Business Rules for Automating Business Policy

Srinivas Krovvidy
Thomson Labs
1375 Piccard Drive, Suite 100.
Rockville, MD 20850

Colleen McClintock
Infinite Intelligence, Inc.
1155 Connecticut Avenue, #500
Washington 20036

Jacqueline Sobieski
Fannie Mae
3900 Wisconsin Avenue
Washington 20016

Abstract

Business policy can be defined as the guidelines and procedures by which an organization conducts its business. Organizations depend on their information systems to implement their business policy. It is important that any implementation of business policy allows faster application development and better quality management and also provides a balance between flexibility and centralized control. This paper views business rules as atomic units of business policy that can be used to define or constrain different aspects of the business. It then argues that business rules provide an excellent representation for business policy. Finally, it presents KARMA, (Knowledge Acquisition and Rule Management Assistant) a system that can be used to specify, verify, validate and implement business rules. KARMA was developed and deployed at Fannie Mae.

1 Introduction

Business policy can be defined as the guidelines and procedures by which an organization conducts its business. Business policy is often documented in manuals and business guidelines and is reflected in an organization's information systems. Organizations depend on their information systems to implement this policy. Often application developers are expected to understand and implement information systems from the documented policy. It is important that any implementation of business policy allows faster application development and better quality management while providing a balance between flexibility and centralized control.

This paper presents business rules as a means of specifying, verifying, and implementing business policy. It shows how business people can specify and implement their business policy as business rules. The paper also demonstrates how to ensure that implemented policy matches exactly with the documented policy by generating executable code from business rules.

Section 2 presents an overview of business rules and different paradigms to implement them. Section 3 introduces KARMA while sections 4 and 5 describe KARMA's verification and validation methodologies for business rules. Section 6 presents code generation techniques followed by conclusions in section 7.

2 Business Rules Overview

The GUIDE (IBM user group) Business Rules project (http://www.guide.org/ap/apbrules.htm) was organized in November 1993 to formalize an approach for identifying and articulating business rules. The final report of this project was released in October 1997. In this report, a business rule is defined as "A statement that defines or constrains some aspect of the business and is intended to assert business structure or to control or influence the behavior of the business." The report also indicates that a formal logic based specification language can be used to express various types of business rules. This report categorizes business rules as follows:

A STRUCTURAL ASSERTION - a defined concept or a statement of a fact that expresses some aspect of the business.

An ACTION ASSERTION - a statement of a constraint or condition that limits or controls some aspect of the business.

A DERIVATION - a statement of knowledge that is derived from other knowledge in the business.

There are primarily two types of technologies that dominate current business rule implementations:

- Data-Oriented Business Rules
- Object-Oriented Business Rules

2.1 Data Oriented Business Rules

Data-oriented business rules use a relational model to define the application's data model. Business rules are used to specify the application's behavior as operations on this relational model. Relational theory allows these operations to be defined as declarative statements. Data-Oriented business rules are also known as repository-driven business rules. Ross (Ross 1997) describes a business rule as a user requirement, and he implies that rules can be seen as database constraints. He suggests extensions to data modeling techniques to represent business rules diagrammatically. Herbst (1995, 1996) defines and structures business rules as a main component of systems analysis and presents a meta-model for business rules. In this approach, business rules are extracted by defining a meta-model consisting of an Event, some Conditions, a Then-Action, and an Else-Action.

Two commercial products, Vision Builder from Vision Software and Usoft Developer, from Usoft Corp support data-oriented business rules. They support business rules defined as constraints on the data. While Vision Builder implements business rules as triggers and stored procedures, Usoft generates application code for all tiers, and uses the repository as a parameter for controlling the environment.

2.2 Object-Oriented Business Rules

Object-Oriented business rules technology is a new adaptation of an existing technology, namely, expert systems. In this methodology, an object model is defined for the application domain and operations are defined on this object model as business rules. These business rules are typically expressed as IF-THEN-ELSE production rules. Traditionally, expert systems have been used for real-time systems, process control systems, and network management systems. However, as embedded information systems have become more popular, expert systems have also been found to be very useful, as they can be easily embedded into conventional business applications.

Currently, nearly all expert system shell vendors (including Brightware, ILOG, Intellicorp, Neuron Data, and Platinum) claim to support business rules in their products; in fact, expert system shells are frequently called "Business Rules Engines" by vendors. However, in most cases, the business rules they support are in the native rule language of the shell. Most business users cannot understand these rules. While these tools allow AI programmers to write production rules, they lack the following kinds of support needed by business users to implement policy:

- They are not easy to define, read, and understand
- They do not provide any tools to verify the logic of the rules
- They lack any tools to manage the rules from specification through implementation
- They create a dependency on the product they support

The next section describes KARMA: a tool that addresses some of these issues faced by business users in implementing business policy.

3 KARMA: A Tool for Automating Business Rules

KARMA was designed, developed, and deployed at Fannie Mae. Fannie Mae is the largest supplier of home mortgage funds, America's largest corporation in terms of assets, and the second largest borrower in the capital markets, exceeded only by the U.S. Treasury. To ensure that the loans that they purchase are of the highest quality, Fannie Mae establishes underwriting guidelines and eligibility criteria, which must be adhered to by those lenders who sell loans to Fannie Mae (Sobieski et. Al. 1996). Because of its strong leadership role, Fannie Mae's policies for loan eligibility set the standard in the mortgage industry. Applying these policies consistently and effectively is critical to Fannie Mae's mission and profitability. Fannie Mae chose business rules as the method of implementing these policies.

Fannie Mae developed a strategic set of tools including KARMA (Knowledge Acquisition and Rule Management Assistant) to define, verify, and validate business rules and business calculations. Ever since their deployment, these systems have had a significant impact on Fannie Mae's ability to respond to changes in the business environment. These tools have also undergone substantial enhancements in subsequent releases.

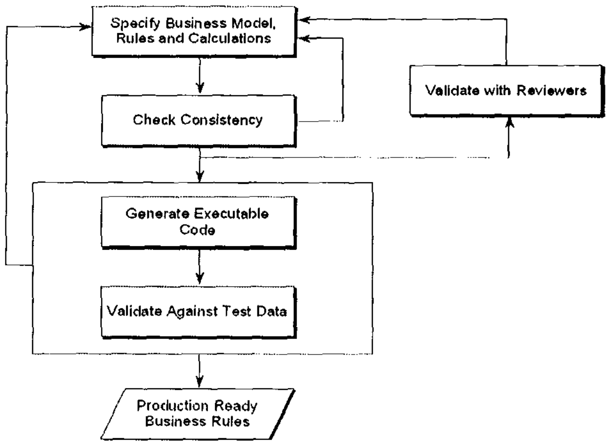

Specify Business Model, Rules and Calculations
Validate with Reviewers
Check Consistency
Generate Executable Code
Validate Against Test Data
Production Ready Business Rules

Figure 1. Business Rules Implementatı

Figure 1 shows the phases of KARMA's business rule implementation. KARMA has a syntax directed editor (Rule Editor) with a point-and-click graphical user interface to support the definition of business rules. These business rules are then checked for inconsistencies and ambiguities. After verifying the integrity of the business rules, they are also validated manually by business experts. Next, they are executed against a set of test cases in a dynamic environment, which allows users to modify rules and instantly analyze results. To enable this validation, executable rules are automatically generated and interpreted by KARMA.

3.1 Business Rule Specification Language

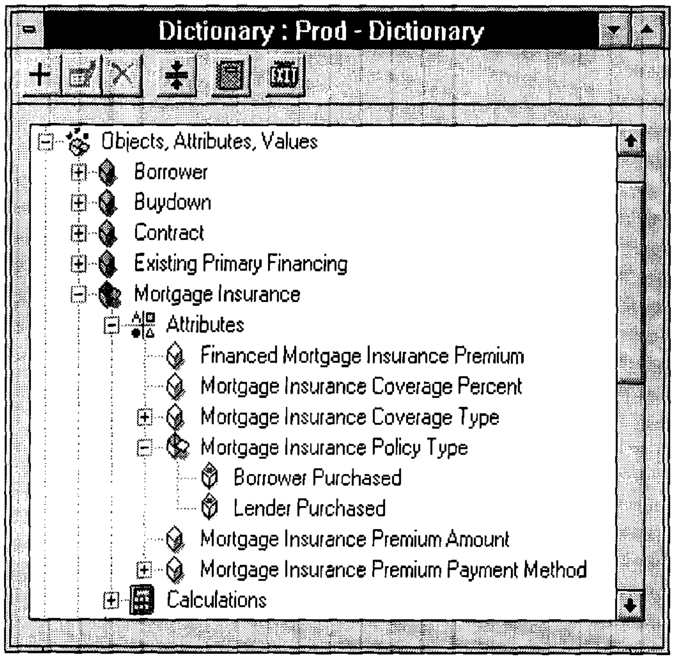
Traditionally, there were expert system languages such as ROSIE that were defined using a formal language that resembled fragments of English language in their syntax

(Hayes-Roth, Waterman and Lenat 1983). These languages provided general purpose programming tools to develop complete AI systems. However, KARMA's business rule specification language only concentrates on defining business model, business rules, and business calculations. It does not include capabilities such as database manipulations and user-interface commands. In particular, KARMA's business rule specification includes:

- Specifying a business model to define the application domain.
- Specifying the business rules and calculations.
- Specifying the properties of business rules which define their unique business-related characteristics to support the management of business rules.

3.1.1 Business Domain Model

The business domain model is defined as a collection of objects representing the business. Each object in turn is defined as a collection of attributes, which define the characteristics, or properties, of the business objects. Each attribute belongs to one of several system-defined or user-defined data types. KARMA provides a point-and-click interface to define the business domain model, as shown in figure 2. Business rules and business calculations are defined using the objects and attributes from the business domain model.

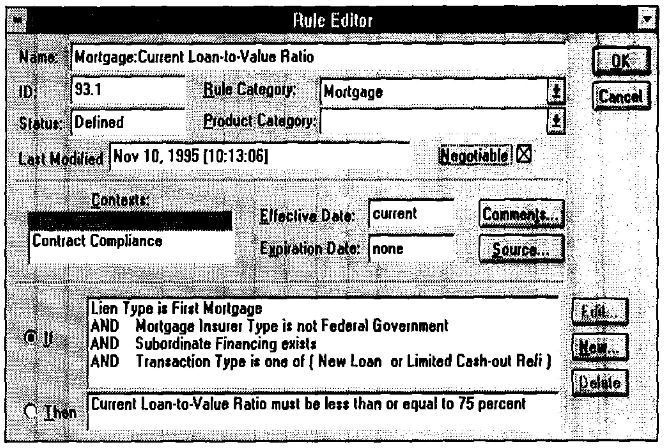

Dictionary : Prod - Dictionary

Objects, Attributes, Values
Borrower
Buydown
Contract
Existing Primary Financing
Mortgage Insurance
Attributes
Financed Mortgage Insurance Premium
Mortgage Insurance Coverage Percent
Mortgage Insurance Coverage Type
Mortgage Insurance Policy Type
Borrower Purchased
Lender Purchased
Mortgage Insurance Premium Amount
Mortgage Insurance Premium Payment Method
Calculations

Figure 2. Data Dictionary Editor

3.1.2 Business Rules

In the business rule specification language (See Business Rules Specification), business rules consist of left-hand-side (antecedent) and right-hand-side (consequent) clauses. Business rules may have one or more clauses ANDed together on the left-hand- side but may only have a single clause on the right-hand-side, as shown in figure 3. This right-hand-side clause either restricts the value of a single attribute or assigns a value to an attribute when the left-hand-side conditions are satisfied. Therefore, business rules are represented as:
IF <clause>
AND <clause> .....
THEN <clause>
Where a <clause> is defined using the clause editor.


Rule Editor

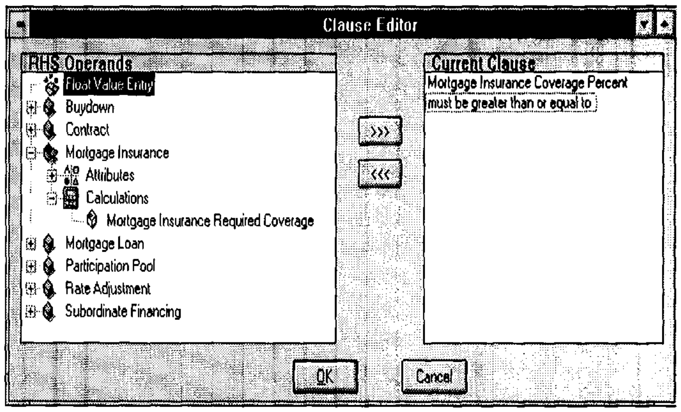
Name: Mortgage:Current Loan-to-Value Ratio
ID: 93.1
Rule Category: Mortgage
Status: Defined
Product Category:
Last Modified Nov 10, 1995 [10:13:06]
Negotiable ☒
Contexts:
Contract Compliance
Effective Date: current
Expiration Date: none

[✓ If]
Lien Type is First Mortgage
AND Mortgage Insurer Type is not Federal Government
AND Subordinate Financing exists
AND Transaction Type is one of (New Loan or Limited Cash-out Refi)

Then
Current Loan-to-Value Ratio must be less than or equal to 75 percent

Figure 3. Business Rule Editor

Clause qualifiers are used to handle multiple instances of objects in business rules. In the absence of clause qualifiers, the default behavior for a rule is to execute once for each instance of an object in working memory which satisfies the clause condition. To modify this default


Clause Editor

RHS Operands
Float Value Entry
Buydown
Contract
Mortgage Insurance
Attributes
Calculations
Mortgage Insurance Required Coverage
Mortgage Loan
Participation Pool
Rate Adjustment
Subordinate Financing

Current Clause
Mortgage Insurance Coverage Percent
must be greater than or equal to

Figure 4. Clause Editor

behavior, the business rule language allows the user to specify either the "EVERY" or "ANY" clause qualifier.

3.1.3 Business Rule Categories

KARMA classifies business rules into three categories:

Concept rules are used to define and/or derive a particular aspect of the business. These rules can use simple

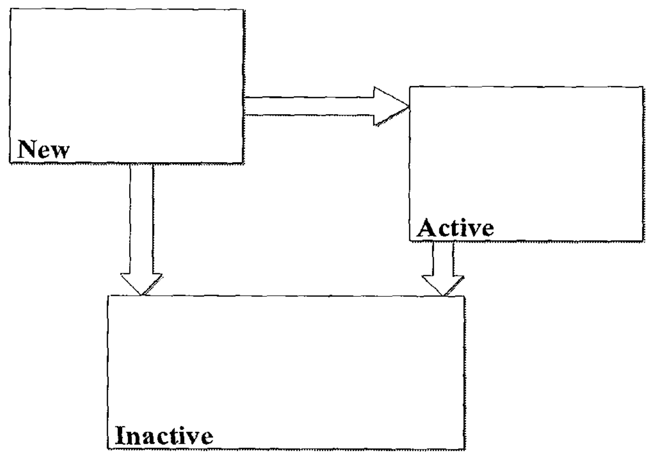


Figure 5a. Business Rule States

assignment statements or complex processing algorithms in their right-hand-side clauses to define concepts.

Constraint rules specify policies or conditions that impose a restriction that can not be violated. Constraint rules are in turn classified as negotiable and non-negotiable. A non-negotiable rule can not be overridden under any circumstances, while negotiable rules can be overridden by override rules.

Override rules are used to implement negotiated exceptions to standard business rules. These negotiated rules are often used to relax the restrictions imposed by negotiable constraint rules. These rules are also called as meta rules, as they are defined on top of the existing constraint rules.

3.1.4 Business Calculations

Oftentimes, the policies in a business domain require simple calculations. In traditional systems the logic behind these business calculations is hidden from the end-user and the burden of debugging these calculations is left to the programmer. However, KARMA's specification language provides a rich set of procedural constructs including IF-THEN-ELSE, ASSIGNMENT, and ITERATION for defining calculations. Additionally, the language allows the user to define calculation logic across multiple instances. Calculations are defined using KARMA's rule editor.

3.2 Business Rule Properties

In addition to the formal syntax supported by the Business Rule Specification Language, the following properties are defined for business rules: Business Rule Context, Business Rule Status, and Business Rule Effective Period. These business rule properties are defined to support the effective management of business rules in a policy-intensive organization.

Business Rule Context refers to the specific business context to which a rule applies. The Business Rule Context is domain dependent, and therefore, is defined by the user. Defining a Context in a business rule provides the ability to implement the rule either globally, across all business processes, or locally, specific to an individual business process.

Business Rule Status indicates the lifecycle of business rules development, beginning with a specification phase where NEW business rules are specified. Once they are verified and validated, they become ACTIVE and are implemented in a production environment. Finally, when a business rule is no longer in effect, it may go into a retired phase and become INACTIVE. Figure 5a depicts these phases as three base states in the life cycle of a business rule. These base states are associated with certain restrictions. For example, an ACTIVE rule can not be deleted or modified. To modify an ACTIVE rule a new version of the rule must be defined. KARMA implements the restrictions imposed by the base states. However, the three base states may not provide sufficient detail to describe the complete life cycle of a business rule within an organization. These states only encapsulate the fundamental states in the business rule life cycle. Different organizations may want to manage the business rule life cycle at different levels of granularity depending on the business processes responsible for defining and managing business rules.

To support these requirements, KARMA allows rule statuses to be derived from an underlying base state. Figure 5b depicts an example of defining a complete life cycle for business rules using the base states. A business rule would be in the NEW state (which allows business rules to be modified and deleted) during its specification, verification, and validation phases.

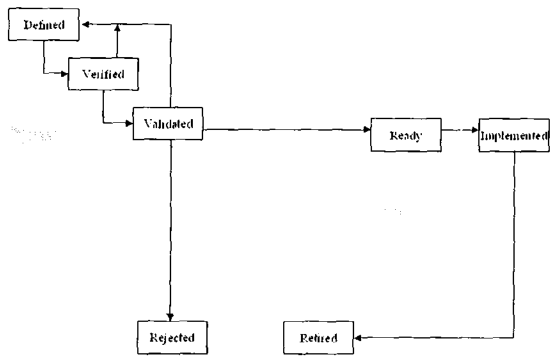


Figure 5b. Business Rule Statuses

These phases can be supported by deriving three statuses from the NEW state: "Defined", "Verified", or "Validated".

Similarly, within the ACTIVE state, the business rule would have a status of "Ready" or "Implemented". A "Ready" status indicates that the business rule is ready for implementation but not yet implemented in production. After deriving required statuses from base states, a state transition table can be defined among derived statuses using a point-and-click interface. KARMA then enforces the defined transitions between rule statuses.

Business Rule Effective period indicates the effective and expiration dates for the business rule. The rule may be an ACTIVE rule in production but the business policy related to the rule may not be effective yet. Effective and expiration dates provide the ability to control the temporal implementation of a business rule independent of the migration process through which a business rule is moved into the production environment.

Business Rule Specification Language (Simplified)

BusinessRuleSpec	: BusinessModel {BusinessRule}* {BusinessCalculation}*
BusinessModel	: {Object}*
Object	: {Attribute}*
BusinessRule	: ConstraintRule
	| ConceptRule
	| OverrideRule
ConstraintRule	: IF {Clause}* THEN Constraint
ConceptRule	: IF {Clause}* THEN Concept
OverrideRule	: IF {Clause}* THEN Override
Clause	: Qualifier Operand OPERATOR {Operand}+
Constraint	: Clause
Concept	: Operand IS Operand
Override	: Operand CAN BE Operand
Qualifier	: ANY
	| EVERY
	|
BusinessCalculation	: NAME {Statement}+
Statement	: IfCondition
	| IterationStatement
	| AssignStatement
	| ReturnStatement
	| CompoundStatement

4 Business Rules Verification

Providing a business rule language and a GUI to ensure that rules are syntactically correct allows business users to independently define business rules. As the number of business rules increases, it becomes increasingly difficult to ensure that business rules are consistent with each other. It is observed that maintaining absolute consistency is not always possible and even desirable (Finkelstein et al. 1994). However, it is useful to identify the presence of inconsistencies in a knowledge-base. The design of KARMA included a consistency-checking component to meet this need.

Several knowledge base verification tools and algorithms have been developed to check the consistency of knowledge base rules (Preece, Talbot, and Vignollet 1997). Since clauses are the building blocks for business rules in the Business Rule Specification Language, a unification-based algorithm based on clause comparison is implemented by KARMA. It performs consistency-checking between clauses and subsequently uses that information to define the relationship between the business rules. KARMA's consistency-checking implementation assumes that:

- No nested clauses exist in the rules
- The consequents have only one clause
- The antecedents may have multiple clauses with an "AND" connector.

KARMA's consistency-checking algorithm detects:

- Conflicting rules
- Redundant rules
- Subsumed rules
- Redundant if-conditions
- Cyclic rules

In terms of business rules, consistency-checking is more than simple verification to show that a system produces expected answers. Consistency-checking includes reporting built-in discrepancies, ambiguities, and redundancies among business rules. Redundant business rules may not be desirable, as they do not satisfy any new business requirements and may cause confusion. Conflicting business rules need to be identified thereby policy makers can decide on how to act on them. Detecting subsumed business rules often leads to the identification of incorrect or incomplete specification of a business policy. Please see the Appendix for more details of the consistency-checking algorithm as implemented in KARMA.

5 Business Rules Validation

Figure 1 shows the verification (consistency-checking) and validation processes within KARMA. Verification primarily focuses on ensuring that business rules are consistently defined and that new rules do not contradict the existing set of rules. The primary emphasis of validation is to make sure business rules implement the business policy accurately and produce the expected results for a large set of test cases. To accomplish this, Business Rules and Calculations are validated in two different ways:

5.1 Validation by Business Experts

Once a business rule has been completely defined by the business user responsible for the associated business policy, it is reviewed by business experts. Collectively, these reviewers approve, modify, or reject the rule. If a rule is rejected, then its life cycle is terminated by making it inactive. A modified rule must pass through the review cycle again. Approved rules become active and are implemented in production. Validation by business experts is supported in KARMA by defining the appropriate rule statuses to support the validation process. Figure 5b shows the business rule life cycle as implemented in our validation process.

5.2 Validation using Test Data

The second form of validation involves analyzing the impact of the business rules using a test data set. A dynamic rule-evaluation capability is integrated with KARMA to validate the business rules with test data. This feature uses A*E (ART*Enterprise®) as a DLL embedded into KARMA. The user can select one or more business rules and request validation. KARMA generates executable code which is interpreted by the A*E inference engine. Since the inference engine is embedded in the system, the requirement for additional compilation is eliminated and the users can validate business rules as they are defined.

6 Code Generation

KARMA generates executable ARTScript rules from the business rule representation. All code that is dependent upon the business rules (business domain model, business rules, and business calculations) is generated by the system. This means that there is no manual maintenance necessary for any application when business policy changes.

In terms of rule generation, the business rule representation stored in the form of a collection of clauses, is the source, and the knowledge base rule (ARTScript) representation is the target. Rules are generated through the use of an intermediate representation (IR). This allows KARMA to generate executable rules in different languages, if necessary.

The IR is composed of three types of constructs:
• **Lexical Nodes** are atomic (i.e., they cannot be decomposed). They describe the syntactic elements on a character-by-character basis.
• **Repetition Nodes** are list nodes, which specify one or more occurrences of a node of any type.
• **Construction Nodes** are nodes that are composed of a fixed number of other nodes which may be lexical nodes, repetition nodes, or construction nodes.
The IR is implemented as a set of C++ classes. All IR classes are subclasses of the lexical, repetition, and construction node classes. Executable code is generated by unparsing the IR nodes (Krovvidy and Wee 1988).

® ART*Enterprise is a registered trademark of Brightware Inc.

In certain cases, the complexity of the Business Rule Specification Language may require KARMA to generate multiple knowledge base rules from a single business rule. More specifically, if a business rule contains one or more clauses with qualifiers (ANY or EVERY), then KARMA generates multiple knowledge base rules to implement the required behavior. KARMA also generates multiple knowledge base rules to implement override rules.

7 Conclusions

When the development of KARMA was started, there were no commercial products available to support business rules. A rule based approach was selected for KARMA since rule based systems offered the most direct representation of business rules. Currently, many expert system shell vendors support business rules defined in their language. However, most business users cannot understand rules in these languages.

Recently some vendors (e.g., Visionware and USOFT) have released business rules products, which are essentially rule-oriented client-server development tools. These tools allow the developer to specify declarative business rules based on a physical database schema. The rule languages used by these business rule tools are targeted towards software developers rather than business users. In addition, these tools can not be used to implement business rules as a service independent of the client application data model. The client-server business rule tools require a shared database. Furthermore, both types of tools (expert system shells and client-server development tools) lack the rule management capabilities that were built into KARMA to support the verification and validation of business rules.

The business rule language presented in this paper does not allow multiple overrides, constraints, or concepts on the right-hand-side of a business rule. This limitation is imposed to maintain the property that each business rule represents a single unit of independent business policy. KARMA's tool-independent business rule language allows applications to share business rules. This can be very important in e-commerce applications where applications share business policy along with data. (http://www.research.ibm.com/rules/home.html) Current efforts also include enhancements to the business rule language to represent more complex business policies in addition to providing business users with an intranet-based capability to develop business rules. In particular, we are planning to develop a business rules tool using XML and related markup technologies to represent the rules embedded in the content of a document such that applications can understand and share rules as well as content. On top of production rules, several researchers are also looking at other re-usable knowledge representation approaches (Menzies 1997). These approaches include identifying patterns in expert systems problem solving behaviors by developing and maintaining an ontology of business terms. We hope to include some of these ideas in our XML based tool. In particular we expect that this tool can help business users develop the vocabulary of their business domain in the form of a re-usable repository and allow them to use the terms from this repository to define their business rules.

References

Finkelstein, A. et al. 1994. Inconsistency Handling in Multi-Perspective Specifications. *IEEE Transactions on Software Engineering* 20(8):569-578.
Hayes-Roth, F, Waterman, D.A. and Lenat, D.B. 1983. *Building Expert Systems.* Reading, Mass.: Addison-Wesley.
Herbst, H. 1996. Business Rules in Systems Analysis: A meta-model and repository system. *Information Systems* 21(2):147-166..
Herbst, H. 1995, A Meta-Model for Business Rules in Systems Analysis. In Proceedings of the Seventh Conference in Advanced Information and Systems Engineering (CaiSE '95), Springer, 186-199.
Krovvidy, S., and Wee, W.G. 1988, Retargetable rule generation for expert systems. In Proceedings of the third international symposium on methodologies for intelligent systems, colloquia program 37-46.
Menzies, T.J. 1997, OO Patterns:Lessons from Expert Systems. *Software Practice & Experience*, 27(12):1457-1478.
Preece, A., Talbot, S. and Vignollet, L. 1997, Evaluation of verification tools for knowledge-based systems. *International Journal of Human-Computer Studies*, 629-658.
Ross, R., 1997. *The Business Rule Book : Classifying, Defining and Modeling Rules, Version 4.0* Database Research Group, Inc.
Sobieski, J. et al. 1996, KARMA : Business Rules from specification to implementation. In Proceedings of the eighth Innovative Applications of Artificial Intelligence, 1536 - 1547.

Appendix:Consistency Checking of Business Rules

Definitions

Inferred Rules

These are rules that can be obtained by using the transitivity property of rules. A new rule can be inferred from two rules if a clause in the consequent of one rule is unifiable with a clause in the antecedent of another. For example, consider the following three rules:

R1: IF (A11 == V11) THEN A12 = V12
R2: IF (A12 == V12) THEN A21 = V21
R3: IF (A21 == V21) THEN A41 must be greater than V41

In this example R1, R2 are concept rules and R3 is a constraint rule. The following inferred rules are generated from these three rules:

R1->R2: IF (A11 == V11) THEN A21 = V21
R2->R3 : IF (A12 == V12) THEN A41 must be greater than V41
R1->R2->R3: IF (A11 == V11) THEN A41 must be greater than V41

Redundant Rules

When two rules' antecedents are equivalent, their consequents are also equivalent. Two antecedents are equivalent when they can be unified and have an equal number of clauses; two consequents are equivalent if they can be unified.

Conflicting Rules

Two rules are said to be conflicting if their antecedents are equivalent but their consequents conflict. Conflicting rules succeed in the same situation but produce conflicting results.

Subsumed Rules

If two rules' consequents are equivalent, and one rule's antecedent consists of the antecedent of the other plus some additional clauses, the more restrictive rule (i.e., the one having more clauses in its antecedent) is subsumed by the other.

Redundant If Conditions

There are two types of unnecessary If-conditions possible. The first type occurs when a clause in one rule's antecedent conflicts with a clause in the other rule's antecedent and all the remaining clauses in both the antecedents and the consequents of the rules are equivalent. The second type of redundant If-condition occurs when two rules' consequents are equivalent, and one rule's antecedent contains a single clause that conflicts with a clause in the other rule's antecedent.

Verification algorithm

For the verification of the rules, the comparison of the clauses is the primary operation. Let C->lhs be the left-hand side operand in the clause C, C->op be the operator in the clause C and C->rhs be the list of right-hand side operands in the clause C. Comparing two clauses C1 and C2 yields the following results with the respective substitution lists:

1. C1 ≡ C2 if {C1->lhs = C2->lhs; C1->op = C2->op; C1->rhs = C2->rhs } SUB_LIST = {} or
   if { C1->lhs = C2->lhs; C1->op = C2->op; C1->rhs ≠ C2->rhs } SUB_LIST = UNIFY(C1->rhs = C2->rhs)

2. C1 ≡ ~C2 if {C1->lhs=C2->lhs;C1->op = ~(C2->op); C1->rhs = C2->rhs } SUB_LIST = {} or
   if { C1->lhs = C2->lhs; C1->op = ~(C2->op); C1->rhs ≠ C2->rhs } SUB_LIST = UNIFY(C1->rhs = C2->rhs)

3. $C1 \subset C2$ if { C1->lhs = C2->lhs, C1->op $\subseteq$ C2->op; C1->rhs $\subseteq$ C2->rhs } SUB_LIST = UNIFY{C1->rhs $\subseteq$ C2->rhs } or
if { C1->lhs = C2->lhs, C1->op = C2->op; C1->rhs $\subseteq$ C2->rhs } SUB_LIST = UNIFY{C1->rhs $\subseteq$ C2->rhs} or
if { C1->lhs = C2->lhs, C1->op $\subseteq$ C2->op; C1->rhs = C2->rhs } SUB_LIST = {}

4. $C1 \neq C2$ In each case, if SUB_LIST $\neq$ {}, then that list must consist of a consistent set of substitutions for each variable. Based on these relationships between clauses, any two rules R_i and R_j are compared as follows:

1. IF the right-hand side clause of $R_i \equiv$ right-hand side clause of R_j with a consistent substitution list for unifying all the clauses on their left-hand sides THEN R_i and R_j are redundant.
2. IF the right-hand side clause of $R_i \equiv$ ~right-hand side clause of R_j with a consistent substitution list for unifying all the clauses on their left-hand sides THEN R_i and R_j are conflicting.
3. IF the right-hand side clause of $R_i \equiv$ right-hand side clause of R_j with a consistent substitution list for subsuming Ri's left-hand side clauses with those of R_j THEN R_i subsumes R_j.
4. IF the right-hand side clause of $R_j \equiv$ right-hand side clause of R_i with a consistent substitution list for subsuming R_j's left-hand side clauses with those of R_i THEN R_i is subsumed by R_j. The left-hand side of a rule R_i subsumes that of R_j in the following cases:
   - All the clauses in the left side of R_i have equivalent clauses in R_j and R_i has at least one more clause than R_j on its left side.
   - At least one clause from the left-hand side of R_i subsumes those of R_j and the rest of the clauses from the left-hand side of R_i have equivalent clauses in the left-hand side of R_j.
5. IF the right-hand side of R_i = right-hand side of R_j AND if we have a clause from the left-hand side of R_i = ~any one clause from the left hand side of R_j and for all the other clauses from the left-hand side of R_i there is an equivalent clause from the left hand side of R_j THEN R_i and R_j have a redundant IF condition.

The following steps are used in the implementation:

- Let S be the set of all business rules.
- Compute S^T, where S^T is the transitive closure of S (Note that ST determines the set of inferred rules.)
- For every $x \in S \cup S^T$
  - for any clause c on the left hand side of x, if there exists another clause d on its left hand side where c $\equiv$ ~d then x has a contradiction within itself.
  - if there exists a clause c on the left hand side of x where c $\equiv$ ~(the right hand side clause of x) then x has a contradiction within itself.
  - if there exists a clause c on the left hand side of x, where c $\equiv$ the right hand side clause of x, then x has a cycle.
  - find the relationship between x and y where $y \in (S \cup S^T)$ and $y \neq x$ using the five rules mentioned earlier.