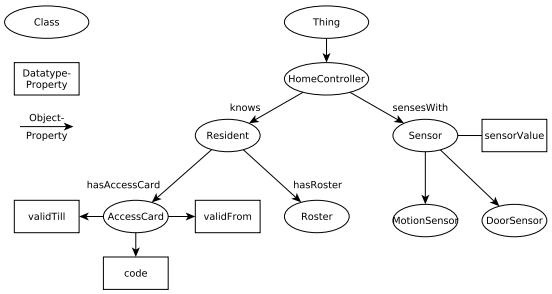


Fig. 6. Part of the Smart Camp ontology

implementation of resources offered by a SHC is depicted in figure 7. This figure showing an LRG includes five resources with the URIs: */, /accesscards, /sensors, /sensors/door* and */sensors/motion*. The root resource (URI: "/") includes data which is semantically marked as *Roster* and also has semantic links (marked as *Sensor* and *AccessCard*) to other resources. The *Sensor* resource also provides semantic links to other resources which have data marked with the semantic term *sensorValue*. The resource with the URI "/accesscards" includes data which is marked with the OWL data type properties for *AccessCard*.

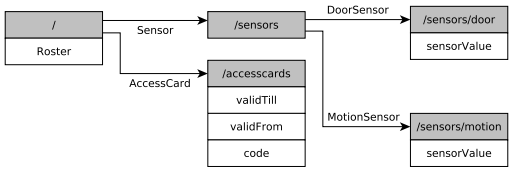


Fig. 7. Possible resources generated with Smart Camp ontology

B. Semantic Agents in Action

We can apply resource composition to our Smart Camp example. One thinkable example is the definition of *HomeController* as the goal of a request. This results in collecting all the information provided by a SHC. Another example could be the goal *sensorValue*. The POPs resulting from the latter case are shown in figure 8. The applied resource discovery process would then constraint the traversal of the LRG to the resources with the URIs */sensors/door* and */sensors/motion* and return their representations since they map the concept *sensorValue*.

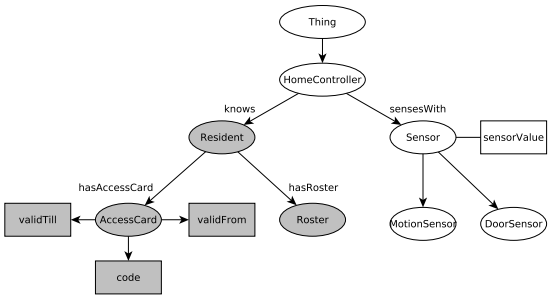


Fig. 8. Usage of ontology paths in example ontology

As a next step, these representations of the resources with the URIs */sensors/door* and */sensors/motion* have to be combined. Below listing shows possible representations of these resources:

```
{
  "@context":{
    "door":"http://www.smartcamp.org/ontology#DoorSensor"
    "valY":"http://www.smartcamp.org/ontology#sensorValue"
  },
  "@type":"door",
  "valY":true
}
```

```
{
  "@context":{
    "motion":"http://www.smartcamp.org/ontology#MotionSensor"
    "valZ":"http://www.smartcamp.org/ontology#sensorValue"
  },
  "@type":"motion",
  "valZ":false
}
```

The result of the actual composition of above representations is given in the next listing. The "@type" value of the new representation is "Sensor" since it is the next common concept in the ontology starting from "MotionSensor" and "DoorSensor" and climbing up the ontology graph (See figure 8). This example shows also the case where two different keys in the contexts point to the same ontological concept ("valY" and "valZ"). These are composed to one key-value pair in the resulting representation.

```
{
  "@context":{
    "motion":"http://www.smartcamp.org/ontology#MotionSensor",
    "door":"http://www.smartcamp.org/ontology#DoorSensor",
    "valY":"http://www.smartcamp.org/ontology#sensorValue"
  },
  "@type":"http://www.smartcamp.org/ontology#Sensor",
  "motion":{
    "value":false
  },
  "door":{
    "valY":true
  }
}
```

C. Resulting Benefits

Relating to the scenario shown in the previous sections, there are several possible changes which can happen. Requirements can change, such that new sensors have to be included or old ones removed with the result that new resources have to be introduced or removed. In the latter case, it would be wise to reorganise the resources since all the sensor information could be stored in a single resource, e.g. the resource */sensors*.

These types of changes would involve changes in depending clients in the initial Smart Camp system. However, there is no need to change the clients in our improved system design using Semantic Web technologies. All the necessary sensor data can be discovered and composed in the LRG of the service provider. This achieves as well, that the functionality of clients can be enriched without recompilation. If new sensors are introduced, clients can easily extract the new data without making any changes to the

clients' implementation in the preface. This is especially beneficial when thinking of a logging unit for sensor information. Consequently, the agile service infrastructure in the context of service bundling in a complex environment is robust, since the composition is based on ad hoc data extraction of any digital or physical asset involved.

VII. Conclusion & Future Work

This work introduced the concept of a semantic agent which is using ontology paths to automate common tasks related to Web services. The adoption of the service infrastructure concept for the applied research project Smart Camp has been shown, and key resulting benefits in terms of Web service evolvability explained.

To assess the actual benefits of Semantic Web technologies and our model devised for the evolvability of Smart Campsservice infrastructure, multiple semantic agents were implemented and executed in an evolution scenario with multiple evolution levels. These evolution levels, which would cause an earlier infrastructure to break, were defined and tested with evolving SHCs. All semantic agents worked as expected and did not break. For the complete details of the research presented here, please refer to [3]. Future research needs suggest the combination of evolvability with scalability and high availability in much bigger service infrastructures. These are denoted here as Smart Cities, with many more types of services offered. In these future complex environments, the digital and physical assets can be included in service bundling in a seamless way, since logical and physical evolvability of the assets is supported.

VIII. Acknowledgments

Particular thanks shall be expressed to Markus Lanthaler for his time and advise whenever needed, and to Professor Elizabeth Chang for providing with DEBII all the support of an inspiring, world class environment.

References

[1] M. Lehman, *On understanding laws, evolution, and conservation in the large-program life cycle*, Journal of Systems and Software, 1:213-221, 1980.

[2] Lukas J. Oslislo, Alex Talevski and Achim P. Karduck, *SMART CAMP: Benefits of Media and Smart Service Convergence*, AINA Workshops, 2011.

[3] Andrej Eisfeld, *Smart Camp: Evolvability of Service Infrastructures with Semantic Web Technologies*, Furtwangen University / Curtin University, Germany, 2012.

[4] Jacek Kopecky, Tomas Vitvar, Carine Bournez and Joel Farrell, *SAWSDL: Semantic Annotations for WSDL and XML Schema*, IEEE Internet Computing, 11:60-67, 2007.

[5] Jon Lathem, Karthik Gomadam and Amit P. Sheth, *SA-REST and (S)mashups : Adding Semantics to RESTful Services.*, ICSC'07, 2007.

[6] Jacek Kopecky, Karthik Gomadam and Tomas Vitvar, *hRESTS: An HTML microformat for describing RESTful web services*, 2008 IEEEWICACM International Conference on Web Intelligence and Intelligent Agent Technology, 1:619-625, 2008.

[7] Jacek Kopeck, Tomas Vitvar and Dieter Fensel, *MicroWSMO: Semantic Description of RESTful Services*, WSMO Working Group, 2008.

[8] David Booth and Canyang Kevin Liu, *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626.

[9] Markus Lanthaler and Christian Guetl, *A Semantic Description Language for RESTful Data Services to Combat Semaphobia*, Proceedings of the 2011 5th IEEE International Conference on Digital Ecosystems and Technologies DEST, 5:47-53, 2011.

[10] Rosa Alarcon and Erik Wilde, *Linking Data from RESTful Services*, Workshop on Linked Data on the Web, 2010.

[11] Areeb Alowisheq, David Millard and Thanassis Tiropanis, *EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies*, 8th International Semantic Web Conference, 2009.

[12] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler and Mark Birbeck, *JSON-LD Syntax 1.0*, http://json-ld.org/spec/latest/json-ld-syntax, 2011.

[13] Douglas Crockford, *RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*, http://tools.ietf.org/html/rfc4627, 2008.

[14] Boris Motik, Bijan Parsia and Peter F. Patel-Schneider, *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/.