Semantic Integration in Telecommunications

Vilho Räisänen

Chief Technology Office/Research,
Nokia Siemens Networks, Finland.
vilho.raisanen@nsn.com

Abstract: In this article, we discuss semantic integration challenges encountered in multi-vendor systems integration as well as ways of addressing them. These challenges are relevant to not only to traditional bespoke Operations Support Systems and Business Support Systems (OSS/BSS) system integration approach in telecommunications, but also to the use of Service-oriented Architecture (SoA) architectural style, for example. In addition to data integration, we discuss the relevance of protocols to semantic integration, both for "incumbent" SoA technologies such as SOAP as well as "challenger" ones. Finally, we describe ongoing implementation work in the area of demonstrator platform leveraging knowledge management, which supports semantic integration and data federation.

1 Introduction

Multi-vendor system integration is bread and butter in telecommunications. Particularly in OSS/BSS, management system typically needs to talk to network elements of various makes and models in order to perform relevant configuration and monitoring tasks. System integration is increasingly streamlined through the use of Commercial, Off-The Shelf (COTS) Information Technology (IT) components also in telecommunications domain. In particular, enterprise management related tools and methodologies are being taken into use. Following of this approach optimally necessitates taking into account the standardised nature of telecommunications domain. In particular, semantic aspects of system integration – having to do with integration of data between the systems – are impacted by this.

The goal of modern enterprise management is to separate business logic from system implementation. Following this principle makes it possible to adjust business logic more easily to changing situations while utilising assets efficiently through the use of adequate governance processes. A generic term for this is Business Process Management (BPM), and it has been exercised in various forms for some time already. Recently, business processes are modelled graphically and executed as scripts (such as Business Process Execution Language, [BPEL]) on process engines. Such systems make possible not only design and execution of business processes, but also advanced simulation and monitoring.

Separation of business logic benefits from adherence to architectural principles for integration of systems that are underlying business processes. Loose coupling of systems promotes flexible participation of systems to business processes and makes the task of systems integration easier. Loose coupling is the cornerstone of Service-oriented Architecture (SoA) [Erl05], an architectural style. In SoA, system's capabilities are exposed by means of one or more service interfaces. Widely used commercial SoA realisations typically use SOAP for transport and Web Services Description Language (WSDL) for service interface description. We shall have more to say about these later on in this article. System integration based on SoA can be performed as a "retrofit" to existing systems or new systems can be equipped with native SoA interfaces from the start. Services provided by individual systems can be composed into higher-level services providing more general capabilities; service interfaces of systems are often called technical services and the term business service is used for composite services.

Large-scale data processing brings challenges of its own. In centralised processing setup, large volumes of data need to be transferred to a central unit that needs heavy-duty processing capability to cope with large data sets.

In large multi-vendor systems, semantic integration of data exchanged between systems turns out to be a major issue. In this article, we study SoA-style system integration from the viewpoint of semantic integration. We discuss interface technologies and modelling paradigms, and describe a demonstration platform that is being developed at Nokia Siemens Networks (NSN) research centre to explore these issues. In the next Section, we shall describe service interface related issues, followed by a Section on semantic integration. The final Section describes the demonstrator platform, and we conclude the present article with a summary.

2 Interfaces

In this Section, we discuss service interfaces of individual systems (technical services) and their relevance to system integration following the SoA paradigm.

A service client calls the interface of a SoA-exposed system and receives either synchronous or asynchronous response. A client needs to know what kinds of service calls are possible as well as the format of requests and replies. Let us next study a couple of ways of implementing service interfaces.

The first approach is based on SOAP encapsulation and WSDL descriptions of service interfaces. In this approach, the definition of service interface consists of the following core parts [2]: Operations, messages, and data.

Operations define the capabilities of interfaces. For example, a service interface relating to customer data management might have operations *AddCustomer*, *QueryCustomer*, *ChangeCustomer* and *DeleteCustomer*. Messages define requests and replies related to operations, and data is transferred within the messages.

The second approach is based on HTTP/REST paradigm, an architectural pattern put forward by Fielding [Fie00]. HTTP/REST is also sometimes associated with an architectural style of its own, called Resource-oriented Architecture (RoA). As we shall see, REST can also be applied in implementing SoA interfaces. The relevant application of REST principles in this context is providing direct access to resources via Universal Resource Identifiers (URIs) by means of four basic HTTP operations: GET, PUT, POST, and DELETE. Implementation of service orientation with HTTP requires that additional design guidelines be followed [RR05]. The customer data operations listed earlier could be implemented with PUT, GET, POST, and DELETE to suitable URLs, respectively, assuming that each customer is modelled as a separate resource. Web frameworks can generate URI structures dynamically from data model.

Let us now step back and compare the two approaches. The first thing that stands out is the separate layer of operations that need to be defined in the first approach. This is due to the fact that actual data is not directly visible but only accessible via operations. Furthermore, message formats and data need to be defined for the operations. In the case of the second approach, operations are pre-determined and refer to resources that are representations of the actual data. Thus, there is no need to keep track of operations and messages separately from data, provided that suitable system-level guidelines are followed in exposing the data.

The difference between the two approaches is manifested in both interface implementations and governance thereof. Each operation in the first approach needs to be implemented, whereas in the second approach, generic HTTP/REST data exposition functionality can generate interface automatically. On system level, governance is more complex in the first case, as there is a need to keep track of operations, messages, and data. In the second approach, the governance is data driven. The difference between the two approaches is underlined further by governance, i.e., the fact that both systems' capabilities and business processes may change, and may give rise to changes in services both "from the top down" and "bottom up" directions. Service interfaces are thus involved in a "meet-in-the-middle" scenario that makes interface governance more challenging.

It should be noted that HTTP/REST interfaces work perfectly well as components of normal SoA stack. In particular, technical service interfaces can be constructed with HTTP/REST, and used in constructing business services.

3 Semantic integration

Service interfaces define the "plumbing" between systems, and alone are not sufficient even with data driven approach. Systems and business processes need the ability to make sense of each others' data (Figure 1). Just to provide a single example, one needs to be able to identify which alarms from network elements are business-critical, and which ones can be processed via normal procedures. Consistent use of models for integration of data within various layers – called semantic integration in this article – allows the use of consistent methodologies for all kinds of data exchanges.

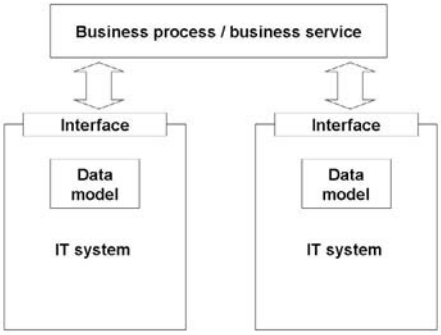

Figure 1: In addition to constructing service interfaces, data models of individual systems need to be integrated with business processes and / or other services. This is called semantic integration in this article.

Basic approaches to linking multiple data models to each other include: Point-to-point integration, Shared Data Model (SDM), and Shared Information Model (SIM).

In point-to-point integration approach, data conversion is made separately for each transaction (system to system or business process to system). This approach does not promote reuse and subsequently does not scale well. Point-to-point integration can be enhanced through data federation, where individual data models are linked to each other through hierarchical organisation of data translations. Federations can be added according to need.

Shared Data Models (SDMs) require standardisation of complete data model and are generally not a feasible alternative, except in specific, well-standardised situations.

Shared Information Models (SIMs) provide higher-level semantics to data models through definition of relations between data items. In the area of telecommunications, TM Forum Shared Information/Data model [SID] – defined in Unified Modelling Language (UML) – defines an industry core model, to be extended by vendors and Communication Service Providers (CSPs) for their own use. Individual data models can be linked to SIM and data conversions generated through mappings. Fundamentally, the goal in SIM is a single common information model, even though it may be organised into multiple layers and domains.

A consequence of the standardised nature of telecommunications domain, SIM approach is the "natural" choice in this triplet for telecom practitioners. Structuring of TM Forum SID into industry core model and stakeholder specific extensions conforms well to the way data are defined in a typical network element. Use of SIMs is not without problems, however: both multi-vendor integration and model governance pose specific challenges.

The multi-vendor challenge comes from the fact that each vendor and CSP may extend the model with their own concepts, which are not necessarily concomitant. When an OSS/BSS system is introduced in CSP's environment, it typically needs to handle multiple variants of SID core model extensions.

Model governance is typically related to service interface governance, as changes in data model may have an impact on service interface too. In the WSDL approach, data model changes are not necessarily visible in the service interface, but may still require changes in interface implementation, if changed aspects of the internal data model are related to operations exposed in the service interface. In a data-driven scenario, data model change is directly visible in service interface. When SIMs are used for semantic integration, changes in data model are reflected in relevant SIM mappings.

The principle of using SIM to amend the semantics of data can be taken further by using a more powerful modelling paradigm. Here, the next "level" is called knowledge modelling, allowing for use of reasoning to deduce more facts from given data set. This is possible due to well-defined semantics of the modelling language and scalability properties for reasoning. Description Logic (DL) [BCM+07] provides a solid basis for reasoning, and is supported by W3C semantic modelling suite consisting of Web Ontology Language [OWL] and Resource Description Framework [RDF]. The W3C suite has the added bonus of having native support for model distribution over HTTP as well as accompanying mechanisms for concurrent support for multiple models.

In addition to reasoning related advantages, knowledge-modelling paradigms complement SIMs by avoiding the need to build modelling constructs for the sole purpose of linking SIM extensions to each other. Model bridging can be built "bottom up" within the knowledge-modelling framework and expert knowledge can be leveraged to perform powerful reasoning.

4 Demonstrator platform

A demonstrator platform has been developed at NSN research unit to explore alternative approaches to semantic integration based on SoA architectural style and leveraging knowledge modelling as well as reasoning. In what follows, we shall first describe design principles, followed by design and selected implementation details.

The overall goal of the demonstrator platform is to provide a building block for performing advanced data integration. A particular goal is the ability to expose data in a format that makes semantic integration easier. As we have seen above, this relates both to the implementation of the interface and the integration paradigm itself.

Given the above "task definition", the basic requirement is support for reasoning and adequate modelling framework. The combination of OWL, RDF, and Description Logic reasoner referred to above seems like a natural choice, in part due to multiple implementations of the Application Programming Interfaces (APIs) and modelling tools.

The goal was to use publicly available implementations. HTTP libraries are available in almost any programming language, but the most up-to-date publicly available for OWL and RDF libraries are based on Java™. Subsequently, Java™ Virtual Machine (JVM) was chosen as the programming environment for the project. A further design choice was the use of a functional programming language for implementation to achieve more concise code base, as well as being well suited to data transformation tasks.

Multiple functional programming languages are available as scripting languages on top of JVM. Clojure [Clo] was chosen for this project due to good interface to Java™ and a fair amount of utility libraries. Clojure is a Lisp-2 programming language with influences from Haskell. Compojure [Com] web framework – based on Jetty [Jet] Java™ libraries – was used for creating HTTP/REST servers. Compojure provides full support for dynamic REST interfaces. The OWLAPI [OWLA] library was chosen for knowledge base and reasoner access, and Pellet [Pel] was selected as the reasoner.

In order to utilise HTTP interface, knowledge base, and reasoner efficiently, Clojure API was created for respective capabilities, utilising and complementing underlying OWLAPI Java™ APIs. The result is a semantic integration platform, the components of which can be used in multiple combinations. The REST/HTTP server capability can be viewed as one component, whereas knowledge management (knowledge base and reasoner) constitute another one.

In what follows, we shall describe the current research implementation targeted at providing a platform for developing applications that can make use of advanced knowledge management capabilities. The applications call the capabilities of the modules via Clojure API. The assumption in the implementation has been that telecommunication systems are exposed by means of HTTP/REST interfaces for reasons discussed above. The platform can provide a HTTP/REST interface towards the telecommunications system to expose its data. The Compojure web framework provides the means to do dynamic data-driven HTTP/REST interface generation. The structuring of the platform is illustrated in Figure 2.

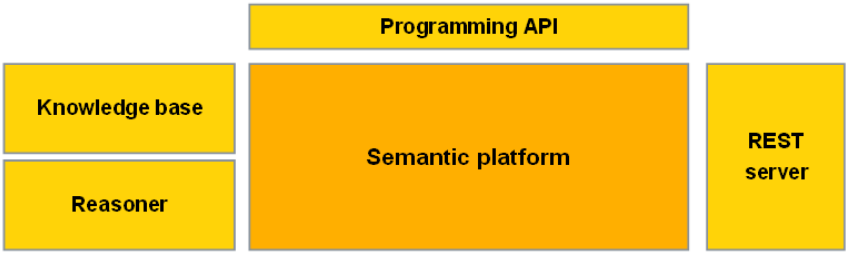

Figure 2: Semantic integration platform and its components.

The platform can be used in two ways: Driven by service interface or driven by the telecommunication system. The alternatives are illustrated in Figure 3.

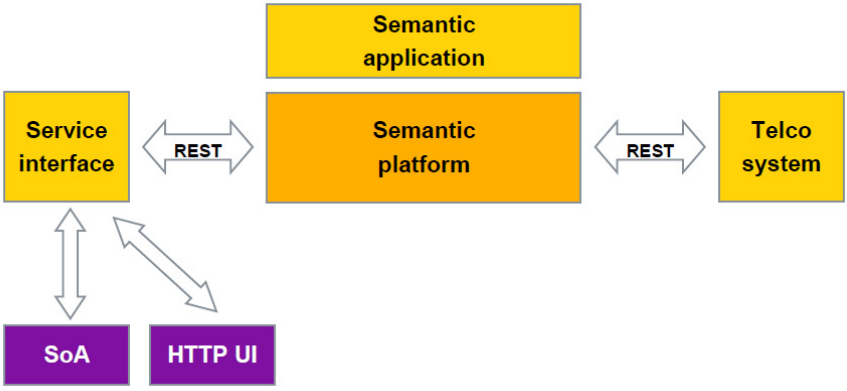

Figure 3: Semantic integration platform use cases.

In the first mode, service interface request triggers a semantic application to be run, possibly triggering further requests to telecommunication systems. An example application of this mode could be advanced telecommunications data processing.

In the second mode, telecommunication system triggers the semantic application, but otherwise the chain of events is the same. Indeed, since interfaces are implemented with HTTP/REST, the application does not even need to know from which direction the query came from.

The same service interface can support both machine users (HTTP/REST clients) and human users (using web browsers). The Clojure API contains HTML rendering functions that are useful for creating human readable pages.

Clients accessing service interfaces don't need to be using the demonstrator platform – a normal HTTP/REST client on any programming platform will do. When systems are already running JVM, the HTTP/REST component of the platform can be utilised in building HTTP server, but basically any HTTP client framework supporting REST operations will do. As discussed earlier, web frameworks supporting dynamic HTML make it possible to expose data dynamically.

5 Usage of semantic integration platform

Let us next consider how this kind of platform could be used in practical semantic integration. We shall assume that both major components are used as a platform for developing semantic applications.

The first prerequisite is gaining access to the telecommunication system. Semantic application run on the platform needs to get data and make sense of it. Data needs to be represented in knowledge base in order to make use of knowledge management capabilities. How exactly this is achieved depends on the way data is exposed. Knowledge base can directly refer to data in RDF format, whereas other data needs to be converted by the semantic application. In addition to triggering reasoning, application may need to "drive" the knowledge base updates to orchestrate knowledge model.

We can now outline the design and implementation of a semantic application. The first step is the definition of the use case. Design phase needs to assess the interfacing strategy for accessing the relevant telecommunication systems, as well as the service interfaces (towards SoA system and/or telecommunications network). The next step is the design of the ontology and RDF facts residing in knowledge base. A well-designed knowledge base goes a long way towards building the application. The use of a suitable user interface such as Protégé [Pro] makes it possible to test reasoning in the UI before implementing the actual semantic application. Once the ontology and the type of data residing in the knowledge base are known, semantic application driving the knowledge capabilities can be developed.

The platform has been used in small OSS/BSS use cases where the use of reasoning reduces the need for programming. Such situations arise naturally in distributed systems where data from multiple sources of needs to be assimilated to perform coordination [Mah07], for example. These tests – to be described in detail in other fora – indicate that most of the programming needed is related to moving data between telecommunication system and knowledge base, and a significant part of actual "heavy lifting" is performed by reasoner with the help of the ontology. Similarly, assimilation of data from multiple sources is made easier by the use of semantic modelling, circumventing the need to map all data to single information model. A prerequisite for these advantages is the use of suitable modelling patterns in the ontology. Generally speaking, though, it is not always optimal to put everything in the knowledge base instead of the program logic, as the reasoning scalability is affected by the modelling patterns.

The anticipated use of the platform in its present form is a supporting capability for normal OSS/BSS functionality for situations that require complex combinations of data from multiple sources. The platform enables creation of advanced capabilities with small amount of code. There's no free lunch available here, but proper modelling patterns need to be used to achieve the goal of minimising application programming. Single platform instance can run simultaneously multiple applications sharing a knowledge base. Clojure provides mechanisms for thread safety based on Software Transactional Memory (STM) [Hal09], whereby multiple instances of one application may be run concurrently.

From a broader viewpoint, the capabilities of the platform provide a basis for distributed data federation. Each server having the components described above can federate its own and other servers' data, process it with knowledge models, and expose the result by means of a REST interface for next level clients. This means that less data needs to be transferred to perform data analysis tasks, and that data processing is distributed.

6 Summary

We have outlined challenges inherent in semantic integration telecommunications domain, and analysed related technologies with focus on system interfaces. HTTP/REST was identified as a complementary technology for SoA architectures for dynamic data exposition purposes. Knowledge management technologies such as Description Logic and W3C modelling standards, provide means of reducing the burden of semantic integration of data models through reasoning and higher level abstractions, as well as providing a basis for approaches which are challenging with only information models, such as linking multiple non-concomitant information models together [SSR+07]. A proof-of-concept type semantic integration demonstrator platform based on HTTP/REST and W3C semantic web standards was described and typical usage scenario presented. Functional programming language Clojure was used to achieve concise code base and Java™ compatibility. Different combinations of the components of the demonstrator platform are envisioned to be relevant to a range of use cases, ranging from simple data exposition to data federation to a platform for developing and running coordination applications in cognitive networks.

Acknowledgments

The author would like to thank (in alphabetical order) Frank Berger, Carlos Cruz, Pedro Diaz, and Veli-Matti Teittinen for comments and input.

References

[BCM+07] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider., *The description logic handbook*, 2nd edition, Cambridge University press, Cambridge, UK, 2007.

[BPEL] See WS-BPEL specification at OASIS web site http://www.oasis-open.org), accessed April 2010.

[Clo] See Clojure website (www.clojure.org), accessed April 2010.

[Com] See Compojure home page (http://github.com/weavejester/compojure), accessed April 2010.

[Erl05] T. Erl, *Service Oriented Architecture*, Prentice Hall, Upper Saddle River, NJ, U.S.A., 2005.

[Fie00] R. Fielding, *Architectural styles and the design of network-based software architectures*, Ph.D. thesis, University of California, Irvine, 2000.

[Hal09] S. Halloway, *Programming Clojure*, Pragmatic Bookshelf, Raleigh, NC, U.S.A., 2009.

[Jet] See Jetty home page (http://jetty.codehaus.org/jetty/), accessed April 2010.
[OWL] See OWL specification at W3C website (www.w3c.org), accessed April 2010.
[OWLA] See OWLAPI home page (http://owlapi.sourceforge.net/), accessed April 2010.
[Mah07] Q.H. Mahmoud (editor), *Cognitive networks*, John Wiley & Sons, Chichester, England, 2007.
[Pel] See Pellet home page (http://clarkparsia.com/pellet/), accessed April 2010.
[Pro] See Protégé home page (http://protege.stanford.edu), accessed April 2010.
[RDF] *S*ee RDF specification at W3C website (www.w3c.org), accessed April 2010.
[RR05] L. Richardson and S. Ruby, *RESTful web services*, O'Reilly, Sebastopol, CA, U.S.A, 2005.
[SID] *Shared Information/Data (SID) model*, TM Forum, www.tmforum.org, accessed April 2010.
[SSR+07] J. Strassner, J. Neuman de Souza, D. Raymer, S. Samurdala, S. Davy, and K. Barrett, The design of a new policy model to support ontology-driven reasoning for autonomic networking, in *Proc. NOMS'07*, Petrópolis, Brazil, 2007.