Using XML Transformations for Enterprise Architectures

A. Stam[1⋆], J. Jacob[2], F.S. de Boer[1,2],
M.M. Bonsangue[1⋆⋆], and L.van der Torre[3]

[1] *LIACS, Leiden University, The Netherlands*
[2] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
[3] *University of Luxembourg, Luxembourg*

Abstract. In this paper we report on the use of XML transformations in the context of Enterprise Architectures. We show that XML transformation techniques can be applied to visualize and analyze enterprise architectures in a formal way. We propose transformational techniques to extract views from an XML document containing architectural information and indicate how to perform a specific form of impact analysis on this information. The transformations are formally expressed with the language RML, a compact yet powerful transformation language developed at CWI, which obtains its power from regular expressions defined on XML documents. We discuss a tool that has been built on top of it to visualize the results of the transformations and illustrate the advantages of our approach: the genericity of XML, the application of a single technique (namely XML transformations) for various tasks, and the benefits of having a model viewer which is in complete ignorance of the architectural language used.

1 Introduction

In this paper, we investigate the use of XML transformation techniques in the context of enterprise architectures, a field in which much work is currently often done by hand, like analysis and the time-consuming creation of views for different stakeholders. The absence of formal methods and techniques hinders the quality of analyses, the consistency between different views and the agility with respect to changes in the architecture. Moreover, architects often want to have their own style of visualization (for cultural and communication reasons within organizations), without having to conform to a certain standard. Many architectural tools depend on a specific architectural language and often only support one visualization style or visual language for it.

In order to overcome these problems, we propose a way of working that introduces flexibility in the architectural language and visualizations used, while

⋆ Corresponding author. Email: `astam@liacs.nl`

⋆⋆ The research of Dr. Bonsangue has been made possible by a fellowship of the Royal Netherlands Academy of Arts and Sciences

adding formality to both the actual creation of views and the analysis of architectures. Our approach is to specify architectural information as XML documents and use XML transformation techniques for creating views and performing analyses on an architecture. Moreover, we propose to use a visualization tool which is independent of a specific architectural or visual language.

In particular, within this paper, we focus on three subquestions:

- Given a set of architectural information described in a single XML document. How can we use XML transformations to select a subset of this information for a specific architectural view?
- How can we transform an XML document containing architectural information into another XML document containing visual information in terms of boxes, lines, etc.? How can we build a *model viewer* which interprets this visual information without having to know anything about the architectural language used?
- How can we use XML transformations to perform analyses on an architectural description? We have chosen to focus 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?

We have carried out the following activities: First, we developed a running example for verification of our ideas and techniques. Then, we developed an XML document containing the architectural information of the running example. We used the ArchiMate language and its corresponding XML Schema, containing the concepts from the ArchiMate metamodel. After this, we developed an XML Schema for visualization information and built a *model viewer*, which is entirely ignorant of the ArchiMate language, and only interprets visualization information and shows this information on the screen. We only used XML transformation techniques for the actual visualization. Then, we selected an easy-to-use transformation tool, namely the *Rule Markup Language* (RML), and built the transformation rules for selection, visualization and impact analysis.

The layout of this document is as follows: In Section 2, we introduce the reader to enterprise architecture and ArchiMate. In Section 3, XML and the Rule Markup Language (RML) are explained. In Section 4 we introduce the running example: ArchiSurance, a small insurance company which has the intention to phase out one of its core applications. In Section 5 we show transformation rules for the creation (selection and visualization) of architectural views, while in Section 6 we illustrate transformation techniques for analysis by means of performing a small impact analysis. In Section 7 we conclude.

2 Enterprise Architecture

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" [11]. Therefore, we can define enterprise architecture [8] as the fundamental organization of an enterprise embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution. 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 presented via (architectural) views. With these views and the information they contain, stakeholders within an organization are able to gain 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 information in relation to the as-is situation of an organization and information in relation to its intended to-be situation. According to IEEE, views conform to viewpoints that address concerns of stakeholders.

2.1 ArchiMate

Within the ArchiMate project [10], a language for enterprise architecture has been developed [7]. This language can be used to model all architectural aspects of an organization. A metamodel containing the concepts in the ArchiMate language is given in Figure 1.

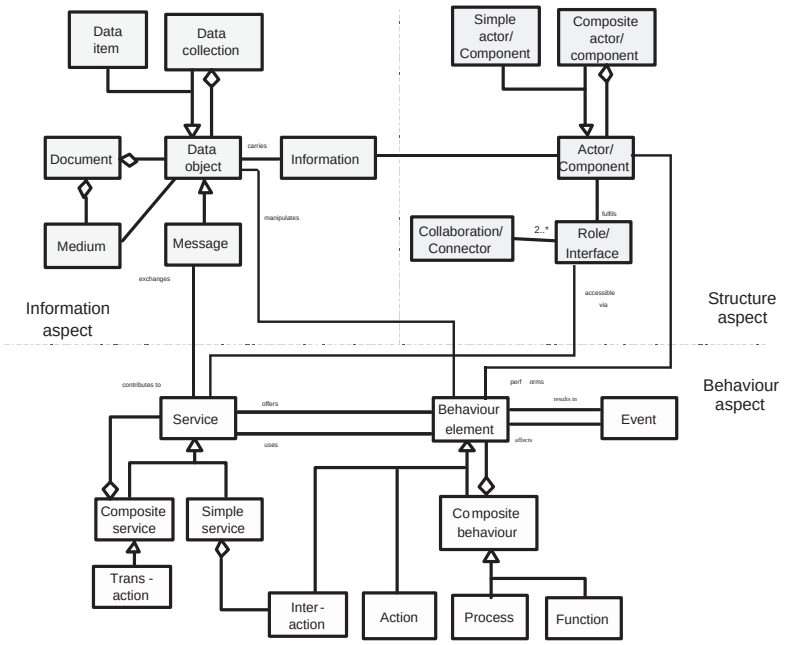

Fig. 1. The ArchiMate metamodel

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.

3 XML and the Rule Markup Language

In this section we introduce XML and present the Rule Markup Language (RML), which is the XML transformation language we have used for creating views and performing analyses of enterprise architectures.

3.1 XML

The eXtensible Markup Language (XML) [4] 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, since XML documents are mostly created by applications for importing or exporting data.

3.2 The Rule Markup Language

In our study, we have used the Rule Markup Language (RML) for the specification of XML transformations. From a technical point of view, other transformation tools like the popular XSLT, RuleML or QVT could have been used as well for the applications described in this paper. However, RML was originally designed to make the definition of executable XML transformations possible for various stakeholders other than programmers, including architects. Thus, instead of creating views and performing analyses by hand, architects can formally specify transformations of the contents of their own XML documents and apply these transformations in order to select, visualize and analyze architectural information.

The Rule Markup Language (RML) [6] consists of a set of XML constructs which can be added to an existing XML vocabulary in order to define RML rules for that XML vocabulary. Specific RML tools can execute these rules, to transform the input XML according to the rule definition. The set of RML constructs is shown in Table 2 with a short explanation of each construct.

Each RML rule consists 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

Elements that designate rules				
`div`	`class="rule"`			
`div`	`class="antecedent" context="yes"`			
`div`	`class="consequence"`			
element	attribute	A	C	meaning
Elements that match elements or lists of elements				
`rml-tree`	`name="X"`	*		Bind 1 element (and children) at this position to RML variable X.
`rml-list`	`name="X"`	*		Bind a sequence of elements (and their children) to X.
`rml-use`	`name="X"`		*	Output the contents of RML variable X at this position.
Matching element names or attribute values				
`rml-X`	...	*		Bind element name to RML variable X.
`rml-X`	...		*	Use variable X as element name.
...	`...="rml-X"`	*		Bind attribute value to X.
...	`...="rml-X"`		*	Use X as attribute value.
...	`rml-others="X"`	*		Bind *all* attributes that are not already bound to X.
...	`rml-others="X"`		*	Use X to output attributes.
...	`rml-type="or"`	*		If this element does not match, try the next one with rml-type="or".
Elements that add constraints				
`rml-if`	`child="X"`	*		Match if X is already bound to 1 element, and occurs *somewhere* in the current sequence of elements.
`rml-if`	`nochild="X"`	*		Match if X does not occur in the current sequence.
`rml-if`	`last="true"`	*		Match if the younger sibling of this element is the last in the current sequence.
A `*` in the `A` column means the construct can appear in a rule antecedent. A `*` in the `C` column is for the consequence.				

Fig. 2. An overview of the RML constructs

vocabulary. The pattern in the antecedent is matched against the input XML. RML constructs can contain variables, which are specified in a familiar way, by using wild-card patterns like `*` and `+` and `?`. 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, 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 a rule is applied to the input, one of the RML tools will by default replace the part of the input that matched

the antecedent, by the output defined in the consequence of the rule; the input surrounding the matched part remains unchanged.

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.

3.3 Comparison with other techniques

XSLT[5] is a W3C language for transforming XML documents into other XML documents.

The *RuleML* [3] 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, which makes it less suitable to use in combination with another XML vocabulary.

The *Relational Meta-Language* [9] is a language that is also called RML, but intended for compiler generation, which is definitely not suited for rapid application development like with RML in this paper.

Another recent approach is *fxt* [1], 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 arbitrary way of doing things. To exploit these kind of systems successfully in a context where a specific XML vocabulary is already in use, there has to be first a translation from this specific 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 transformations.

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.

4 Running Example

Throughout this paper, we use a running example to illustrate our ideas. Though our example is small compared to a real-life enterprise architecture, its sole purpose in this paper is to illustrate the use of XML transformation techniques, for which it contains sufficient complexity. Analyzing the performance of RML when applied to larger architectural models is a topic for future research.

A small company, named ArchiSurance, sells insurance products to customers. Figure 3 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, which is 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.

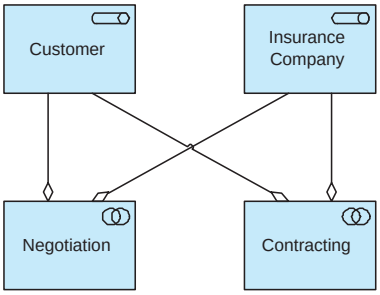

Fig. 3. a Business View of ArchiSurance

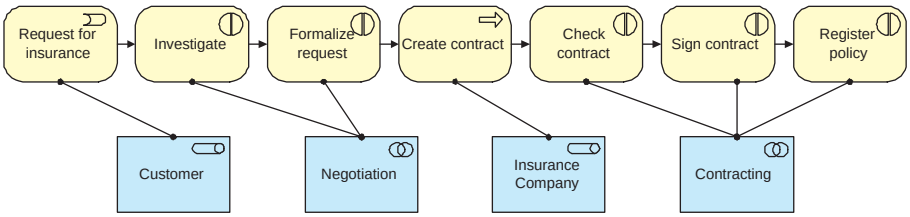

Fig. 4. a Process View of ArchiSurance

Within Figure 4, the business process for selling an insurance product to a customer is shown in a Process View, together with the roles and collaborations that are involved in executing the individual steps within the process.

Figure 5, an Application View, 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 6, a Service View is presented: the process for selling products is shown again, now together with the services that are used within each step.

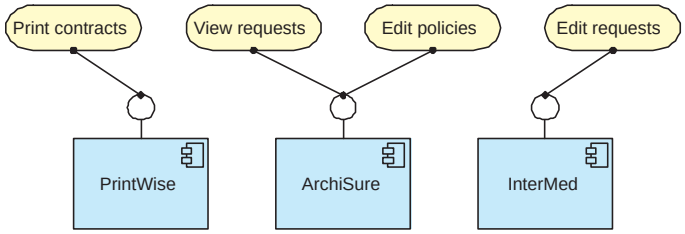

Fig. 5. an Application View of ArchiSurance

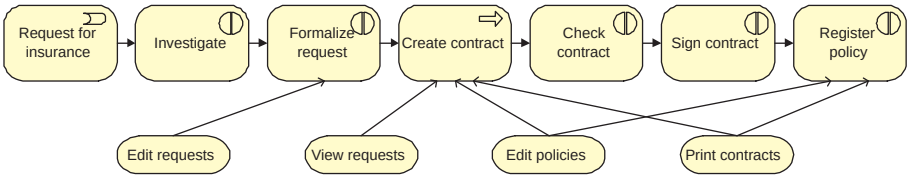

Fig. 6. a Service View of ArchiSurance

4.1 An XML description of the example

Although the four views (Business View, Process View, Application View, Service View) 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, an XML Schema has been developed which can be used for storage or exchange of architectural information. Based on this Schema, we have created a single XML document that contains all information about ArchiSurance. The use of a single document is no problem in this small case, but when architectural models are larger, a single document could be difficult to maintain. Investigating the use of more XML documents for a single architectural model is a topic for future research.

For illustration, a fragment of the XML document is shown below. It contains the XML equivalent of Figure 3.

```
<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>
```

5 Selection and Visualization

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 to formally define selection and visualization schemes, as will be illustrated in the following sections.

5.1 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* and abstracts from all other related concepts. 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. By writing down the RML rules, we ensure that we do not omit certain concepts, which often happens when this work is being done by hand.

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. Nevertheless, this recipe is fairly easy to understand and can be specified by architects themselves for any kind of selection they want to make.

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>
```

5.2 Visualization

One of the research questions is about the creation of a "dumb" model viewer, i.e., it is ignorant of any architectural language. By this, we can illustrate the way in which XML transformations can be used for creating several visualizations for a single XML document.

For this purpose, we made a specific XML schema which can be interpreted by a 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 7 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.

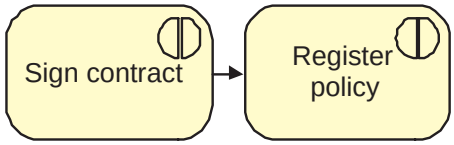

Fig. 7. Example of the visualization technique used

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

```
<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>
```

The technique presented here is quite powerful yet easy to understand: from the same architectural description, it is possible to define different visualization styles, like ArchiMate (which is used in the running example), UML[2], 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. They can do this in a formal way by capturing their visualizations in XML transformation rules.

6 Analysis

Next to selection and visualization, we investigated ways to use XML transformations for analysis of enterprise architectures. Our aim is 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, e.g., 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 8 and Figure 9, the output of the model viewer is given for two views. The change of color is done by the visualization rules, based on the attributes added during the analysis.

By performing impact analysis in the way presented above, we treat the set of architectural concepts as a graph with nodes and edges and define that a node A has impact on another node B when A is selected and there is a relation from

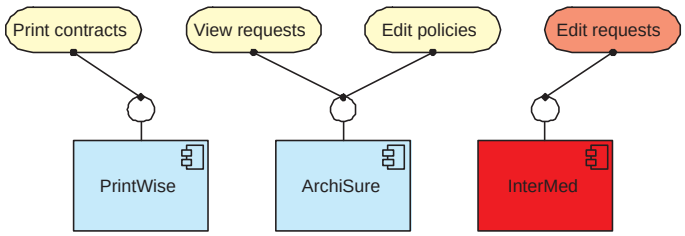

Fig. 8. The Application View with a selected InterMed application

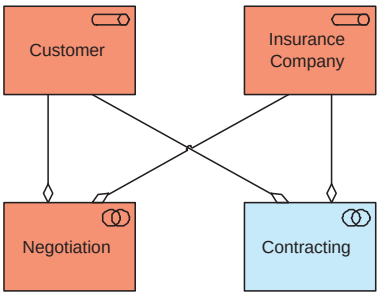

Fig. 9. The resulting Business View after the impact analysis

A to B. But we could easily change our interpretation of the term "impact", for example by introducing several kinds of impact (e.g., extend, replace, delete) or treating different kind of relations between nodes in a different way. In all cases, our interpretation of the term "impact" is formally defined in the XML transformation rules.

7 Conclusions

The research reported on in this paper shows promising results. The use of XML transformation techniques for selection, visualization and analysis of enterprise architectures 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 application of XML transformation techniques for enterprise architectures is a good example of the way in which formal techniques, in this case, formally defined transformation rules, can be applied effectively by end users in the context of enterprise architecture, i.e. the enterprise architects themselves.

Based on the reported investigations, our answers to the research questions set out earlier are the following:

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

This can be done via transformation rules which filter out certain concepts and create a new XML document containing a selection out of the original document. These transformation rules are easy to understand and can be defined by the architects themselves to create their own selections.

Question 2 How can we transform an XML document containing architectural information into another XML document containing visual information in terms of boxes, lines, etc.? How can we build a *model viewer* which interprets this visual information without having to know anything about the architectural language used?

What is needed for this, is a separate "intermediate" language for visualization information. Via XML transformations, we can transform an ArchiMate XML document into an XML document containing only visual elements. The latter document can then be interpreted by a model viewer which only has to know how to interpret the visual elements in this document. By separating the visualization step from the viewer, architects gain much often demanded flexibility: they are able to create their own visualizations in a formal way, i.e. defined in transformation rules.

Question 3 How can we use XML transformations to perform our specific form of impact analysis?

We can do this analysis by iterative selection of the elements which have a relation with the "element to be changed". By including or excluding certain relation types, architects gain insight in the mutual dependencies between the entities within an architecture.

Acknowledgments This paper is based on research in the context of the ArchiMate[4] 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.

References

1. A. Berlea and H. Seidl. fxt a transformation language for XML documents. *Journal of Computing and Information Technology*, 10(1):19–35, 2002.
2. G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.
3. The Rule Markup Initiative community. URL: `http://www.dfki.uni-kl.de/-ruleml/`.
4. World Wide Web Consortium. Extensible markup language (XML). URL: `http://www.w3.org/XML/`.
5. World Wide Web Consortium. XSL transformations (XSLT) version 1.0, W3C recommendation, November 1999. URL: `http://www.w3c.org/TR/xslt`.

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

6. J. Jacob. The rule markup language: a tutorial. URL: `http://-homepages.cwi.nl/~jacob/rml`.
7. H. Jonkers, R. van Buuren, F. Arbab, F.S. de Boer, M.M. Bonsangue, H. Bosma, H. ter Doest, L. Groenewegen, J. Guillen-Scholten, S. Hoppenbrouwers, M. Jacob, W. Janssen, M. Lankhorst, D. van Leeuwen, E. Proper, A. Stam, L. van der Torre, and G. Veldhuijzen van Zanten. Towards a language for coherent enterprise architecture description. In M. Steen and B.R. Bryant, editors, *Proceedings 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003)*, pages 28–39. IEEE Computer Society Press, 2003.
8. James McGovern, Scott W. Ambler, Michael E. Stevens, James Linn, Vikas Sharan, and Elias K. Jo. *A Practical Guide to Enterprise Architecture*. Prentice Hall PTR, 2003.
9. M. Pettersson. RML - a new language and implementation for natural semantics. In M. Hermenegildo and J. Penjam, editors, *Proceedings of the 6th International Symposium on Programming Language Implementation and Logic Programming, PLILP*, volume 884 of *LNCS*, pages 117–131. Springer-Verlag, 1994.
10. The Archimate Project. URL: `http://www.telin.nl/NetworkedBusiness/-Archimate/ENindex.htm`.
11. IEEE Computer Society. IEEE std 1471-2000: IEEE recommended practice for architectural description of software-intensive systems, Oct. 9, 2000.