UServ Case Study, Conceptual Design with ARD+ Method

Grzegorz J. Nalepa
Institute of Automatics,
AGH – University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
gjn@agh.edu.pl

Abstract

The paper is dedicated to an analysis of the classic business rules study called UServ. The importance of this study comes from the fact, that it serves as a benchmark example for a number of design and implementation methods for rules. In this paper, the analysis is conducted from the knowledge engineering point of view. The results of the analysis are also used to present the design of UServ using ARD+, a design method developed within the HeKatE project. The project provides a unified hierarchical design solution, that does not suffer from the well-known semantic gaps. In the paper the design gap is exposed; it is due to the lack of integration of the original SBVR, BPMN and rule methods used in the study.

Introduction

The rule-based knowledge specification method is one a classic one in AI (Russell & Norvig 2003). Rules in various forms find applications in different areas of engineering and business. The latter is especially true with the so-called business rules approach, that has been gaining momentum recently. While it could be considered a rediscovery of the rule-based expert systems (RBS) studied in AI in the '80 and '90, it is very successful at providing efficient solutions for rules deployment in business systems.

Practical support for business rules design remains an active development area, with several communities providing different approaches. This paper discusses a classic business rules case study called UServ, designed with the BPMN method from OMG. Then an alternative design using the ARD+ method developed by the HeKatE project is presented. The project aims at providing advanced AI methods to support design and business software analysis.

Business Rules Design Approaches

Business Rules (Ross 2003; von Halle 2001) (BR for short) are a rediscovery of the rule-based expert systems studied for years in AI. In last years BR became popular among non-engineers, especially in business and management environments. This is due to the fact, that rules are a very natural, intuitive yet powerful knowledge specification tool.

Current development of the BR systems includes: rule engines, (BRE) rule management (and interchange), (BRMS) rule design, rule analysis. To some extent this corresponds to the classic research issues in the RBS. What is important, is the fact, that the emphasis is different. Since the technology is commercially driven, a fast rule deployment is usually the single most important issues. This why the development of fast *rule engines* is the priority. These engines are often accompanied by number of rule acquisition tools, able to read rules in number of formats, e.g. classic spreadsheets files. These tools use only simple syntax checkers.

Number of companies provide mature solutions, with FairIsaac and ILOG being the leaders. There are several mature open-source solutions, especially JBoss Rules (formerly Drools) on the RedHat JBoss platform, and Jess. Another group of engines provides a business process perspective, e.g. VisualRules from Innovations. In order to import, integrate and manage large rulebases BR management systems (BRMS) are used.

Specialized *design* methods are not introduced in most cases. The design support is limited to the use of decision tables, or decision trees. This limitation is especially when there is a need to design a new and complex system.

However, recently there has been a lot of new developments in the area of BR design. When BR systems began to play an important role in enterprise management, there was a need to integrate them on a higher level, possibly at the design stage. This is why, there have been multiple attempts to provide new design methods, that could be integrated with UML-based object-oriented applications. Two most important include REWERSE *URML*, and *OMG PRI*.

Another influential group of developers works within the W3C framework, focusing on the data-perspective, including XML-based markup languages, metadata representation with RDF/S and ontologies. W3C aims at providing a platform for rules on the Web (possibly the Semantic Web) through the RIF standard (*Rule Interchange Format*).

What seems unfortunate, is the fact that quality issues are often not considered, with quality assurance being treated as an optional product, not a process, or important feature of the design. There are very few companies providing specialized rule analysis tools, with LibRT being one of the exceptions. Quality of BR is not such a "hot" topic among researchers, as were the RBS evaluation, validation, verifi-

cation, and analysis in the '90s.

HeKatE Design Approach

The HeKatE project (`hekate.ia.agh.edu.pl`) aims at providing unified modeling methods for rules. These methods should support the design from the system description in the natural language, through a formalized multilevel design, to a final, automated implementation. The process should be supported by intelligent tools helping in the design, as well as translation and integration of the model.

Currently, this approach is based upon a concept of a three-level design, including conceptual, logical, and physical design. At every level another design method is used. The methods are integrated, with the integration formally described, in order to evade well-known *semantic gaps* that are common in the software engineering. The project aims at supporting general business software design, and business rules, but is based on the previous experiences with design and implementation of RBS in AI.

At this stage, the project provides *ARD* conceptual design method for attributes, where ARD stands for *Attribute Relationship Diagrams*, and *XTT* logical design method for rules, where XTT stands for *eXtended Tabular Trees*. While ARD allows to identify system attributes and their functional dependencies, XTT provides means to build decision rules using these attributes. The ARD method has been recently largely reworked towards supporting more complex design cases with a fully formalized design. For the preliminary formal description of the ARD+ method see (Nalepa & Wojnicki 2008b). At the same time XTT is also being enhanced toward a general rule programming model, for the progress see (Nalepa & Wojnicki 2007).

Supporting tools are an important area of development within the project. At the moment, the ARD+ design is supported by a prototype Prolog tool-chain, that allows for an automated visualization of the model using the *GraphViz* tool. For the description of the prototype see (Nalepa & Wojnicki 2008a). The enhanced XTT design is supported by a prototype visual editor using a Qt library.

The ARD+ method aims at capturing relations between *attributes* in terms of *Attributive Logic* (Ligęza 2006; Ligęza & Nalepa 2007). *Attributes* denote certain system *property*. A *property* is described by one or more attributes. ARD+ captures *functional dependencies* among system 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 attribute*, an attribute describing some general, abstract aspect of the system to be specified and refined e.g.: `WaterLevel`. *Physical attribute* is an attribute describing a well-defined, atomic aspect of the system, e.g. `theWaterLevelInTank1`. Conceptual attributes are *finalized* during the design process, into possibly multiple physical attributes that cannot be finalized (they are present in the final rules).

The gradual design process is based on the notions of *transformations*. There are two transformations allowed: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, more attribute about the given property. *Split* – transforms a complex property into a number of properties and defines functional dependencies among them. Attributes are unique, the same attribute cannot describe more than a single property. The number of transformations in a single design step, also referred to as "level transition" is limited to *one per node*.

During the design process, upon splitting and finalization, the ARD model grows. This growth is expressed by consecutive diagram levels, making the design more specific. This constitutes the *hierarchical model*. The implementation of the model is provided through storing the lowest available, most detailed diagram level, and additional information needed to recreate all of the higher levels, the so-called *Transformation Process History*, TPH for short. A TPH forms a tree-like structure then, denoting what particular property is split into or what attributes a particular property attribute is finalized into.

A Prolog-based prototype providing the ARD+ design method has been built. It serves as a proof of concept for the ARD+ design methodology and prototyping environment. It is designed as a multi-layer architecture: providing low and high-level predicates to manipulate the knowledge base and to represent the design. In the knowledge base attributes, properties, dependencies and TPH are represented as Prolog facts, using dynamic knowledge modification capability. Examples of low-level predicates and transformations are:

```
ard_att_add(+Attribute)
ard_att_del(?Attribute)
ard_depend_add(ToProperty,FromProperty)
ard_depend_del(ToProperty,FromProperty)
ard_finalize(+Property,+ListOfNewAttributes)
ard_split(+Property,+PropList,+DependList)
```

The low-level visualization primitives generate data for the visualization tool-chain. The tool-chain is based on Unix (or Unix-like) environment and uses SWI-Prolog (`www.swi-prolog.org`), GraphViz (`www.graphviz.org`) and ImageMagick (`www.imagemagick.org`).

In this paper the focus is to apply, present and test the ARD+ method on the classic UServ business rules case study (BRForum 2005), using the Prolog tool-chain (Nalepa & Wojnicki 2008a).

The UServ Case Study

The UServ case (BRForum 2005) is a benchmark case study from the *Business Rules Forum*. Since it has been published, it has been serving as one of the benchmark use cases for number of design tools for business rules from different groups. For a in-depth analysis and implementation from the REWERSE-I1 group, see `http://hydrogen.informatik.tu-cottbus.de/moodle/course/enrol.php?id=24`.

From the original description one can learn that UServ Financial Services provides a number of financial products,

including: insurance products, as well as banking services. UServ aims at satisfying the complete financial services of its clients. With this focus on complete relationship services, clients are rewarded for their loyalty as they deepen their relationship with UServ by increasing their financial portfolio. UServ plays a balancing act between rewarding their best clients and managing the risk inherent in providing on-going service to clients whose portfolios are profitable, but violate the eligibility rules of individual products. UServ's business rules are an essential component for managing the risk. The business rules address eligibility, pricing and cancellation policies at both the individual product and portfolio level. The case study focuses on UServ's vehicle insurance products, but differentiates the basic business rules from those that apply to preferred and elite clients.

The case description includes the following parts: the general description, business rules, use scenarios, business concepts model, and business process model. The original case description does not include hints, which information has been *acquired* from the customer (the UServ Financial Services), and which was actually *inferred* by the designers. Since the *core* knowledge about the company is contained within the rules, and processes, one can imagine, that these were the parts acquired, whereas the business concepts model has been actually inferred. It should also be observed, that while the rules offer a *declarative* knowledge, the processes present the *sequential* perspective.

The business rules model includes:

1. 2 Client Segmentation Business Rules (called *RS* here)
2. Eligibility Business Rules (*RE*), including:
   (a) 14 rules for Automobile Eligibility (*REA*), including:
      i. 5 rules for Potential Theft Category (*REAT*),
      ii. 5 rules for Potential Occupant Injury Category (*REAI*),
      iii. 4 rules for Auto Eligibility (*REAE*).
   (b) 12 rules for Driver Eligibility (*RED*), including:
      i. 9 rules for Driver Age Category (*REDA*),
      ii. 3 rules for Driving Record Category (*REDR*).
   (c) 14 rules for final Eligibility Scoring (*RES*), including:
      i. 3 rules for Automobile Eligibility scoring (*RESA*),
      ii. 4 rules for Driver Eligibility scoring (*RESD*),
      iii. 2 rules for Client Segment scoring (*RESS*),
      iv. 5 rules for the main Eligibility Scoring (*RESM*).
3. Pricing Business Rules (*RP*), including:
   (a) Auto Premiums (*RPP*), 13 rules,
   (b) Auto Discounts (*RPA*), 5 rules,
   (c) Driver Premiums (*RPD*), 10 rules,
   (d) Market Segment Discounts (*RPS*), 2 rules, and
   (e) a single rule for the Base Premium (*RPB*).

This makes a total of 73 rules, of different types, e.g. fact, calculating, and constraint satisfaction rules.

The business process model includes two basic processes, shown on the highest level (Manage UServ called *PM* here), that are (top-to-down):

1. Vehicle Insurance Application Process, (*PA*)
2. Policy Renewal Process (*PR*).

They both use the basic subprocesses, that is:

1. Policy Processing (*PP*), based on the eligibility scoring, that makes use of
2. Policy Scoring (*PS*), that uses the eligibility rules for car and driver.

The top level processes *PA* and *PR* implicitly make use of the pricing rules. The model is built using the *Business Process Modelling Notation* (BPMN) (OMG 2006a).

The business concepts model is put forward with use of the *SBVR* notation, that is the *Semantics of Business Vocabulary and Business Rules* (OMG 2006b). It tries to captures the relationships between different *objects* (mostly in terms of the OOA/P, UML) that could possibly identified in the system, and its environment. It should be helpful to better understand the system, as well as provide hints for the future OO-based model and implementation. The use scenarios serve as the test cases for the implementation. The UServ case description does not include any information about implementations of this study, as it is supposed to aim as a reference *model*.

Before moving to the actual design with the HeKatE methodology, it is worth emphasizing, that from the point of view of a classic knowledge engineer, this model seems to be unclear, redundant, and possibly inconsistent. First of all the declarative rule-based perspective is implicitly, unclearly mixed with the sequential process-based perspective. This is not to deny a possible use of these two perspectives, but to point out an inconsistent model notation. To some extent, this is due to the limitations of the BPMN itself, as well as lack of a clear integration between the BPMN and rule semantics. The process model also introduces some extra information (see Process Vehicle Insurance Application), *not* present in the rules, that can be possibly useful for the implementation, but there is no clue on *how* this information could actually be used. It is also worth pointing out, that for a classic *rule engine* the process model is mostly both redundant and useless anyway, because the rules should carry the *whole* information. On the other hand, the process-based perspective also implicitly offers a *hierarchical* perspective which *is valuable* for the HeKatE approach, as well as some newer rule engine tools, such as *RuleFlow* available in the new 4th version of the Drools platform.

Another inconsistency concerns the so-called business concepts model. This model is somehow inferred from the rules, and processes. However, it contains much more information than these models, and one can only imagine, that some of this information has been inferred by the designers, whereas other comes form additional "stories" told by the customers. From the formal point of view, the relation of concepts in the model is unclear and inconsistent with the concepts present in rules and process.

We shall now move to the UServ design with the HeKatE ARD+ method, that offers a hierarchical conceptual design solution for rules, and possibly captures all of the information needed to build a rule-based system for the UServ. In here the Prolog prototyping environment will be used.

UServ Design with ARD+

Let us begin with the highest, most general concepts about the UServ company. The processes and rules use abbreviations previously defined. We will begin with the hierarchical process model, which in a general sense is similar to the hierarchical ARD+ approach. On this level (L0), corresponding to the *PM* diagram at process model, we may state that the system provides a `NewPolicy`, for some customers. The level 0 is described in Prolog as:

```
ard_att_add('NewPolicy'),
ard_property_add(['NewPolicy']).
```

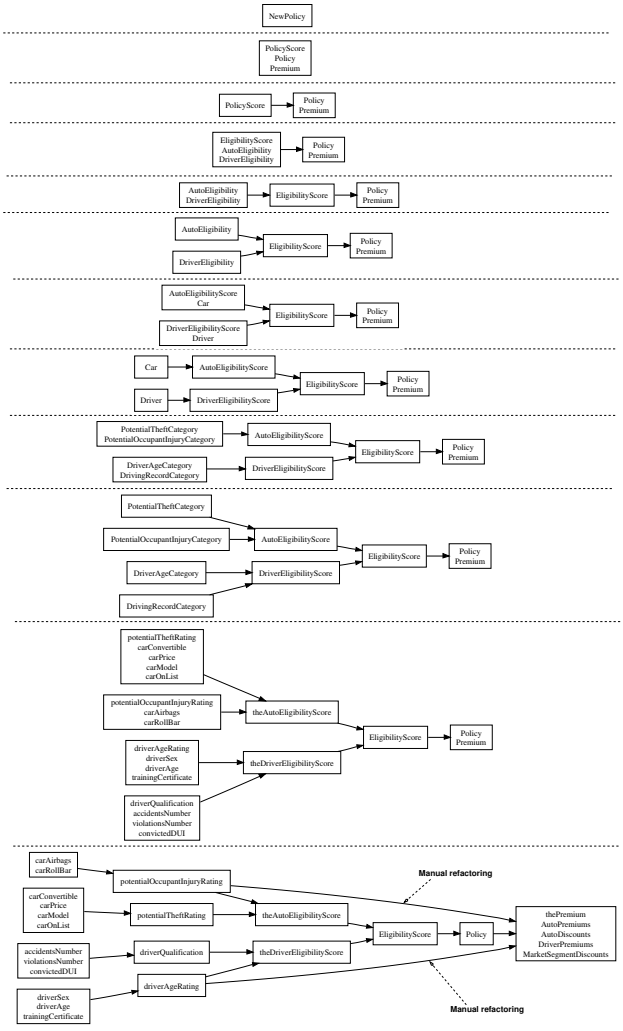

Figure 1: UServ ARD+ model

While analyzing the *PA* and *PR* diagrams one can see that besides a `Policy`, we also get a `Premium`. Next, looking at the *PP* diagram, we see that only thing we learn, is that the policy decision and the premium is based on the `PolicyScore`. So on this level (L1) the general concept of `Policy` gets *finalized* to these three concepts. The finalization on level 1 corresponds to:

```
ard_att_add('Policy'),
ard_att_add('Premium'),
ard_att_add('PolicyScore'),
ard_finalize(
    ['NewPolicy'],
    ['PolicyScore','Policy','Premium']).
```

To capture the functional dependency between them, we move to the next level (L2), where a single ARD+ property grouping them, gets *split* into two functionally dependant properties. The split in Prolog looks like this:

```
ard_split(
    ['PolicyScore','Policy','Premium'],
    [ ['PolicyScore'],['Policy','Premium']],
    [
     [['PolicyScore'],['Policy','Premium']]
    ]).
```

Analyzing the *PS* diagram, it can be observed, that the `PolicyScore`, involves the `EligibilityScore`, `AutoEligibility` and `DriverEligibility`; this is the third (L3) level of the ARD+ model, reached through finalization. To capture the functional dependency between these three a split is needed, observed in the level 4.

Looking at the *PS* diagram, we see, that the `AutoEligibility` and `DriverEligibility` are functionally independent, this is captured through a split on the level 5 of ARD+. Calculating the actual `AutoEligibilityScore` and `DriverEligibilityScore` is done using information about the `Car` and `Driver`, see level 6, for finalizations (by the way, the names "Auto" and "Car" are used interchangely in the original study, which is misleading). We can see that there is a functional dependency between `AutoEligibilityScore` and `Car`, and `DriverEligibilityScore` and `Driver` respectively, captured by the split at level 7.

This is an interesting point in the original study, since the formalism used so far, that is the BPMN cannot express a more detailed information. This the point, where the *designer* should somehow relate and connect the declarative rule model to the hierarchical sequential process model. This is the very point where a clear *semantic gap* between these two views is exposed. However, this gap is not present in the ARD+. We shall continue with our analysis, using the rulesets mentioned previously. Let us focus on the car, that is the *REA* rulesets.

The actual notion of `Car` is related to the `PotentialTheftCategory` and `PotentialOccupantInjuryCategory`. Now, with the *RED* ruleset, `Driver` is related to the `DriverAgeCategory` and `DrivingRecordCategory`. Both of these specifications are captured with the finalization in level 8. In both cases these aspects are functionally independent, as seen in the level 9.

Getting into more details of rulesets *REA* and *RED*, and analyzing the actual attributes of the rules, we can introduce the final concepts, that are in fact the so-called *physical attributes* in the ARD+ method. These are `carConvertible`, `carPrice`, `carModel`, `carOnlist`, that determine the value of the `potentialTheftRating`, `carAirbags`,

carRollBar, that determine the value of the potentialOccupantInjuryRating, for the *REA* set, and sex, age, trainingCertificate, that determine the value of the driverAgeRating, accidentsNumber, violationsNumber, convictedDUI that determine the value of the driverQualification, for the *RED* set. These get introduced through finalization on level 10, and split to show the dependence on next levels.

In the *RP* set the pricing rules are. These are used to calculate the premium. The Premium is in fact thePremium AutoPremiums, AutoDiscounts, DriverPremiums, and MarketSegmentDiscounts. The premium only gets calculated when the Policy is actually granted, so the policyDecision is positive. These attributes get introduced in the right part of the ARD+ diagram, in the Policy,Premium property, at level 12; furthermore they should be splited on the next level.

This is the point, at which the straightforward design encounters problems. While analyzing the *RP* set, it can be realized, that some important dependencies have been missed. These are the dependencies between AutoPremiums and attributes of the car, as well as DriverPremiums and attributes of the driver. They should have been introduced somewhere at the higher level. This is where the automatic refinement feature of ARD proves to be helpful.

Automated Design Refinement

In order to represent the missing dependencies a manual refactoring is used as follows:

```
ard_depend_add(
  ['potentialOccupantInjuryRating'],
  ['thePremium','AutoPremiums','AutoDiscounts',
   'DriverPremiums','MarketSegmentDiscounts']).
ard_depend_add(['driverAgeRating'],
  ['thePremium','AutoPremiums','AutoDiscounts',
   'DriverPremiums','MarketSegmentDiscounts']).
```

They are indicated on the lowest level observed in Fig. 1 Now the question is how this change can be integrated with the existing model.

In the hierarchical ARD+ model only the lowest level is stored. Every split or finalization is recorded in the TPH. Using this information *any* previous level can be integrated. Moreover, modifications introduced at the lowest level can be *automatically* integrated at higher levels. So, in case when some dependencies or properties have been missed during the design, they can be added at the lowest level, and be automatically integrated into the design (in case of properties the TPH needs to be modified).

Methodological Observations

Number of observations on the ARD+ process can be made.

The ARD design is not deterministic one, for the very same system, the designer can make different decisions abut the actual sequence of split or finalizations. This also means, that the number of design steps, or levels can be different – it is possible to transform every node of the diagram at every level, but it does not mean it has to be done.

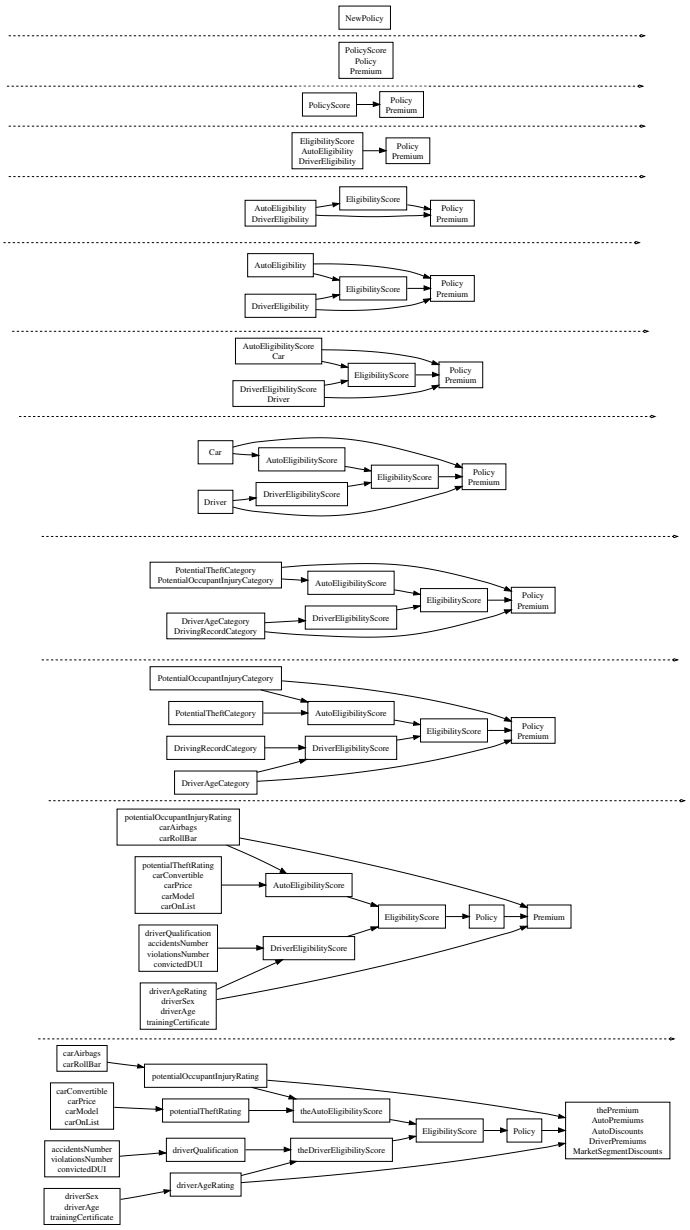

Figure 2: Automatically refined diagram

The transformations of not neighboring nodes are independent of each other, so the sequence of transformations can be different. The automatic *collapsing* of the diagram (recreating higher levels using TPH) can in fact generate different diagram than the original one. So, in fact, collapsing, and then recreating the diagram back to the bottom level can be thought of as a process of the *minimization*.

Having a system description given, there are number of approaches to the design. In the study presented in this paper, the design process was mimicking the original design with BPMN in order to compare the two approaches. Another reason was actually to get the final rules present in the study. But there could be another approach to the UServ de-

sign, where the model is build the design from scratch, simply by reading the rules, and neglecting the BPMN model at all. In this case notions of Client, Car, and Driver could be captured on a higher abstraction level.

A general rule concerning transformation sequence can be formulated: "finalize to physical attributes as late as possible". As long as there are conceptual attributes describing a property, there is a possibility of further specification, thus discovering, or specifying new knowledge.

Upon a *finalization* transformation number of attributes describing a property increases. For a *split* transformation number of properties increases since the source property is split into some number of properties which is indicated by multiple vertices in the diagram.

An interesting concept is related to the ARD design patterns, that is some repeating patterns in designing systems with ARD. Some typical split patterns can be observed, as well as split–finalize sequences. This is a research in progress, that definitely needs more use cases.

In future ARD+ tools should provide heavy visual hinting, guiding the designer through possible diagram transformations, including the support for the patterns mentioned above. The tools should also support bidirectional design, where the designer is able to see the results of the refactoring on-line. These feature is currently only partially supported by the Prolog prototype, but is fully doable.

For importing existing rule-based systems, a semi-automatic ARD model building could be supported. In this case, the model could be built from original rules by a backwards chaining generator, that would analyze and identify functional dependencies, and generate new conceptual attributes (this idea is actually similar to unification in Prolog).

The focus of the ARD is on the design phase, and the initial transition from user-provided specification to rule specification that connects rules with concepts. So it is the phase of a *conceptual design*, that is also addressed by the *SBVR* (OMG 2006b). The main difference is, that SBVR is based on the MOF, and uses UML-derived constructs. While it facilitates the integration with UML-centric design tools and approaches, it also makes it inherit the problems UML has. Most of the complex designs are created *gradually*, so the design method should take the design *process* into account, and the tools used should effectively support it. It is worth noting that UML, does not support the process at all. This makes it difficult to use in real-life for both SE and KE.

Future Work

The paper discusses practical approach to the conceptual design of business rules. In the paper a new refined designed method, called ARD+ is applied to the classic benchmark case study from the Business Rules Group. The method, originally aimed at the classic rule-based systems, aims at providing a unified, hierarchical design, superior to this of the BPMN, or SBVR methods.

The original contribution of this paper consists in: an in-depth discussion of the *UServ*, a classic and benchmark business rules case study, that outlines some of the limitations of the original design, from the knowledge engineering point of view, an application of the ARD+, a conceptual design tool within the HeKatE methodology, and practical guidelines for the future design tools for ARD+.

The ARD+ version presented in this paper is a rework of the original prototype method. Future work will be mainly focused of the more complex design cases, that would allow for developing practical *refactoring* solutions within ARD+. One of the approaches is to formulate semi-formalized *design minipatterns*, used at the ARD+ level transitions.

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

References

BRForum. 2005. Userv product derby case study. Technical report, Business Rules Forum.

Ligęza, A., and Nalepa, G. J. 2007. Knowledge representation with granular attributive logic for XTT-based expert systems. In Wilson, D. C.; Sutcliffe, G. C. J.; and FLAIRS., eds., *FLAIRS-20 : Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference : Key West, Florida, May 7-9, 2007*, 530–535. Menlo Park, California: Florida Artificial Intelligence Research Society.

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

Nalepa, G. J., and Wojnicki, I. 2007. Proposal of visual generalized rule programming model for Prolog. In Seipel, D., and et al., eds., *17th International conference on Applications of declarative programming and knowledge management (INAP 2007) and 21st Workshop on (Constraint) Logic Programming (WLP 2007) : Wurzburg, Germany, October 4–6, 2007 : proceedings : Technical Report 434*, 195–204. Wurzburg : Bayerische Julius-Maximilians-Universitat. Institut für Informatik: Bayerische Julius-Maximilians-Universitat Wurzburg. Institut für Informatik.

Nalepa, G. J., and Wojnicki, I. 2008a. An ARD+ design and visualization toolchain prototype in prolog. In *FLAIRS2008*. submitted.

Nalepa, G. J., and Wojnicki, I. 2008b. Towards formalization of ARD+ conceptual design and refinement method. In *FLAIRS2008*. submitted.

OMG. 2006a. Business process modeling notation (bpmn) specification. Technical Report dtc/06-02-01, Object Management Group.

OMG. 2006b. Semantics of business vocabulary and business rules (sbvr). Technical Report dtc/06-03-02, Object Management Group.

Ross, R. G. 2003. *Principles of the Business Rule Approach*. Addison-Wesley Professional, 1 edition.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition.

von Halle, B. 2001. *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. Wiley.