Chapter 10

Using XML Transformation for Enterprise Architecture

Authors: F.S. de Boer, M.M. Bonsangue, J.F. Jacob, A. Stam, L. van der Torre

10.1 Introduction

In this paper, we investigate the use of XML transformation techniques in the context of Enterprise Architectures. We have split up this research question into two subquestions:

- How can we use XML transformation for the generation of views on an architecture (selection and visualization)?
- How can we use XML transformation for the analysis of architectures?

First, we will introduce the reader to the term Enterprise Architecture, the ArchiMate project, and XML. Second, we will give a short overview of our research methodology.

10.1.1 Enterprise Architectures

A definition of Architecture quoted many times is the following IEEE definition: "the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution" [Soc00]. Therefore, we can define Enterprise Architecture [MAS+03] as the Architecture of an enterprise. It covers principles, methods and models for the design and implementation of business processes, information systems, technical infrastructure and organizational structure.

Architectural information is usually contained in Architectural models. With these models and the information they incorporate, stakeholders within an organization are able to get more insight into the working of the organization, the impact of certain changes to it, and ways to improve its functioning.

Usually, we can distinguish between architectural descriptions that cover the as-is situation of an organization and descriptions that cover its intended to-be situation. According to IEEE, views are part of an architectural description. Views conform to viewpoints which are useful for certain stakeholders.

10.1.2 ArchiMate

Within the ArchiMate project, a language for Enterprise Architecture has been developed [JvBA+03]. This language can be used to model all architectural aspects of an organization. An overview of the concepts in the ArchiMate language is given in Figure 10.1.

As can be seen, the language contains concepts for several aspects of an organization. The individual concepts can be specialized for multiple domains, like the business domain, application domain or technical domain. Thus, a *Service* can be a business service, an application service or a technical service, for example.

10.1.3 XML

The Extensible Markup Language (XML) [XML] is a universal format for documents containing structured information so that they can be used over the Internet for web site content and several kinds of web services. It allows developers to easily describe and deliver rich, structured data from any application in a standard, consistent way.

Today, XML can be considered as the lingua franca in computer industry, increasing interoperability and extensibility of several applications. Terseness and human-understandability of XML documents is of minimal importance,

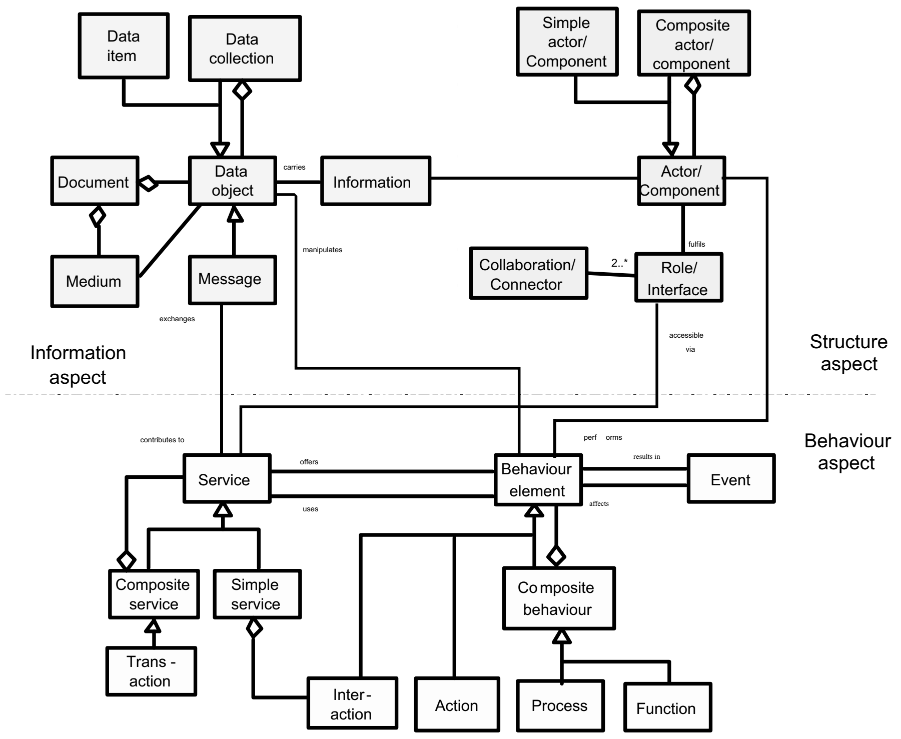

Figure 10.1: The ArchiMate metamodel

since XML documents are mostly created by applications for importing or exporting data.

10.1.4 Research methodology

First, we have developed a running example for verification of our ideas and techniques. During the development, we have refined the research questions as follows:

1. Given a set of architectural information described in a single XML document. How can we use XML transformation to select a subset of this information for a specific architectural view?

2. Is it possible to use XML transformation for *visualization*? I.e. is it possible to transform an XML document containing an architectural

description into another XML document containing visual information in terms of boxes, lines, etc.?

3. How can we use XML transformation to perform analyses on an architectural description? At first, we have chosen to specifically look at a specific form of *impact analysis*: given an entity within the architectural description which is considered to be modified or changed, which other entities in the description are possibly influenced by this change?

The second step consisted of developing an XML document containing the architectural information of the running example. As a basis for the architectural description, we have used an XML Schema containing the concepts from the ArchiMate metamodel.

Thereafter we developed an XML Schema for visualization information and built a *model viewer* which interprets this visualization information and shows this information on the screen. The aim was to keep the model viewer as "dumb" as possible, in order to make full use of XML transformation techniques for the actual visualization.

In the fourth step, we selected a transformation tool, namely the *Rule Markup Language*(RML), and built the transformation rules for selection, visualization and impact analysis.

10.1.5 Document layout

In Section 10.2 the Rule Markup Language (RML) is introduced. In Section 10.3 we will introduce the running example: ArchiSurance, a small insurance company which has the intention to phase out one of its core applications. In Section 10.4 we show transformation rules for selection and visualisation of architectural views, while in Section 10.5 we illustrate transformation techniques for analysis by means of performing a small impact analysis. In Section 10.6 we conclude.

10.2 The Rule Markup Language

RML stands for Rule Markup Language. It consists of a set of XML constructs that can be added to an existing XML vocabulary in order to define RML rules for that XML vocabulary. These rules can then be executed by RML tools to transform the input XML according to the rule definition.

The set of RML constructs is concise and shown in Table 2.1 with a short explanation of the constructs.

Rules defined in RML consist of an antecedent and a consequence. The antecedent defines a pattern and variables in the pattern. Without the RML constructs for variables this pattern would consist only of elements from the chosen XML vocabulary. The pattern in the antecedent is matched against the input XML. The variables specified with RML constructs are much like the wild-card patterns like `*` and `+` and `?` as used in well known tools like `grep`, but the RML variables also have a *name* that is used to remember the matching input. Things that can be stored in RML variables are element names, element attributes, whole elements (including the children), and lists of elements.

If the matching of the pattern in the antecedent succeeds then the variables are bound to parts of the input XML and they can be used in the consequence of an RML rule to produce output XML. When one of the RML tools applies a rule to an input then by default the part of the input that matched the antecedent is replaced by the output defined in the consequence of the rule; the input surrounding the matched part is kept intact.

RML does not define, need, or use another language, it only adds a few constructs to the XML vocabulary used, like the wild-card pattern matching. RML was designed to make the definition of executable XML transformations also possible for other stakeholders than programmers. This is of particular relevance when transformations capture for instance business rules. In this way it is possible to extend the original model in the problem domain XML vocabulary with semantics for that language. Similarly, it is also possible to define rules for constraining the models with RML.

10.2.1 Comparison with other techniques

XSLT is a W3C language for transforming XML documents into other XML documents.

The *RuleML* [com] community is working on a standard for rule-based XML transformations. Their approach differs from the RML approach: RuleML superimposes a special XML vocabulary for rules. This makes the RuleML approach complex and thus difficult to use in certain cases.

The *Relational Meta-Language* [Pet94] is a language that is also called RML, but intended for compiler generation, which is much more roundabout and certainly not usable for rapid application development like with RML in this paper.

Another recent approach is *fxt* [BS02], which, like RML, defines an XML syntax for transformation rules. Important drawbacks of fxt are that it is rather limited in its default possibilities and relies on hooks to the SML programming language for more elaborate transformations.

Other popular academic research topics that could potentially be useful for rule based XML transformations are term rewriting systems and systems based on graph grammars for graph reduction. However, using these tools for XML transformations is a contrived and roundabout way of doing things. To use these kind of systems, there has to be first a translation from the problem XML to the special-purpose data structure of the system. And only then, in the tool–specific format, the semantics is defined. But the techniques used in these systems are interesting, especially for very complex or hard transformations, and it looks worthwhile to see how essential concepts of these techniques can be incorporated in RML in the future.

Compared with the related work mentioned above, a distinguishing feature of the RML approach is that RML *re-uses* the language of the problem itself for matching patterns and generating output. This leads in a natural way to a much more usable and clearly defined set of rule based transformation definitions, and an accompanying set of tools that is being used successfully in practice.

10.3 Running Example

Throughout this paper, we will use a running example to illustrate our ideas. The architectural description of this example can be found in the models in Figures 10.2, 10.3, 10.4 and 10.5.

A small company, named ArchiSurance, sells insurance products to customers. Figure 10.2 contains a Business View of the company. Two *roles* are involved, namely the insurance company and the customer, which work together in two *collaborations*, namely negotiation, i.e. the set of activities performed in order to come to an appropriate insurance for a customer by discussion and consultation, and contracting, i.e. the set of activities performed in order to register a new customer and let it sign a contract for an insurance policy.

Within Figure 10.3, the business *process* for selling an insurance product to a customer is shown, together with the *roles* and/or *collaborations* that

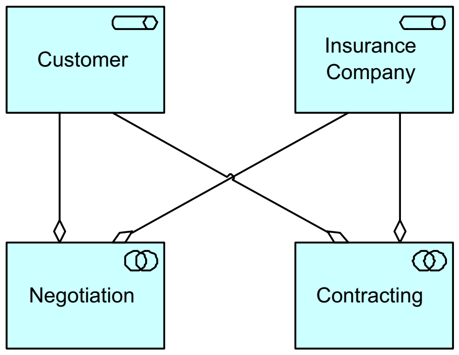

Figure 10.2: a Business View of ArchiSurance

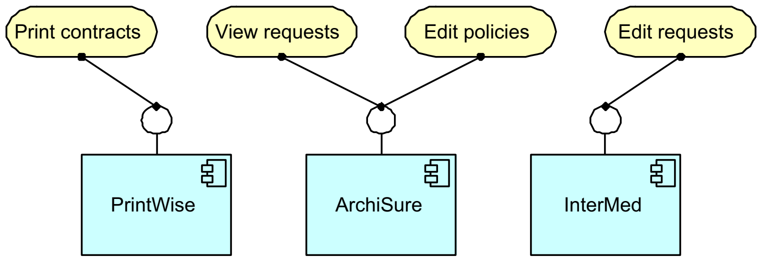

Figure 10.3: a Process View of ArchiSurance

are involved in executing the individual steps within the process.

Figure 10.4 shows the software products (*components*) that are used within the ArchiSurance company and the *services* they offer. *ArchiSure* is a custom-made software application for the administration of insurance products, customers and premium collecting. *PrintWise* is a out-of-the-box tool for official document layout and printing. *Intermed* is an old application, originally meant for intermediaries to have the possibility to enter formal requests for insurance products for their customers. The application is now used by employees of the insurance company, since no intermediaries are involved in selling insurance products anymore. Actually, the company would like to phase out this application.

In Figure 10.5, the *process* for selling products is shown again, together with the *services* that are used within each step.

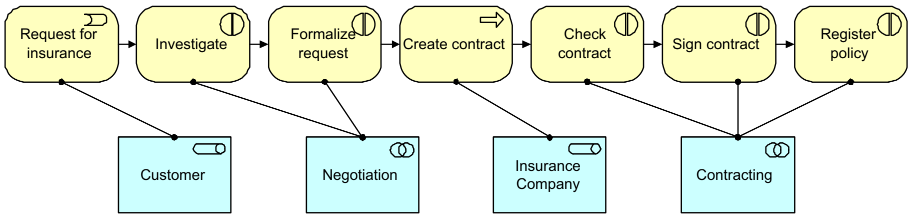

Figure 10.4: an Application View of ArchiSurance

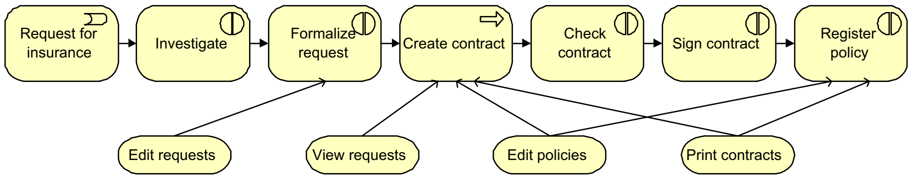

Figure 10.5: a Service View of ArchiSurance

10.3.1 An XML description of the example

Though the four views are depicted separately, they are clearly related to each other via the concepts they contain. In this small example, it is possible to imagine the *big picture* in which all ArchiSurance information is contained.

Within the ArchiMate project, a XML Schema has been developed which can be used for storage or exchange of architectural descriptions. Based on this Schema, we have created an XML document that contains all information about ArchiSurance. For illustration, a fragment of this XML document is shown below. It contains the XML equivalent of Figure 10.2.

```
<role id="002" name="Customer"/>
<role id="003" name="Insurance Company"/>
<collaboration id="004" name="Negotiation"/>
<collaboration id="006" name="Contracting"/>
<composition id="035" name="composition">
    <from href="004"/>
    <to href="002"/>
</composition>
<composition id="036" name="composition">
    <from href="004"/>
    <to href="003"/>
</composition>
<composition id="041" name="composition">
    <from href="006"/>
    <to href="002"/>
</composition>
<composition id="042" name="composition">
```

```
    <from href="006"/>
    <to href="003"/>
</composition>
```

10.4 Selection and Visualisation

The initial XML document contains the concepts and relations based on the ArchiMate metamodel. It does not contain information about which concepts are relevant for which views, nor does it describe how to visualize the concepts. We can use RML rules for both tasks, as will be illustrated in the following sections.

10.4.1 Selection

Within a single view, usually a selection of the entire set of concepts is made. For example, the Business View in our example only contains *roles* and *collaborations*. For this purpose, RML rules have to filter out all unnecessary information from the XML document and thus create a new document that only contains those concepts and relations that are relevant for the view.

We have created the following recipe for selection:

1. add a specific *selection* element to the XML document which is going to contain the selected concepts;

2. iterate over the document and move all relevant concepts into the specific *selection* element;

3. iterate over the document and move all relevant relations into the specific *selection* element;

4. remove all relations within the *selection* element that have one "dangling" end, i.e. that are related at one side to a concept that does not belong to the selection;

5. remove all elements outside the *selection* element.

Note that the step for removing relations with one "dangling" end out of the selection is necessary, because one relation type (e.g. association) can be defined between several different concept types.

The following RML rule illustrates the way all instances of a specific concept are included in the selection:

```
<div class="rule">
    <div class="antecedent">
        <model>
            <rml-list name="list1"/>
            <collaboration rml-others="other">
                <rml-list name="childs"/>
            </collaboration>
            <rml-list name="list2"/>
            <selection>
                <rml-list name="selection"/>
            </selection>
            <rml-list name="list3"/>
        </model>
    </div>
    <div class="consequence">
        <model>
            <rml-use name="list1"/>
            <rml-use name="list2"/>
            <rml-use name="list3"/>
            <selection>
                <rml-use name="selection"/>
                <collaboration rml-others="other">
                    <rml-use name="childs"/>
                </collaboration>
            </selection>
        </model>
    </div>
</div>
```

10.4.2 Visualization

As is described in the introduction, we wanted to keep the model viewer as "dumb" as possible, in order to illustrate the way in which XML transformations can be used for creating several visualizations for a single XML document.

For this purpose, we have made a specific XML schema which can be interpreted by the model viewer without having to know anything about the ArchiMate language. The following XML fragment illustrates this language.

```
<container height="80" id="014" type="interaction" width="100" >
      <box color="khaki1" height="80" type="round" width="100" x="0" y="0" z="0" />
      <label fieldname="name" halign="center" text="register policy" x="50" y="40" z="1" />
      <icon height="15" type="splitcircle" width="15" x="75" y="10" z="1" />
</container>
```

```
<container height="80" id="013" type="interaction" width="100" >
      <box color="khaki1" height="80" type="round" width="100" x="0" y="0" z="0" />
      <label fieldname="name" halign="center" text="sign contract" x="50" y="40" z="1" />
      <icon height="15" type="splitcircle" width="15" x="75" y="10" z="1" />
</container>

<arrow from="013" id="020" to="014" type="triggering" >
      <line type="solid" width="1" z="0" />
      <headarrowtip size="10" type="filledarrow" z="1" />
</arrow>
```

The intermediate visualization language has two main constructs: containers and arrows.

Containers are rectangular areas in which several visual elements can be placed. The exact location of those visual elements can be defined relative to the size and position of the container. Each container has a unique identifier which can be used to refer to the original elements in the architectural description.

Arrows are linear directed elements. They have a head and a tail, which both have to be connected to containers (via their identifiers). They also have unique identifiers themselves.

In the example above, two containers and one arrow are defined. In Figure 10.6 the output of the interpretation of this XML fragment by the model viewer is shown. As can be seen in the XML fragment, some visual elements, like "split circle", are built into the model viewer. This has mainly been done for reasons of efficiency.

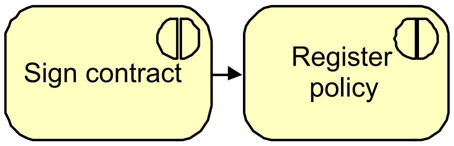

Figure 10.6: Example of the visualization technique used

For the transformation of the original XML model to the visualization information, we have created scripts that transform each concept into its corresponding visualization. An example is given below.

```
<div class="rule">
    <div class="antecedent">
        <interaction id="rml-id" name="rml-name" color="rml-color"/>
    </div>
    <div class="consequence">
        <container id="rml-id" type="interaction" width="100" height="80" color="rml-color">
            <box x="0" y="0" z="0" width="100" height="80" color="khaki1" type="round"/>
            <label x="50" y="40" z="1" halign="center" text="rml-name" fieldname="name"/>
            <icon x="75" y="10" z="1" width="15" height="15" type="splitcircle"/>
        </container>
    </div>
</div>
```

This example rule transforms an *interaction* concept into a visual representation.

The technique presented here is quite powerful: from the same architectural description, it is possible to define different visualization styles, like ArchiMate, UML, etc. In the context of enterprise architectures, this is especially useful since architects often want to have their own style of visualization (for cultural and communication reasons within organizations), without having to conform to a standard defined outside the organization.

10.5 Analysis

Next to selection and visualisation, we have investigated ways to use XML transformation for analysis of enterprise architectures. Our aim was to create a technique for impact analysis, i.e. given an entity within the architectural description which is considered to change, which other entities are possibly influenced by this change?

We have created the following recipe for this analysis:

1. add a specific *selection* element to the XML document which is going to contain the concepts that are considered to be possibly influenced;

2. add a special attribute to the element describing the entity under consideration, which can be used for for example visualisation (in order to make it have a red color, for example);

3. make the element describing the entity under consideration a child of the *selection* element;

4. iterate over all relations included in the analysis and, if appropriate, add a special attribute to them and make them a child of the *selection* element;

5. iterate over all concepts and, if appropriate, add a special attribute to them and make them a child of the *selection* element;

6. repeat the previous two steps until the output is stable;

7. remove the *selection* element, so that we have one list of concepts and relations, of which some concepts have a special attribute which indicates that the change possibly has impact on them.

An example of the output of the analysis is given below. The component "InterMed" is considered to change. It has two new attributes. The *selected* attribute indicates that it belongs to the entities which are possibly influenced by the change, while the *special* attribute indicates that this entity is the unique entity considered to change. The remaining elements describe concepts and relations that are all selected, because they are directly or indirectly related to the "InterMed" component.

```
<component id="082" name="InterMed" selected="yes" special="yes"/>

<composition id="104" name="composition" selected="yes" >
    <from href="082" />
    <to href="094" />
</composition>

<interface id="094" name="Interface" selected="yes" />

<assignment id="112" name="assignment" selected="yes" >
    <from href="094" />
    <to href="090" />
</assignment>

<service id="090" name="edit requests"  selected="yes" />
```

Within Figure 10.7 and Figure 10.8, the output of the model viewer is given for two views. The change of color is done by the visualisation scripts, based on the attributes added during the analysis.

10.6 Summary

Our conclusions about each research question are as follows:

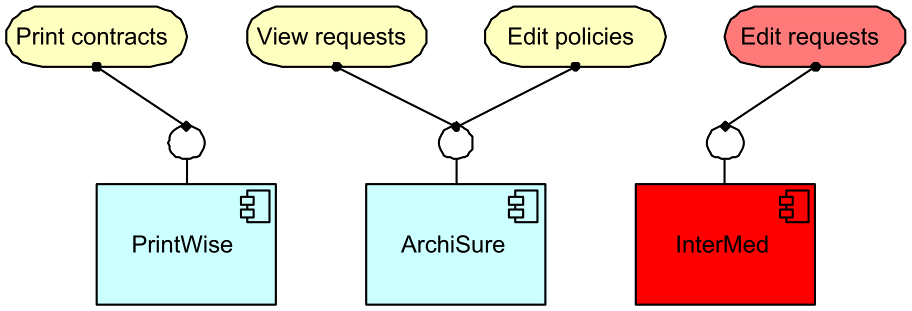

Figure 10.7: The Application View with a selected InterMed application

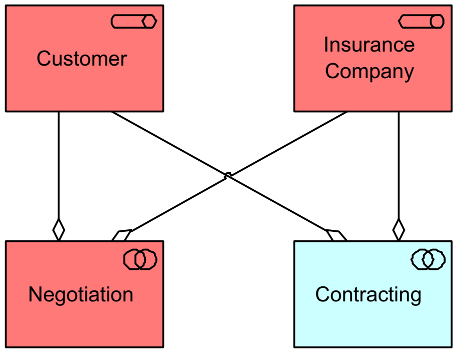

Figure 10.8: The resulting Business View after the impact analysis

10.6.1 Question 1

How can we use XML transformation to select a subset of a set of architectural information for a specific architectural view? In Section 10.4 we have illustrated a way to filter out certain concepts and create a new XML document containing a selection out of the original document.

10.6.2 Question 2

Is it possible to use XML transformation for *visualization*? In Section 10.4 we have shortly described an "intermediate" language for visualization information and illustrated how we can transform an ArchiMate XML document into a Visualization XML document.

10.6.3 Question 3

How can we use XML transformation to perform a specific form of impact analysis on an architectural description? A technique to perform this specific analysis is described in Section 10.5.

10.6.4 Conclusions

The research reported on in this paper shows promising results. The use of XML and transformation techniques for it has several benefits: XML is well-known, the transformation techniques are generic and tools for it are improving rapidly. Transformation rules are well understandable and can be adapted quickly for specific needs or purposes.

The use of XML transformations for visualization proves to be specifically interesting: in many cases, enterprise architecture tools have a fixed way of visualizing information, which hinders architects in representing information in the way they want to. By separating the "visualization step" from the "viewer", architects gain much often demanded flexibility.

Acknowledgments This paper results from the ArchiMate[1] project, a research initiative that aims to provide concepts and techniques to support architects in the visualization and analysis of integrated architectures. The ArchiMate consortium consists of ABN AMRO, Stichting Pensioenfonds ABP, the Dutch Tax and Customs Administration, Ordina, Telematica Institute, CWI, University of Nijmegen, and LIACS.

[1](`http://archimate.telin.nl`)