JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2007**

Vol.6, No. 1, January - February 2007

# A Framework to specify Declarative Rules on Objects, Attributes and Associations in the object model

**C. Anantaram**, Tata Consultancy Services Limited, 249 D&E Udyog Vihar Phase IV, Gurgaon 122016, India.

**Abstract**

There are various mechanisms for creating an application object model. These mechanisms are used for modeling the application Meta model and the user models. Some of the mechanisms allow constraints to be specified on object entities. Constraints are expressions and evaluate to either true or false, and are used to specify invariants or act as guards on objects, their attributes or associations. However, they do not derive new states from existing states, nor do constraints create new objects or assert an association dynamically. Thus it becomes cumbersome to specify business rules during modeling, especially the kind of business rules that have an impact on the association between objects. We present a framework for specifying declarative rules on objects, attributes and associations in the object-model for a domain. Our framework permits the specification of an association which has related business rules, such that the rules are apparent during modeling, and also provides a mechanism to evaluate the rules before the association is created between run-time instances of the classes. Using such a framework, some of the business rules can be stated during modeling and need not be buried in the code or be separately defined in a rule language. We discuss the framework and its advantages during modeling.

## 1 INTRODUCTION

Real-world objects are best modeled as self-contained entities that contain both data and functionality. In the object-oriented methodology, this is expressed as objects with attributes and operations. Modeling tools and methodologies support the software development process. For modeling the application, there are various standards such as Unified Modeling Language (UML), Entity-Relationship model (E-R model), etc. Most modeling tools and methodologies are based on fixed, explicitly stated models such as the UML model, E-R models, etc.

Rules are the heart of business logic. Businesses implement their information systems upon the pillars of business rules. Usually, business rules have been buried deep

Cite this article as follows: C. Anantaram: "A Framework to specify Declarative Rules on Objects, Attributes and Associations in the object model", in *Journal of Object Technology*, vol. 6, no. 1, January-February 2007, pp. 91-106 http://www.jot.fm/issues/issue_2007_01/article2

inside the code. With the advent of business rule engines, rules could be specified separately from the main processing flow of the business application. This allows the application to be flexible and configurable to dynamic changes in the business environment, as the rules could be easily changed and modified. Generally, a business policy is coded as a set of business rules and workflow processes [Streng94][Date2000]. While this provided some temporary relief, the absence of specification of business rules in the object model during modeling imposed various restrictions. Model developers could not explicitly specify when a particular relationship (modeled as an 'association') had rules. The rules which were known during modeling were hidden in code.

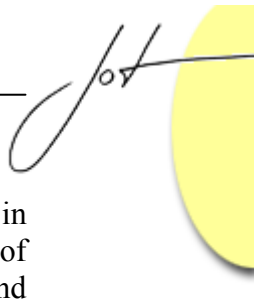
Some mechanisms have been developed to attach rules with methods present in the model. However, no mechanism exists to express rules on an object, its attributes and its associations in a declarative way using the power of logic.

Rules can be used to specify semantic checks based on object properties and associations. Allowing rules on the object, attributes and associations will provide evaluating conditions and also carrying out some actions. New states can be derived from existing states and this new state can be used for further processing. Application developer will be able to write rules to create objects, instantiate associations based upon some conditions, define rules on classes and their attributes. This will be useful for modeling integrity constraints on the model, derive new attribute values and specify rules to instantiate an association based on the evaluation of a rule. Thus one can model strategies in the business components.

We present a framework for model developers to specify declarative and procedural rules on the objects, attributes and associations in the object model itself, using the power of logic programming to make new derivations and assertions on the model elements.

## 2 RELATED WORK

The Object Management Group's Object Constraint Language (OCL) [OCL03] and Meta Object Facility (MOF) [MOF03] standards have been incorporated to include constraints (or invariants) on objects and associations. OCL is a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or can be used to specify queries over objects described in a model. While the OCL expression itself cannot alter the state of the corresponding executing system (i.e. it cannot change anything in the model), an OCL expression may contain operations / actions that when executed can alter the state of the system. OCL expressions are not directly executable and the evaluation of an expression is instantaneous. OCL expressions have been used for a number of different purposes such as a query language, as invariants on classes and types in the class model, as type invariant for Stereotypes, as pre- and post conditions on Operations and Methods, as Guards, and as derivation rules for attributes in a UML model. Invariants can be described on classes using the Universal(∀) and Existential(∃) operators to provide a



constraint on all elements of a collection, or test the existence of at least one element in the collection, respectively. Derived-value expressions allow the specification of expression states which can be used to set the value of an attribute or an association-end from the evaluation of the expression. Guards are expressions that are linked to state transitions, and are evaluated at the moment the transition is attempted.

During the evaluation of an OCL expression, two important points are assumed:

1. States of objects in the model cannot change during the evaluation
2. An expression must be true for all instances of that type for which the expression is defined.

The above points form a sort of restriction, as they do not allow dynamic creation of objects based on the constraints and also do not permit a constraint to be "false" for some of the objects in its collection. Hence constraints do not permit the model developer to specify an association that is "conditionally instantiated" for a set of objects. Further, the constraints by themselves do not derive any state (for example, like the conclusion that is derived in a rule-execution) and hence cannot be used in further reasoning (i.e. there is no derived state which could be used in other constraints or business rules of the application).

OMG is evaluating a couple of RFPs, viz. 'production rules representation' RFP [PRRP05] and the 'Business semantics of business rules' RFP [SBVR05]. Both these RFPs are focused on defining business rules from the business perspective and do not provide logic programming power on object models.

Some of the other work in the area has been to model rules as component objects, separate from business objects and application logic [RuleMachines05]. While the business user is free to define and modify the rules, these business rules are themselves not defined in the object model. This approach is more like the traditional business rules approach in applications, wherein rules are actually described separately from the application and is not concerned with stating rules in object models. It is significantly different from the approach that we are taking in this paper.

Adaptive Object-Models have been used to address the need for change by casting information like business rules as data rather than code. Using objects to model such data and coupling an interpretation mechanism to that structure, a domain-specific language allows users themselves to change the system following the evolution of their business. A system with an Adaptive Object-Model has an explicit object model that it interprets at run-time. If the object model is changed, the system changes its behavior. Since objects have states and respond to events by changing state, the Adaptive Object-Model defines the objects, their states, the events, and the conditions under which an object changes state [Yoder02]. However, this work does not address the aspect of defining rules during modeling such that the rules could be specified in the object model itself.

In the framework that we describe, we provide the flexibility of defining rules on objects, attributes and associations in the object model by enabling logic programming power (Prolog-kind) in terms of binding, unification, backtracking etc. over object

models. Our framework enables the specification of rules during modeling to qualify association with conditions, and enables the creation of that association at run-time between the objects that satisfy the conditions at run-time. Previously, such business rules were not modeled and were buried deep inside the code. Model developers and model-maintainers would be oblivious to such rules and the object model may not actually reflect the true state of the run-time model. Our framework attempts to address these aspects.

## 3 TERMINOLOGY

In order to make the paper complete we present our explanation of some terminology that we will refer.

### Object Meta Model

The Object Meta Model is an object-oriented representation of the Domain for which the software application is envisaged. Typically it consists of entities of the domain categorized as 'Objects' with 'attributes'. These entities are identified by the Domain Analyst and created in an object-modeling tool. Objects may be related to each other through 'associations'. The Meta model is exported from the object-modeling tool to the software application in order that the application can use the entities of the domain model to create the actual model of the domain at run-time.

### User Model

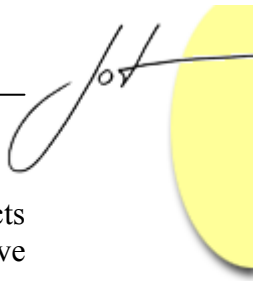
The User Model consists of the 'Object' entities and their associations that are specified in the Meta model of the domain. These Object entities are treated as 'Classes of objects' that are instantiated into real-world objects at run-time. Thus the User Model is an instance of the Meta Model at run-time.

### Rulebase

Rulebase is the collection of all rules that are possible on the object entities and their associations in the Meta model. These consist of conditional statements of the form "if <conditions> then <actions>". The conditions consist of the object entities combined with other entities through mathematical relations. The actions are specific computational instructions that can be performed on the object entities when the conditions are met.

### Declarative object-model-enabled Rule Engine

The declarative object-model-enabled rule engine is an inference engine that can read, parse, interpret, translate and execute rules in the Rulebase. While the engine could use one of the mechanisms for rule execution, viz. forward-chaining using RETE [Forgy82] or backward-chaining using WAM [Ait-Kaci91], it would need some extensions. The



engine has to have the ability to perform operations on a User Model and can treat objects as Facts and as Parameters for Logical Predicates. The Rule Engine has the declarative power of logic programs, such as Unification and Binding of logical variables, Assertion and Retraction of facts, Resolution, Backtracking, Freeing of bound variables, Existential and Universal quantifiers etc.

### Domain Application

The domain application consists of the core software that is written for a specific purpose. Its entities are based on the User model and create the real-world objects at run-time. The workflow and control of the application directs what functions are carried out in the domain.

## 4 FRAMEWORK TO DEFINE RULES ON OBJECT MODELS

In our framework we have focused on three important questions that we believe should be addressed in object modeling:

1. how to specify an association which has related business rules, such that the rules are apparent during modeling
2. how can the business rules attached to an association be evaluated before the association is created between run-time instances of the classes
3. how can the association be dynamically created for some of the objects and deleted depending on the dynamic states of the objects.

In the following sections we discuss our framework that helps address the above questions.

### Core Framework

Our framework focuses on the specification of associations in a declarative way, using the power of logic programming, through which it is possible to derive an implementation to create, delete and maintain associations in object models. Primarily, in the framework there are four main aspects:

a) Treating classes and associations as predicates in a logical framework and allow it to bind to their instances.

   Every class and association in the object model is automatically treated as a logical predicate with the class name or association name as the name of the predicate. The instances of the classes and associations in the model are treated as facts in a factbase. A predicate that represents a class will have one logical variable that would bind to instances of that class at runtime. On the other hand, a predicate that represents an association would have two logical variables, such that the variables would bind to the two objects between which the association is created. A predicate can take a `TRUE` or a `FALSE` value depending on whether the instance of the class or association exists or does not exist in the factbase at

runtime. A predicate can also have an associated rule that will be evaluated to determine if the predicate is `TRUE` or `FALSE`, in case the relevant facts do not exist.

b) Treating class instances and association instances as facts in a factbase for logical reasoning.

When the logical variable of the predicate representing a class does not have an associated value, we call it a free variable. The variable can bind to object instances of that class at runtime. Similarly, the logical variables of the predicate representing an association, binds to object instances of the classes between which the association is defined. Thus we treat the runtime class instances and association instances as facts in a factbase. When the logical variables are bound to a value and the corresponding class or association instance does not exist, then the predicate would return a value `FALSE`. On the other hand, if the instance does exist then the predicate would return a value `TRUE`.

Our framework assumes that these predicates automatically get a `first` and a `next` method to permit the fetch of the first instance from the factbase and the next subsequent set of instances. In the execution of a predicate rule, when all instances are considered, next method will return `FALSE`.

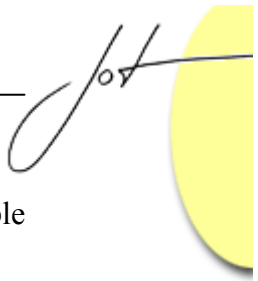
c) Allowing class instances and association instances as predicate parameters to permit binding, unification etc.

Business rules require a complex set of conditions that need to be evaluated and, if their conditions are `TRUE`, a conclusion is asserted (or derived / set). In our framework, the rules could have any number of predicates, some of which could represent classes or associations in the model. Other predicates in the rulebase can have parameters that bind to class instances or association instances of the model. This facilitates using the instances for logical operations such as unification, binding, freeing, and logical reasoning.

d) Allowing creation of class instances and association instances at run-time and make assertions and retractions on them.

Each predicate that represents a class or an association would have the following methods automatically created: assert, retract, bind, free, test, first and next. Class instances and association instances can be created at runtime with the assert method, and deleted with the retract method. The bind method will search for an instance and bind it to the predicate variables. The free method would un-bind the variables. The test method would check the existence of a given instance in the factbase. The first and next methods would get the first and subsequent instances from the factbase.

The above aspects are explained in the later sections with examples. It should be noted that our framework does not propose a formal specification language for addressing the



above aspects or its formal implementation mechanism. However, we discuss a possible mechanism to implement the framework using a pseudo business rules format.

## Mechanism to implement framework in a pseudo rule language

Business rule formats that are defined in literature for business applications follow either a Event-Condition-Action format, or a Logic-based format (viz. Data-driven rules or Goal-driven rules), or a procedural rule format [Date2000] [Dayal88] [Knolmayer94] [Streng94] [Tsalgatidou90] [Winston92] [SBVR05] [PRRP05]. These formats do not support the use of the power of first-order-logic constructs and logic programming in the rule language to define declarative rules on an object model. While a pure first-order-logic based rule language might be difficult for business users to define and modify rules, we believe that a fine mix of procedural rules with Prolog-like restricted first-order logic constructs would make a powerful framework for defining both declarative and procedural rules. A fine mix would allow seamless invocation of procedural rules from declarative rules and vice-a-versa.

In our approach, we treat classes and associations in an object model as predicates in a logical rule. This enables us to treat class instances and association instances as facts and make assertions and retractions on them. In an object model, this implies that we can make assertions of new objects (i.e. create new objects based on some conditions) and create instances of associations (i.e. associations can be dynamically created based on conditions specified in the rule). These are powerful techniques as they permit the modeling of dynamic states of an object model during the Meta-model and User model phase.

To illustrate this further, let us consider the following object meta-model as shown in Figure 1. We discuss the mechanism to specify the business rules on any such model, in a pseudo business rule language.

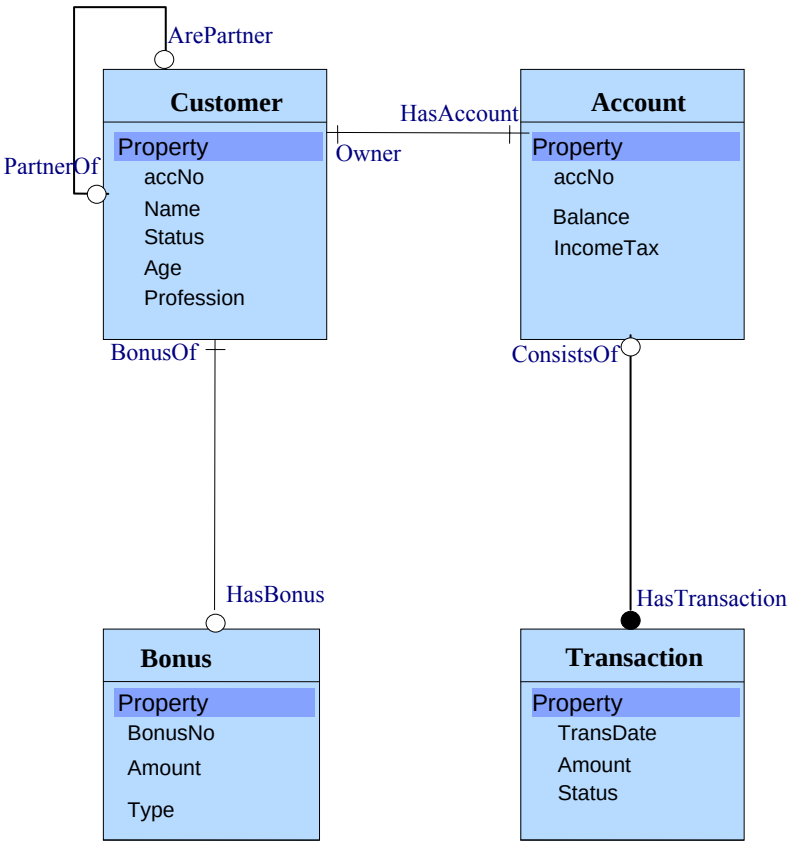



Figure 1: A sample object meta-model from a financial bank domain

## Declaring logical variables

The rule language permits the specification of logical variables that can bind to a value and can be freed from a binding. The binding should be permitted on object or association instances. An example would be a variable called `CustA` that could bind to object instances say "`Customer_Ram`", or "`Customer_Shyam`".

## Declare logical predicates

The rule language permits the definition of predicates that can bind to instances of a class or an association. For example, let us say that `Customer` is a predicate that can bind to instances of `Customer` class. We need to define its `arity` (i.e. the arguments that it will take) and the `type` in the model to which it will be attached; for example, the predicate `Customer` is attached to `Customer` class. Any assertions would occur for creating new objects in the `Customer` class, while any retractions would occur on the objects existing in the `Customer` class. A psuedo code of the predicate definitions is shown below.

```
PREDICATE Customer
ARITY 1
MODEL_TYPE Customer
```

Similarly, the predicate `HasAccount` would be attached to the association `HasAccount` and would have an arity of 2. The type to which this predicate would be attached would be both `Customer` class and the `Account` class. Any assertions and retractions would involve the instances of both the classes, and in-turn the instance of the association.

```
PREDICATE HasAccount
ARITY 2
MODEL_TYPE Customer
MODEL_TYPE Account
```

### Define Predicate rules

Definition of rules for predicates would allow a predicate to get a value either from the factbase or from the rules. For example, we could define a predicate called `Bonus` that could either be an asserted fact or be derived by a rule as shown below:

```
PREDICATE_RULE Bonus
IF ( THERE_EXISTS(Customer(CustA)) AND
CustA.Status  EQ 'Privileged' )
THEN assert_fact (Bonus (CustA));
```

### Manage a factbase

Asserting facts into a factbase and retracting facts from the factbase are basic mechanisms required. This should be implemented in such a way that the instances of classes or associations are also treated as part of the factbase. What this implies is that when an `assert_fact` operation is invoked with a class or association as the parameter, object instances and association instances can be created. Similarly when a `retract_fact` operation is invoked with a class or association instance as the parameter, the relevant instance is deleted. For example, in the rule shown in the previous section, `Bonus` is a class for which a new instance is created for `CustA`.

### Binding of facts

When predicates are given with unbound logical variables then facts in the factbase that are asserted for that predicate are searched and are bound to the variables of the predicate. For example, if we have a predicate `Bonus(x)` then it would return the fact `Bonus(x=CustA)`, where `x` is bound to `CustA` instance of `Customer` class. Since in our mechanism we include both class instances and association instances, this would work for both classes and associations.

## Backtracking rules

When the variable arguments of predicates in a rule are bound to some facts, these bindings are carried forward to other unbound arguments of the rule. At times the new bindings may not retrieve any fact or may not lead to a solution. When this occurs, the bindings need to be undone and the next possible fact is considered. This process is called backtracking. Implementing backtracking would also facilitate the search for multiple solutions. For example, in the `PredicateRule Bonus` above, say that we have two facts `Customer_Ram` and `Customer_Shyam` and at the first instance, let `CustA` bind to `Customer_Ram`. If the `Status` attribute of `Customer_Ram` is not '`Privileged`' then we need to backtrack to the previous predicate, unbind `CustA` and bind `CustA` to `Customer_Shyam`. Then the rule proceeds with evaluation of `Customer_Shyam`.

## Searching for single and multiple solutions

The pseudo rule language allows Universal(∀) and Existential (∃) operators representing operations such as "For All" and "There Exists". When combined with backtracking this can allow the search for single or multiple solutions (much in the same lines as an SQL statement). For example, `PredicateRule Bonus` could be defined as "multiple solutions" with the Universal (∀) operator preceding the definition. This would allow every `Customer` with `Status` as '`Privileged`' to be given a bonus.

## Mixing procedural and declarative rules

Using declarative rules inside procedural rules allows a mechanism wherein users can specify both procedural actions as well as the declarative rules. To do this, we treat every predicate rule as a function call that evaluates on its factbase (which can comprise of its internal factbase as well as the objects and association instances). The predicate rule returns either `True` or `False` value that may be used by the procedural rule. Similarly in a declarative rule, a procedural rule can be invoked by considering the rule reference as a predicate evaluation that returns either a `True` or `False` value.

Using such a mechanism, rules can be used to specify semantic checks based on object properties and associations, create objects at runtime, instantiate associations based upon some conditions, define rules on classes and their attributes. This can also be used for modeling integrity constraints on the model, derive new attribute values and specify rules to instantiate an association based on the evaluation of a rule, thus providing a mechanism for declarative modeling for object behavior.

# 5 EXAMPLES

## Rules on association

Let us assume that we want to model the rule that "Two customers are partners if and only if they hold the same bank account". In the present modeling mechanisms what can be modeled is shown in Figure 2.

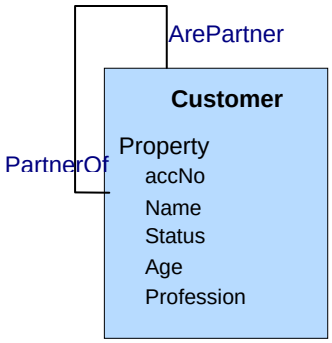



Figure 2: Object model of an association ArePartner

The fact that two `Customers` are `Partners` if and only if they hold the same account is not modeled; it would most probably coded in the application. The information is lost from the model. Instead, if the following rule could be added to the model on the association '`ArePartner`' then it makes the modeling more complete.

```
IF (THERE_EXISTS (Customer (custA)) AND
   THERE_EXISTS (Customer (custB)) AND
   THERE_EXISTS (HasAccount (custA, acc)) AND
   THERE_EXISTS (HasAccount (custB, acc)) AND
   custA not_the_same_as custB )
THEN    assert_fact (ArePartner (custA, custB))
```

Figure 3: Rule on the association ArePartner

The rule in Figure 3 states that the association '`ArePartner`' is instantiated only if there exists two unique customers who have the same account. With a `FOR_ALL` at the rule property level, this rule can automatically detect and assert the association '`ArePartner`' for all customers who have the same account in the bank. Further, when one of the `Customer` withdraws, this fact can be retracted.

## Rules on attributes

Let us assume that we have a rule that states "For a customer, whose account balance is less than minimum balance, assert the fact that he cannot withdraw". We can define a backtracking multiple-solution rule as follows (Figure 4):

```
RULE MinBalanceCheck
BACKTRACK MULTIPLE_SOLUTION
IF      ( THERE_EXISTS ( Customer(custA)) AND
        THERE_EXISTS ( HasAccount(custA, acc)) AND
        acc.Balance < MinimumBalance)
THEN    CanNotWithdraw (custA);
```

Figure 4: Rule on attributes

This rule will get all the customers from the model, check if the `Customer` has an `Account` using the association `HasAccount` and check if the `Balance` in the `Account` is less than `MinimumBalance`. If such a `Customer` exists, then method `CanNotWithdraw` is called with `Customer` object as a parameter.

As a summary, the figure below (Figure 5) shows how the rules can be written on the model.

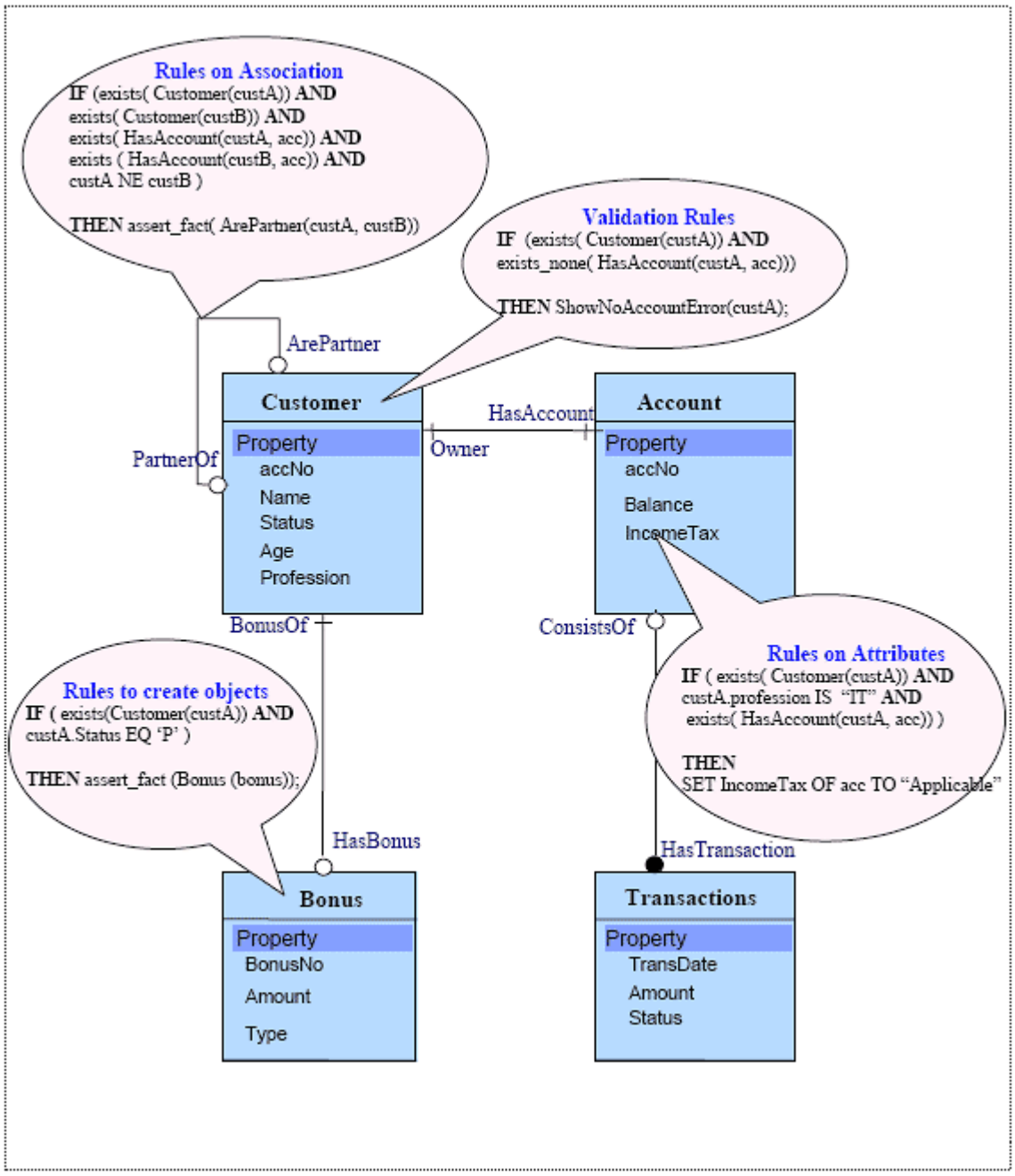



Figure 5. Writing rules on the objects, attributes and associations

## 6 ADVANTAGES OF THE FRAMEWORK

There are a number of advantages of the proposed framework. The first is the ability to specify business rules on associations in an object model, during modeling. Such business rules may specify the conditions when an association should be instantiated between

runtime objects in the domain. Previously such rules have normally not been explicitly stated in the object model, and were either hidden in the code or stated separately in a rule language. The framework facilitates the definition and thus enables an object model to be more representative of the real application world. The framework also permits the definition of logical rules on objects and their attributes during modeling. While all rules are not expected to be defined in the modeling stage, this mechanism permits the "modeling" of those rules that are known or apparent during modeling. Thus a model can be considered more complete than one without such definitions. Another advantage could be in the area of model validation. A model can have both constraints and rules. These rules and constraints would be checked to determine if there is any violation. While OCL permits the definition of constraints, our framework permits logical rules in addition to constraints. Run-time instantiations of associations based on conditions specified in the rules is possible. This is a major advantage over existing methods. Associations can be instantiated dynamically based on some conditions. Operations can then be performed on them. Derivations can be made on the model by defining rules on the objects. This means that based on a set of conditions, logical inferences can be derived on the objects. These are major advantages over a constraint language like OCL wherein logical reasoning which is available in logic programming cannot be carried out.

We have implemented these concepts in experimental extensions of commercial products of our company, viz. the MasterCraft Component Modeler (an object modeling and code generation tool) and the Infrex Rule Engine (a business rules engine), and have observed the advantages.

## 7 CONCLUSIONS

We have proposed a framework for specifying declarative rules on objects, attributes and associations. The framework permits definition of some of the business rules during modeling that need not be buried in the code or specified separately, especially rules on associations. Using a declarative rule language and rule engine it is possible to apply these concepts. Further extensions can be envisaged in terms of incorporating more powerful pattern matching in the rule formats, generating XML compatible / BRML compatible output format from the rules.

## REFERENCES


[Ait-Kaci91] Hassan Ait-Kaci, Warren's Abstract Machine: A Tutorial Reconstruction, MIT Press, 1991.

[Date2000] C. J. Date, What Not How: The Business Rules Approach to Application Development, Addition Wesley Longman Inc., 2000.

[Dayal88] U. Dayal, A.P. Buchmann, and D.R. McCarthy, "Rules Are Objects Too: A knowledge Model for an Active, Object-Oriented Database Management System", in: K.R. Dittrich (ed.) *Advances in Object–Oriented Database Systems*, Springer, Berlin, (1988), pp. 129-143.

[Forgy82] C.L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence*, 19,(1982) 17-37.

[Knolmayer94] G. Knolmayer, H. Herbst, M. Schlesinger, "Enforcing Business Rules by the Application of Trigger Concepts", *Proceedings of Program Informatics Research, Information Conference Module 1*, Berne, 1994.

[MOF03] Meta Object Facility (MOF) 2.0 Core Specification version 2.0, ptc/04-10-15, October 2003.

[Nilsson82] N. J. Nilsson, *Principles of Artificial Intelligence*, Springer Verlag, Berlin, 1982.

[OCL03] OCL 2.0, OMG Final Adopted Specification, ptc/03-10-14, October 2003.

[PRRP05] Production Rule Representation Proposal, OMG Draft 1, 2005.

[RuleMachines05] *Business Rules and the Component Object Model*, Whitepaper, RuleMachines Corporation, 2005.

[SBVR05] Semantics of Business Vocabulary and Business Rules (SBVR), bei/2005-03-0, Version 7.0, OMG.

[Streng94] .J. Streng, "BPR needs BIR and BTR: The PIT framework for Business Reengineering", *T.J. Proceedings of Second SISnet Conference*, Barcelona, 1994.

[Tsalgatidou90] .Tsalgatidou, V. Karakostas, P. Loucopoulos, "Rule-Based Requirements Specification, and Validation", in: B. Steinholtz, A. Solvberg, L. Bergman (Eds.), *Proceedings of the Fourth International Conference on Dynamic Modelling and Information Systems Engineering*, Berlin et al: Springer, (1990), pp. 251-263.

[Winston92] .H. Winston, *Artificial Intelligence*, Third Edition, Addison-Wesley, 1992, Pages 119-161.

[Yoder02] Joseph Yoder, and Ralph Johnson, "The Adaptive Object-Models Architectural Style", in *Working IEEE/IFIP Conference on Software Architecture*, 2002.

## About the author

**C. Anantaram** is a Senior Consultant in Tata Consultancy Services Limited, New Delhi. He received his M.Sc.(Tech.) from BITS, Pilani and Ph.D. degree in Computer Science from IIT Bombay, India. His areas of interest are knowledge engineering, object oriented technology, natural language interfaces, and integrating AI in business applications. He is a member of Government of India's Working Group on Metadata and Data standards for e-Governance applications. His email is c.anantaram@tcs.com.