# Prototype of a Framework for Ontology-aided semantic conflict resolution in enterprise integration

Željko Vuković*, Nikola Milanović*, Gregor Bauhoff **
* Faculty of Technical Sciences, University of Novi Sad, Serbia
** PI Informatik GmbH, Berlin, Germany

zeljkov@uns.ac.rs, mnikola@gmail.com, bauhoff@pi-informatik.de

***Abstract* — Enterprise integration carries the need for resolution of various semantic conflicts. These conflicts come in many forms and each of those may appear in a different context. Conflict detection and resolution can be made easier if a semantic description of the involved systems is available. We have developed a prototype, based on existing software - *Talend Open Studio ESB*, for a framework where a user may attach an ontology to interface elements and have those interfaces mapped automatically. We present how this prototype was tested on a scenario for which a solution was previously developed manually at Model Labs GmbH/PI Informatik GmbH, Berlin.**

## I. INTRODUCTION

Developing enterprise integration solutions presents various challenges: unreliable and slow networks, heterogeneous applications, inevitable changes over time [1]. Differences between applications may be technical or semantic. Data may be stored in different format (e.g. 32 vs. 64 byte, little- vs. big-endian), interface elements with the same semantics may have different names, interface elements with the same name may have differing semantics and so on. One system may return data as a collection, while another one may expect elements of that collection one at a time. Manually detecting and resolving these conflicts is a tedious, error prone work. Often, a lot of glue code is needed to make systems work together [1].

We have developed a prototype for a framework that can help automate some of the steps in conflict resolution. Interfaces and their elements can be semantically described using ontologies in order to facilitate this automation.

In this paper, we describe a prototype for this framework and how we have tested it on a real-world integration scenario.

## II. RELATED WORK

In [2] a framework is given for conflict analysis and composition at the component level. Components that originate in object oriented middleware are represented canonically on common denominator basis. The framework is model based. A classification of semantic conflicts is given in [3]. Here, three dimensions are used for classification: naming, abstraction and level of heterogeneity.

One (meta) model-based platform for integration is given in [4] along with the accompanying methodology. It allows for a tight cooperation with the domain expert. The platform enables semi-automatic conflict analysis. An example of using ontologies expressed using the Web Ontology Language (OWL) and available on a network is shown in [5]. It was concluded that by adding classes to an existing ontology, enterprise integration was possible in hours time. An approach called ODSOI (Ontology-Driven Service-Oriented Integration) was proposed for using a combination of ontologies and web services in [6] to address some problems of enterprise integration, along with a vision of an integration framework.

Following Model-Driven Architecture and using a Domain Specific Language (DSL) was proposed in [7]. A DSL called Guaraná was proposed for design and automatic deployment of integration solutions. Another (internal) DSL for enterprise integration called Highway is presented in [8]. It is based on Apache Camel and Clojure. An executable DSL that is platform-independent and message-based is described in [9].

In [10] chapter 10 discusses using Ontology Architectural Patterns in order to achieve semantic enterprise interoperability.

## III. SCENARIO DESCRIPTION

To test our approach, we have chosen to recreate an existing integration solution. The scenario is part of a project management portal. The portal is designed to query a web service in order to get data about projects, tasks, etc. However, some legacy data is stored in a SAP system. An integration solution is therefore needed that will expose a web service. Upon getting a Simple Object Access Protocol (SOAP) request for that service it should fetch data from SAP and then wrap the data in a SOAP response. Data is retrieved from the SAP system by placing a Business Application Programming Interface (BAPI) call to a function. A schematic view of the scenario is shown in Fig. 1.

The existing solution was manually coded in Java and uses Hibersap[1] for mapping data from SAP to an object model. It is being executed in the SAP Process Integration

[1] A library for mapping Java classes to SAP backend via annotations, http://hibersap.org

(PI) middleware. Had there been another source of legacy data (e.g. a database) the process of manually writing a mapper would have to be repeated. It could be argued that this scenario is not overly complex and that a simpler solution might have been devised. However, it is important to remember that one of the challenges of developing integration solutions (as for software development in general) are the inevitable changes. For this reason, even for a simple scenario, a flexible solution might present a good investment.

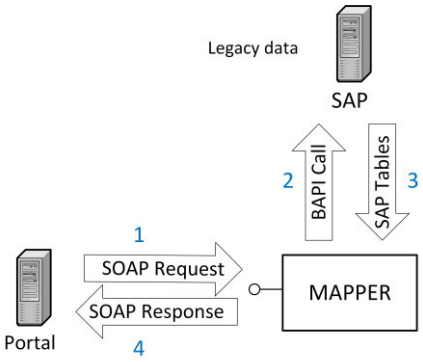


Figure 1. Integration scenario

## IV. BASE PLATFORM

Rather than developing a prototype from scratch, we have decided to modify and extend one of the existing IDE tools for integration. We needed a tool that can visually represent the integration solution, be extendable and offer access to the internal object representation of the solution that the user has built, so that we could use it for analysis and code generation. After researching several available platforms, for this purpose we have chosen Talend Open Studio for ESB (TOS). It is a tool that enables users to develop, test, deploy and administrate integration solutions. A user can graphically lay out the integration solution either as a TOS Job (in *Integration view*) or an Apache Camel route (in *Mediation view*). The Mediation view uses standard Enterprise Integration Pattern (EIP) graphical representation established in [1]. Jobs and routes can cooperate. Both jobs and routes can be run either inside TOS or deployed to a standalone runtime environment. Talend's runtime is based on Apache Karaf and Apache Camel. When a job is run from within TOS, real-time performance and error information is available on the graphical editor for each component and connection. The application comes with connectors for a large number (more than 800) of data sources: files, databases, web services, REST, e-mail, open and proprietary protocols, big data, cloud, .NET etc. An SDK is available for component development. TOS has built in code generation (based on Java Emitter Templates), which further shortened our development process. Various orchestration tools are available based on time, system or user events.

Source code for TOS is freely available on GitHub. The architecture of TOS is that of an Eclipse Rich Client Platform. This allowed us to easily extend and modify parts of the application.

## V. SOLUTION

The web service definition (WSDL) file was loaded into TOS. This makes the service available and loads service metadata. From the loaded WS definition, a Job can be created automatically that contains TOS components necessary for the WS request and response. The request component can be configured in terms of address and port on which the WS will be available. Accessing data from SAP is done using the tSAPInput component. Connection to the SAP server is configured in the tSAPConnection component. The tSAPInput component makes a Business Application Programming Interface (BAPI) function call to the SAP system and receives several tables as a response, describing the structure of managed projects (name, planed begin and end date, actual duration, related projects, subproject hierarchy etc.). Extraction of the BAPI function metadata (input and output parameters) was done by writing a Python script that converts a TXT file exported from SAP GUI to an XML representation that can be used for loading into TOS. Automated interface extraction for SAP is available as an extension in TOS, but only in the Enterprise version.

The TOS Job is laid out as in Fig. 2. When a request is received for the WS, connection to the SAP server is initiated. If the connection is successful, the SAP input component fetches data by making an SAP BAPI call. Error handling is possible in TOS, but is omitted here for clarity. Returned data is then fed to a tXMLMap component. This component does the necessary mapping between the format in which data is received from SAP and the format in which it needs to be sent as a SOAP/XML response.

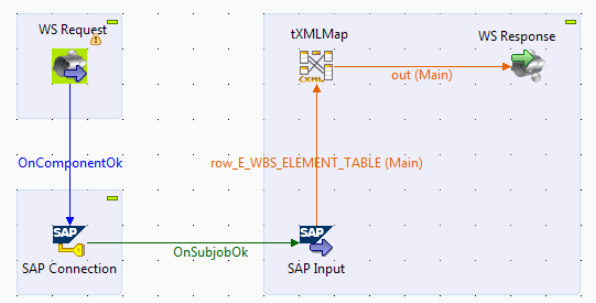


Figure 2. Talend Open Studio job with components laid out and connected

The stock tXMLMap component in TOS has an Auto Map feature, which makes a connection between input and output interface elements if they have the exact same name. Another component, tMap works the same, but accepts different kinds of input and output formats. We have used these two components as a basis for the prototype of our framework.

First, we have extended metadata property editors in TOS. These metadata editors already allow the user to choose name, type, length, date format, etc. for interface elements (rows in TOS terminology). We have added a way for the user to load an ontology file specified in OWL and then to annotate interface elements with one or more ontology elements.

The ontology can be used to formally specify knowledge about systems and their interfaces. This can then be used in the process of conflict detection and resolution. For loading, manipulating and persisting of the ontologies, we are using the Apache Jena framework. In order to use it in the OSGi environment, we've encapsulated Jena as an Eclipse plugin [11].

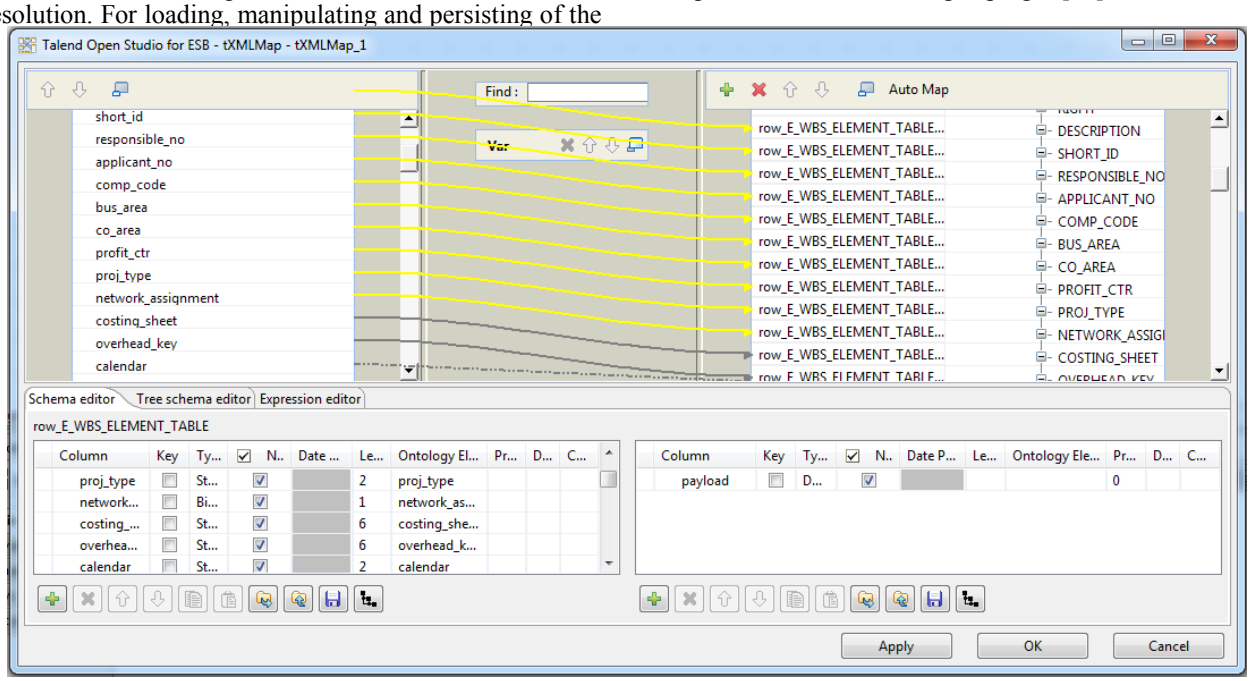


Figure 3. The modified tXMLMap component with automatically mapped input and output interface elements. The button next to the Save button is used to load an ontology.

In general, knowledge about involved systems could be in multiple ontologies and should therefore be merged before they are used. Since ontology merging is not trivial operation and is out of the scope of our research, at this time we assume that it has been already performed and we operate with a single ontology that describes all systems that are involved.

Once interface elements are annotated with ontology elements, this semantic description can be used by the Auto Map feature. Our current implementation is rather simple: it will map those input and output elements that are annotated with the same ontology element. It could be argued that this strategy may offer a marginal time saving in the overall integration process and even take longer, depending on the scenario. It can certainly be said that it does not provide any conflict resolution. However, our current goal was only to establish a solid platform where semantic data will be available in a powerful, real world environment. Further work on developing the actual conflict detection and resolution is to follow.

The mapper is not limited to one input and one output interface. Multiple input and output interfaces can be involved in a single mapping. Likewise, one interface element may be annotated with multiple ontology elements. When an element of the output interface is matched with more than one input element, the way in which those two elements will be merged may also present a semantic conflict. Dealing with these merge conflicts is another point for future research. In the prototype we have opted for the simplest possible solution: data from the input elements is converted to strings and those strings are concatenated.

Figure 4. Elements from multiple input interfaces mapped to a single output interface element

It is important to note that after automatic mapping is performed, the user has the ability to manually review and edit the mappings.

The user interface of the map component can be seen in Fig 3. and Fig. 4. Input interface is on the left and output interface on the right.

## VI. FUTURE WORK

During the development of this prototype, a general workflow needed to develop an integration solution became apparent and is shown in Fig. 5. As stated earlier, the first step - processing ontologies is out of the scope of our research. The final step, code generation for target ESB runtimes is already available in TOS. We plan to focus on the three inner steps: finding mappings, detecting and resolving semantic conflicts and building mapping expressions, which are then used by the TOS code generator.

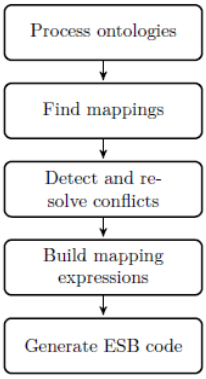


Figure 5. General framework workflow

Given the vast possible kinds of conflicts and the fact that each of them may appear in a different context for each integration scenario, we conclude that developing a fully automated solution for each such case would be next to impossible. For this reason, we plan to develop the framework in such a way that it may be customizable by the user. One way that we see as adequate for this is developing a conflict resolution DSL. Each step of the framework workflow could then be parametrized by the user, without the need for coding in some general purpose language like Java. The user could specify how each type of conflicts will be handled in the given context. A sensible default should exist for each conflict type.

To make the DSL usable and productive, an editor will need to be available for it that offers the usual coding aids to the user: syntax coloring, code completion, instant error checking and highlighting [12]. To develop such an editor and integrate it into our existing prototype, which is Eclipse based, a tool like Xtext or Spoofax/IMP may be used.

## VII. CONCLUSION

Extending an existing integration platform has tremendously cut the time needed to develop a working prototype that we can use to test our framework. Using this prototype, we were able to import an ontology that describes the involved interfaces. We have then annotated elements of those interfaces with ontology elements. This information was then used by the Auto Map feature that we have developed to correctly connect elements of the input and output interfaces. After the automatic mapping, the user is able to visually inspect the results, review them and modify if necessary. These mappings were then used to generate code that successfully replaced an integration solution that was previously manually coded and used in a real-world application. Using such a powerful tool, along with the ability to map elements automatically shortens the integration solution development time, while allowing for a very flexible solution that can be modified as necessary if any of the involved systems, or enterprises themselves change with time.

We plan to further research this subject and develop a way for the end user to describe conflict resolution rules for within the context of their own scenario. One way we see fit for this purpose is developing a DSL in which the user could describe how each type of conflict will be handled.

## ACKNOWLEDGMENT

Part of the research was done at the offices of PI Informatik, GmbH in Berlin. We wish to extend our gratitude for their help and hospitality.

## REFERENCES

[1] G. Hohpe, B. Woolf. “Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions”. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[2] A. Leicher. “Analysis of Compositional Conflicts in Component-Based Systems”. PhD thesis, Fakulät IV - Elektrotechnik und Informatik der Technischen Universität Berlin, 2005.

[3] A. Ouksel, E. Naiman. “A classification of semantic conflicts in heterogeneous database systems”. Journal of organizational computing, 1995.

[4] R. Kutsche, N. Milanovic, G. Bauhoff, T. Baum, M. Cartsburg, D. Kumpe, J. Widiker. “BIZYCLE: Model-based interoperability platform for software and data integration”. TU Berlin, 2008.

[5] S. Stoutenburg, L. Obrst, D. Nichols, P. Franklin, K. Samuel, M. Prausa. “Ontologies in OWL for Rapid Enterprise Integration“. OWLED 2007 Workshop on OWL: Experiences and Directions

[6] S. Izza, L. Vincent, P. Burlat, “A Unified Framework for Enterprise Integration. An Ontology-Driven Service-Oriented Approach”. Interoperability of Enterprise Software and Applications INTEROP-ESA, 2005

[7] H. Sleiman, A. Sultán, R. Frantz, R. Corchuelo. “Towards Automatic Code Generation for EAI Solutions using DSL Tools”. JISBD, 2009

[8] V. Kovanović, D. Đurić. “Highway: a domain specific language for enterprise application integration”. 5th India software engineering conference. ACM India, 2012

[9] M. Shtelma, M. Cartsburg, N. Milanovic. “Executable Domain Specific Language for Message-Based System Integration”. MODELS 2009, USA

[10] Y. Charalabidis. “Revolutionizing Enterprise Interoperability through Scientific Foundations“. IGI Global, 2014

[11] J. McAffer, J. Lemieux, C. Aniszczyk. “Eclipse Rich Client Platform”. Addison-Wesley Professional; 2 edition, 2010

[12] M. Voelter, “DSL Engineering: Designing, Implementing and Using Domain-specific Languages”, CreateSpace Independent Publishing Platform, 2013