ARD+ Design and Visualization Tool-Chain Prototype in Prolog

Grzegorz J. Nalepa and **Igor Wojnicki**
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 presents a prototype design tool-chain for the ARD+ conceptual design method for rules, called VARDA. The tool-chain is implemented in a Unix environment with the use of Graphviz visualization tool and SWI-Prolog.

Introduction

An effective design support is a complex issue. It is related to the design methods as well as the human-machine interface. What is often not emphasized, is the role of the design *process*. Since most of the complex designs are created *gradually*, and are often *refined* or *refactored*, the design method should take this process into account, and the supporting tools should effectively use it.

In order to solve these problems, the HeKatE project aims at providing both design methods and tools that support the design process. Currently HeKatE provides the preliminary *conceptual design* with the ARD+ method (Attribute Relationships Diagrams). The main logical design is conducted with the use of XTT method (eXtended Tabular Trees) (Nalepa & Ligęza 2005).

The main focus of the paper is to present the prototype of VARDA (*Visual ARD Rapid Development Alloy*). It is a rapid prototyping environment for ARD+, built with use of the SWI-Prolog for the knowledge base building, and Graphviz tool for a real-time design visualization. These tools are combined by the Unix environment, where the ImageMagick tool provides an instant visualization of the prototype at any design stage.

Conceptual Design of Rules with ARD+

The ARD method aims at capturing relationships between *attributes* in terms of *Attributive Logic* (Ligęza 2006). *Attributes* denote certain system *property*. A *property* is described by one or more attributes. ARD captures *functional dependencies* among these *properties*. A simple property is a property described by a single *attribute*, while a complex property is described by multiple *attributes*. It is indicated that particular system property depends functionally on other properties. Such dependencies form a directed graph with nodes being properties.

There are two kinds of attributes adapted by ARD: *Conceptual Attributes* and *Physical Attributes*. A *conceptual attribute* is an attribute describing some general, abstract aspect of the system to be specified and refined. Conceptual attributes are being *finalized* during the design process, into possibly multiple physical attributes. A *physical attribute* is an attribute describing a well-defined, atomic aspect of the system. There are two transformations allowed during the ARD+ design. These are: *finalization* and *split*. *Finalization* transforms a simple property described by a conceptual attribute into a property described by one or more conceptual or physical attributes. It introduces a more specific knowledge about the given property. *Split* transforms a complex property into a number of properties and defines functional dependencies among them.

During the design process, upon splitting and finalization, the ARD model grows. This growth is expressed by consecutive diagram levels, making the design more and more specific. This constitutes the *hierarchical model*. Consecutive levels make a hierarchy of more and more detailed diagrams describing the system. The implementation of such hierarchical model is provided through storing the lowest available, most detailed diagram level at any time, and additional information needed to recreate all of the higher levels, the so-called *Transformation Process History* (TPH).

Prolog Prototype

A software prototype providing the ARD+ design and visualization method has been built. It is designed as a multilayer architecture (see Fig. 1):

- knowledge base to represent the design,
- low-level primitives: adding and removing attributes, properties and dependencies,
- transformations: finalization and split including defining dependencies and automatic TPH creation,
- low-level visualization primitives: generating data for the visualization tool-chain, so-called DOT data,
- high-level visualization primitives: drawing actual dependency graph between properties and the TPH.

As an implementation environment of choice the Prolog language is used. It serves as a proof of concept for the ARD+ design methodology and prototyping environment.

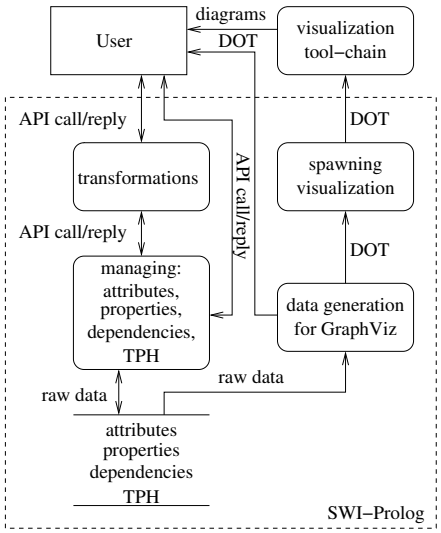

Figure 1: Prolog prototype architecture

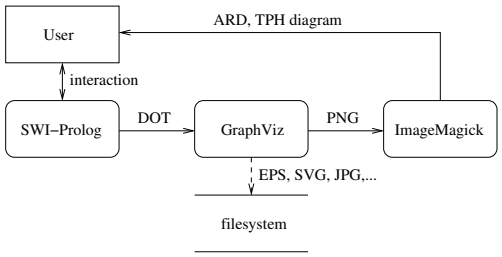

Figure 2: Visualization tool-chain

Switching to other environments such as Java, C++, Ajax, or Eclipse platform is possible. Prolog was chosen because it offers a rapid development environment for knowledge-based systems.

The low-level visualization primitives generate data for the visualization tool-chain. The high-level visualization primitives spawn the tool-chain, which renders appropriate graphs representing the diagrams. These primitives can be used at any time during the design process in the Prolog interactive shell. They launch the visualization tool-chain in parallel with the shell.

At the design stage, a proper visualization of the current design state, is the key element. It allows to browse the design more swiftly and identify gaps, misconceptions or mistakes more easily. Both ARD and TPH diagrams are directed graphs. Therefore, a graph visualization primitives are needed. Instead of reinventing these concepts, or implementing them from scratch, a tool-chain of well proved tools to provide actual visualization is assembled. The tool-chain is based on three components: SWI-Prolog (`www.swi-prolog.org`), GraphViz (`www.graphviz.org`), and ImageMagick (`www.imagemagick.org`).

The interaction between the visualization tool-chain and the rest of the system is given in Fig. 1. The detailed interaction between the tool-chain components is given in Fig. 2.

There are two scenarios the visualization is performed:

1. generating diagrams for an already designed system described in Prolog,
2. generating diagrams during the design process.

The first scenario can be executed as follows:

```
swipl -q -f 'ard-design.pl' -t go.
   | dot -Tpng | display
```

Assuming that `ard-design.pl` file contains the design coded with appropriate Prolog clauses, and predicate `go` triggers GraphViz data generation. The generated data is processed by GraphViz (`dot` utility) generating a PNG output which is passed to ImageMagick (`display`) which displays it and allows for annotation. In addition to the functionality described above, GraphViz can be successfully applied to generate the diagrams in other formats and store them in the file system. It is indicated as a dotted flow between `GraphViz` and `filesystem` in Fig. 1.

Generating diagrams during the design process is provided by two Prolog predicates: `sar` and `shi` that generate the appropriate GraphViz source code, and spawn both GraphViz and ImageMagick subsequently. These predicates are accessible from the interactive Prolog shell, and display the ARD or the TPH accordingly. The tool-chain is executed in parallel with the interactive Prolog prompt which allows to display several diagrams simultaneously.

Implementing VARDA with Prolog was a conscious decision. Currently it has over 700 lines of Prolog code. A rough estimate is that it corresponds to several thousands lines of Java code. While some graph editing Java solutions (JGraph) or frameworks (Eclipse EMF) could be helpful, the development of an ARD editor in Java would be much more complicated and time consuming. VARDA is a free software licensed GNU GPL, and it can be obtained from: `https://ai.ia.agh.edu.pl/wiki/hekate:varda`.

Conclusion

The original contribution of this paper is the presentation of a Prolog-based prototype tool-chain for the refined ARD+ method. The tool-chain uses the Graphviz visualization tool and the SWI-Prolog environment. It allows for a rapid prototyping of the ARD model, with an automated, real-time visualization.

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

References

Ligęza, A. 2006. *Logical Foundations for Rule-Based Systems*. Berlin, Heidelberg: Springer-Verlag.

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