Selected Interoperability Mechanisms Implementation with the Use of the SOA Approach

G. Bliźniuk, R. Dawidczyk
grzegorz.blizniuk@wat.edu.pl

Institute of Computer and Information Systems
Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw, Poland

In this article a short discussion of interoperability was presented according to proprieties of Information Technology (IT) systems. A proposal about some assuring interoperability of IT systems interoperability, so called “conversion rules” of interoperable data was mentioned. The next, a concept of providing successful and efficient implementation of interoperability mechanisms for cooperating inside IT was described. All main considerations were illustrated in short examples.

Keywords: interoperability condition, conversion rules, SOA.

1. Introduction

Interoperability is a non-functional propriety of cooperating information systems and cooperating information technology systems. The quintessence of interoperability of information systems is the ability to interchange information and make a proper use of it. For information technology systems this condition is understood as interchange and the proper use of data. The reason of this point of view is the assumption about nonphysical integration of systems throughout one new system. Each analyzed system is viewed as a “black box” with specific interfaces, dedicated for effective cooperation of systems.

Presented understanding of quintessence of interoperability was suggested and shared among many descriptions of this term, from which some are very interesting. According to description from [6], interoperability is understood as an ability of two or more systems or components to exchange and use of information. According to [1], this is described as an ability to efficiently transfer and use unified information across different businesses and information technology systems. This is very important, to look at interoperability throughout the perspective of businesses and business processes. Similar understanding of interoperability can be read in [5].

According to [1] there are certain prerequisites for IT systems to interoperate. One of them is the need for establishing a common domain of data that is supposed to be interchanged (formulas 5–7). That common domain is the collection of data, which shares the same meaning among two independent IT systems. It is a rare situation, when data being the subject of interchange can be exchanged without any additional expenditure on the implementation of a mechanism for their conversion. More likely, as described in article [1], in that kind of situation the use of so-called “conversion rules” (see formulas 8–10) is necessary. To provide an example of bits of data that have the same meaning but different format, and so requires conversion to be exchanged successfully, one may consider the difference in formats of dates in two given systems. Different date notations differ in numeric or word data representation, separating characters, number of digits or characters representing each period.

In terms of interoperability layers, implementation of a conversion mechanism aims at fulfilling the syntactic and semantic interoperability aspects: ensuring that data that has to be shared (is both correctly sent and received) can be properly interpreted by exchanging participants [2], [1].

The most popular approach when implementing an interface for data exchange between independent systems is the SOA concept using the Web-Services standard. Multiple technologies can be used for implementation The protocol that is gaining a particular acceptance is SOAP, which relies on HTTP for data transmission and XML format for structuring messages sent between participants.

1 See: description of formula 7 in article [1]
Thus, implementation of selected conditions is based on the author’s assumption, that interoperable IT systems have interfaces compliant with SOAP and Web-Services principles, and concerns those dimensions only.

Creating software responsible for data conversion for each cooperating system separately is an inefficient, time consuming and costly task. In the process of analysis, all of the necessary conversion rules have to be identified and documented. Afterwards, rules have to be designed, implemented, and tested – for each portion of exchanged data. It is obvious that generated code is not likely to be used as a solution in a different case of cooperation. This situation is illustrated in figure 1, where IT systems use different conversion mechanisms.

That analysis led authors to an idea of creating a comprehensive tool, a Converter, which would provide universal data conversion methods. With such software, providing conversion methods for data exchange requires less workload. As a result, creating data conversion routines is less of a challenge. After determining the necessary conversion rules, instead of implementing them, only the configuration of the Converter is required. The use of the Converter (marked as “C”) in a situation as shown in figure 1 is illustrated in figure 2.

2. Requirements and Functionalities of the Converter

The main objective of the Converter application is to convert messages from the format provided by the sending system to the format accepted by the receiving system. This is achieved by converting data sent within the SOAP XML document, as well as the XML structures. Figure 3 shows how the Converter is involved in the communication between two IT systems. System “A” is sending a request addressed to system “B”. Instead of sending it to the participating system, the request is sent to the application. It should determine the network address of the receiving system and conversions that are to be used based on sender’s network address and request name. The request name is the invoked SOAP method. All this information should be provided beforehand as the configuration of the Converter that is stored in database of the application. Afterwards, the converted message is sent to the receiving system, which generates the response document. The response received by the Converter is translated (if necessary) and sent back to the sender i.e. system “A”.

The implemented functionalities of the Converter are shown on use case diagram in figure 4.
Fig. 4. Use case diagram for the Converter

"Sender" and "Receiver" are IT systems participating in communication and the most important actors specified on the use case diagram. The use case "message sending" is associated with those systems. The relation is directional and indicates that the sender is the system that initiates the sequence of events, which make up the communication. In SOAP-based architecture, web modules usually provide a number of different methods within a web service. Therefore, for some methods the role of the Converter is simply to forward the received message to the receiver. It is a rare situation, in which data exchanged between systems is compliant and can be sent directly without further workload. For methods that require conversion, the "message sending" use case is extended by the "conversion" use case. It represents conversion routines being executed to convert defined data accordingly to the configuration. It is possible to execute more than one conversion routine for a transferred message. In fact, it is desired to use multiple general conversion rules, rather than implementing a single one. It is in this way the general rules implemented in the Converter can be used in different combinations and for different requests. "Logging" is a use case included in both (already described) use cases. Converter software keeps log information of transferred messages, as well as of errors and exceptional events.

The human actor's role of "Administrator" is to provide configuration for the Converter as illustrated by the following use cases:

- definition of conversion – the collection of conversion rules that are supposed to be applied when converting a request or response.

One of the requirements is to provide a convenient way to administrate and configure the application. In order to cope with that need an administration panel has been implemented in the form of a web page. It provides the possibility to add, remove and modify connections between systems, served requests, executed conversion rules and their parameters. Only authorised users have access to the Converter administration panel as shown in the "authorisation" use case.

The rules implemented within the Converter are divided into two categories; routines converting data wrapped inside XML tags and procedures for converting required structures of the document. Structure modifying rules enable inserting, deleting or modifying names of XML nodes. More complex structural rules allow splitting single nodes with their values into multiple nodes containing portions of original data and vice versa: structure modifications that allow merging multiple nodes into one. On the other hand, data conversion rules allow replacing contents of a given node: modifying date values or formats, doing simple arithmetic or changing/replacing data using regular expressions. For more details and examples refer to [2], chapter 2.2.

3. Implementation Details

The Converter is a tool aimed at supporting communication between systems, which use principles of Web Services and SOAP as a means of data transmission. SOAP in turn relies on web protocols like HTTP, HTTPS or SMTP for data transmission. Hence the author’s decision to implement a translator software as a web application. Among a wide selection of internet technologies, the choice was PHP and Apache HTTP server. The use of such application platform has its advantages. One of those is that there is no need to implement lower level interface for data transmission and negotiation, which allows concentrating the engineering efforts on logical aspects of the application. The environment provides ready solutions for performance and multi-threading aspects. Other advantages are the popularity of the Apache Server and PHP language as well as their open source nature, which is reflected in the amount of documentation and community support, as well as deployment and maintenance.

• definition of conversion – the collection of conversion rules that are supposed to be applied when converting a request or response.

One of the requirements is to provide a convenient way to administrate and configure the application. In order to cope with that need an administration panel has been implemented in the form of a web page. It provides the possibility to add, remove and modify connections between systems, served requests, executed conversion rules and their parameters. Only authorised users have access to the Converter administration panel as shown in the "authorisation" use case.

The rules implemented within the Converter are divided into two categories; routines converting data wrapped inside XML tags and procedures for converting required structures of the document. Structure modifying rules enable inserting, deleting or modifying names of XML nodes. More complex structural rules allow splitting single nodes with their values into multiple nodes containing portions of original data and vice versa: structure modifications that allow merging multiple nodes into one. On the other hand, data conversion rules allow replacing contents of a given node: modifying date values or formats, doing simple arithmetic or changing/replacing data using regular expressions. For more details and examples refer to [2], chapter 2.2.

3. Implementation Details

The Converter is a tool aimed at supporting communication between systems, which use principles of Web Services and SOAP as a means of data transmission. SOAP in turn relies on web protocols like HTTP, HTTPS or SMTP for data transmission. Hence the author’s decision to implement a translator software as a web application. Among a wide selection of internet technologies, the choice was PHP and Apache HTTP server. The use of such application platform has its advantages. One of those is that there is no need to implement lower level interface for data transmission and negotiation, which allows concentrating the engineering efforts on logical aspects of the application. The environment provides ready solutions for performance and multi-threading aspects. Other advantages are the popularity of the Apache Server and PHP language as well as their open source nature, which is reflected in the amount of documentation and community support, as well as deployment and maintenance.
costs. Finally, since the fifth revision, PHP is an object-oriented scripting language with decent built-in support of XML documents.

The Converter uses the database to store the configuration. The details of database architecture, including conceptual and physical structure models, are described in [2], chapter 2.5. The PostgreSQL was chosen for DBMS. It has the same open source advantages, as PHP and Apache. Additionally, it most likely resembles popular commercial database systems.

The Converter software was designed to run on a LAPP platform: Linux operating system, Apache HTTP server (<<webServer>>), PostgreSQL DBMS (<<database>>) and PHP programming language module "mod_php" (<<interpreter>>). The <<device>> node is a physical hardware device, on which those components are installed.

The system was modelled and implemented in accordance with object-oriented programming paradigms. UML notation was used for software engineering and development. The most important classes specified, that make up the core of the application, are illustrated in the class diagram, figure 6. The class "Converter" is the fundamental part of the logic. It keeps the information about network addresses of both cooperating systems through the communication process and the name of the request. That information is obtained at the constructor method. Only one instance of this class is created during a single messaging process. Implemented routines coordinate the work of objects used during conversion.

The "Converter" class aggregates "Rule" classes. "Rule" is an abstract class. Every conversion rule implemented in the Converter is a type of "Rule". The goal of such a design is to unify the behaviour of all conversion rules implemented in the application. These rules are shown on a diagram in figure 8. Every inheriting class has to provide implementation of an abstract method "run". The body of this method in inheriting entities contains the logic and activities of a given conversion routine.

Both classes are associated with "SOAPMessage" class that is implemented to store the XML message in the form of the DOM object. It provides a convenient way to select and manipulate XML nodes and access their contents or attributes, basing on the built-in PHP classes, such as "DOMNode", "DOMDocument" and "DOMXPath" [7]. The relations result from the built-in DOM API structure.
Fig. 7. Messaging and conversion process flow

Activities performed by the application during messaging process are described in the diagram in figure 7. After receiving a request, the XML document is extracted and deserialized from its string form into an object structure. An instance of "SOAPMessage" is created and, at that moment, it is described as "received". The request name is obtained. Afterwards there is a decision, based upon the read configuration data, whether the message contents require conversion. That is a realisation of assumptions introduced in the diagram in figure 4, as a "conversion" use case that extends the basic message sending. In case a conversion is required, required conversions are executed. Thus, the stored message becomes converted. Regardless of whether the conversion has been made or not, the message is sent to the receiver system. If a response message is received, the messaging and conversion process is invoked again, as shown in the activity diagram.

Conversion routines are implemented within classes, the logic of a single conversion rule per single class. When a certain conversion is to be made during the messaging process, the class that contains the rule implementation is constructed and the "run" method is executed. In order to standardize every conversion class, each one extends the "Rule" class and provides implementation for abstract methods. Every conversion rule requires the name of the node upon which it is executed, and certain arguments, which control the conversion activity that is to be made. The conversions are divided into two categories: structural and value. Structural conversions are implemented to change the structure of the XML document, whereas value conversions operate on the values of document nodes. Implemented conversion classes are visible in the diagram in figure 8 and described below:

1. "Mathematical" is a conversion rule that allows simple mathematical formulas to be executed on the contents of a given XML node.
2. "Date" allows modification of the date information.
3. "RegExpReplace" – changes the value (contents) of the XML node in accordance with the provided regular expression.
4. "XMLSplit" – implemented to provide a possibility to split a XML node that contains multiple information into multiple nodes.
5. "XMLJoin" – opposite to the above: provides a possibility to merge multiple nodes into one.
6. "ChangeName" – changes the chosen XML node name.
7. "Remove" – removes the selected XML node.

Examples of some of these rules are presented in the next section, for full reference see [2].

Not all of the implemented or used built-in classes are mentioned in this article. Those may be classified as helper classes that aid the application in the messaging process, and include the following:

- "Database" class – acting as a way to reach the configuration data stored in the database
- "Logger" class – reporting activities made during the messaging process and encountered exceptions
- "HttpMessage" class – built-in entity used to send a converted request to the receiving system and to obtain the response content.

For details about the application structure, refer to [2].

4. Selected Examples of Messages

To see the effects of conversions done by the Converter, a simple assessment application has been implemented. It sends a declared SOAP message to the Converter, receives the request and generates a response message. The Converter is required to be configured to send the processed request to the assessing application. In effect, it is possible to see messages generated during the messaging process (figure 3), and allows the administrator to test configured conversions in detail before deploying software in a production environment. For more information on testing and the assessment software, refer to [2].

The examples of messages presented in this section are not full SOAP messages. They have been trimmed, so that only the contents of the "Body" section of the SOAP message with the actual data is visible. Furthermore, it is unimportant, whether discussed messages are requests or responses. The aim is to discuss the conversions, based on the comparison of raw and converted messages.

The first example (see Script 1) shows the use of "XMLSplit" and "ChangeName" conversion rules. The Converter has been configured to change the name of the node that contains the request/response name from "getPersonResponse" to "getPersonNameResponse" as an example of the "ChangeName" rule. The presented message contains a name and surname of a person. The Converter uses the rule "XMLSplit" to split that single node into two separate nodes, one for the name and the other for the surname, thus, nodes "name" and "surname" are generated. The assumption is that the format of the raw message containing the name data is organized as "name surname" with a space as a separator.

```
Raw message:
<getPersonResponse>
  <person>Adam Abacki</person>
</getPersonResponse>

Converted message:
<getPersonNameResponse>
  <name>Adam</name>
  <surname>Abacki</surname>
</getPersonNameResponse>
```

Script 1. Conversion example with "ChangeName" and "XMLSplit" rules

The next example illustrates the use of rules "XMLJoin", "Remove" and "Date". The raw message contains information about a date of an exemplary event, and is listed below. Within the message "eventDate", event name (node "name") and event date (structure "date") are sent. The aim is to convert the date structure into a universal format "CCYY-MM-DD hh:mm:ss" and contain it within a new "dateTime" node, and to add five hours to the resulting time variable (as an example of the different time zone).

```
Raw message:
<eventDate>
  <name>event</name>
  <date>
    <year>2010</year>
    <month>08</month>
    <day>15</day>
    <time>15:43:00</time>
  </date>
</eventDate>
```

Script 2. Raw message for conversion example with "XMLJoin", "Remove" and "Date" rules.

"XMLJoin" is the first rule invoked for the conversion process. It merges the nodes, which are in the "date" node structure, and creates a new node "dateTime" where the result is put.

---

2 See chapters: 2.2 and 3.3
3 See chapter: 2.3
4 See chapter: 3
The structure of the message after this step is listed below.

Message after "XMLJoin" rule:

```xml
<eventDate>
  <name>event</name>
  <date>
    <year>2010</year>
    <month>08</month>
    <day>15</day>
    <time>15:43:00</time>
  </date>
  <dateTime>2010-08-15
  15:43:00</dateTime>
</eventDate>
```

Script 3. Message obtained after executing the "XMLJoin" rule.

This conversion proves, that the order of invoking rules is important. The next steps are to remove the redundant "date" node (with all the contents) using the "Remove" rule and, finally, to modify the "dateTime" node value using the "Date" rule. The converted message is:

Converted message:

```xml
<eventDate>
  <name>event</name>
  <dateTime>2010-08-15
  20:43:00</dateTime>
</eventDate>
```

Script 4. Fully converted message for the conversion example with "XMLJoin", "Remove" and "Date" rules.

The last example demonstrates a conversion that is supposed to change the unit in which a speed value is transferred in a message. Conversion rules used are: "Mathematical" and "RegExpReplace". The "Mathematical" rule is used to calculate the value of speed from mph indication into kph. The "unit" node is converted with the "RegExpReplace" rule, which changes the string with the name of the unit from "mph" into "km/h". Messages are:

Raw message:

```xml
<speed>
  <value>125</value>
  <unit>mph</unit>
</speed>
```

Converted message:

```xml
<speed>
  <value>200</value>
  <unit>km/h</unit>
</speed>
```

Script 5. Conversion example with "Mathematical" and "RegExpReplace" rules.

5. Summary and Further Works

The most important conclusion is the fact that it is possible to implement a tool and mechanisms, which are compliant with definitions described in the introduction of this article, related with [1], [5] and [6]. However, it is important to consider formats and data compliance in cooperating systems through the prism of enabling successful exchange of this data and to both establish and present activities performed throughout the messaging process.

It was possible to construct and implement the Converter application in such a way, that the conversion rules are in fact components that are not inseparable from the rest of the system, but can be added, modified or removed. It is an important advantage, especially when the number of different formats of data that represent the same information is taken into consideration.

Further works, concentrated on increasing the range of functionalities of the Converter may be focused on implementing more conversion rules capable of dealing with different data formats. Those may be general rules, such as described "XMLSplit" and "XMLJoin", or more specific that, for example, extracts the gender from the PESEL number. It is more advantageous to implement less general rules rather than more specific rules, and achieve detailed conversions with a combination of general rules. The desired state of the application is to implement a number of rules that make it possible to convert any format of message into another.

Other improvements can be made within the configuration module of the application [2]. Those include: increasing the security of the application, improving the interface by implementing browser side validation. A number of exceptions resulting from configuration errors could be decreased.

Important modification that would improve the configuration module would be the possibility to aid the administrator by uploading and analysing the WSDL document (Web Service description), and generating example requests, responses and examples of converted messages.

5 PESEL is a governmental unique personal number dedicated for all citizens and residents in Poland

6 See chapter: 2.6 in [2]
6. Bibliography


---

Implemetacja wybranych mechanizmów interoperacyjności w podejściu SOA

G. Bliźniuk, R. Dawidczyk


Słowa kluczowe: warunki interoperacyjności, reguły konwersji, SOA.