RULES FEST

Using Constraint Solvers as Inference Engines to Validate and Execute Rules-Based Decision Models

JBoss® by Red Hat

FICO™

VisionArts communications
strategic thinking • creative action

Jacob Feldman, Ph.D.

Founder & CTO, OpenRules, Inc.

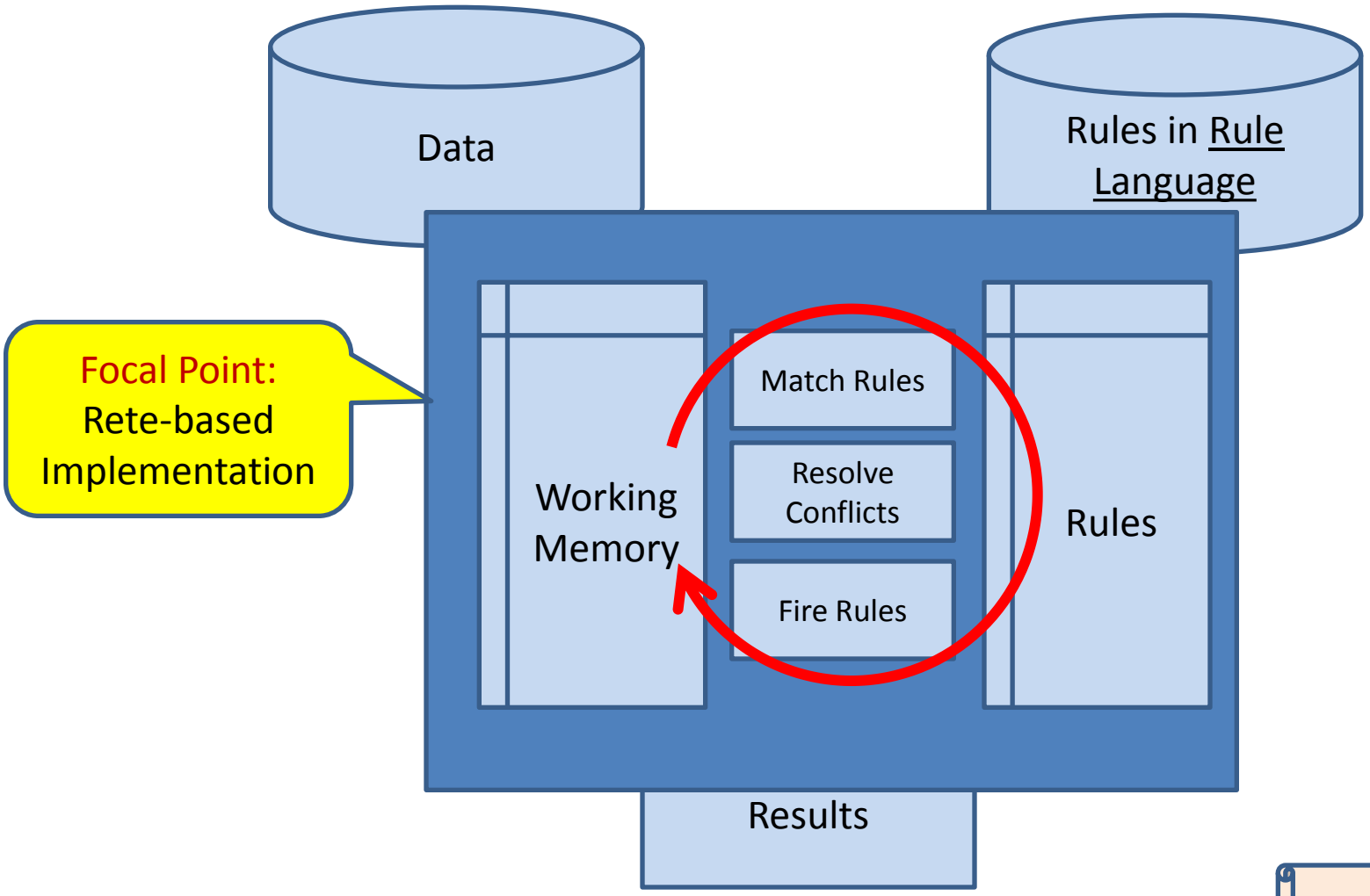
JSR-331 Specification Lead

OPEN RULES

Two Types of Rule Engines

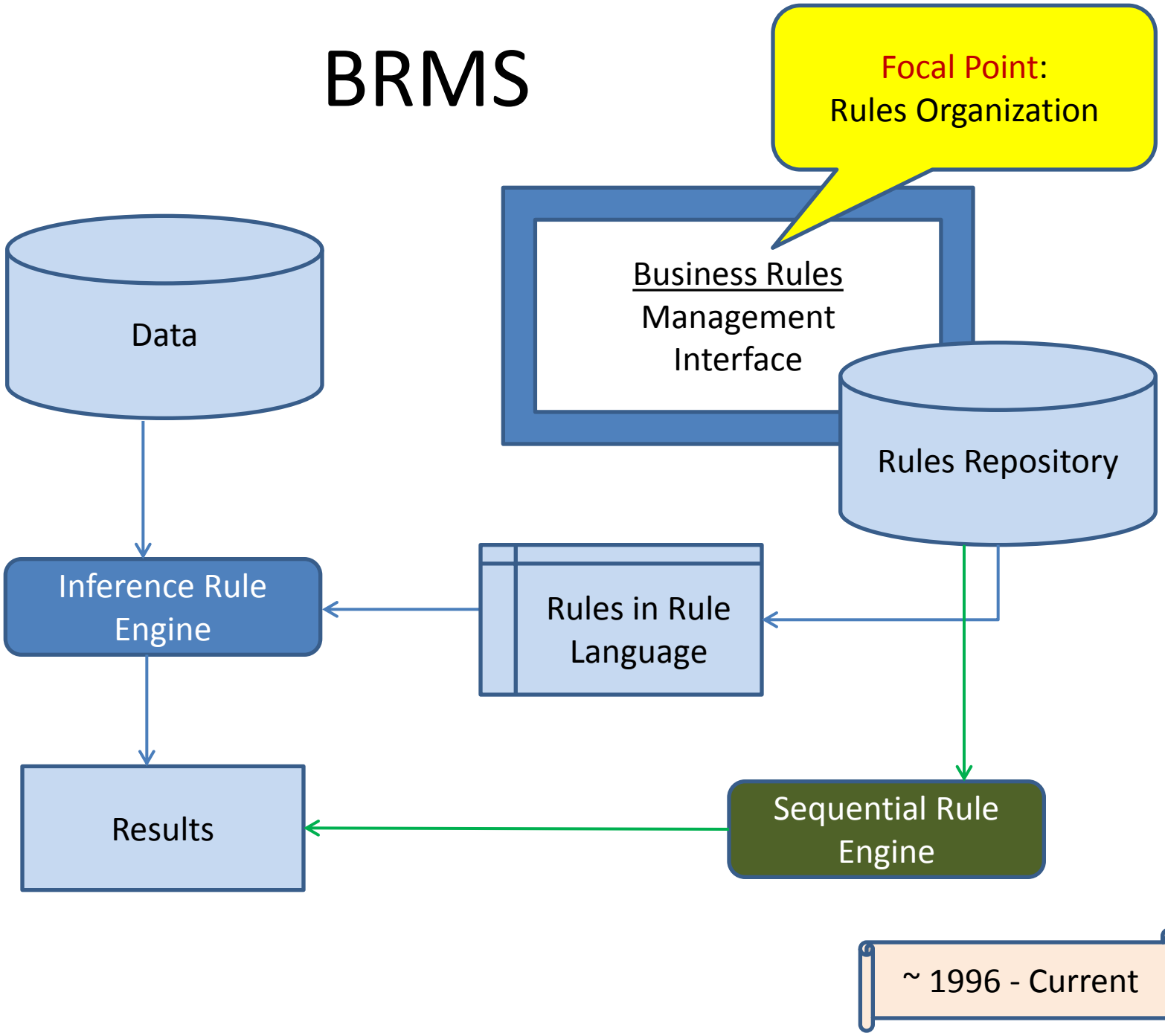
- **Inferential** Rule Engines
  - support a pure declarative representation of business rules
  - Rete-based
- **Sequential** Rule Engines
  - rely on user-defined sequencing of rules and rule families

Rule Engines



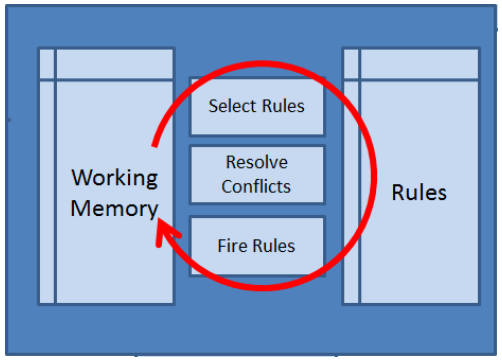
~ 1982-1995

BRMS



Inference Engines: Rete Domination

- It works!
- A great algorithm! Keeps improving…
- A good platform for other systems (e.g. JBoss CEP, Planner)
- Why no alternatives?
  - The best within its framework:
  - Multiple commercial and open source implementations
  - No needs for alternative
  - Inference frequently is not needed



From BRMS to BDMS: Decision Management Systems

- The newest Decision Modeling approach specifies such an organization of decisions and supporting rules that allows us to completely automate their execution (without coding!)
- Real orientation to Business Users
- Current Decision Modeling Efforts:
  - **OMG DMN** (Decision Modeling Notation)
  - **The Decision Model** from KPI

~ 2007 - Current

Rule Family Examples

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 Po				Is High
Is Go				Is Low
Is Po				Is Medium
				Is High

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

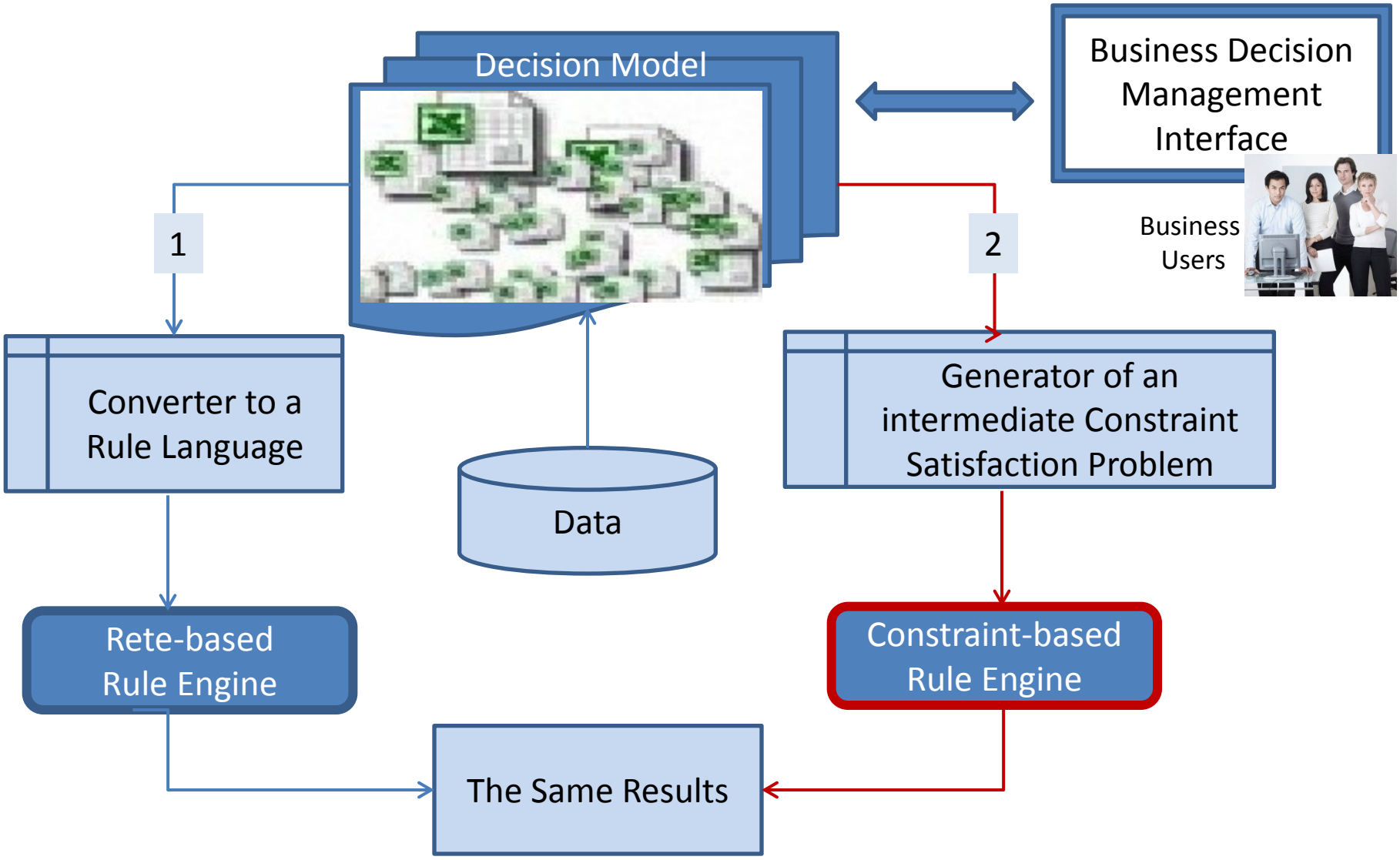
Rules: To Order or Not to Order

- The Decision Model promotes two important principles:
  - The order of Rules inside Rule Families should not matter. In particular, rule overrides are not allowed
  - Rule Families (even when they are inferentially linked) can be defined in any order
- Obviously, sequential rule engines cannot be used to satisfy these principles

Implementing Decision Models

- Only Rule Engines with true inference capabilities can support these principles
- It brings a new incentive to implement inference engines that are capable to execute decision models
- Question: Is it necessary to convert a decision model to a rule language that can be executed by a Rete-based engine?

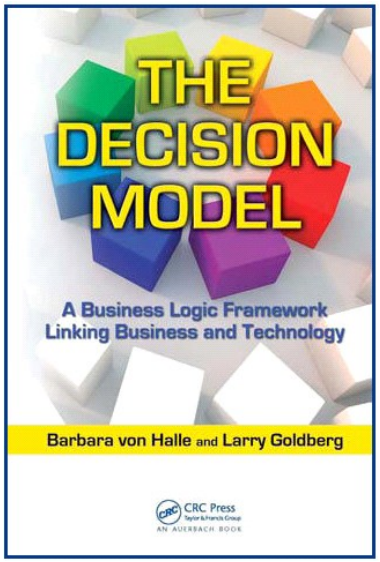
Two Ways of Executing Decision Models



Constraint-based Rule Engine

- Constraint-based Programming (CP) and Rules-based Programming (BR) are similar declarative technologies that deal with similar decision support problems
- However, until recently their input was different:
  - Business Rules were expressed using different flavors of a Rule Language oriented to Rete Engines
  - Constraint Satisfaction Problems (CSP) were expressed using different flavors of a Constraint Language oriented to Constraint Solvers
- These days CP and BR become standardized:
  - The Decision Model as a common input for CP and BR engines
  - The Decision Model becomes a de-facto standard for BDMS (?)
  - JSR-331 provides a standard API for different Java-based CP solvers

The Decision Model



- The Decision Model (TDM) is oriented to business users allowing them to describe their decisions and supporting rule families without help from developers
- TDM authors claim that today ~50% of the US financial institutions adopted or consider to adopt TDM
- TDM does not use any rule language
- TDM specifies strict rules organization principles
- Current implementations:
  - Sapiens, OpenRules, Corticon, New Wisdom, …

JSR-331 – Java Specification Request

- JSR 331 is “Constraint Programming API” standard developed ithin the Java Community Process (JCP) www.jcp.org
- JSR-331 covers key concepts and design decisions related to the standard representation and resolution of constraint satisfaction and optimization problems
- Reached The Final Draft phase
- Currently has 3 working implementations

OpenRules Approach

- Since its inception in 2003, OpenRules supported two engines:
  - Sequential Rule Engine
    - for efficient execution of hierarchies of complex decision tables defined in Excel files (with Java snippets)
  - Rule Solver
    - A constraint-based engine that was used when real inference and/or optimization were required
    - However, an input for Rule Solver used to be created by a user familiar with Constraint Programming concepts

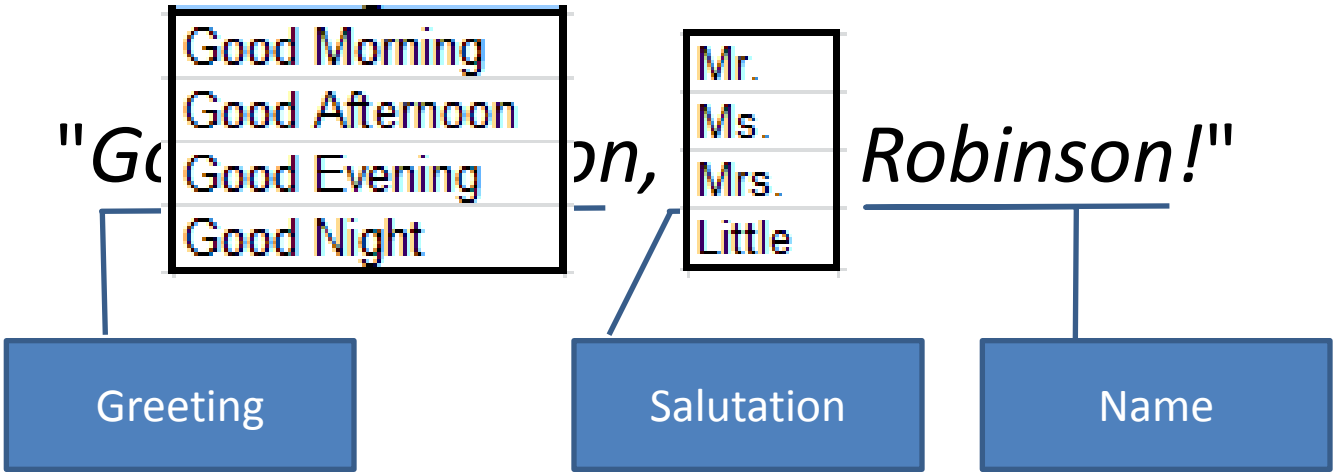
News from OpenRules

- **A new Rule Solver can execute inferential decision models "as is"!**
- No additional coding required
  - A user do not have to learn neither a rule language or a constraint language
- A new Rule Solver can handle real inference (similarly to Rete)

Actions are better than words: Rule Solver Demo

- Simple Decision Model “Hello Customer”
- No order between Rule Families
- Conflicts inside Rule Families

Starting with a Decision



Decision DetermineCustomerGreeting	
Decisions	**Execute Rule Families**
Define Greeting Word	:= DefineGreeting()
Define Age Group	:= DefineAgeGroup()
Define Salutation Word	:= DefineSalutation()

Rule Families

RuleFamily DefineGreeting

Condition		Condition		Conclusion	
Current Hour		**Current Hour**		**Greeting**	
>=	0	<=	11	Is	Good Morning

RuleFamily DefineAgeGroup

Condition		Condition		Conclusion	
Age		**Age**		**Age Group**	
>=	0	<=	7	Is	Little

RuleFamily DefineSalutation

Condition		Condition		Condition		Conclusion	
Gender		**Marital Status**		**Age Group**		**Salutation**	
				Is	Little	Is	Little
Is	Female	Is	Married	Is Not	Little	Is	Mrs.
Is	Female	Is	Single			Is	Ms.
Is	Male					Is	Mr.

Defining Business Glossary

Glossary glossary			
Fact Name	**Business Concept**	**Attribute**	**Domain**
Gender	Customer	gender	Male,Female
Marital Status		maritalStatus	Single,Married
Age		age	1-120
Age Group		ageGroup	Little,Young,Adult
Current Hour		hour	0-24
Greeting	Response	greeting	Good Morning,Good Afternoon,Good Evening,Good Night
Salutation		salutation	Mr.,Ms.,Mrs.,Little

Defining Test Data (in Excel)

Data Customer customers			
name	maritalStatus	gender	age
Customer Name	**Marital Status**	**Gender**	**Age**
Robinson	Married	Female	24
Smith	Single	Male	6

Executing Decision Model

- Using Sequential Rule Engine

```
Decision DetermineCustomerGreeting: Define Greeting Word
Conclusion: Greeting Is Good Morning
Decision DetermineCustomerGreeting: Define Age Group
Conclusion: Age Group Is Little
Decision DetermineCustomerGreeting: Define Salutation Word
Conclusion: Salutation Is Little
Decision has been finalized
Decision: Good Morning, Little Smith!
```

Running
Sequential Rule Engine
and Rule Solver
in Eclipse

- Using Inferential Rule Solver

```
=== Before Posting Rule Family Constraints
Marital Status[Single]
Greeting[Good Morning,Good Afternoon,Good Evening,Good Night]
Age[6]
Age Group[Little,Young,Adult]
Gender[Male]
Salutation[Mr.,Ms.,Mrs.,Little]
Current Hour[0]
=== Summary after execute() ===
Marital Status[Single]
Greeting[Good Morning]
Age[6]
Age Group[Little]
Gender[Male]
Salutation[Little]
Current Hour[0]
Decision: Good Morning, Little Smith!
```

Let's change the decision order

- Ordered Decisions

Decision DetermineCustomerGreeting	
Decisions	**Execute Rule Families**
Define Greeting Word	:= DefineGreeting()
Define Age Group	:= DefineAgeGroup()
Define Salutation Word	:= DefineSalutation()

- Unordered Decisions

Decision DetermineCustomerGreeting	
Decisions	**Execute Rule Families**
Define Greeting Word	:= DefineGreeting()
Define Salutation Word	:= DefineSalutation()
Define Age Group	:= DefineAgeGroup()

Define Age Group goes after Define Salutation

Executing Decision Model Again

- Using Sequential Rule Engine

```
Decision DetermineCustomerGreeting: Define Greeting Word
Conclusion: Greeting Is Good Morning
Decision DetermineCustomerGreeting: Define Salutation Word
Decision DetermineCustomerGreeting: Define Age Group
Conclusion: Age Group Is Little
Decision has been finalized
Decision: Good Morning, null Smith!
```

Now Salutation remains undefined (null)

- Using Inferential Rule Solver

```
=== Before Posting Rule Family Constraints
Marital Status[Single]
Greeting[Good Morning,Good Afternoon,Good Evening,Good Night]
Age[6]
Age Group[Little,Young,Adult]
Gender[Male]
Salutation[Mr.,Ms.,Mrs.,Little]
Current Hour[0]
=== Summary after execute() ===
Marital Status[Single]
Greeting[Good Morning]
Age[6]
Age Group[Little]
Gender[Male]
Salutation[Little]
Current Hour[0]
Decision: Good Morning, Little Smith!
```

Rule Solver continues to work well!

Let's create an invalid Rule Family

- Correct Rule Family

RuleFamily DefineSalutation							
Condition		Condition		Condition		Conclusion	
Gender		**Marital Status**		**Age Group**		**Salutation**	
				Is	Little	Is	Little
Is	Female	Is	Married	Is Not	Little	Is	Mrs.
Is	Female	Is	Single			Is	Ms.
Is	Male					Is	Mr.

- Incorrect Rule Family

RuleFamily DefineSalutation							
Condition		Condition		Condition		Conclusion	
Gender		**Marital Status**		**Age Group**		**Salutation**	
				Is	Little	Is	Little
Is	Female	Is	Married	Is Not	Young	Is	Mrs.
Is	Female	Is	Single			Is	Ms.
Is	Male					Is	Mr.

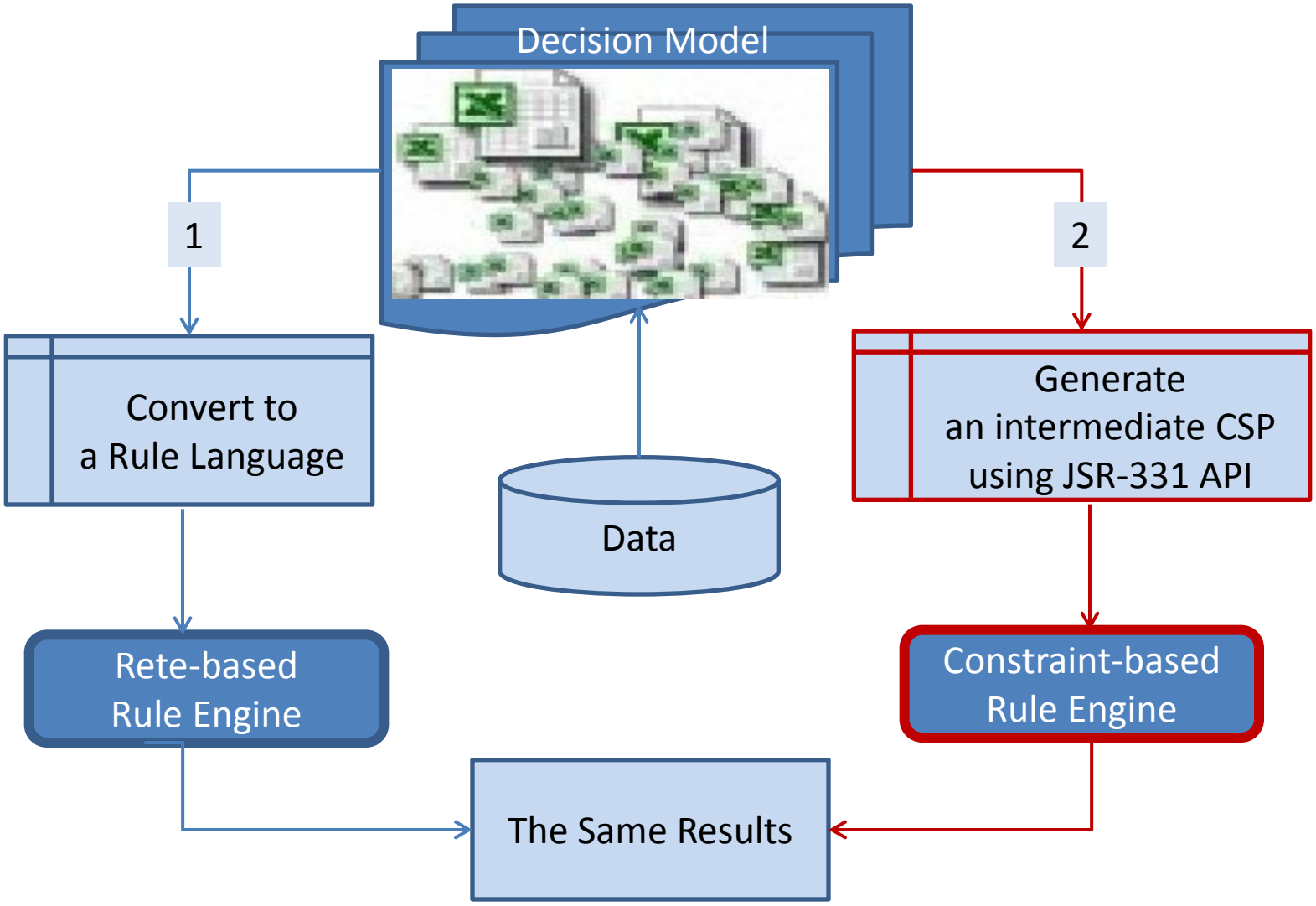
A Little Male is covered twice

- Rule Solver diagnoses the error

* Failure to post constraint: IF Gender = Male AND Age Group != Young THEN Salutation = Mr.

- Unfortunately, a sequential Rule Engine produces "correct" results – a user will not notice a problem!

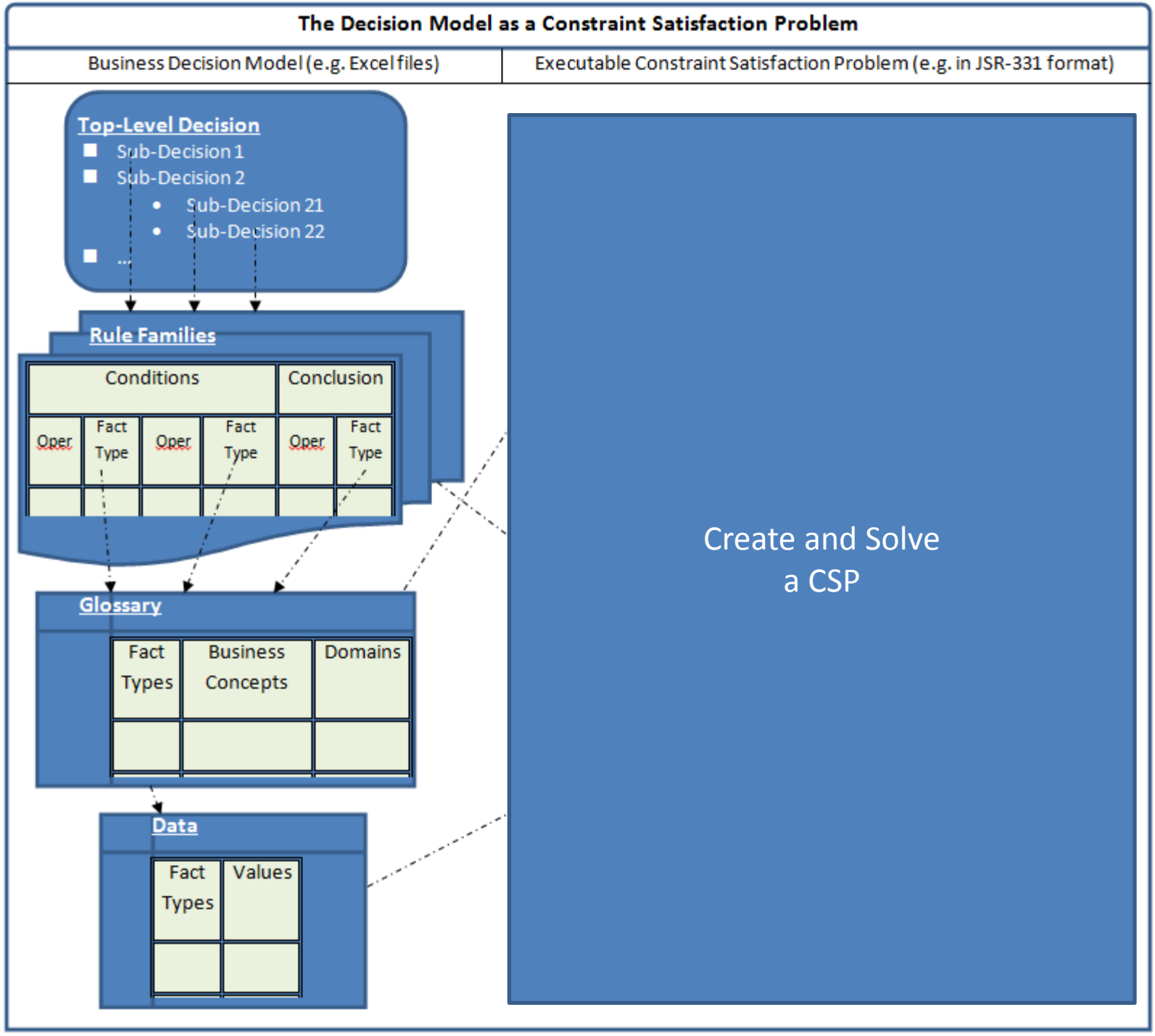
Rule Solver Validates and Executes Decision Models



How Rule Solver Works

- Takes an Excel-based Decision Model as an input
- Generates a constraint satisfaction problem (CSP)
  - Done on the fly, no code generation
  - Uses JSR-331 “Constraint Programming API” standard
- Solves the CSP using any Constraint Solver compliant with JSR-331 (there are at least 3 CP solvers available today)
- Produces errors (if any) in business terms of the initial decision model
- Saves the results in the business objects

How Rule Solver Works



BR and CP are Similar

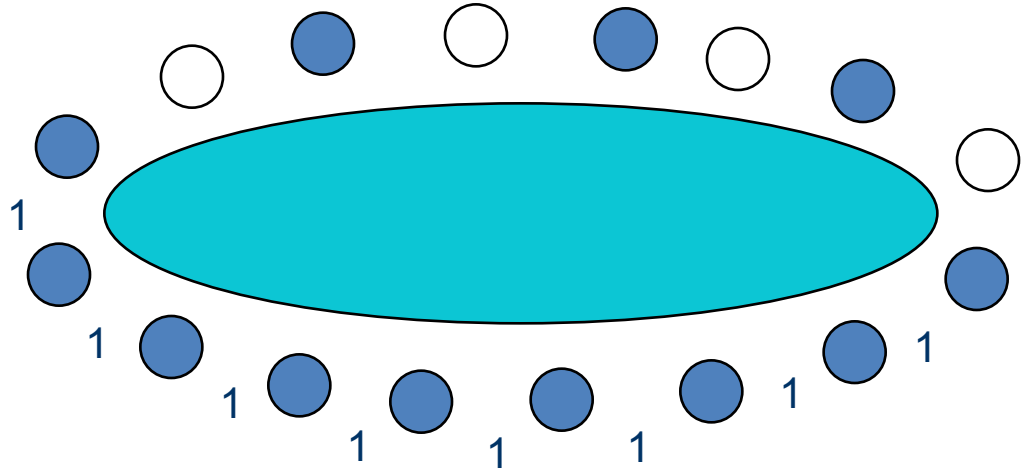
Business Rules (BR)	Constraint Programming (CP)
Business Objects with some yet unknown characteristics	Decision variables with yet unknown values from a finite domain
Rules specify relations among these objects	Constraints specify relations among these variables
Rule Engine executes these rules using Rete algorithm to find unknown characteristics of the business objects	Constraint Solver uses different search strategies to find an assignment of possible values to variables that satisfies all constraints

BR and CP are Different

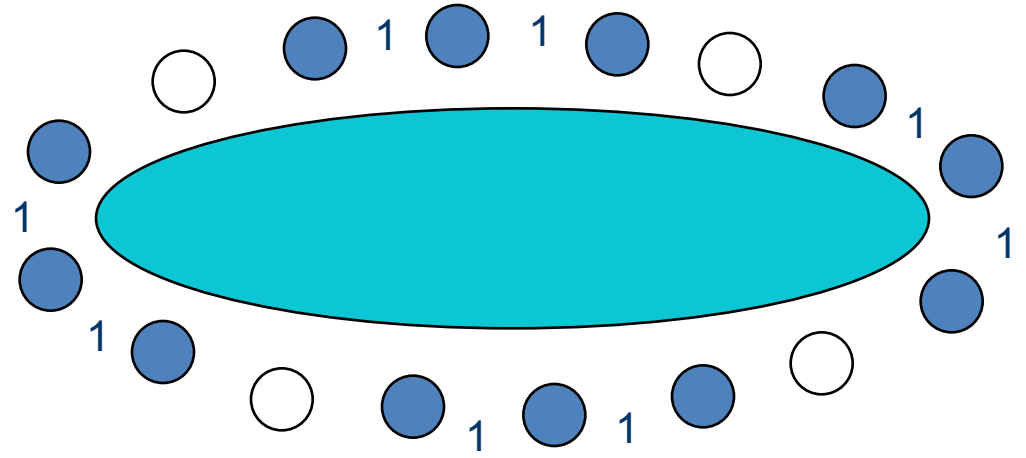
- BR Advantage:
  - Rules Repository is managed by business people while Constraint Store is usually under control of software developers
- CP Advantage:
  - Rules usually have to consider All (!) possible combinations of the problem parameters
  - Constraints do not have to cover all situations but rather define an optimization objective and allow a search algorithm to find an optimal solution
  - CP allows to choose different search strategies (while BR relies only on one Rete algorithm)

Miss Manners: Minimizing Rules Violation

- All rules violations may have an attached cost, e.g. 1. As a result, this solution has the minimal total constraint violation cost 8.



- However another solution looks “better”:



JSR-331 demonstrates the proper implementation

Presented at RulesFest-2009

Consistency Validation (1)

- Validate Rules Consistency using Constraint Propagation:
  - Rule Solver simply posts all 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

Consistency Validation (2)

- Validate Rules Consistency using Test Data:
  - 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 constraints (with data constraints already posted)
    - 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 displays the current state of all instantiated (or only partially instantiated) variables corresponding to the fact types.

Completeness Checking

- Rules Completeness:
  - Rule Solver determines whether all constrained variables have been instantiated for all conclusion fact types
  - If not, Rule Solver points to the fact types with remaining possible values from their domains. This prompts a user to identify which rules should be extended to cover the remaining situations
  - 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

OpenRules Decision Models

- Executable Decision Models
  - Can be created and tested by business people using MS Excel or Google Docs
  - A user may decide to use either Sequential or Inference rules execution modes
- Custom Decision Models
  - Easily configurable decision model templates defined in Excel specify:
    1) what is actually allowed;
    2) new custom features
  - Open source

Summary

- The Decision Model provides a common input for Rete-based or Constraint-based rule engines
- Rule Solver can be used instead an real (non-sequential) rule engines
- Rule Solver validates compliance with The Decision Model inferential principles
- Rules Solver may go beyond traditional BR problems

www.OpenRules.com