Representing and Solving Rule-Based Decision Models with Constraint Solvers

Jacob Feldman

OpenRules, Inc., 75 Chatsworth Ct.,
Edison, NJ 08820, USA
jacobfeldman@openrules.com

Abstract. This paper describes how constraint solvers could serve as rule engines in the context of modern business decision management systems. Decision models are based on rule families oriented to business users and frequently represented as Excel decision tables. The proposed approach uses exactly the same representation of decision models as a rule engine. The developed Rule Solver loads a decision model from multiple Excel files, generates a constraint satisfaction problem, and then validates it for consistency, diagnosing possible conflicts. Finally, it solves the problem, delivering results using the same terms as business rules. In fact, a user may switch between a rule engine and a constraint solver without changing the rules themselves. Additionally, Rule Solver can find solutions or find an optimal decision when business rules only partially define a problem. Rule Solver is implemented as an advanced component of the popular open source business decision management system "OpenRules".

Keywords: Decision Model, Rule Family, Constraint Satisfaction, Rule Engine, Constraint Solver.

1 Introduction

In recent years, decision management services have become key components of real-world business applications in banking, insurance, healthcare, telecommunication, advertising, and many other industries. They are frequently based on predictive analytics, business rules management, complex event processing, optimization, and other business intelligence technologies. According to IDC [1] the decision management software market is expected to exceed $10B by 2014 – doubling in the five years from 2009. Business rules management systems (BRMS) and rule engines are already "must-have" components for decision management. At the same time, constraint programming (CP) has been successfully used for years to find optimal solutions for complex industrial problems. However, it is only now CP is starting to penetrate the world of business decision support applications. In this paper, we show how existing CP solvers can be effectively used as an execution mechanism for complex decision models.

There are two major implementations of rule engines available on the market today:

1) Inferential rule engines that support a pure declarative representation of business rules;
2) Sequential rule engines that rely on user-defined sequencing of rules and rule families.

The famous Rete algorithm was invented by Charles Forgy almost 40 years ago [2] and it still remains the major foundation for most implementations of inferential rule engines [4], [5], [6], [7]. Sequential rule engines are used by many commercial and open source engines ([20], [21], [8], [9]), which recognized that in many practical applications manual rules sequencing is a preferred mode. Even vendors of the major Rete-based rule engines added special sequential modes to more effectively compete with their sequential counter-parts. However, the newest decision management methodologies ([3], [22]) without insisting on any particular rule engine implementation, require that there should be no rules ordering within rule sets, and between rule sets. These declarative principles simply cannot be supported by sequential rule engines which makes Rete again the dominating implementation approach.

Over the years, Rete went through many enhancements, but until now there were no practical alternatives to Rete for implementation of non-sequential rule engines in the business rules world. At the same time, Prolog-based tools and different constraint solvers (see the list of products in [23]) have been successfully used for years to resolve complex optimization problems defined in terms of rules and constraints. However, these tools usually require a deep understanding of the underlying technologies, and are mainly oriented to software developers, not to subject matter experts. This fact limits the real-world acceptance of these technologies. Making these tools available to business users through their favorite interfaces such as Excel-based decision tables and/or different BRMS rule editors, can bring constraint-based technologies to the practical decision management world.

In this paper we propose a new, constraint-based approach to the implementation of inferential rule engines that can execute rule-based decision models [3]. The proposed approach is functionally similar to Rete-based rule engines in its support of declarative principles for business rules organization. On the one hand, it allows a user to execute decision models using exactly the same business rules without any additional coding. On the other hand, it does not require explicit sequencing of rules inside a rule family or a strict execution order of related rules families. For every input dataset, a constraint-based rule engine executes all related business rules and either infers a decision or diagnoses conflicts among rules and input data. Additionally, it can find solutions when business rules only partially define a problem. When an optimization objective is defined by the rules, it also can find an optimal decision instead of forcing a user to specify enormous amount of rules to compare different decisions. The proposed approach is implemented as a component of the open source business decision management system “OpenRules” [8], in which it is called “Rule Solver”. We will use this name throughout this paper when referring to the proposed approach and its implementation.

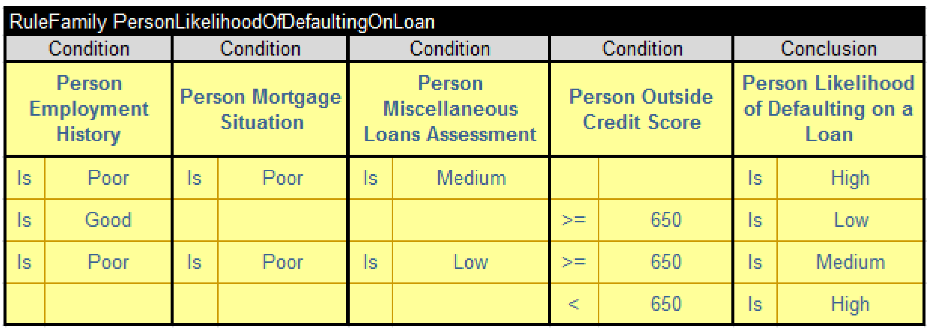
2 The Decision Model

Rule Solver does not deal with any particular rule language, but rather executes rule-based decision models created by business users in accordance with the methodological approach known as "The Decision Model" [3]. The Decision Model was introduced two years ago by Barbara von Halle and Larry Goldberg and quickly gained popularity as a practical methodology for developing Business Decision Management Systems (BDMS) for large financial services, insurance, health care, and other industries. According to the authors, "The decision model is a representation of fact-based business logic within a scope of a single business decision". Examples of such decisions are "Determine Loan Eligibility", "Define Insurance Premium" or "Determine Medical Treatment". The Decision Model is oriented to subject matter experts (business analysts who are not software developers) providing them with a strictly defined notation, as well as concepts and principles that they should follow to build maintainable decision support systems. The Decision Model is defined as technology agnostic, meaning that different BRMS products may provide different implementations of the same decision models.

The Decision Model is defined in [3] as "an intelligent template for perceiving, organizing, and managing business logic behind a business decision (specifically, a representation of business logic statements that together lead to a single business decision, and which complies with the 15 Decision Model principles)". The rigor of the Decision Model is embodied in these 15 principles that are divided into structural, declarative, and integrity principles. In particular, they specify how to organize and to connect Rule Families.

2.1 Rule Families

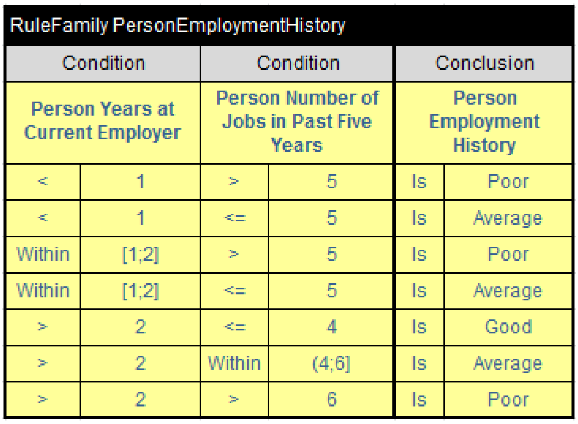
Rule Families are the heart of the Decision Model and they are defined as traditional decision tables but with certain limitations. A Rule Family is a decision table that consists of 0 or more conditions and only one conclusion. If there is more than one condition, then all of them are connected by the logical operator "AND". Examples of Rule Families are shown in Figures 1 and 2.



RuleFamily PersonLikelihoodOfDefaultingOnLoan									
Condition		Condition		Condition		Condition		Conclusion	
Person Employment History		Person Mortgage Situation		Person Miscellaneous Loans Assessment		Person Outside Credit Score		Person Likelihood of Defaulting on a Loan	
Is	Poor	Is	Poor	Is	Medium			Is	High
Is	Good					>=	650	Is	Low
Is	Poor	Is	Poor	Is	Low	>=	650	Is	Medium
						<	650	Is	High

Fig. 1. Rule Family "PersonLikelihoodOfDefaultingOnLoan"

This Rule Family consists of four AND'ed condition columns and a single conclusion column. Each column is associated with one fact type (e.g. "Person Employment History"). Multiple rows (rules) specify the fact using an operator (e.g."<=" or "Is") and a value (e.g. "Poor").



RuleFamily PersonEmploymentHistory

Condition		Condition		Conclusion	
Person Years at Current Employer		Person Number of Jobs in Past Five Years		Person Employment History	
<	1	>	5	Is	Poor
<	1	<=	5	Is	Average
Within	[1;2]	>	5	Is	Poor
Within	[1;2]	<=	5	Is	Average
>	2	<=	4	Is	Good
>	2	Within	(4;6]	Is	Average
>	2	>	6	Is	Poor

Fig. 2. Rule Family "PersonEmploymentHistory"

Conditions and conclusions may use different business fact types shared by different Rule Families. The fact types used in conditions of one Rule Family may be defined by conclusions of other Rule Families. For example, Figure 2 shows a Rule Family that specifies the value for the fact "Person Employment History" used by the dependent Rule Family in Figure 1. This is an example of so called "inferential relationships". According to the "Declarative Inferential Relationship" principle [3], there should be no implied sequence in the path among Rule Families related through such inferential relationships.

The Decision Model includes other important principles that support the integrity of Rule Families, among which are some that are especially important for automatic execution of the Decision Model.

The "Declarative Body" principle states that "the entries in the body of a Rule Family are unordered" [3]. In particular, it means that a user may insert new rules into a Rule Family without worrying about any particular order. This principle imposes very strong requirements not only upon the rule engine that will execute the Decision Model, but also on the designer of Rule Families. It excludes the (sometimes very convenient) ability override rules and forces a Rule Family author to consider almost all possible combinations among condition values.

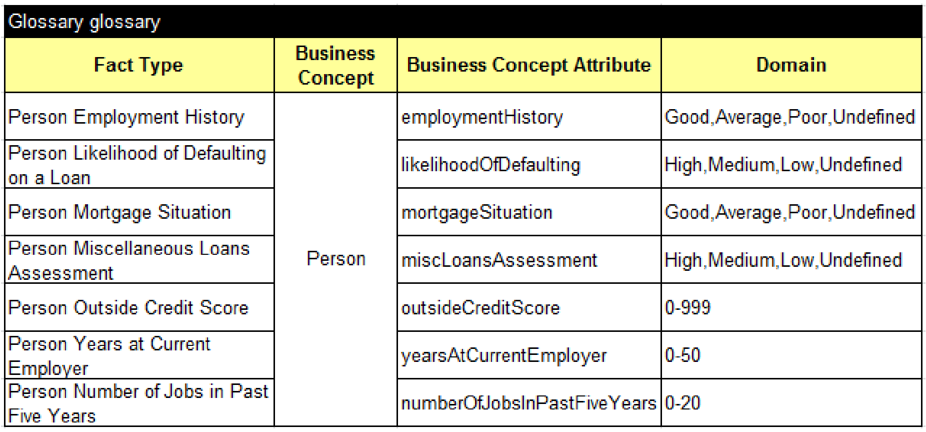
The "Rule Family Consistency" principle states that "a Rule Family should be free of inconsistencies such as overlapping conditions or more than one conclusion" [3]. The conditions need to cover only a subset of the fact type's domains that are within scope of a Rule Family. At the same time, a Rule Family must result in at least one conclusion value for any set of valid input values for condition fact types.

2.2 Business Glossary

Fact types used by all Rule Families are collected in one special table called the “Business Glossary”. Usually a glossary defines the following information about fact types:

- Business names of the fact types defined exactly in the same way they are used in Rule Families;
- Business Concepts to which these fact types belong;
- Domains with all possible values of the fact types;
- Technical names of fact types for integration of the Decision Models with actual business object models used by programmers.

Figure 3 shows an example of the business glossary for rule families presented in Figures 1 and 2.



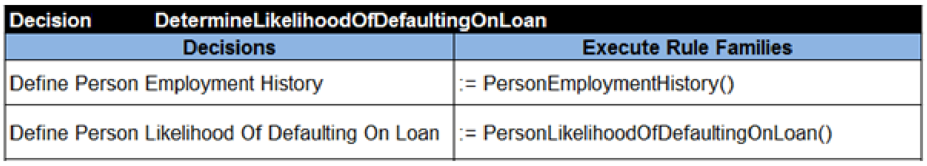
Glossary glossary

Fact Type	Business Concept	Business Concept Attribute	Domain
Person Employment History	Person	employmentHistory	Good,Average,Poor,Undefined
Person Likelihood of Defaulting on a Loan		likelihoodOfDefaulting	High,Medium,Low,Undefined
Person Mortgage Situation		mortgageSituation	Good,Average,Poor,Undefined
Person Miscellaneous Loans Assessment		miscLoansAssessment	High,Medium,Low,Undefined
Person Outside Credit Score		outsideCreditScore	0-999
Person Years at Current Employer		yearsAtCurrentEmployer	0-50
Person Number of Jobs in Past Five Years		numberOfJobsInPastFiveYears	0-20

Fig. 3. Example of a Business Glossary

2.3 Top-Down Design

The Decision Model promotes a top-down design approach that starts with a top-level decision which can be described through sub-decisions and their associated Rule Families. For the above examples of Rule Families, the proper Decision table is presented in Figure 4.

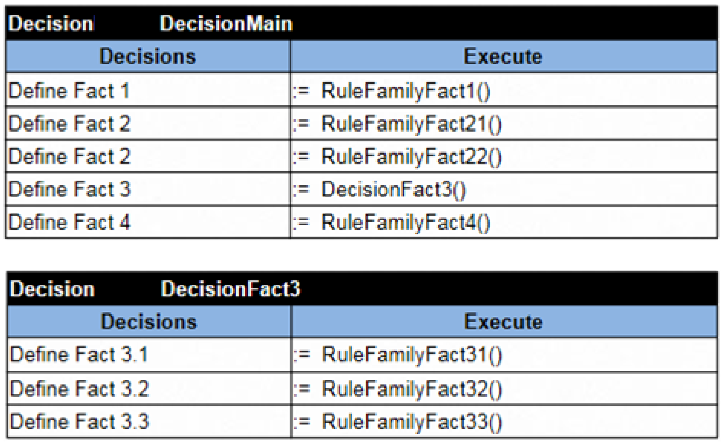


Decision DetermineLikelihoodOfDefaultingOnLoan

Decisions	Execute Rule Families
Define Person Employment History	:= PersonEmploymentHistory()
Define Person Likelihood Of Defaulting On Loan	:= PersonLikelihoodOfDefaultingOnLoan()

Fig. 4. The Decision Model “DetermineLikelihoodOfDefaultingOnLoan”

The Decision Model can be considered as a hierarchy of decisions presented in a graphical form [3] or a tabular form [8]. For example, using the table of the type "Decision" we may present a top-level decision and its sub-decisions in Excel tables similar to the ones on Figure 5.



Decision	DecisionMain
Decisions	**Execute**
Define Fact 1	:= RuleFamilyFact1()
Define Fact 2	:= RuleFamilyFact21()
Define Fact 2	:= RuleFamilyFact22()
Define Fact 3	:= DecisionFact3()
Define Fact 4	:= RuleFamilyFact4()

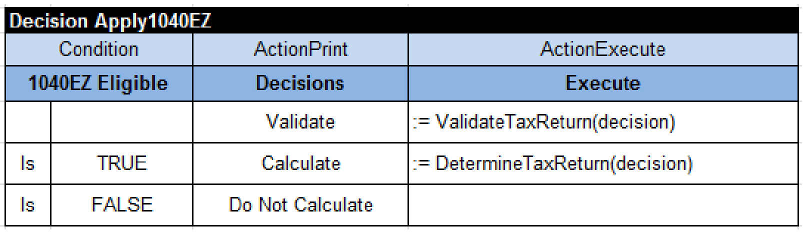
Decision	DecisionFact3
Decisions	**Execute**
Define Fact 3.1	:= RuleFamilyFact31()
Define Fact 3.2	:= RuleFamilyFact32()
Define Fact 3.3	:= RuleFamilyFact33()

Fig. 5. Decisions and Sub-Decisions

The first table specifies a top-level decision using 5 sub-decisions and Rule Families that implement their business logic. For instance, the decision "Define Fact 2" is defined by two Rule Families "RuleFamilyFact21" and "RuleFamilyFact22". At the same time, the decision "Define Fact 3" is defined using a separate decision table "DecisionFact3".

According to the Decision Model [3], there should be no inferential dependencies among inferentially related Rule Families. Correspondingly, the order of fact definitions inside the above decision tables should not matter during the execution of these models.

The decision can be also defined through other decisions using different conditions. For example, Figure 6 demonstrates a situation when the first sub-decision validates your data and the second sub-decision executes complex calculations but only if the data validation was successful.

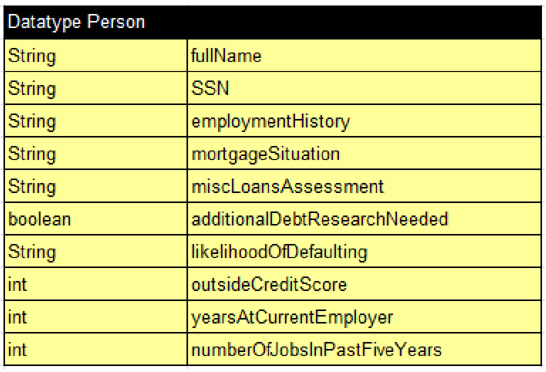


Decision Apply1040EZ			
Condition		ActionPrint	ActionExecute
1040EZ Eligible		**Decisions**	**Execute**
		Validate	:= ValidateTaxReturn(decision)
Is	TRUE	Calculate	:= DetermineTaxReturn(decision)
Is	FALSE	Do Not Calculate	

Fig. 6. Conditional Sub-Decisions

2.4 Test Cases and Real Data

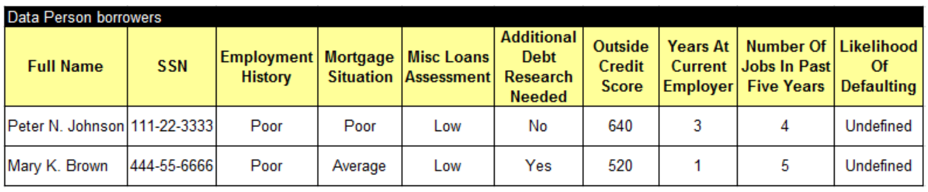
The test cases like Rule Families and all other components of the Decision Model can be defined by business people directly in Excel. Figure 7 shows an Excel table that defines a data type for the business concept “Person” (defined in the glossary in Figure 3 above.)



Datatype Person	
String	fullName
String	SSN
String	employmentHistory
String	mortgageSituation
String	miscLoansAssessment
boolean	additionalDebtResearchNeeded
String	likelihoodOfDefaulting
int	outsideCreditScore
int	yearsAtCurrentEmployer
int	numberOfJobsInPastFiveYears

Fig. 7. An example of a Datatype table used for Decision Model testing

Instead of an Excel-based data type, we may use a regular Java class Person that is defined as a Java bean by the Java application in which this Decision Model is going to be incorporated. Figure 8 shows an Excel table that contains concrete test instances of type Person called “borrowers”.



Data Person borrowers

Full Name	SSN	Employment History	Mortgage Situation	Misc Loans Assessment	Additional Debt Research Needed	Outside Credit Score	Years At Current Employer	Number Of Jobs In Past Five Years	Likelihood Of Defaulting
Peter N. Johnson	111-22-3333	Poor	Poor	Low	No	640	3	4	Undefined
Mary K. Brown	444-55-6666	Poor	Average	Low	Yes	520	1	5	Undefined

Fig. 8. An example of a Data table with test instances

When the Decision Model is integrated with a Java or .NET application, actual data instances can be used by the same Decision Model without any changes in the business logic.

The described components of the Decision Model “DefinePersonLikelihoodOf DefaultingOnLoan” are sufficient for Rule Solver to either execute this model inferring a correct decision or to inform a user about possible inconsistencies.

3 Constraint-Based Implementation

Formally, the Decision Model can be described as follows:

- There is a set of business objects $X = \{ X_1, \ldots, X_n \}$
- Each business object X_i has fact types $F_i = \{ f_1, \ldots, f_m \}$ with possible values $D_j = \{ v_{j1}, \ldots, v_{jk} \}$ for each property f_j
- There is a set of rules $R = \{ R_1, \ldots, R_r \}$, where a rule R_k defines relationships between different fact types by specifying the allowed combinations for all fact types in that rule.

The rules from set R are grouped into Rule Families that are organized in accordance with the Decision Model principles. Execution of the Decision Model should cause the assignment of values to all fact types that satisfy the rules.

This representation demonstrates that the Decision Model is quite similar to a typical constraint satisfaction problem (CSP) where fact types F_i correspond to constrained variables with known domains D_j and where rules R_k correspond to conditional constraints.

Thus, to use a constraint solver as a rule engine that is capable of executing a decision model compliant with The Decision Model principles, we need a tool that can do the following:

- read the decision model created by business analysts directly from the rule repository (i.e. from a set of Excel files) without requiring the manual transfer of the model into any CP language
- generate a CSP that corresponds to this decision model
- validate the consistency of the model by checking the consistency of the generated CSP and point to possible conflicts using the business terms of the initial decision model
- execute the decision model against concrete data using the following steps:
  - instantiating all constrained variables for which input data is defined
  - posting all constraints that correspond to rules from all Rule Families
  - if constraint propagation by itself does not find single values for all fact types (does not instantiate all constrained variables), then run a constraint solver's search strategy that finds one or more solutions.

OpenRules Rule Solver provides the described functionality by downloading all decision model tables directly from Excel files and then automatically generating and solving a corresponding constraint satisfaction problem. Rule Solver is based on the standard Java Constraint Programming API defined by the Java Specification Request (JSR) 331 [19]. The use of the JSR 331 allows a user to not commit to a particular CP vendor and to try different underlying solvers before choosing the most suitable one based on its technical and business applicability. A user may switch between different underlying CP solvers compliant with the JSR 331 without any changes in the code. Below we will use a simple example to describe how Rule Solver works.

3.1 Fact Types as Constrained Variables

First, Rule Solver creates a CSP instance using the class RuleSolver inherited from the JSR 331 class Problem:

```
RuleSolver rs = new RuleSolver();
```

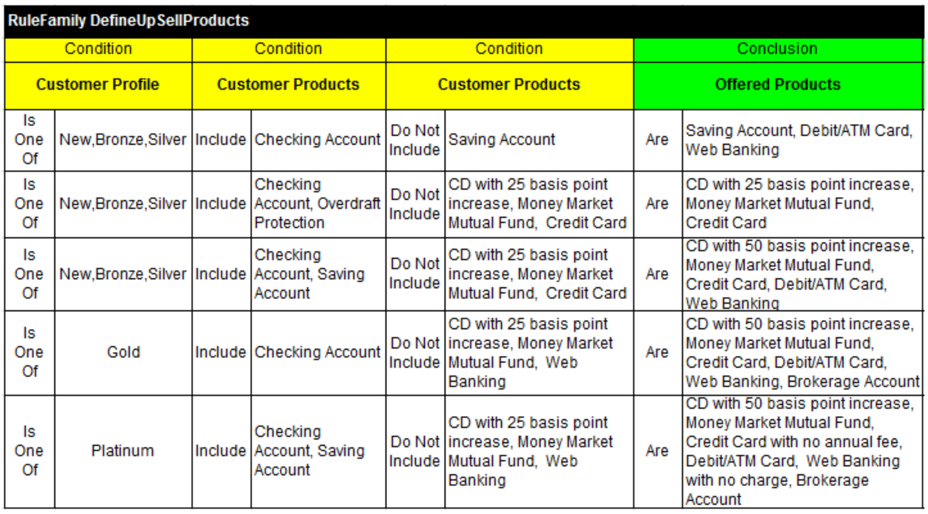
Then it iterates through the glossary and for each fact type it creates a constrained variable of one of the following types:

- Var for integer constrained variables
- VarBool for Boolean constrained variables
- VarReal for real constrained variables
- VarString for string constrained variables.

Rule Solver automatically converts fact type domains from the glossary, to the domains of the constrained variables as they are specified by JSR 331. While the glossary does not specify a particular type of the fact types, the concrete types of variables are defined based on the provided data instances. For example, a constrained variable that corresponds to the fact type "Person Outside Credit Score" will be created using the following JSR 331 method:

```
   rs.variable("Person Outside Credit Score", 0, 999);
```

The Decision Model may use aggregated fact types, for example arrays of strings. Consider the Rule Family in Figure 9 that specifies up-selling rules.



RuleFamily DefineUpSellProducts							
Condition		Condition		Condition		Conclusion	
Customer Profile		Customer Products		Customer Products		Offered Products	
Is One Of	New,Bronze,Silver	Include	Checking Account	Do Not Include	Saving Account	Are	Saving Account, Debit/ATM Card, Web Banking
Is One Of	New,Bronze,Silver	Include	Checking Account, Overdraft Protection	Do Not Include	CD with 25 basis point increase, Money Market Mutual Fund, Credit Card	Are	CD with 25 basis point increase, Money Market Mutual Fund, Credit Card
Is One Of	New,Bronze,Silver	Include	Checking Account, Saving Account	Do Not Include	CD with 25 basis point increase, Money Market Mutual Fund, Credit Card	Are	CD with 50 basis point increase, Money Market Mutual Fund, Credit Card, Debit/ATM Card, Web Banking
Is One Of	Gold	Include	Checking Account	Do Not Include	CD with 25 basis point increase, Money Market Mutual Fund, Web Banking	Are	CD with 50 basis point increase, Money Market Mutual Fund, Credit Card, Debit/ATM Card, Web Banking, Brokerage Account
Is One Of	Platinum	Include	Checking Account, Saving Account	Do Not Include	CD with 25 basis point increase, Money Market Mutual Fund, Web Banking	Are	CD with 50 basis point increase, Money Market Mutual Fund, Credit Card with no annual fee, Debit/ATM Card, Web Banking with no charge, Brokerage Account

Fig. 9. An example of a Rule Family with aggregated fact types

Here the fact type "Customer Products" is an array of strings that represents banking products that a customer already has. The fact type "Offered Products" represents additional products a bank is ready to offer to a customer based on the customer's profile and the set of existing products. Rule Solver represents such fact types using constrained set variables (the JSR 331 standard type VarSet) and posts the proper constraints defined on these set variables.

3.2 Rules as Conditional Constraints

While processing the Decision Model tables and related Rule Families, Rule Solver creates conditional constraints in the form:

```
conditionConstraints.implies(conclusionConstraint)
```

where "conditionConstraints" are accumulated by using the method "and" defined for the JSR-331 class Constraint. For example, the rule

```
IF Person Years at Current Employer < 1
AND Person Number of Jobs in Past Five Years > 5
THEN Person Employment History = Poor
```

may be implemented in Java using the JSR-331 interface:

```
Var var1 = rs.getVar("Person Years at Current Employer");
Constraint c1 = rs.linear(var1, "<", 1);

Var var2 = rs.getVar("Person Number of Jobs in Past Five Years");
Constraint c2 = rs.linear(var2, ">", 5);

Constraint conditionConstraints = c1.and(c2);

VarString var3 = rs.getVarString("Person Employment History");
Constraint conclusionConstraint = rs.linear(var3, "=", "Poor");

rs.add(conditionConstraints.implies(conclusionConstraint));
```

However, Rule Solver never generates Java or any other code. Instead, at run-time, it simply creates an instance of different JSR 331 classes and adds them to the already created constraint satisfaction problem (an instance of the class "RuleSolver"). All instances of constrained variables and constraints are added to the problem "on the fly". How does Rule Solver actually generate this CSP? It does not use any special code parser and/or generator. Instead, it effectively relies on the existing OpenRules's templatization mechanism.

OpenRules uses different rule templates to implement all tables included into the default (not constraint-based) implementation of the Decision Model. Such tables as "Decision", "RuleFamily", and "Glossary" are actually implemented based on rule templates defined in several configuration Excel files. For example, the file "RuleFamilyExecuteTemplates.xls" contains a template with the fixed name "RuleFamilyTemplate" and all Rule Families are created based on it. This template is a regular OpenRules "single-hit" rules table. It means that it is trying to execute rules in top-down order by evaluating their conditions. When all conditions inside a rule are evaluated as TRUE, the rule's conclusion (and possibly other related actions) will be executed and all remaining rules will be ignored.

Rule Solver provides another configuration file “RuleFamilySolveTemplates.xls” that substitutes the template “RuleFamilyTemplate” with a different implementation that is actually a special “multi-hit” rules table. This rule table executes all rules inside every Rule Family. However, instead of evaluating rule conditions it simply creates new constraints similar to `c1` and `c2` above, and then “AND”s all previously defined conditions similarly to `c1.and(c2)`. Thus, all conditions from one rule will form a constraint `conditionConstraints` described in the previous example. Then the conclusion will be converted to the `conclusionConstraint` that is based on the constrained variable associated with the conclusion’s fact type, operator, and value. Finally, Rule Solver creates a new constraint `conditionConstraints.`**`implies`**`(conclusionConstraint)` and adds it to the problem. According to the JSR 331, this constraint states that if the constraint `conditionConstraints` is satisfied then the constraint `conclusionConstraint` also should be satisfied.

While the “RuleFamilyTemplate” may contain more complicated constructions, the very fact that the generated CSP can be reconfigured by simply changing the template directly in Excel, makes this approach extremely flexible, extensible, and customizable for different needs.

3.3 Consistency Validation

Rule Solver provides a user (a business analyst who creates and maintains the rules within the Decision Model) with several consistency validation modes.

Mode 1. Validate rules consistency. In this mode, Rule Solver simply posts all already added constraints one-by-one with constraint propagation turned on. If a constraint fails to be posted, a user will be notified that the associated rule is in conflict with the rules, for which the corresponding constraints were previously posted.

Mode 2. Validate rules consistency using test data. In this mode, before posting any constraints, Rule Solver is trying to instantiate constrained variables, for which the proper test data is defined. If an error occurs, the user will be informed about invalid data. If there are no errors in the data, then Rule Solver will try to post all automatically defined constraints for all involved Rule Families. The constraints again will be posted one-by-one with constraint propagation on. If a constraint fails to be posted, the user will be notified that the associated rule is in conflict with previously posted constraints (rules). To help a user find the reason for the conflict, Rule Solver will display the current state of all instantiated (or only partially instantiated) variables corresponding to the fact types.

Mode 3. Validate rules completeness. If the previous modes do not produce errors, Rule Solver validates whether the Rule Family consistency principle has been satisfied. This principle states that “a Rule Family must result in at least one conclusion value for any set of valid input values.” So, Rule Solver determines whether all constrained variables have been instantiated for all conclusion fact types. If all Rule Families have been fully defined in accordance with the Decision Model principles, constraint propagation will be sufficient to determine a decision.

In real-world decision management environments, not all Rule Families are created at the same time, and as a result, the Decision Models frequently can be found to be incomplete, but still producing satisfactory results on the data that have been used. However, a user may not even know about potential problems with such decision models. In this case Rule Solver provides a user with a list of fact types that remain undefined. These fact types are displayed with all remaining possible values from their domains that may have been reduced. This information prompts a user to identify which rules should be extended to cover the remaining situations.

It is important to emphasize that Rule Solver validates consistency of not only one Rule Family but of all Rule Families included in the Decision Model and related through inferential relationships! Otherwise, it may be extremely difficult for the author of the rules to predict how adding or modifying a single rule in one Rule Family may affect the execution logic of dependent Rule Families. There could be hundreds and even thousands of Rule Families in real-world decision support applications, and it would be humanly impossible to maintain their consistency relying only on test cases. Rule Solver helps business users to keep their entire Decision Models in a consistent state.

3.4 Finding Solutions for Partially Defined Decision Models

In some practical situations a creator of a decision model cannot strictly specify all possible combinations of values for all conditions. Instead, users frequently cover only a subset that according to the Decision Model is "within scope" [3]. Unfortunately, this means that such incomplete Decision Models will not produce any decision for certain data sets. Rule Solver helps a user to deal with this problem by simply executing the default search strategy after all data and rule constraints have been posted. Rule Solver offers a user the following options:

- find a single solution that satisfies all currently specified rules (constraints);
- find several solutions by specifying a limit for the maximal number of solutions or by limiting the amount of time during which solutions may be calculated;
- find a solution that minimizes (or maximizes) an optimization criteria defined by a user as an expression of the existing fact types.

In this manner, Rule Solver goes well beyond traditional inference rule engines by empowering business users with a new functionality without forcing them to specify rules for all possible situations.

4 Related Work and Future Development

The integrated use of business rules and constraint programming has been described in several works [12], [13], [15], [16]. In most cases, business rules are used to define a specific business problem and then CP is used to solve the problem. The early versions of Rule Solver [8] offered generic rule templates that allowed a user to directly use CP concepts represented through business rules. The closest approach to the one described in this paper was proposed in [14] where an automatically generated

CSP was used to validate the consistency of a stand-alone classification rules table. However, that previous approach did not validate the consistency of multiple decision tables and, more importantly, was not able to execute the rules. To the best of our knowledge there were no known software products that use a constraint solver as a truly declarative (not sequential) rule engine.

With the Decision Model gaining in popularity as a decision management methodology, several vendors extended their product offerings to enable the creation and management of decision models in accordance with [3]. Such products as SAPIENS [9], interGREAT [10], and RuleGuide [11] provide powerful graphical interfaces for the creation and validation of decision models and OpenRules 6.0.1 [8] released in March 2011 became the first business rules product that allows business users to define and execute their decision models.

The approach described in this paper has been implemented as an advanced Rule Solver component of the OpenRules BDMS [8]. It allows a user to check if custom decision models are compliant with the Decision Model principles [3]. In cases when these principles have been violated, Rule Solver shows a user how to improve their models. In addition to traditional rule engine functionality, Rule Solver can deal with practical situations where a custom decision model does not cover all possible combinations of fact types. Instead of simply failing to find a decision, Rule Solver can offer a user either a feasible or an optimal solution.

The proposed approach has not yet been tested on large industrial problems. It was also not possible to do a performance comparison with Rete-based rule engines since there are still no available Rete engines that implement The Decision Model. However, the automatically generated CSPs are simple from the CP perspective, they are highly constrained and do not require an optimal search for many practical situations. When we conducted performance tests using relatively small rule sets and the default search strategies of several open source CP solvers, the high performance results came as no surprise. CP solvers have proven records of solving much more complex constraint satisfaction problems to compare with ones that automatically generated from the decision models. However, we plan to conduct further tests using more complex rules with multiple inferential dependencies and data coming from real-world projects.

From a practical perspective, the performance should not be an issue as it is not an issue for most existing rule engines. What is especially important is the fact that the Decision Models can be executed "as is" without any conversion of the original Excel-based rule families (created by business users) and without additional coding. The creators of business rules, who usually have no idea about Rete or any other rule engine algorithm, do not have to know anything about CP either. They may continue to use only business terms to define their business logic and the system will communicate with them in the same terms. As a result, business users can test and maintain their decision models themselves without help from software developers. The same decision model can be used with Rule Solver to validate its consistency, but then a user may switch back to a conventional rule engine to execute the model.

OpenRules plans to extend Rule Solver by covering more types of business facts with more operators (and related constraints) defined on these facts. We also plan to add the ability to minimize rule violations in accordance with the approaches described in [17] and [18].

Since Rule Solver's implementation is based on the JSR-331 standard [19], it remains independent of the underlying CP solvers. It also allows any JSR-331 compliant CP solver to use Rule Solver as a front-end for integration with business rules products. At the same time, different BRMS vendors may use the proposed approach to extend their product offerings by adding constraint-based rule engines.

References

1. Worldwide Decision Management Software 2010-2014 Forecast: A Fast-Growing Opportunity to Drive the Intelligent Economy. IDC Report for December 2010 (2010), `http://www.idc.com/getdoc.jsp?containerId=226244`
2. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19, 17–37 (1982)
3. von Halle, B., Goldberg, L.: The Decision Model: A Business Logic Framework Linking Business and Technology. Auerbach Publications/Taylor & Francis Group, LLC (2009)
4. IBM WebSphere ILOG JRules, `http://www-01.ibm.com/software/integration/business-rulemanagement/jrules/`
5. FICO Blaze Advisor business rules management, `http://www.fico.com`
6. JESS, the Rule Engine for the Java platform, `http://jessrules.com`
7. Drools, The Business Logic Integration Platform, `http://www.jboss.org/drools`
8. OpenRules, Open Source Business Decision Management System, `http://openrules.com`
9. Sapiens International Corporation N.V, `http://www.sapiens.com`
10. inteGREAT Enterprise 2010, `http://www.edevtech.com/index.html`
11. RuleGuide, New Wisdom Software, `http://www.newwisdomsoftware.com`
12. Bousonville, T., Focacci, F., Le Pape, C., Nuijten, W., Paulin, F., Puget, J.F., Robert, A., Sadeghin, A.: Integration of rules and optimization in plant powerops. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 1–15. Springer, Heidelberg (2005)
13. Feldman, J., Korolov, A., Meshcheryakov, S., Shor, S.: Hybrid use of rule and constraint engines, Patent no: WO/2003/001322, World Intellectual Property Organization
14. Feldman, J., Korolov, A., Meshcheryakov, S., Shor, S.: Consistency validation for complex classification rules. Patent no: WO/2003/017060, World Intellectual Property Organization
15. Feldman, J., Freuder, E.: Integrating business rules and constraint programming technologies for EDM. In: The 11th International Business Rules Forum (2008)
16. van der Krogt, R., Feldman, J., Little, J., Stynes, D.: An Integrated Business Rules and Constraints Approach to Data Centre Capacity Management. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 568–582. Springer, Heidelberg (2010)
17. O'Sullivan, B., Feldman, J.: Using hard and soft rules to define and solve optimization problems. In: The 12th International Business Rules Forum (2009)
18. Feldman, J.: Rules Violations and Over-Constrained problems. October Rules Fest (2009)
19. Java Request Specification (JSR) 331: Constraint Programming API. Java Community Process, `http://www.jcp.org/en/jsr/detail?id=331`
20. Corticon, Business Rules Management System, `http://corticon.com`
21. Visual Rules, Business Rules Management System, `http://visual-rules.com`
22. Ross, R.G.: Decision Analysis Using Decision Tables and Business Rules, `http://www.brsolutions.com/b_decision.php`
23. ACP, Association for Constraint Programming System, `http://www.4c.ucc.ie/a4cp`