1. Introduction

2. Object-Orientation

3. Business Rule Modeling
   3.1 Trigger Events
   3.2 Pre-Conditions
   3.3 Actions
   3.4 Guidelines in Modeling Business Rules
   3.5 Instantiation and Destruction

4. Business Rules Implementation
   4.1 Rules Processor
   4.2 Rules Repository
   4.3 Rules Manager

5. Business Rules and Relational Databases
   5.1 The Shortcoming of Existing Relational Databases
   5.2 The Object-Relational Mapping
   5.3 Data Normalization
   5.4 Accessor Procedures and Methods
   5.5 Containers
   5.6 Calculated Fields
5.7 Data and Referential Integrity

5.8 Instantiations

5.9 Events

1. **Introduction**

   Business rules is a general term applied to the attempt of bridging the gap between decision makers and information systems. Traditionally, once the decision making body within a business formulates a plan for action, it must rely on a team of computing experts to firstly translate the plan from the business domain to the IS domain and secondly to execute the plan within the IS domain. Business rule sidesteps the IS personnel and allows decision makers to directly implement their plans through the information channel. What business rules is trying to accomplish is ambitious, but the mission is multi-faceted and can be broken down into smaller, more manageable goals. Business rules also has ancillary benefits because thinking about business decisions within a computing context in turn forces a degree of rigor on the formulation of business rules themselves. Decision makers need to be more precise and systematic in the way they make decisions.

2. **Object-Orientation**

   The Law of Demeter states that a method definition should only contain messages sent to the receiver, the superclass of the receiver, the parameters, and temporary variables.

3. **Business Rule Modeling**

   A business rule contains a triggering event, a pre-condition, and an action. Event hooks inserted into method definitions create triggering events. When an object encounters an event hook during method execution, the event hook provides a way for passing control flow to a rules processor. The event hook provides the rules processor with access to all the public method objects. The pre-condition is a logic statement of the states of the public method objects. If the pre-condition evaluates to true, the public method objects will execute a set of pre-specified methods prescribed in the action component of the rule before returning control to the original method.
1. **Trigger Events**

   Inserting event hooks inserted into method definitions create trigger events. Event hooks play two important roles in business rules. First, it signals to the rules processor that some event has occurred which could necessitate further action as specified by preestablished business rules. Second, it passes by reference all the public method objects to the rules processor. This enables the rules processor to check the state of the public method objects in the pre-condition component of the business rule.

   The **triggering object** is the receiver of the method that contains the event hook. The **triggering method** or **triggering message** is the method itself that contains the event hook.

2. **Pre-Conditions**

   The trigger passes all the public method objects to the rules processor. A business rule can take advantage of these pointers to check the states of various public method objects before any action is carried out. The pre-condition component of a business rule is where the state checking takes place. The pre-condition is a boolean expression formed from the states of public method objects.

   The pre-condition expression can only use the states of the public method objects because event hooks and any subsequent execution is still within the method definition of the triggering method. As such, the Law of Demeter dictates that messages can only be sent to the public method objects.

   A pre-condition is not a necessary component in a business rule. A missing pre-condition implies that the pre-condition always evaluates to true.

3. **Actions**

   The action component performs a series of method executions by sending messages to the public object methods. The execution of the methods is contingent on the evaluation of the pre-condition.
Only if the pre-condition evaluates to true will the messages in the action component be sent.

4. Guidelines in Modeling Business Rules

To formulate meaningful rules, the three pieces of the business rule must be strung together through careful analysis -- triggering event, pre-condition, action. Some design guidelines are necessary to restrict the structure of a rule; otherwise the rules become too multifarious to be effective. Following are guidelines to adhere to when designing business rules:

**Every rule must have an event and an action.**

Without triggering events, the event-driven domain is missing and rules become mere restatement of methods. On the other hand, triggering events without actions is equivalent to sentences without verbs: They are not sentences. Without actions, the entire processing domain is missing and rules become alarms that warn of certain happenings.

The only non-required component in a rule is the pre-condition. Even then, a missing pre-condition always implies a pre-condition that evaluates to true.

**Triggering events and pre-conditions must be class-based.**

This is a subtle, yet crucial point, towards understanding rules. Designing class-based rules enforces an instance-blind approach to rules. No components of the business rules can base themselves on a particular instance. They should base themselves on the classes of the public method objects passed into the rules processor through the event hook.

For example, the following rule is class-based:

When a customer places an order

If the customer has satisfactory credit rating
Then the customer’s sales representative can take the customer’s order.

In contrast, the following rule is instance-based:

When John Doe places an order.

If John has satisfactory credit ratings

Then Jane Doe can take John’s order.

The class-based rule is more universal in coverage. It applies to all instances of the class, whereas the instance-based triggering event only works for a single instance. To enforce class-based business rules is to enforce the power of class encapsulation and reuse that object-orientation offers. Furthermore, during rules design, the rule maker should only have knowledge of the class of the public method objects, but not which instance will actually end up triggering. To have an instance in mind while designing rules is similar to enumerating exceptions to a rule before making a rule.

5. **Instantiation and Destruction**

Instantiation is the process of creating a new object. The class of the new object implements the instantiation service for the new object; and as such, inserting event hooks into the instantiating methods would always return the class as the triggering object. Business managers might be interested in singling out the new instance for special processing. For example, the following rule might be desirable:

When a new employee joins the company, the security department must issue him an ID.

The above rule violates a rule guideline. The business manager has based a rule on an instance, the new employee. This violates the guideline that rule triggers and pre-conditions must be class-based.

The class performs instantiation to ensure that newly created objects can trigger events at the earliest opportunity.

4. **Business Rules Implementation**
The rules system bears three responsibilities. Firstly it is responsible for providing an interface for rule makers to define new rules and to edit or delete existing rules. Secondly, it is responsible for maintaining the rules database for storing rules. Thirdly, it is responsible for recompiling event handlers whenever it modifies pertinent rules.

1. **Rules Processor**

   A rules processor is responsible for installing rules into the domain model. The processor first scans the rules repository, which stores all the rules. The processor will pass each rule through any language translators. Next the processors will insert the pre-conditions and actions into the event handler associated with the rules event. These event handlers are then recompiled and incorporated into the runtime domain model.

   The rules processor is the most critical part of a rules system. It is also the most complicated aspect of a rules system. Because it is the only bridge between the rules system and the domain model, the rules processor must reconcile any language differences. It is quite common for the rules system to reside in a relational database application while the domain model rests on a more pure object-oriented language. The rules processor must be able to convert rules phrased in something not too far from the natural language, stored in a relational database to object-oriented code.

2. **Rules Repository**

   The rules repository provides storage for defined rules and reusable rule components.

3. **Rules Manager**

   The rules manager is the interface for the rules repository. At the minimum, a rules manager should be able to display existing rules and display existing rules components available in creating new rules. Further enhancements could include a class hierarchy browser for defining new events and an expression builder for assisting in the construction of the pre-condition.
The rules manager is also where the user can direct the rules processor to implement the business rules in the domain model. When the rules processor encounters an error, it reports the error back to the rules manager.

The rules manager is also the place where a natural language translation interface can be installed. Natural language translation allows computer codes to be interpreted into English and vice versa. Implementing business rules in object-oriented syntax lessens the need for natural language translation because a good object model should have nomenclature that closely resembles natural language constructions. A motto of object-oriented design has been: "Model every message send after a sentence, replete with a subject, a verb, and an object."

The rules manager can easily contain a simple phrase translator. A phrase translator allows users to substitute more complicated statements with user-defined string constants or phrases. Whenever the user types in the phrase, the translator will provide recursive translation before any compilation.

5. Business Rules and Relational Databases

Most businesses employ some form of relational database to store and process information. Business rules can play a valuable role in relational databases by centralizing rules commonly found in databases such as validation rules, calculation rules, referential integrity rule, etc. Furthermore, by installing an object-oriented layer to work with the relational database, data and method encapsulation becomes possible, paving way to more powerful procedures and therefore more comprehensive implementation of business rules.

To fully tap the power of a business rules system, an object-oriented layer for interacting with the databases is a necessity. This is because object-orientation offers an intuitive foundation for business rules. A relational database lacks such a foundation. Without an object-oriented layer, business rules will be restricted to simple validation and referential integrity rules. Complex rules will be difficult to formulate and to understand.

1. The Shortcoming of Existing Relational Databases
Most commercial database systems, already offer means for dealing with data and referential integrity, but these rules must be put in place during design time effectively putting them out of reach of the non-programmer. Nor are these rules centralized so there is no easy way of examining all the rules in force. More powerful database systems have incorporated hybrid forms of object-orientation. These systems are one step closer to the object-oriented business rules paradigm preached thus far. The object-orientation could relieve the business rules architect of the trouble of having to create an object-oriented layer before considering a business rules system, but the degree to which the database system can support object-orientation varies from database to database. Because a business rules system requires almost a pure object-oriented environment to fully demonstrate its prowess, additional coding would have to provide some form of object-oriented support.

2. **The Object-Relational Mapping**

Object and relational models share similar data modeling concepts. In object-orientation, objects are grouped into classes. In relational databases, objects are grouped by tables. Object-orientation emphasizes the dynamics, functional-oriented portrait of objects while relational databases emphasizes the static, data-oriented portrait of objects.

The following table list additional modeling concepts equivalences between the two modeling methodologies:

<table>
<thead>
<tr>
<th>Object-Orientation</th>
<th>Relational Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Table</td>
</tr>
<tr>
<td>Instance</td>
<td>Row</td>
</tr>
<tr>
<td>Attributes</td>
<td>Fields</td>
</tr>
<tr>
<td>Collections</td>
<td>Joined Tables</td>
</tr>
<tr>
<td>Methods</td>
<td>Stored Procedures</td>
</tr>
<tr>
<td>Pointers</td>
<td>Primary Keys</td>
</tr>
</tbody>
</table>
3. Data Normalization

Data normalization in relational databases is equivalent to class parsing in object-orientation. Having normalized data means that the database designer has adequately categorized heterogeneous data into separate tables and has established join conditions between tables to handle one-to-many relationships. Because object-orientation treats each table as a single class, having normalized data, and therefore properly parsed class hierarchy, is critically important for development of business rules.

Also important in data normalization is to ensure that proper key fields have been identified and that uniqueness of keys is enforced. Because relational databases do not support pointers, the object-oriented layer would need to use the key fields to mimic pointer behavior.

4. Accessor Procedures and Methods

In the realm of object-orientation, methods are bound to classes. In relational databases, procedures and functions provide the functionality of methods. Binding procedures and functions to tables will mimic object-orientation. Sometimes, stored procedures that encapsulate procedures within a table or database the binding can accomplish the task of binding. Often, stored procedures are either not available or not fully encapsulated, in which case the binding of procedures to tables has to go through a table that records the correspondence between tables and procedures.

The most important object-oriented methods to model in the relational database are the **accessors**. Attributes of an object are retrieved and changed through the use of accessors. Each attribute generally gives rise to two different accessor methods, a **get-method** to return the current value of the attribute variable and a **set-method** to change or initialize the current value of the attribute variable. To introduce accessors into relational databases, each field must have two procedures, a get-procedure and a set-procedure. The exact syntax used in these procedures will vary from database to database, but for systems that support SQL, the following two statements will work:
Having properly implemented accessors means that a rules processor can completely control the data aspect of a table, which is extremely important in implementing business rules. Further, because the relational database focuses on data, having accessors means that the business rules system can endogenize up to 75% of the rules.

5. Containers

In an object-oriented environment, an object can contain multiple instances of other objects. Pointers simplify the containment relationship between objects so that no special attention is required. Because relational databases use keys to replace pointers, the containment relationship between objects needs to be explicitly maintained.

Relational databases impose an important restriction on the containment relationship in that the contained objects, or children, must keep track of their containers, or parents. Parents keeping a record of their children cannot be easily done in the relational world.

6. Calculated Fields

Calculated fields provide numerous opportunity for business rules. To stage calculated fields for rules modeling, accessors are created as for non-calculated fields. The only difference lies in the set-procedure. For calculated fields, the resulting value is based on a function with other fields within the same table or within a related table as arguments. Because the set-procedure for calculated fields is different than for non-calculated fields, the set procedure will be termed the recalculating procedure.

Besides introducing recalculating procedures that applies the function for updating calculated fields, rules analysis would actually delve into the function itself and look for decompositions that are
both mathematically and logically viable. The decomposition generates different functions that could be used in business rules, giving rule makers greater choice in rule design.

As an example of the decomposition, suppose that a grocery store has a table containing all products with the following fields:

- Product ID
- Cost
- Markup
- Retail Price

*Final Retail Price* is a calculated field using the following formula:

\[ \text{Final Retail Price} = \text{Cost} \cdot (1 + \text{markup}) \cdot (1 + \text{tax}) \]

The above formula breaks down into two computational steps as follows:

\[ \text{Retail Price} = \text{Cost} \cdot (1 + \text{markup}) \]

\[ \text{Final Retail Price} = \text{Retail Price} \cdot (1 + \text{tax}) \]

The computational decomposition is congruous with logical decomposition. The first step applies the markup to the cost; the second step applies the sales tax. The single step calculation uses a single procedure. The two-step calculation can use two procedures. Business rules modeling prefer the decomposition because it gives the rule maker more flexibility and finer control over calculated fields. If the rule maker decides to compute pre-tax prices, he must delve into the blackboxed application, change the formula, and recompile -- an impossible task for a rule maker without programming capabilities. By having logically and computationally divided independent calculations and incorporated them into business rules, the business manager has more enhanced capabilities over changing the internal workings of the application without having to resort to recoding.
7. **Data and Referential Integrity**

Databases have standard rules to preserve data integrity and referential integrity. Because data and referential integrity rules can be easily handled by the business rules system, they should not be considered an issue in the underlying database design. Doing so would detract from the mission of business rules system to centralize all rules processing.

8. **Instantiations**

Instantiation is the process of creating new objects based on the blueprints laid out in the class definition. For relational databases, instantiation amounts to adding a new row to a table. In relation databases, instantiation must accomplish the important task of assigning a unique key value to the new row.

For business rules, instantiation should do the minimum necessary to create the object and leave further initialization for rules.

9. **Events**

Rules require triggering events. In object-orientation, methods explicitly trigger events. To implement event hooks in a relational database, a procedure call can be placed before and after a method execution. For example, two event hooks can be added to a set-procedure as follows:

```
BeforeField1Change(ID,newvalue)
UPDATE table1
SET field1 = newvalue
AfterField1Change(ID,newvalue)
```

The event hooks passes control to the event handler functions. It also passes all the public method objects to the event handler for use in the pre-condition and the action. The first parameter is always the triggering object. In the relational database, a primary key (ID) serves as the pointer to the triggering object.