Technical Report: Development and Verification of Rule Based Systems - a Survey of Developers

Valentin Zacharias

FZI Research Center for Information Technologies at the University of Karlsruhe,
zach@fzi.de,
http://www.fzi.de/ipe

Abstract. While there is great interest in rule based systems and their development, there is little data about the tools and methods used and the issues facing the development of these systems. To address this deficiency, this paper presents the results from a survey of developers of rule based systems.
The results from the survey give an overview of the methods and tools used for development and the major issues hindering the development of rule based systems. Recommendations for possible future research directions are presented.
The results point to verification and validation, debugging and overall tool support as the main issues negatively affecting the development of rule based systems. Further a lack of methodologies that appropriately support developers of these systems was found.

1 Introduction

With the application of rules on the Semantic Web, the continuing rise of business rule approaches and with initiatives to standardize and exchange rules, there is currently a renewed and increasing interest in rule based software systems and their development. At the same time, however, there is little data and overview about the way rule bases are used, developed and which challenges developers of these systems face; in sum there is little data that could be used to set priorities for research and development.

To address this shortcoming this paper presents a survey of the methods and tools used and the issues facing the development of rule based systems. Based on prior experience [1] and much older surveys (see related work) verification and particularly debugging issues were identified as particular important and the survey was designed to focus on these.

This paper starts with a short related work section introducing two much older studies that addressed similar questions, results from these survey are also cited throughout the text where similar questions were used. The next sections introduce the survey and presents data about the participants, their experience and the rule base systems they develop. The core results from the survey are then

grouped into three sections, (1) Methods and Tools Used for Development, (2) Verification, Bugs and Debugging and (3) Issues and Comparison to Procedural Programming. The conclusions highlight the most important findings and use these to derive possible directions for future research.

2 Related Work

In 1991, Hamilton et al. [2] questioned 70 people with respect to the state of the practice in knowledge-based system verification and validation. The goal of this survey was to document the state of the practice. The insight gathered during the survey and follow up interviews was used to develop recommendations for further V&V research. The findings from this survey included that 62% of developers judged the developed expert system to be less accurate than the expert and 75% judged it to be less accurate than expected.

Also published in 1991, O'Leary [3] used a mailed survey to query 34 developers of knowledge-based systems. This poll had the specific goal of testing a number of hypotheses with respect to prototyping of knowledge based systems. The core finding was that the prototyping development methodology is found to lead to more robust models and that it does not increase the validation difficulty.

In the broader world of *conventional* software engineering (i.e. using procedural or object-oriented languages) Cusumano et al. [4] compared the practices used in software development across countries, finding in particular that Indian software development projects used the most elaborate practices, combining traditional techniques such as specification and reviews with modern ideas like pair programming. Zhao and Elbaum [5,6] explored the use of quality assurance tools and techniques in open source projects. The most interesting findings from these studies include, that while over half of the projects spend more than 20% of the development time on testing, only 5% compute any test coverage measures and the use of regression testing is not widespread. Finally, Runeson et al. [7] summarize the (mostly experimental) empirical data with respect to quality assurance methods. Among other things they found that on average only 25% to 50% of faults are found during inspection and only a slightly higher 30% to 50% during testing - concluding that on average half of the faults remain.

3 Survey Construction

The goal of the survey was to be an exploratory study of the methods and tools used, and the issues facing the developers of rule based systems. The survey focused on verification and in particular debugging as very important questions that in the author's experience are particularly problematic for the development of rule based systems. Some questions were also derived from specific hypotheses, described in detail below together with the results for the questions.

The survey was designed to be answerable in less than 15 minutes; included 17 questions spread over three pages and was conducted using the SurveyMonkey [8] service. The survey with all questions in their original layout can be accessed

at http://vzach.de/papers/survey08.pdf. Participants were asked to answer all questions with respect to the largest rule base in whose development they had been involved with in the past 5 years.

4 Participants

Participants were recruited through emails sent to public mailing lists concerned with rule based systems[1], Jess and mailing lists of academic institutes; invitations were also published on some blogs concerned with rule based systems[2] . A chance to win a camera was given as additional incentive to motivate people to participate. 76 people opened the survey and 64 answered most questions; one reply was removed because it consisted of obviously nonsensical answers.

	Mean	Median	Std. Deviation
Person Month Development			
PM for entire software	59	15	148
PM for rule base	9	5.5	15
Size of Rule Base			
Number of rules	1969	120	8693
Size of average rule	9.3	5	17
Size of largest rule	24	11	39
Developers involved			
Rule developers	3	2	4
Other software developers	3	1	8
Domain experts that created rules	1.5	1	2
Domain experts as consultants	1.9	1	2.5
Domain experts for V&V	1.7	1	2.4
Others	0.6	0	1.6

Table 1. Measures of the size of the rule base

For the purpose of analysis the wide variety of systems used by the respondents was grouped into five groups:

[1] The mailing lists where the CLIPS mailing list, RuleML 'all' mailing list, the RIF initiative mailing list, the JESS users mailing list, the GNU Prolog users mailing list, the SWI Prolog mailing list, the TU Prolog users mailing list, the Pellet users mailing list, the XSB users mailing list, the Jena developer mailing list, the Drools developer mailing list and the Semantic Web mailing list of the W3C

[2] Smart Enough System at http://smartenoughsystems.com/wp/, James Taylors Decision Management http://www.ebizq.net/blogs/decision_management/, Enterprise Decision Management Blog http://www.edmblog.com/ and finally the author's own blog at http://vzach.de/blog

– **Prolog:** 7 results; consisting of tuProlog(2), SWI Prolog (2), Visual Prolog (1), XSB (1) and Yap Prolog (1)
– **Declarative Rules:** 11 results; consisting of F-logic - ontoprise (3), SWRL - KAON2 (2), SWRL - Protege SWRL Tab (2), SWRL - PELLET (1), Jena Rules (1), WSML-Flight(1), Ontology Works Remote Knowledge Server (1) and Iris (1)
– **Business Rule Management Systems (BRMS):** 17 results; consisting of JBoss /Drools (8), Fair Isaac Blaze Advisor (3), Yasu Quick Rules (SAP) (2), BizTalk (1), NxBre (1), Acumen Business-Rule Manager(1), OpenRules (1) and Ilog JRules/.Net rules (1)
– **Shells:** 24 results, consisting of Jess (12), Clips (9), Mandarax (1), Jamocha (1) and KnowledgeWorks/LispWorks (1)
– **Other:** 1 results, using a 'Proprietary IT software engine'

The size of the reported systems varied widely (see Table 1); the average rule base consists of 2000 rules, has 9 conditions/body atoms per average rule and is developed in 9 person months. However, the average size is strongly influenced by a small number of very large systems, half of the systems have not more than 120 rules. On average the rule base is part of a much larger software system that takes almost 60 person months to develop. The largest system in the survey has 63000 (partly learned) rules and is used for disease event analysis. The most time consuming took 100 person months to develop and is used to determine parameters to operate a medical imaging system. Slightly over 50% of the projects involve at least one domain expert that creates rules herself.

On average the people filling out the survey had 6.6 years of experience with rule based systems and 15 years experience with creating computer programs in general.

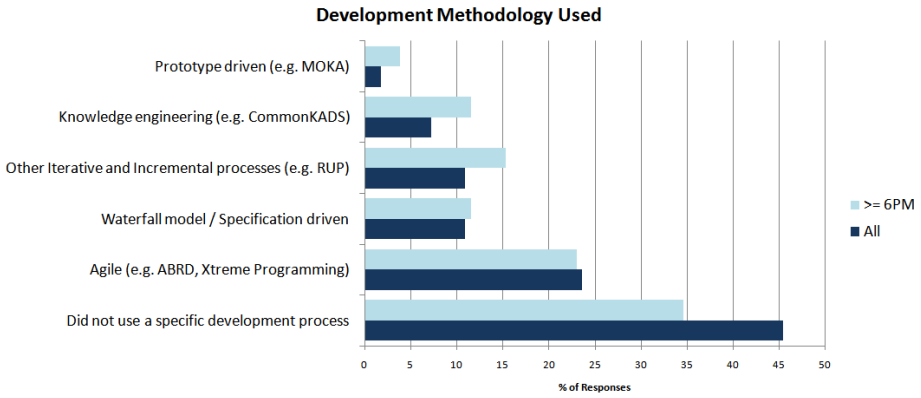

Fig. 1. *The development methodology used, for all responses and for the 26 responses where the rule base development took at least 6 person months.*

The tasks of the rule bases (entered as free text) include workflow management, diagnosis, scoring, ontology reasoning, tutoring and planning. The rule bases are created for a multitude of different domains, including insurance, finance, health care, biology, computer games, travel and software engineering.

38% of the rule bases are commercially deployed, 26% are deployed in a research setting and 10% are under development with deployment planned. The remaining 26% are prototypes, 10% are prototypes that are also used by others than the developers.

5 Methods and Tools Used for Development

Participants of the survey where given a multiple choice question to describe the methods used for the development of the rule base. The answers (see figure 1) showed that indeed a large part of rule based systems are created without any specific development process and that the rise of agile and iterative methods [9,10] is also visible for rule based systems. In 1991, Hamilton et al. [2] used a similar question and found that the most used model was the cyclic model (41%) and that 22% of the respondents followed no model[3].

	Percent responses
Simple text editor	33%
Textual rule editor	28%
Constraint language, business language rule editor	10%
Graphical rule rditor	26%
Spreadsheets based rule editor	12%
Decision trees rule editor	9%
Rule Learning	5%
An IDE that allows to edit, load, debug and run rules	46%

Table 2. Use of tools for the development of rules

The next questions asked participants for the tools used for the development. The results show that almost half of the respondants use an integrated development environment (IDE)[4] [11], . For editing rules the most widely used tools are still textual editors, with 33% and 28% of respondents stating that they use a simple text editor or a textual rule editor with syntax highlighting. With 26% of respondents using them, graphical rule editors are also widespread (see table 2).

[3] However, care should be taken when comparing these numbers, since the sample of the surveys differs considerable: on average the projects discussed [2] were considerable larger.

[4] Such as the Ontoprise's Ontostudio [11], Ilog Rule Studio [12], the Visual Prolog IDE [13] or the SWRL tab of Protege [14].

The results show that the overwhelming majority of rule bases is still created manually; is not learned or generated as some expect. Further it shows that text editors - even simple ones without syntax highlighting - are still used, meaning that even simple typos that could be prevented by more elaborate tools will still happen in practice. Finally the questions about methods used revealed a preference for agile processes and even no development process at all - meaning in particular that tools cannot rely on the availability of formal specifications.

6 Verification, Bugs and Debugging

To gain an overview of the verification state of practice, the survey included a multiple choice question that asked participants to check all V&V tools or methods that they use in their project.

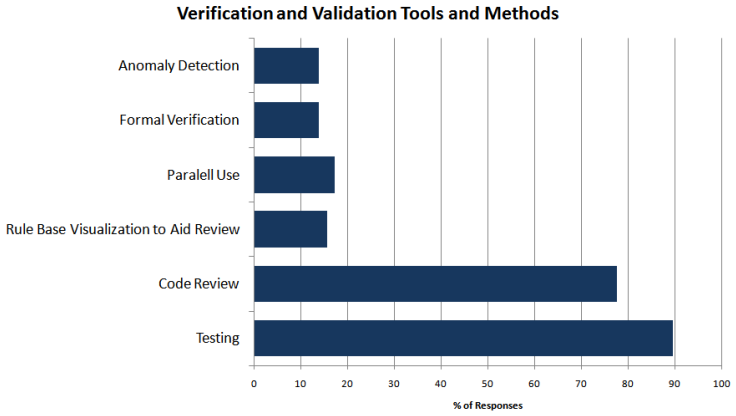

Fig. 2. *The verification and validation methods and tools used, in percent of respondants*

The results show (a summarize is shown in figure 2) that verification is dominated by testing (used by 90%) and code review (used by 78%). 74% of respondents do testing with actual data, 50% test with contrived data. Advanced methods of test organization are used by a minority, with only 31% doing regression testing and 19% doing structural testing with test coverage metrics. Code review is done equally by domain experts (53%) and developers (57%), most projects combine both (73% of projects that perform a code review, have both domain experts and developers do it). The system is used parallel to development in 17% of the projects; in 16% it is used by developers; in 14% by domain experts.

O'Leary [3] posed a similar question to the developers of expert systems in 1991, asking about the validation effort spent on particular methods. In the

average over all responses, he found that most effort is spent on testing with actual data (31% of validation effort), followed by testing with contrived data (17.9%), code review by domain expert (17.6%), code review by developer (13%), parallel use by expert (12%) and parallel use of system by non-expert (7%).

6.1 Debugging Tools

Debugging is dominated by procedural debuggers, i.e. debuggers similar to the ones used in procedural programming; tools that allow to specify breakpoints, to suspend the inference engine and to explore the stepwise execution of the program[5]. 37% of the projects used a command line procedural debugger and 46% a graphical procedural debugger. Explanations are used by almost a quarter (23%) of the respondents for debugging. An overview of these results are shown in figure 3.

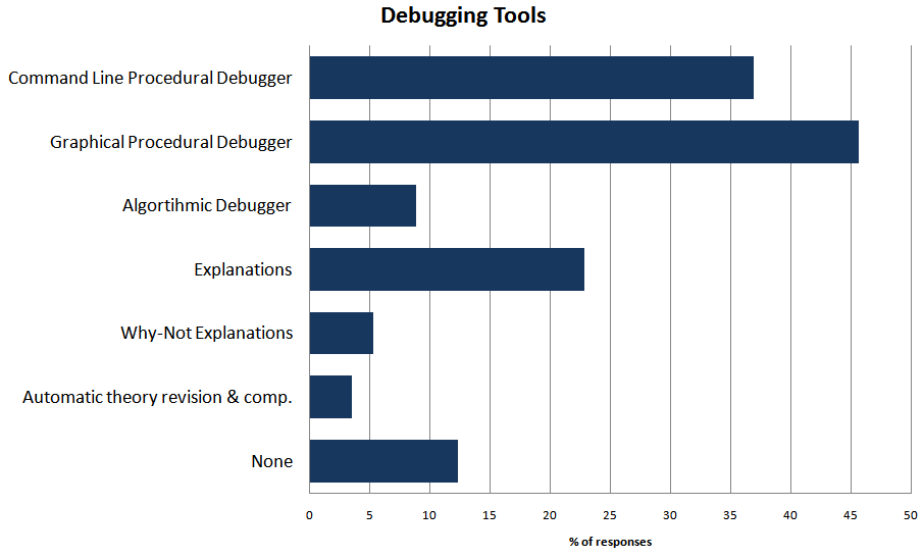

Fig. 3. *Tools used for debugging*

Surprisingly widespread is the use of Algorithmic Debugging [16] and Why-Not Explanations (e.g. [17,18]), considering that to the best knowledge of the author there is no widely available and currently maintained implementation of either of these debugging paradigms. For the systems used by three of the five people that professed to use Algorithmic Debugging (JBoss rules/Drools and Fair Isaac Blaze Advisor) no mentioning of any such debugger could be

[5] An overview and description of the different debugging paradigms for rule based systems can be found in [15].

found and it seems likely that the short explanation for this debugging paradigm given in the survey (*'system tries to identify error by asking user for results of sub computations'*) was insufficient to convey the meaning of this concept. Similarly for Why-Not explanations three of the four respondents use systems for which such no mentioning of such debuggers could be found. The remaining two responses for Algorithmic Debugging and the remaining one for Why-Not explanations use Prolog dialects, where such debuggers have existed/exist.

6.2 Bugs, Symptoms of Faults in the Rule Base

Debugging is the process of tracking down a fault based on error revealing values. The difficulty of this process and the kind of tools that can support it, depends on the error revealing values and how well these allow for the identification of the fault. In the author's experience based on F-logic, most faults in rule based systems cause a query to not return any result (the so called *no-result-case*) or to return fewer results than expected. This stands in contrast to the development with modern object-oriented languages where in many cases at least a stack trace is available. This is problematic because a *no-result-case* gives only very little information for fault localization and means that most explanation approaches are not suitable for debugging (because these rely on the existence of a result). A question was included in the survey about the common symptoms of faults in the rule base to check whether this patterns of error revealing values holds for rule based systems in general.

	frequent	seldom	never
A query/test would not terminate	7	57	30
A query/test did not return any result	38	47	11
A wrong result was returned	53	39	5
A part of the result missing	31	42	20
The rule engine crashed	9	47	38

Table 3. Bugs, symptoms of faults in the rule base in percent of responses. To 100% missing percent: respondents selected *not applicable*

The results (see table 3) show 'wrong results' as the most frequent bug, followed by 'no results' and 'partial results'. Most participants encounter not terminating tests and crashing rule engines only seldom. The results show also that 60% of participants frequently and 34% sometimes encounter a fault showing itself in the failure of the rule base to conclude an expected result/result part. For the developers of the system using *declarative rules* (see section 4), the no-result case is the most frequent bug. These results underline the need for debugging approaches to support users in diagnosing bugs based on missing conclusions.

7 Issues and Comparison to Procedural Programming

On the last page of the survey participants were asked to rank a number of possible issues as *Not an Issue*, *Annoyance* or *Hindered Development*. An average score was obtained for the issues by multiplying the *annoyance* answers with 1, the *hindrance* answers with two and dividing by the number of all answers. The aggregated answers for this question are shown in the table 7[6].

	Average	Not an Issue	Annoyance	Hindrance
Debugging	1	12	28	12
Determining completeness	0.76	18	27	6
Supporting tools missing/immature	0.67	26	17	9
Editing of rules	0.66	24	23	6
Determining test coverage	0.65	25	19	7
Inexperienced developers	0.58	31	13	9
Rule expressivity	0.5	33	12	7
Keeping rules base up to date	0.5	30	19	4
Understanding the rule base	0.47	31	19	3
Runtime performance	0.41	35	14	4
Organizing collaboration	0.41	35	14	4

Table 4. Issues hindering the development of rule based systems. Numbers show the actual number of participants that selected an option. Please note that the 'Rule Expressivity' option was phrased in a way that asked also for things that could not easily be represented, not only things that could not be represented at all.

The results show the issues of debugging, validation and tool support as the most important ones. The issues of probably the largest academic interest - runtime performance and rule expressivity, are seen as lesser problem. This is particulary interesting in the light of the fact that of the 7 survey participants that stated they were hindered by rule expressivity, none used a declarative rule system (for which these questions are debated the loudest).

These findings of verification and validation issues as the most important ones are similar to the finding of Leary [3]. He found that the *potentially biggest problems* were determining the completeness of the knowledge base and the difficulty to ensure that the knowledge in the system is correct.

In a final question participants were asked how a rule base development process compares to the development of a conventional program (created with procedural or object oriented languages) of similar size. A number of properties was given to be ranked with *Rule base superior*, *Comparable*, *Conventional*

[6] Please note that the *Rule Expressivity* option was phrased in a way that asked also for things that could not easily be represented, not only things that could not be represented at all.

program superior and *Don't know*. The aggregated score for each property was determined by subtracting the the number of *conventional program superior* answers from the *rule base superior* answers and dividing the result by the number of answers other than *don't know*.

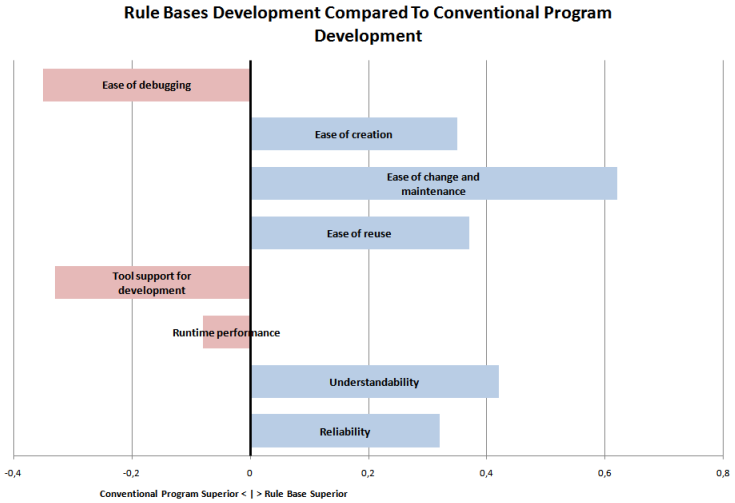

Fig. 4. Participants opinion about the strength and weaknesses of rule base development compared to that of 'conventional' programs. Positive numbers indicate that the majority thought rule bases to be superior, negative numbers that they thought conventional programs to be superior.

The participants of the survey judged rule bases to be superior in most respects. The largest consensus was that rule bases are indeed easier to change and to maintain. Easy of creation, ease of reuse, understandability and reliability are also seen as the strong points of rule based systems. A small majority saw conventional programs as superior in runtime performance; most saw rule bases as inferior in ease of debugging support and tool support for development.

8 Conclusion

The developers of rule based system are indeed seeing the advantages in ease of change, reuse and understandability that were expected from rule based (and declarative) knowledge and program specification. However, there are also issues hindering the development of these systems with verification and validation and the tool support for development topping the agenda.

Particularly debugging is seen most frequently as an issue negatively affecting the development of rule bases and as one area were rule base development is

lacking behind procedural and object oriented program development. This relative difficulty of debugging may be caused either by a lack of refined debugging tools or by intrinsic properties of rule bases that make these hard to debug[7]; in either case it should be a priority topic for researchers and practitioners in the area of rule based systems. The results from the survey further show that many of the older innovative approaches for the debugging of rules are not used widely in practice; resuscitating these and refining them for practical use may be one way to tackle this debugging challenge of rule based systems[8]. That 60% of the projects frequently and another 34% sometimes encounter a fault showing itself in the failure of a rule base to conclude an expected result/result part show that the diagnosis of bugs based on missing conclusions must be a core feature of any new debugging tool.

Missing and immature tools support for the development of rule bases as the other big issue can be seen as another motivation for the already ongoing efforts to standardize rule languages. A tool supporting a standartized (and widely accepted) rule language could be developed with more resources, because its potential market would be larger than one for a tool supporting just one language in a fragmented rule landscape.

Finally this survey together with a literature research reveals a lack of methodological support for the development of rule bases: 38% of the larger projects in the survey use agile or iterative methods; at the same time there is no established agile or iterative methodology for rule based systems. Exploring, describing and supporting the transfer of agile and iterative methods to the development of rule based systems should be one major topic for the future development of rule based systems; notable first movements in this direction are the open sourcing of Ilogs Agile Business Rule Development methodology [19] and the authors own work on the adoption of eXtreme programming for rule development [20].

9 Acknowledgements

The author thanks Hans-Joerg Happel, Andreas Abecker, Michael Erdman, Katharina Siorpaes and Merce Mueller-Gorchs for their support in creating and realizing this survey.

References

1. Zacharias, V.: Rules as simple way to model knowledge: Closing the gap between promise and reality. In: To Appear: Proceedings of the 10th International Conference on Enterprise Information Systems. (2008)
2. Hamilton, D., Kelley, K.: State-of-the-practice in knowledge-based system verification and validation. Expert Systems With Applications **3** (1991) 403–410
3. O'Leary, D. In: Design, Development and Validation of Expert Systems: A Survey of Developers. John Wiley & Sons Ltd. (1991) 3–18

[7] [1] details some hypothesis on what those properties may be
[8] See [15] for an overview including the older debugging approaches

4. Cusumano, M., MacCormack, A., C., Kemerer, F., Crandall, B.: Software development worldwide: The state of the practice. IEEE Software **20** (2003)
5. Zhao, L., Elbaum, S.: A survey on quality related activities in open source. SIGSOFT Software Engineering Notes **25**(3) (2000) 54–57
6. Zhao, L., Elbaum, S.: Quality assurance under the open source development model. Journal of Systems and Software **66** (2003) 65–75
7. Runeson, P., Andersson, C., Thelin, T., Andrews, A., Berling, T.: What do we know about defect detection methods? IEEE Software **23** (2006) 82–90
8. SurveyMonkey: Surveymonkey. http://www.surveymonkey.com/ (2008) (accessed 2008-05-29).
9. Larman, C., Basili, V.: Iterative and incremental development: A brief history. IEEE Computer **June** (2003) 47–56
10. MacCormack, A.: Product-development practices that work. MIT Sloan Management Review (2001) 75–84
11. Ontoprise: Ontostudio. http://www.ontoprise.de/ (2007) (accessed 2007-02-27).
12. Ilog: Ilog business rule management systems. http://www.ilog.com/products/businessrules/ (2008) (accessed 2008-29-04).
13. : Visual Prolog. http://www.visual-prolog.com/ (accessed 2008-08-19).
14. Golbreich, C., Imai, A.: Combining swrl rules and owl ontologies with protege owl plugin, jess and racer. In: Proceedings of the 7th International Protege Conference. (2004)
15. Zacharias, V.: The debugging of rule bases. In: to appear in Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Global (2009)
16. Shapiro, E.: Algorithmic program debugging. PhD thesis, Yale University (1982)
17. Chalupsky, H., Russ, T.: Whynot: Debugging failed queries in large knowledge bases. In: Proceedings of the Fourteenth Innovative Applications of Artificial Intelligence Conference (IAAI-02). (2002) 870–877
18. Becker, M., Nanz, S.: The role of abduction in declarative authorization policies. In: in Proceedings of the 10th International Symposium on Practical Aspects of Declarative Languages (PADL). (2008)
19. Ilog: Agile business rule development. http://www.ilog.com/brms/media/ABRD/ (2008) (accessed 2008-05-31).
20. Zacharias, V.: The agile development of rule bases. In: Proceeedings of the 16th International Conference on Information Systems Development. (2007)