An Approach for Generating Semantic Annotations in Telecommunication Services

Simeona Harahap Cruz Pellkvist
University of Applied Sciences Technikum Wien
Höchstädtplatz 5, A-1200 Vienna, Austria
simeona.pellkvist@technikum-wien.at

Peter Reichl, Anna Fensel, Joachim Zeiss
FTW Telecommunications Research Center Vienna
Donau-City-Str. 1, A-1220 Vienna, Austria
{reichl | fensel | zeiss}@ftw.at

***Abstract*— This paper describes the design and implementation of the "Semantic Generator" engine, which is used to transform and generate data from semantic formats, i.e., RDF, OWL or N3, to semantic and non-semantic formats, i.e., RDF/XML or text. The proposed lightweight approach maximizes the reuse of existing, widespread technologies while also allowing easy integration of new technologies. The generator engine's capacity is demonstrated and evaluated for two use cases with different requirements. On the one hand, it performs annotation generation of mobile service descriptions. On the other hand, the engine is used for mapping SIP messages from and to ontologies in real-time scenarios. In both cases, information available in a semantic format is mapped to the resulting semantic or non-semantic annotations and vice versa.**

Keywords - Semantic annotation, mobile services, microservice description, Session Initiation Protocol, IP Multimedia Subsystem.

I. INTRODUCTION

In the last years, semantic technologies have gained increasing interest in various fields of IT (Information technology), like Semantic Web and Business Process Management. Recently, there are also ongoing research efforts and projects on how these technologies can be leveraged in the field of telecommunications [1]. In particular, ontologies in the knowledge layers of telecommunication architectures play an increasing role for service platforms and mobile communications. As the integration of Telco, Internet and the Web takes place, in order to achieve interoperability, telecommunication systems and services tend to rely on knowledge represented with the use of shared schemata, i.e., on ontologies similar to those envisioned in the Semantic Web [2]. For example, when users move between different service domains, service delivery platforms might be necessary to dynamically change the service resources in order to provide the best user experience based on the context information or user preferences, e.g., by switching from one network operator to another, or between similar service components provided by the different service providers. Given efficient interoperability and semantic policies, it could be possible to substitute one service with another, if they can be proven to be sufficiently similar [3]. Therefore, as it is the case with services in general, semantic annotations will facilitate accurate service description, discovery and composition of telecommunications network services.

Among the many research projects in the engineering field using semantic technologies, we will focus on two particularly interesting ones, which are carried out at the Telecommunications Research Centre Vienna (FTW), Austria, i.e., m:Ciudad[1] and BACCARDI[2]. m:Ciudad addresses the question of how to create and share micro-services between the users of the mobile devices, where services should be created and shared to other users on the spot. In this project, service descriptions are generated out of information, which is available in semantic formats like RDF (Resource Description Framework) [4] and OWL (Web Ontology Language) [5] on the Web. For testing purposes, the service descriptions are generated automatically according to defined rules and schemata; and the users could share their own set up services to groups of friends or an organization. On the other hand, BACCARDI is an application-oriented research project with a broad focus on next generation fixed and wireless networks based on the Session Initiation Protocol (SIP) and using IMS (IP Multimedia Subsystem) as a testbed platform. One of the tasks in this context is to lift the header information of SIP `INVITE` and `BYE` messages to semantically described resources (i.e., RDF/N3 (Notation 3)). Furthermore, the transformation of semantically described resources to SIP route headers should be enabled to make round trips possible. Eventually, this allows for the execution of high-level policies on a technical level [6].

This paper is structured as follows: Section II presents the problem statement and a brief survey on relevant related work. Section III introduces our general approach and the Semantic Generator engine as our proposal to solve the problem. Section IV discusses the application of our prototype for the mentioned two case studies and presents results from our evaluation. Section V summarizes and concludes the paper with a brief outlook on the future work.

II. PROBLEM STATEMENT AND RELATED WORK

The need for a transformation process that translates semantic data to semantic or non-semantic formats (and vice versa) with some rules set up in between is common to both projects. On a higher level, such processes are generally needed

[1] m:Ciudad is a FP7 STREP that focuses on enabling end-user-generated mobile services. The project is running from 2007 to 2010, the consortium is comprised from 8 partners from several EU countries, coordinated by Robotiker-Tecnalia, Spain.
URI: http://www.mciudad-fp7.org.

[2] BACCARDI (Beyond Architectural Convergence: Charging, Security, Applications, Realization and Demonstration of IMS) is an Austrian COMET project which has been carried out at the Telecommunications Research Center Vienna from 2008 to 2010 in close collaboration with Telekom Austria, mobilkom austria, Alcatel-Lucent Austria, Kapsch CarrierCom and TU Vienna.
URI: http://www.ftw.at/ftw/research/projects/ProjekteFolder/COM-4.

for automatic generation of semantic annotations and service profiles, as well as for the purpose of testing and benchmarking large-scale search mechanisms. As large amounts of semantic annotations are time consuming and expensive to produce by hand, the suggested service annotation generation solution is of paramount importance, especially considering the tremendous growth of the semantically annotated resources. Thus, the main contribution of this work is an *approach and a prototype providing a configurable and flexible way to generate and transform telecommunication services annotations*.

Related work provides several frameworks and applications, developed and applied to transforming and generating services using semantic technologies. In [7], an approach is chosen to generate Web services automatically from a service graph model. An abstract model of services is created, and for this model a code generation is run to generate implementation files. However, this approach does not use semantic resources as an input. The approach chosen in [8] makes use of software agents to parse HTML (Hyper Text Markup Language) files and generates XML (eXtensible Markup Language) from them, enriching them with specific XML tags. Unfortunately it works on HTML only; therefore this approach is not applicable to our problem as well. The authors of [9] show how semantic data can be transformed to non-semantic data exploiting the capabilities of Model Driven Architecture in the domain of Business Process Engineering.

While these approaches work well in their specific domains, our goal is a more general solution, which can be used for a broad scope of applications and facilitates building user applications on proven traditional tools, while enabling an easy integration of arbitrary semantic schemata. In our solution, practically every semantic resource available on the Web could serve as an input. This allows crawling and extracting relevant information from large volumes of interlinked ontologies and semantic annotation resources, particularly, the whole Linked Open Data Cloud [10].

III. THE SEMANTIC GENERATOR ENGINE

A. Approach

While currently there are no ready to use applications available that can solve the entire transformation problem introduced previously, there are at least some frameworks available, which can be leveraged to fulfill parts of the requirements. Therefore, we combine these frameworks in order to use the advantages of each tool and integrate them to an engine that produces the required results.

More specifically, the transformations addressed in our work involve:

- Querying information from semantic resources,
- Querying information from SIP messages,
- Combining the data that has been extracted, and
- Formatting and splitting the data according to the requirements.

The engine is required to be able to read the semantic resources – in our case, RDF/XML, N3, and OWL – and also to write, at least in RDF/XML format (for the m:Ciudad case) and in text/N3 format (for the BACCARDI case). Moreover, the tool must able to reformat and split combined data according to given rules. In addition, querying should be performed similar to the well-known and standard SQL syntax in order for achieving an easy and familiar usage. The extracted data should be able to be combined with or without any repetition.

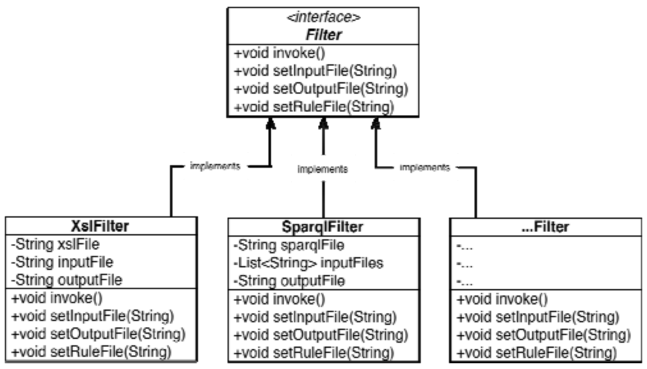
For an efficient employment of the available tools they have to be combined using design patterns [11][12], which are well known from software design. In general, a design pattern is a template, which could be applied for several situations in a certain condition in order to achieve a general solution for a number of common problems, which happen repeatedly. In order to achieve this we use "Pipes and Filters Pattern" and "Adapter Pattern".

In software engineering, Pipes and Filters usually mean that the output or result of one application is used as input for another one. The Pipes and Filters pattern in our case represents an architectural pattern for the overall application. Different forms of the inputs are passed to the first filter and are processed there. This result is transferred to the second filter for which it represents the input. This process repeats to the next filter and so on. Therefore, this method is suitable to transform the semantic data, in our case into an RDF/XML..

The Adapter pattern is also known as wrapper pattern (or wrapper) and describes a technique used to make classes, which have different interfaces that are compatible to each other. An adapter may also be used to convert data to a suitable format. In our case, the adapter is used to wrap external resources; in our case, XSLT (eXtensible Stylesheet Language Transformation) and SPARQL (SPARQL Protocol and RDF Query Language).

B. Main Engine

The main generator engine is the core integration layer built upon the Pipes and Filters and the Adapter pattern. It represents an abstraction mechanism for the other frameworks, provided they have serializable input and output, and rules for controlling this transformation. This is achieved by implementing the common Filter interface. Fig. 1 shows the important core classes of the generator, specifically the filter interface that is implemented for example by the SPARQL filter and the XSLT filter.



<interface>
Filter
+void invoke()
+void setInputFile(String)
+void setOutputFile(String)
+void setRuleFile(String)
implements
implements
implements
XslFilter
-String xslFile
-String inputFile
-String outputFile
+void invoke()
+void setInputFile(String)
+void setOutputFile(String)
+void setRuleFile(String)
SparqlFilter
-String sparqlFile
-List<String> inputFiles
-String outputFile
+void invoke()
+void setInputFile(String)
+void setOutputFile(String)
+void setRuleFile(String)
...Filter
-...
-...
-...
+void invoke()
+void setInputFile(String)
+void setOutputFile(String)
+void setRuleFile(String)

Figure 1. Class diagram (excerpt)

1) *Semantic Query Filter*

Two semantic query filters have been implemented: the SPARQL filter based on ARQ [13] using the Jena [14] toolkit, and a SeRQL (Sesame RDF Query Language) based filter using the Sesame [15] toolkit. The SPARQL based filter is our main filter for querying semantic resources in this engine, whereas the SeRQL based filter is provided for comparison to the SPARQL based filter, playing an important role for the engine has and involving three main steps:

- First, the filter takes the semantic files, such as RDF file, N3 file, OWL file, as the input.
- Using specific sets of rules, the input files are then stored where the query is stored, either in SPARQL or in SeRQL.
- Finally, the output is an XML file from the results of the SPARQL and SeRQL query language. Both of them use the XML result format from SPARQL such that they can be easily replaced by each other.

2) *SIP Query Filter*

In this filter, the engine is assigned to separate the SIP messages into different fields for reasoning purpose. The input of the SIP query filter consists of SIP messages, which are in text format. The algorithm translates the SIP messages into N3 serialization to be passed to a reasoner, which is supposed to infer information over semantically annotated data, based on logical rules. In our engine, this serialization can theoretically be set to another format file, depending on which rule is set. The input is read from the `InputStream` and saves the result as a string. JAIN API [16] is used to parse the string into a SIP message, before the default Jena model is initialized. The filter is set to find out if the SIP message is a SIP request or a SIP response. When the message is already defined, its content then is separated to different fields as Subject - Predicate - Object "sentences" in the Jena model under the condition that the related field is not null. The message is handled differently according to the SIP request or SIP response. The result is written to an `Output-Stream`.

3) *Combining Filter*

The purpose of this filter is to combine the queried output results from the semantic filter. The input for this filter consists of the output results that are written in an XML file format from a semantic query filter. This filter has different sets of rules for different purposes of output result; the rules can be defined as requested. The filter combines the rules and orders the elements, generates a maximum of results and optionally considers uniqueness. The output of this filter is in XML file format. The filter reads the config file, where it can be specified if every input shall only be used once. Furthermore, this filter will take care of randomizing the results. The output will be written as a result in XML.

4) *Formatting Filter*

The output result of the previous filter is not formatted yet with respect to the user's design, hence this procedure is done in this filter. Two formatting filters have been implemented: one based on XSLT, another one based on XQuery (XML Query Language).

IV. Case Studies

Having outlined our general approach in Section III, we will now discuss two case studies where the suggested transformation approach has been applied, i.e., the research projects m:Ciudad project and BACCARDI.

A. *m:Ciudad*

As already mentioned earlier, the m:Ciudad case study is focusing on semantic descriptions of mobile micro-services. For this purpose, a tool is required, which is able to transform available semantic data written in RDF or OWL file form to XML file form.

During the transformation, the data of semantic descriptions in the form of RDF or OWL should be extracted, combined/integrated, and reformatted to different RDF formats according to the requirements. As input and output formats may change, the transformation should be able to be adapted in a flexible way.

In order to create mobile service descriptions, RDF and OWL files constitute the input for our engine. These inputs are queried and combined using the SPARQL filter over the Linked Open Data Clouds [17], and the result is persisted to an XML file. The output file of the first filter then becomes the input file for the XSLT filter, which has again XML files as output. This process could be developed to filters that are piped, to support additional input or transformations, i.e., the parsing/writing of SIP messages or replacing XSL transformations by another XML transformation language.

The implementation of this engine, which queries RDF- and OWL-based knowledge bases, generates any required number of various service annotation datasets required for our testing and benchmarking purposes. These datasets are compliant with the m:Ciudad's schemata of the Service Profile, i.e., the basic annotation of a service, and the Service Capability, i.e., annotation of basic service behavior and requirements. Each element has been formulated in different fields according to what has been set on the rule. The number of created datasets is large enough to be used, particularly, for performance evaluation of the micro-service employment algorithms.

Figure 2. and the subsequent listings illustrate the m:Ciudad solution in more detail. **Listing 1** represents one of the input files, i.e., the bloggers.rdf, which is an RDF store conforming to the FOAF (Friend of a Friend) ontology. **Listing 2** shows the configuration file that drives the transformation process. **Listing 3** shows the configuration rule for the first filter in the pipe, i.e., the SPARQL filter. An SQL-like query asks the name from the FOAF ontology. In this query, abbreviations for namespaces are defined using the keyword `'PREFIX'`. Variables begin with a `'?'`. The query listed here basically translates to: Find an entity `x` that has a `foaf:name`, bind this name to the variable `Capabili-tyName` and return it. **Listing 4** shows the intermediate result for the capability name. **Listing 5** shows the configuration file of the combining filter as a result of the SPARQL query above. In order to reformat the information and structure it according to a service profile, an XSL stylesheet is used, which is shown in **Listing 6**. As the intermediate result contains all the result rows in one file, it is necessary to split them in different files; this is achieved using a Xalan specific instruction (xalan:write) for every `sparql:result` element. For every sparql:binding, one element is created in the example. E.g., if the binding is for `CapabilityName`, a `udlcp:Capability` element is created. An attribute is created for it, which has an attribute `datatype`, whose value is set to string (using the corresponding XML schema datatype). The value of the `sparql:literal` or `sparql:uri` element in the input is put as content of the created element. Finally, **Listing 7** shows the resulting service capability. The capability name derives from the foaf:name in the foaf ontology.

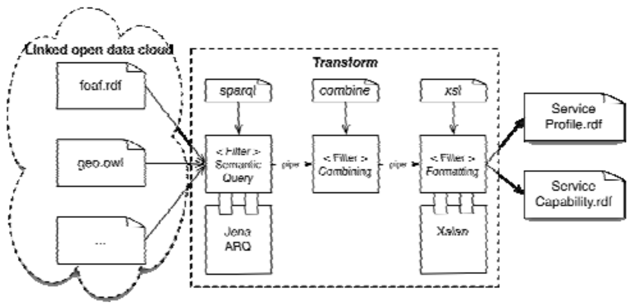


Figure 2. mCiudad solution

```
<foaf:Agent rdf:nodeID="ni93487857">
<foaf:name> Pasta N Pizzor </foaf:name>
<foaf:weblog>
 <foaf:Document
rdf:about="http://www.example.com/pastapizzor">..
```

Listing 1. Input file: Friend-of-a-Friend (FOAF) RDF

```
<generatorConfiguration>
<workflow name="CapabilityWF">
<filter class="id.shc.genie.SparqlFilter">
<input>http://danbri.org/foaf.rdf</input>
<output>../work/CapabilityName.xml</output>
<rule>../etc/CapabilityName.sparql</rule>
</filter>...
```

Listing 2. Transform: Workflow Configuration file

```
PREFIX foaf : <http://xmlns.com/foaf/0.1/>
SELECT ? CapabilityName
WHERE { ?x foaf :name ?CapabilityName . }
```

Listing 3. SPARQL filter configuration

```
<sparql>
<head> <variable name="CapabilityName"/> </head>
<results><result>
<binding name="CapabilityName">
<literal xml:lang="en"> Pasta N Pizzor </literal>
</binding></result>...
```

Listing 4. SPARQL Profile result

```
COMBINE 100 :
CapabilityID unique
CapabilityName
CapabilityDescription
...
```

Listing 5. Combining filter configuration

```
<stylesheet version="1.0"
  extension-element-prefixes="xalan"
...
<output method="xml" encoding="ISO-8859-1" in-
dent="yes" />
...
<template match="sparql:result">
<xalan:write se-
lect="concat('capability-',position(),'.rdf')">
<udlcp:Capability> <apply-templates />
</udlcp:Capability>
</xalan:write>
</template>

<template match="sparql:binding">
<if test="@name = 'CapabilityName '">
<udlcp:CapabilityName>
<attribute name="datatype"namespace="&rdf;#">
&xsd;#string
</attribute>
<value-of select="sparql:literal|sparql:uri"/>
</udlcp:CapabilityName>
</if>
...
```

Listing 6. Formatting filter configuration: XSL Stylesheet

```
<udlcp:Capability
xmlns:udlcp="http://www.mciudad-
fp7.org/schemas/udlcp#">
  <udlcp:CapabilityID>59</udlcp:CapabilityID>
  <udlcp:CapabilityName>Pasta N Piz-
zor</udlcp:CapabilityName>
  ...
```

Listing 7. Output file: Service Capability RDF/XML

Summarizing, the engine supports two different languages for semantic queries, based on two different frameworks and two different XML transformation languages, in order to investigate how the flexibility of the engine is able to cope with different applications.

B. BACCARDI

In the BACCARDI use case, semantic web based policy definitions are enabled, which compose and control application services hosted in an IMS core network. In the BACCARDI Service Oriented Data-driven Architecture (BACCARDI SODA) working group, a N3-based semantic reasoner, the so-called "policy engine" [17][18], is used to make semantic policy-based decisions on how to combine or modify behavior of application services, which communicate via SIP or ISC (IMS Service Control), respectively (ISC is an extension of SIP for call and service control purposes in IMS). Instead of talking to the application services directly, the SODA architecture proposes to intercept SIP `INVITE` and `BYE` messages to cancel, redirect or manipulate message information based on policy decisions of the reasoner, and thus to compose and control service behavior in real time.

In the BACCARDI case, SIP messages have to be transformed to N3 statements, which are subsequently sent to the reasoner over HTTP. The answers received from the reasoner in terms of N3 have to be parsed, and accordingly SIP headers are manipulated.

For this purpose, SIP messages have to be intercepted by a SIP proxy servlet. These messages have to be translated to N3, for which a special filter is needed. As this is the only transformation step, it shall be able to directly embed the filter into the corresponding servlet, without having to use the engine for setting up the transformation. Figure 3. depicts the basic solution for the BACCARDI case, while Figure 4. shows how servlet container, SIP proxy servlet, SIP2N3 filter and the reasoner service work together.

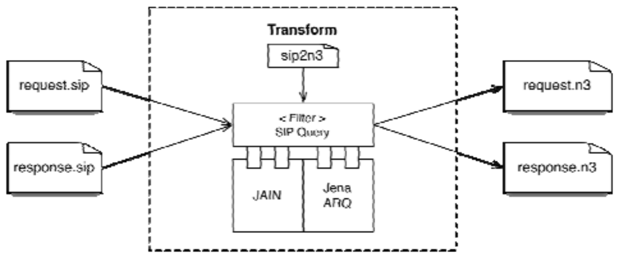


Figure 3. BACCARDI solution

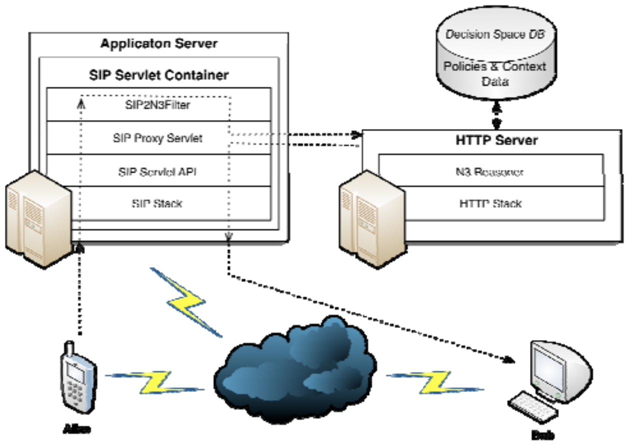


Figure 4. Setup for Baccardi

For the BACCARDI use case, the result has been applied to different scenario episodes. The result is shown in log files, demonstrating the reaction of the engine to the scenarios that have been set up. As a specific example, we now consider the "Forward" scenario, which is applied to check when the SIP user agent makes a call and this call is then forwarded to another participant. Suppose the caller is defined as Alice and the callee is defined as Bob, while Charlie is define as the one whom the call shall be forwarded to. **Listing 8** shows how the original log file of the SIP `INVITE` message looks like. This SIP request is translated to N3 when passing the engine, and forwarded to the reasoner. As we can see in **Listing 9**, the SIP `INVITE` message has been translated to N3 for reasoning purposes. The reasoner answers with "FORWARD" using Charlie's username/number. **Listing 10** shows the response of the reasoner mock, and **Listing 11** shows the modified forward SIP `INVITE` message, where Bob's phone doesn't ring, instead Charlie gets Alice's calls. Charlie answers the call and the answer 200 `OK` message from the forward number, see **Listing 12**.

During our practical experiments, we observed that the engine gives proper response and translates the original messages to N3 for reasoning purposes. In the BACCARDI SODA case, the implementation of this engine has successfully transformed the header information from SIP `INVITE` and `BYE` messages to semantic annotated data instances according to the BACCARDI SODA ontology. The header information from SIP `INVITE` and `BYE` messages are transformed to N3; also SIP route headers can be manipulated, based on the results of the N3 based reasoner according to the BACCARDI SODA ontology, in order to send, cancel or forward SIP messages to other application servers.

```
INVITE sip:Bob@128.131.202.184:22244 SIP/2.0
Max-Forwards: 70
Content-Length: 255
To: "Bob"<sip:Bob@128.131.202.184:22244>
Contact: <sip:Alice@128.131.202.184:62901>
Cseq: 1 INVITE
Content-Type: application/sdp
From: "Alice"<sip:Alice@test.com>;tag=684e0942...
```

Listing 8. Scenario Forward: Original SIP invite message

```
<request> sip:contact
"<<sip:Alice@128.131.202.184:62901>>" .
<request> sip:cSeq 1 .
<request> sip:protocol "SIP/2.0" .
<request> sip:content_type "application/sdp" .
<request> sip:request_url
"<sip:Bob@128.131.202.184:22244>" .
<request> sip:max_forwards 70 .
<request> sip:to
"sip:Bob@128.131.202.184:22244" .
<request> sip:content_length 255 .
<request> sip:from "sip:Alice@test.com" .
<request> a        sip:INVITE .
...
{<response> soda:action ?a} => [] .
{<response> soda:add_header ?b} => [] .
{<response> soda:delete_header ?c} => [] .
{<response> soda:append_value ?d} => [] .
```

Listing 9. Scenario Forward: SIP invite message translated to N3

```
<response> soda : action soda : forward . <re-
sponse> soda:forward_address
"sip:Charlie@128.131.202.184:51267".
```

Listing 10. Scenario Forward: Reasoner answer

```
INVITE sip:Charlie@128.131.202.184:51267 SIP/2.0
Max-Forwards: 70
Content-Length: 255
To: "Bob"<sip:Bob@128.131.202.184:22244>
Contact: <sip:Alice@128.131.202.184:62901>
Cseq: 1 INVITE
Content-Type: application/sdp
From: "Alice"<sip:Alice@test.com>;tag=684e0942...
```

Listing 11. Scenario Forward: Modified SIP invite message

```
SIP/2.0 200 OK
Record-Route:
<sip:128.131.202.184:5060;lr;fid=server_1>
Content-Length: 253
```

```
To:
"Bob"<sip:Bob@128.131.202.184:22244>;tag=126fb82d
Contact: <sip:Charlie@128.131.202.184:51267>
Cseq: 1 INVITE
Content-Type: application/sdp
From: "Alice"<sip:Alice@test.com>;tag=684e0942...
```

Listing 12. Scenario Forward: 200 OK message

An emulation of the reasoner has been developed to test if the SIP proxy servlet and filter reacts properly to the SIP messages. It is implemented as an HTTP servlet, and allows the configuration for different scenarios, i.e., the unchanged forwarding or redirection of a SIP message, the cancellation of the related SIP dialog or the manipulation of header files in the SIP message based on the response of the reasoner. Both, the SIP proxy servlet and the BACCARDI SODA "policy engine" represent a semantic IMS SCIM (Service Capability Interaction Manager), which controls services across application servers in the IMS network.

V. Conclusions and Future Work

In this paper we have demonstrated how to successfully apply semantic technologies for scalable and flexible data transformation and generation within a single prototypical engine, i.e., the Semantic Generator. The chosen approach and the corresponding solution has been validated and tested in two different case studies from two ongoing telecommunications industrial projects.

For the m:Ciudad case, we have successfully queried semantic resources from the Web, aggregated and combined the results, and transformed them, thus generating mobile service descriptions according to different configurable set of rules. The engine was able to successfully generate 10,000+ datasets of service profiles and capabilities. In the BACCARDI case, we have transformed the different SIP messages to N3 for reasoning purposes. The developed filter has been embedded in a SIP proxy servlet, and this approach has been evaluated for different test scenarios. The results have been documented by screen capturing and log files, which show that the chosen approach and the developed solution fulfill the requirements concerning functionality.

It could be argued if the development of a common architecture has really been necessary, or if two different architectures for solving the two problems should be preferred. While it is not necessary to use the same architecture, this approach has certain advantages, for example it extends maintainability and enables combination of filters. For instance, the SIP2N3 filter cannot only be used in a real-time scenario like in the BACCARDI project, but also to enable lifting of information from log files for enabling reasoning or for providing input to service generation. Further, it allows a step-by-step and bottom-up style of development, thus reducing the entry barrier to semantic technologies for users who are currently using traditional technologies. The generator engine enables them to continue to use their proven tools while facilitating and promoting the use of semantic technologies.

Summarizing, the proposed approach has been successfully evaluated for different applications. The developed engine is flexible, and its behavior can be changed easily by adapting configuration files. Furthermore, the extensible architecture of the engine also allows the user to create their own filters according to their needs with reasonable effort, which underlines once more the efficiency of our solution.

Acknowledgment

Part of this research has been funded by the Austrian Government and the City of Vienna within the competence center program COMET.

References

[1] A. V. Zhdanova, N. Li, and K. Moessner: Semantic Web in Ubiquitous Mobile Communications. The Semantic Web for Knowledge and Data Management (Ed.: Ma, Z.), IGI Global, August 2008.

[2] S. Tarkoma, C. Prehofer, A.V. Zhdanova, K. Moessner, and E. Kovacs: SPICE: Evolving IMS to next generation service platforms. In: Proceedings of the 3rd Workshop on Next Generation Service Platforms for Future Mobile Systems (SPMS 2007) at the 2007 International Symposium on Applications and the Internet, IEEE Computer Society Press, 2007.

[3] T. van Do and I. Jorstad: A service-oriented architecture framework for mobile services. In: Proceedings of Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop AICT/SAPIR/ ELETE, 17-20 July 2005, pp. 65 – 70.

[4] http://www.w3c.org/RDF, 2009-08-24, 2010-02-09.

[5] G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. Springer-Verlag, 2003.

[6] A. V. Zhdanova, J. Zeiss, A. Dantcheva, R. Gabner, and S. Bessler: A Semantic Policy Management Environment for End-Users and its Empirical Study. Volume 221/2009 of Studies in Computational Intelligence. Springer Berlin / Heidelberg, 2009.

[7] E. Cho, S. Chung, and D. Zimmerman. Automatic Web Services Generation. In HICSS, pages 1–8. IEEE Computer Society, 2009.

[8] D. Camacho and M. D. R-Moreno: Web Data Extraction using Semantic Generators. In: International e-Conference of Computer Science (IeCCS 2006), LNCS, pp. 34–38, 2007.

[9] C. Blamauer and D. M. R. Lintner: Integrating Semantic Business Process Management and Viewbased Modelling. Master's thesis, Vienna University of Technology, 2009.

[10] C. Bizer, T. Heath, and T. Berners-Lee: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems, Special Issue on Linked Data, 2009.

[11] E. Gamma, R. Helm, J. Vlissides, and I. R. Johnson: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[12] F. Buschmann, R. Meunier, H. Rohnert, and P. Sommerlad: Pattern-oriented Software Architecture - A System of Patterns. J. Wiley and Sons Ltd., 1996.

[13] http://jena.sourceforge.net/ARQ/documentation.html, 2010-02-04.

[14] http://jena.sourceforge.net/, 2010-02-04.

[15] http://www.openrdf.org/doc/sesame2/2.3.1/users/index.html, 2010-02-09.

[16] https://jain-sip.dev.java.net/, 2010-01-08.

[17] J. Zeiss and O. Jorns: Using Semantic Reasoning and Privacy Policies in Ubiquitous Envi- ronments. UBICOM2007 Workshop: First International Workshop on Security for Spontaneous Interaction, IWSSI 2007, Innsbruck, Austria, 16th September, 2007.

[18] S. Bessler and J. Zeiss: Using Semantic Policies to reason over User Availability. Second International Workshop on Personalized Networks, Pernets'07, Philadelphia 2007.