XML-based Knowledge Translation Methods for XTT-based Expert Systems*

Grzegorz J. Nalepa[1] **and Igor Wojnicki**[1]

Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
<*gjn@agh.edu.pl*>, <*wojnicki@agh.edu.pl*>

Abstract *The paper discusses a concept of representing a rule-based knowledge base of an expert system with a hybrid method combining decision trees and tables. The XTT method provides means for logical formulation of a hierarchical rulebase with explicit inference control. It also has a well-defined visual representation, which is useful in the design process. On the machine readable level the XTT knowledge base is encoded in a custom XML format. The paper introduces new methods of automatic XSLT translation of this format into the SVG format. The same method could be used to translate XTT to Prolog code that provides an executable prototype. XSLT is a purely declarative XML-based language for defining document translations. SVG is a upcoming standard for describing vector graphics using XML. Methods presented in the paper provide a purely declarative platform-free translation of XTT knowledge. The translations improve practical design of complex rulebases. They can be implemented in several tool-agnostic methods presented in the paper.*

1. Introduction There are two aspects of Knowledge Engineering which are omnipresent in contemporary design of Knowledge Based Systems (KBS). The first one is the knowledge base design process, the second one is knowledge base representation. The knowledge base design process should take into account expressing knowledge, both facts and rules, gradual Knowledge Base improvement and refinement, validation and verification. The process should be based on a methodology which tackles the above and should also be supported by technologies and tools.

As a result of the design process a knowledge base is to be formulated. There should be a formalism describing it. This formalism should support all of the design stages.

These days, markup languages used in web technologies play an important role in the data encoding formats. XML-based encodings provide standard parsing tools and processors, as well as interoperability between different environments. These are also applied in the artificial intelligence applications used in the web, especially reasoning engines and so called business rules applications.

Research presented in this paper concerns building transparent knowledge models, that provide clear visual representation. The logical structure of the model can then be captured and encoded with the use of an XML-based format. The above aspects are well defined within the XTT (see Sect. 2,3) methodology for designing rule-based systems [1]. XTT stands for *eXtended Tabular Trees*, and provides a structured knowledge representation for decision rules. The actual encoding issues are given in Sects. 4 and 5. The XML-based logical model can then be translated to a visual one as presented in Sect. 6. The directions for future work are given in the final section.

2. Multi-Aspect Knowledge Representation In the field of intelligent systems finding an effective knowledge representation method is non-trivial. Not only the method must be suitable for

*The paper is supported by the *HEKATE* Project funded from 2007–2009 resources for science as a research project.

the problem domain, but it should also allow for an efficient design process. In order to provide possibility of analysis, the method should posses strong formal foundations. To allow gradual refinement of a once designed knowledge base, as well as integration with other systems, possibly developed in the future, it has to support the knowledge interoperability.

To fulfill these requirements in the field of the rule-based expert systems (RBS), the XTT (*eXtended Tabular Trees*) [4] has been proposed. It is an integrated method for the design, analysis, and implementation of RBS. The method is based on the idea of a hierarchical logical design of the rule base, using a structured, representation. The method aims at combining extended attributive decision tables, with explicit inference control, that allows for building decision-tree-like meta control structure. On the high level, this logical design can be supported by the conceptual design that allows for specifying functional dependencies between systems attributes. The ARD (*Attribute Relationship Diagrams*) method [3] serves this purpose. It allows for visualizing the gradual specification of the functional relations of attributes. The process specifies attributes on different levels of abstraction. Formal system analysis is possible thanks to an automatic transformation of the knowledge base, into a corresponding representation in Prolog [5].

One of the principal ideas around the XTT method, is to provide three aspect of the knowledge representation in every design, these are:

1. logical interpretation,
2. visual representation,
3. machine-readable encoding.

The *logical* interpretation of the XTT knowledge based is possible thanks to its formal definition in an extended calculus, that makes use of restricted predicate calculus with an extended attribute calculus, see [4, 2]. XTT provides *visual* support of the design, in the conceptual and logical design phases, described in Sect. 3. In order to provide design tool independence, as well as knowledge base interchange possibility, the logical XTT model can be *encoded* on the machine-readable level using an XML-based formats, described in Sect. 5. These aspects are summarized in the Table 1.

	logical	visual	encoding
ARD	relations	graphs	ARDML
XTT	rules	tables,trees	XTTML

Table 1. Multi aspect representation.

3. Visual Rule Representation

The integrated design process centered around XTT involves three main phases: conceptual, logical, and physical. In this process the two custom knowledge representations are used: *ARD* in the conceptual phase, where gradual refinement of the system attributes relationships, takes place; and *XTT* where rules which express system behavior are specified. The XTT knowledge base is then described in a *Prolog*-based representation. This process is constantly supported on the *visual level*, which is mainly important at two first stages. The visual representation should be provided by appropriate visual editors supporting the design.

The design begins with specifying system attribute relationships with ARD diagrams. An ARD diagram defines a *functional dependency* between two groups of attributes, as seen in Fig. 1. In this figure both the old format [3, 2], as well as the new one can be observed. In both cases the interpretation is as follows: in order to determine the values of $Y1 - Ym$ the values of $X1 - Xm$ have to be known. The ARD model is a hierarchical model, with the highest level being most abstract, and the lower levels being more specific. New, lower levels are created be the use of diagram *splits*: vertical and horizontal, as well as attribute substitution. On the higher levels the so-called conceptual attributes can be used. These are generalized, not-yet-specified attributes, that are being refined, and specified, in the subsequent levels. A part of an example ARD design is presented in Fig. 2.

The last level of the ARD diagrams serves as the prototype for XTT table headers. A single XTT table is presented in Fig. 3. Table rows correspond to decision rules, having the same attributes, thus operating in the same *context*. This

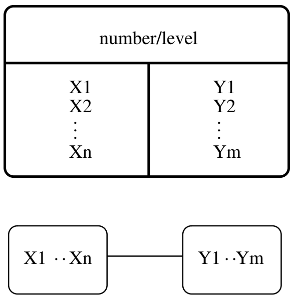


Figure 1. A generic ARD diagram.

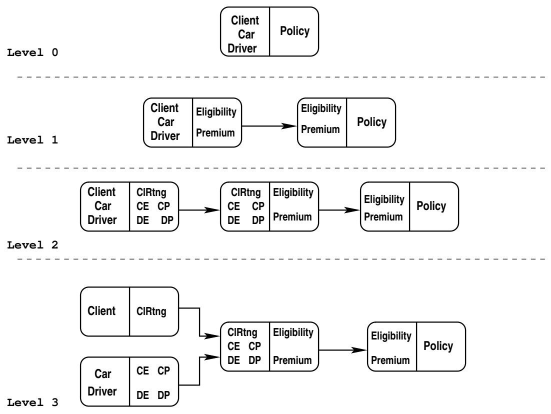


Figure 2. An example ARD design.

is an extended rule syntax, including inference control at the rule level, and dynamic memory operations (assert +, and retract −) in the decision part. Tables are linked in a graph-like structure.

Let us now show, how these visual representations can be encoded on the machine readable level, using XML-based formats. In this approach, ARD and XTT models can be translated to different formats, using XML-specific XSLT translation.

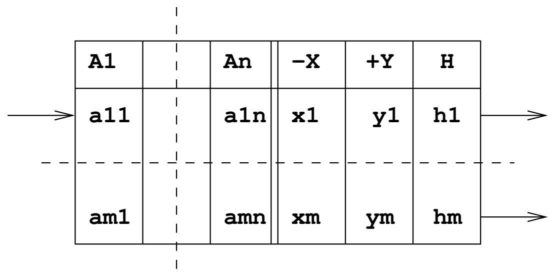


Figure 3. A generic XTT table.

4. Knowledge Markup with XML

XML (Extensible Markup Language) was introduced as a general markup meta-language for data representation and exchange [6]. The main idea is to provide a structured and flexible way of representing both data and relationships among data. An XML-complying document is a tree-like structure. The structure is built out of tags. Each tag can have attributes and a value. Each attribute can have a value. The tag value can be another tag, set of tags, or arbitrary value. In this way a tree-like hierarchy of tags and their values can be easily created.

XML is text-based and it is entirely human readable. There are certain rules regarding tags but there are no restrictions as to the values.

The XML is a meta-language. It defines constructs such as tags, attributes, values but it does not define a set of available tags or their semantics. Its purpose is to serve as a framework for defining other hierarchical, tag-based languages such as: RDF, XSLT, RuleML, SVG, XHTML etc.

Since any XML-based document is well structured and organized, there are tools for automatic language translation and code generation available. Processing an XML-based document by them is far more simple than doing it with traditional scanner-parser-generator approach.

There are number of XML-based markup languages for knowledge representation. One can design a custom XML format based on the pure XML. This is the simplest and most straight-forward approach. It was for example used in the early prototypes of *RuleML* (see `www.ruleml.org`). A more advanced, but recently more common approach makes used on some custom languages built on top of XML, the best example being RDF. This formulation is now used in both RuleML and *R2ML* (see `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML`). In some most complex cases, where richer knowledge representation methods such as ontologies are used, one can used higher-level languages such as *OWL* (see `http://www.w3.org/TR/owl-guide`). A recent use of advanced markup-representation and transformation can be found in the *Eclipse Modeling Framework* (EMF) (see `www.eclipse.org/projects/emf`).

XSLT (Extensible Stylesheet Language Transformation) is an XML based language suitable for transformations of XML documents into other documents including XML based ones. Since XML can be successfully used to express knowledge at different abstraction levels (see above) providing uniform translation mechanisms is an asset. It allows interoperability with other applications and paradigms regarding knowledge exchange, and last but not least, it also provides basis for knowledge presentation and visualization.

There are the following scenarios available regarding knowledge transformation for:

- interoperability purposes,
- execution purposes,
- validation and verification purposes,
- visualization purposes.

The above transformations can be both XML or non-XML based. However, it is worth pointing out that a transformation into XML-based representation allows a reversed XSLT-based transformation (assuming that the transformation is lossless).

The transformation scenario for interoperability purposes includes transformations to/from RDF, RuleML (see above), XTTML, ARDML (see Sec. 5) and possibly others including i.e. Eclipse Project.

The transformation scenario for execution purposes might include transformations into Prolog language or serialization into Java language classes and others.

A transformation for validation and verification purposes results in a Prolog code which self analyzes knowledge regarding its completeness, determinism etc.

A transformation for visualization purposes includes generation of XHTML human-readable description of the knowledge base or SVG-based diagrams, or other XML-based formats for third party applications. These descriptions can utilize well-known knowledge representation paradigms such as decision tables, decision trees, XTTs (see Sect. 3), frames, but also classes or objects, well known from software engineering.

5. XTT Markup Representation In the case of XTT markup representation some specific problems had to be solved. Since the comprehensive design process (see Sect. 3) includes three stages which define and refine attributes, functional relationships, and rules; there is a need to express this process in terms of XML. There are three XML-based languages formulated for expressing attributes, functional relationships among them and expressing rules. They are denoted ATTML (ATTribute Markup Language), ARDML (ARD Markup Language) and XTTML (XTT Markup Language) respectively. Dependencies among these languages, regarding their purpose are given in Fig. 4 and 5.

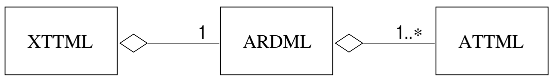


Figure 4. XTTML/ARDML/ATTML relationships.

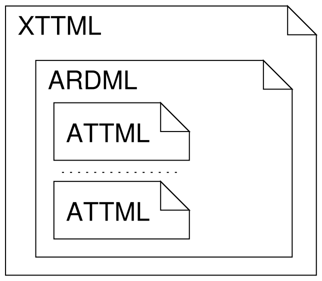


Figure 5. XTTML/ARDML/ATTML relationships, implementation.

ATTML serves the purpose of expressing attributes. It specifies attribute names, descriptions, domains and constraints. An example is given below:

```
<ATTML version="1.0">
 <attribute_list>
  <xtt_attribute name="a11"
   realtions="input" type="integer">
   <low_constraint brackettype="square"
    limit="0.0" ignore="false"/>
   <hi_constraint brackettype="round"
    limit="2.0" ignore="false"/>
  </xtt_attribute>
  <xtt_attribute name="a12"
   realtions="middle" type="integer">
  </xtt_attribute>
```

ARDML expresses functional relationship among attributes. It uses ATTML extensively working on a set of attributes.

Finally, XTTML represents rules which are built out of interconnected conditional and assignment statements. A single XTTML document is created based on a single ARDML document. An example is given below:

```
<XTTML version="2.0">
 <xtt_list_table>
  <xtt_table table_type="initial"
   name="t1">
   <row label="">
    <cell content="= 1"/>
    <cell content="= 10"/>
   </row>
   <context_size conditional="0"
    assert="0" retract="0"
    action="2"/>
   <column_attributes>
    <ca>factor</ca>
    <ca>arg</ca>
   </column_attributes>
  </xtt_table>
```

6. Automatic XTT Markup Transformation

Since the knowledge translation possibilities described in Sect. 4 are vast, this section focuses on visually appealing transformation of ARD and XTT XML-based representations into SVG, see Fig. 6

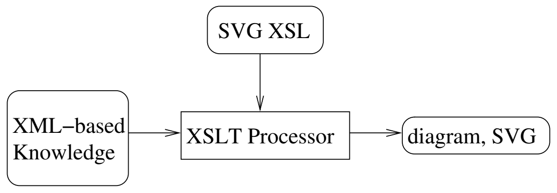


Figure 6. Presenting Knowledge as a diagram.

Knowledge base design at the conceptual level (see Sect. 3), stored as an ARDML document can be visualized as an SVG diagram (see Fig. 7).

Knowledge base, including facts and rules, at the logical level, is stored as an XML document complying with the XTTML specification. The

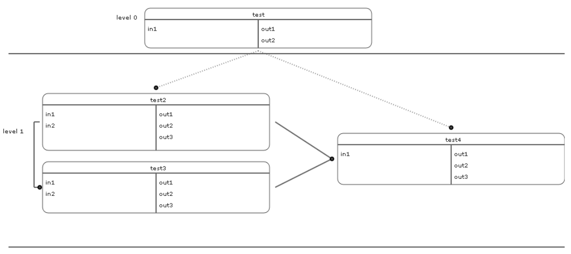
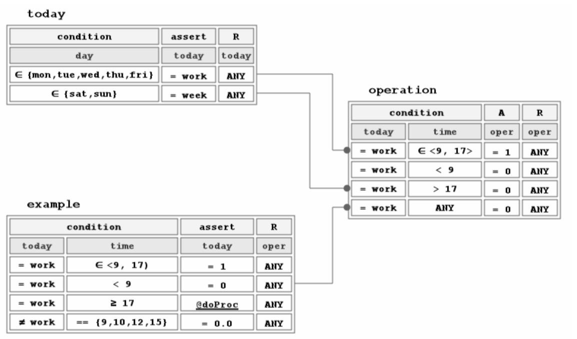


Figure 7. An ARD based Knowledge base transformed into vector graphics (SVG).

XTTML specification may be generated by any visual editor for XTT. Using appropriate transformation (XSLT) the designed knowledge can be visualized as scalable graphics using the SVG standard. A result of such process is given in Fig. 8.



today

condition	assert	R
day	today	today
∈ {mon,tue,wed,thu,fri}	= work	ANY
∈ {sat,sun}	= week	ANY

operation

condition		A	R
today	time	oper	oper
= work	∈ <9, 17>	= 1	ANY
= work	< 9	= 0	ANY
= work	> 17	= 0	ANY
= work	ANY	= 0	ANY

example

condition		assert	R
today	time	today	oper
= work	∈ <9, 17)	= 1	ANY
= work	< 9	= 0	ANY
= work	≥ 17	@doProc	ANY
≠ work	== {9,10,12,15}	= 0.0	ANY

Figure 8. An XTT based Knowledge base transformed into vector graphics (SVG).

7. Future Work ARDML, XTTML and ATTML have been evolving to match needs of contemporary rule-based system design. Subsequent releases of these languages are more and more capable. These new capabilities are followed by new features of the assisting technologies: ARD/XTT editors and XSL Transformations.

It is planned to use XSLT to translate XTTML into full featured, working Prolog language code, which is an automatic physical implementation of the designed knowledge base, becoming the executable design. It is not yet clear though whether XSLT is powerful enough to provide such advanced translation capabilities, which are close to code generation, than to "pattern matching" approach provided by XSLT.

Furthermore expressing structured knowledge designed with ARD/XTT approach using RDF seems promising. It could allow automatic knowledge merging, exchange and discovery. It is considered currently as work in progress.

References

[1] Peter Jackson. *Introduction to Expert Systems*. Addison–Wesley, 3rd edition, 1999. ISBN 0-201-87686-8.

[2] Antoni Ligęza. *Logical Foundations for Rule-Based Systems*. Springer-Verlag, Berlin, Heidelberg, 2006.

[3] Grzegorz J. Nalepa and Antoni Ligęza. Conceptual modelling and automated implementation of rule-based systems. In Tomasz Szmuc Krzysztof Zieliński, editor, *Software engineering : evolution and emerging technologies*, volume 130 of *Frontiers in Artificial Intelligence and Applications*, pages 330–340, Amsterdam, 2005. IOS Press.

[4] Grzegorz J. Nalepa and Antoni Ligęza. A graphical tabular model for rule-based logic programming and verification. *Systems Science*, 31(2):89–95, 2005.

[5] Grzegorz J. Nalepa and Antoni Ligęza. Prolog-based analysis of tabular rule-based systems with the xtt approach. In Geoffrey C. J. Sutcliffe and Randy G. Goebel, editors, *FLAIRS 2006: proceedings of the 19th international Florida Artificial Intelligence Research Society conference*, pages 426–431. AAAI Press, 2006.

[6] W3C. Extensible markup language (xml) 1.0, w3c recommendation. Technical report, W3C, 1998.