This article describes three models that are necessary in order to implement a successful IBM® ODM business rule project, the domain conceptual and logical model, the executable object model (XOM), and the business object model (BOM), and the roles they play in developing an IBM ODM business rule solution. It describes the IBM tooling and the analysis process used to create the models and includes examples. The analysis process draws heavily on the Term–Fact analysis process described in Ronald Ross's *Principles of the Business Rule Approach*, and the Unified Modeling Language (UML). The IBM ODM Rule Designer and Decision Center components facilitate creation of the business rule vocabulary and grammar, the Business Object Model (BOM), the rule project organization, and the actual business rules. The rule testing and simulation environment, Decision Validation Services (DVS), reflects the XOM in a Microsoft® Excel® format that business users use to test the rules they define.

**Introduction**

Business rule technologies, including IBM Operational Decision Management (ODM) rules, have been steadily gaining in popularity over the past two decades. Business rules offer the promise of enhancing business agility by enabling business users to manage business rules that have been externalized from business application source code. Business user rule management includes maintenance of a repository of rules that have been developed to enforce business operational policies. Using the tools provided by the business rule management system, BRMS, components of ODM rules, business users are able to view the business rules that enforce the business policies and internal and external regulations governing the way the business operates. Business users can read the individual rules, identify the rule’s status, determine who last modified the rule and when. Business users are able to edit rules in the business rule repository, test the rules using sophisticated testing tools, and ultimately deploy newly modified or newly authored rules into the operational production environment with little or no assistance from the IT staff.

None of this rule management activity would be possible were it not for three models that must be developed in order to enable business rule development. These three models are:

- The Domain Class Model represents the conceptual and physical entities and the logical relationships between those entities that reside in the business's problem domain. The
Domain Class Model forms the foundation for the creation of the other two models required for a successful business rule implementation.

- The Execution Object Model, referred to as the XOM, which is developed from the domain class model, serves as the basis for the creation of the business vocabulary and grammar that are used to author business rules, and provides the object instances against which the rules are evaluated and ultimately executed.
- The Business Object Model, referred to as the BOM, provides the business vocabulary and grammar that are used to author business rules. The BOM is created from the XOM.

This article describes the process of creating the three models providing examples of the process in actions and describes the use of the IBM tools that simplify the process.

**Overview of the Unified Modeling Language Notation**

In this article, we use the Unified Modeling Language (UML) exclusively in modeling the business problem domain. Therefore, it’s important to ensure that you understand the notation.

**Class notation**

The class is represented by a rectangular structure that is typically divided into three sections. The uppermost section of the class notation contains the class name. The class name is commonly written using camel case, a form in which words are run together with the first letter of each word capitalized, for example `ClassName`. The central section of the class notation contains the names of characteristics or attributes of the class along with the type of the value of the attribute separated from the attribute name by a colon. The attribute names are typically nouns. When the attribute name contains multiple words the words are run together, and the first letter of the first word is lower case while all subsequent words are capitalized, for example `attributeName : type`. The third section of the class notation contains the names of behaviors or methods that are performed by instances of the class. The method names are typically verbs and are written in the same manner as the attribute names with trailing parentheses, for example `writeToStream()`.

**Figure 1. UML class notation**

![UML class notation](image)

**Relationship notations**

Classes may be related to one another in multiple ways. There are four commonly used relationships:

1. Generalization/specialization
2. Composition
3. Aggregation
4. Association

With the exception of the generalization/specialization relationship, the UML relationships can indicate navigability, the roles that the classes perform in the relationship, and the multiplicity of instances of classes in the relationship.
In UML 2.0, bidirectional navigability is indicated by a straight line. Unidirectional navigability is indicated by an open arrow head (->) at the target class end of the relationship. In a business rule context, navigability indicates that one class "knows" about another class.

The roles performed in a relationship are represented by a label at the performing class's end of the relationship, for example Figure 2 indicates that zero or more instances of the Engine class perform the engines role in an aggregation relationship with the MotorizedVehicle class.

**Figure 2. Role in an aggregation relationship**

![Aggregation relationship diagram](image)

Multiplicity is represented in several ways:

- * - indicates 0 to many, optional
- n..* - indicates n to many, at least n are mandatory
- 1..* - indicates 1 to many, at least one is mandatory
- n..m - indicates a minimum of n instances and a maximum of m instances

I'll use an example in this article to describe each of the relationships, except the composition relationship, along with unique characteristics of each relationship.

**UML generalization/specialization notation**

The UML generalization/specialization notation is used to establish the relationship between a super class and its subclasses. The super class is a generalization of the subclass and conversely the subclass is a specialization of the super class. The generalization/specialization relationship is used to represent the inheritance of attributes and behaviors, where the subclass inherits the attributes and behaviors of the super class and may add attributes and behaviors that are unique to the subclass. The subclass may also override the implementation of an inherited behavior, thus introducing polymorphism, one of the distinguishing features of object-oriented analysis and design. The colloquial term for the generalization/specialization relationship is "IS-A" indicating that the Subclass "IS-A" super class. The generalization/specialization relationship is represented by a straight line with a triangular arrow head. The arrow head is attached to the super class with the blunt end of the line attached to the Subclass.

**Sentence rules for using generalization**

If one of the following sentences makes sense in the domain that is being modeled, generalization applies:

- A subclass is a kind of super class.
- A subclass is like a super class.
UML composition relationship
The UML composition relationship is used to indicate that one class is a component of another class. Aside from the generalization/specialization relationship, the composition relationship is the strongest of relationships represented in UML. The composition relationship indicates that the individual components of the relationship cannot exist in isolation. Deletion of one of the components causes the deletion of the other component. The UML composition relationship is represented by a filled diamond anchored on the composite class with an arrowhead anchored on the component class. The composition relationship is always represented as a unidirectional navigable relationship.

Performing Term–Fact analysis
Term–Fact analysis is described in Principles of the Business Rule Approach, by Ronald G. Ross. Term–Fact analysis is the foundation for creating a business domain model. In his book, Mr. Ross...
describes terms as words and phrases that are used by business people in the daily execution of their business responsibilities and facts as combinations of terms that describe what business people know to be true about their business.

Terms
Terms are individual words or phrases that have meaning to business people in the performance of their daily business duties. Terms can describe abstract business concepts such as contract or concrete business entities such as a physical product. Terms can describe characteristics of a business concept such as a contract’s start date or a product’s size and shape. In the UML modeling context, when a term describes a business concept that concept can be modeled as a class. When the term describes an attribute of a business concept, that term can be used as the name of an attribute of a class.

Understanding facts
Facts are combinations of terms that describe what business people know about their business. Facts can describe the relationships between business concepts and can therefore be modeled as relationships between classes in a UML class model. Facts can describe the behaviors of business concepts and can be modeled as methods defined for the classes that model the business concepts. Facts can also describe the manners in which entities in the business domain interact. Facts provide the logical glue for the business domain class model.

Describing business policies and regulations
Business policies and regulations are positive statements that describe the way businesses must operate and interact during the daily conduct of commerce. Business policies and regulations are composed of facts, as described by Mr. Ross. Business policies and regulations require enforcement in order to be effective. Software rules such as those created in IBM ODM rules can be used to enforce business policies and regulations automatically. Business policies and regulations are composed of facts and lend themselves to the term–fact analysis process. In Principles of the Business Rule Approach, Ross provides a dozen templates that provide standard formats for expressing business policies. Six of the most common templates are shown in Listing 1. These templates are not only useful in writing rules that enforce the business policies, they are also highly useful in modeling the business domain where the policies must be enforced.

Listing 1. Six commonly used business policy templates

Rejected:
<Subject> must/must not <fact> [if/when <condition statement>] <start date> <end date>

Permission Statement:
<Subject> may/need not <fact> [if/when/while <condition statement>] <start date> <end date>

Computation Rule:
<Subject> = <mathematical formula> [if/when/while <condition statement>] <start date> <end date>

Derivation Rule:
<Subject> must/must not be taken to mean <logical expression> [if/while/when <condition statement>] <start date> <end date>

Inference Rule:
<Subject> must/must not be considered to be a <term> [if/while/when <condition statement>] <start date> <end date>

Imprint Rule:
<Term> must/must not be set to <term/value> [if/when/while <condition statement>] <start date> <end date>
A modeling example: planning distribution for a Canadian refinery

The example described here is fictitious, however the descriptions are based on real-life situations.

The Acme Refining Company currently plans product distribution using MicroSoft Excel. The distribution planning system no longer meets the planning needs, so the Acme Refining Company has launched a project to design a rule-based distribution planning solution. The following information has been gathered during a discussion with a business SME who works at the Acme Refining Company.

About the Acme Refining Company

Acme Refining Company is a medium-sized, privately owned oil refinery located in Saskatchewan, Canada. The Acme Refining Company produces four products:

- 87 Octane, Regular Gasoline
- 89 Octane, Mid-grade Gasoline
- 91 Octane, Premium Gasoline
- Low Sulphur Diesel Fuel

The Acme Refinery is located at a large complex occupying several hundred square acres. The refinery consists of a crude oil storage tank farm, the crude oil refinery, two tank farms where the refined gasoline and diesel fuel are stored, a railroad siding, hundreds of miles of pipe, hundreds of pumps, thousands of valves, and a distribution plant from which products are distributed to the Acme Refining Company’s 200 customers.

The refinery receives crude oil via railroad tanker car, each car contains approximately 30,000 U.S. gallons or 24,980 imperial gallons of crude oil. Oil is pumped from the tanker cars, that are positioned on the railroad siding, into the oil storage tanks.

The refinery stores crude oil in a tank farm consisting of 50 large tanks each of which has a 5,000 cubic meter capacity, approximately 1,320,850 U.S. gallons or 1,099,883 imperial gallons. The total capacity of Acme Refining Company’s crude oil tank farm is approximately 54,994,171 imperial gallons.

Each crude oil storage tank has a "collection area" at the bottom of the tank that allows for the collection of sediments and impurities. This "collection area" holds approximately 20,000 imperial gallons in the bottom of each tank. Oil is pumped into the storage tanks from the top of the tank. As a result of the placement of the inflow pipe, the capacity of each tank is reduced by 10,000 imperial gallons. Thus, for business purposes, each tank can contain approximately 54,964,171 imperial gallons. Oil is pumped into and out of the tanks of the tank farm using 30 electric pumps that are controlled by software that runs at the main pumping station. Each pump is able to pump 10,000 imperial gallons per hour.

The Acme Refining Company's refinery is able to refine diesel and one of the three grades of gasoline at a time. The refinery is capable of refining 7,200,000 imperial gallons per week when operating at 100%. The plant operates at 80% of capacity in order to accommodate maintenance...
requirements within the refinery, thus producing 5,760,000 imperial gallons of fuel weekly. Of the 5,760,000 imperial gallons produced each week, 1,152,000 imperial gallons are diesel and 4,608,000 imperial gallons are gasoline. The refinery produces 87 octane gasoline four days a week, 89 octane gasoline two days a week and 91 octane gasoline one day a week. The refinery operates three, eight-hour shifts daily, seven days a week.

The Acme Refining Company maintains two separate fuel storage tank farms that contain their gasoline and diesel fuel products. These tank farms contain 21,997,668 imperial gallons of gasoline in 20 tanks and 10,998,834 imperial gallons of diesel fuel in 10 tanks. Each fuel storage tank has a 5,000 imperial gallon collection area at the bottom where sediment is captured. The sediment containing fuel is never delivered to the refinery's customers.

Each of the refined fuel tank farms contains loading docks that are used to load the tanker trucks that haul the fuel to the company's customers. Each loading dock can accommodate three tanker trucks. There are four loading docks in the gasoline tank farm and two loading docks in the diesel tank farm. The loading docks have a pump for each loading station. Each loading station can pump 3500 imperial gallons per hour.

In addition to refining crude oil, the Acme Refinery distributes its refined products to remote locations in northern Canada using tanker trucks each capable of carrying between 9,000 and 9,300 U.S. gallons or 7,494 to 7,744 imperial gallons. The Acme Refining Company maintains a fleet of 30 tank trailers and 32 tractors. 10 trailers are used to transport diesel with the remaining 20 trailers used to transport gasoline. Acme Refining Company is able to lease tankers from independent truckers to augment their internal delivery capabilities as necessary. In addition, the Acme Refining Company also sells their product to independent truckers who deliver to non-contract customers.

All the equipment used at and by the refinery, including the tractors that pull the fuel tankers, the loading docks, and the pumps require periodic maintenance and are subject to emergencies that result in a reduction in the overall output capacity of the refinery.

The Acme Refining Company's customers are long-term contract customers who have agreed to buy predetermined quantities of the company's products, so the demand is fixed. The preponderance of the company's customers are located in remote regions near the Arctic Circle and are reachable only over ice roads. Because the company's customers are so remote, they all maintain tank farms for the storage of gasoline and diesel fuel. Ice roads are available to the company's fuel tankers from late September until mid-June. Therefore, the company must maximize its distribution to customers that are reachable only via ice road to ensure that the customers have sufficient fuel on hand to last through the summer months from mid-June to late August.

The Acme Refining Company schedules major maintenance during the summer months but continues to refine gasoline and diesel fuel, which it sells on the open market to local customers.
Modeling the Acme Refining Company domain

Modeling for business rules follows an iterative approach. This article describes the first iteration of the models needed to support rule authoring for a distribution planning process. The modeling approach is to start with as simple a model as possible, while applying the principles of abstraction, encapsulation, and polymorphism. The model is designed to support extensibility by enabling addition of classes, attributes, associations, and behaviors as knowledge of the distribution process expands.

Modeling the plan

The business rules solution will ultimately create the distribution plan. Every plan has a planned start date and a planned end date, and these dates constitute the planned duration of the plan. A distribution plan is a plan that has an aggregation of individual delivery plans. Each delivery plan identifies the product to be delivered, the amount of that product, the delivery date, the tank farm that is the origin for the delivery and the tank farm that is the destination for the delivery, the customer for which the delivery is being made, and the tractor and tanker trailer designated to make the delivery.

Figure 7. Plan models

Modeling the domain facilities

The Merriam-Webster online dictionary defines a facility as: "something (as a hospital) that is built, installed, or established to serve a particular purpose".

Delivery of gasoline and diesel fuel is a logistics activity and requires the ability to calculate the length of time required to move between two locations, in this case, two tank farm facilities. In addition, there is a need to calculate the length of time required to load and unload a tanker truck in order to create a delivery plan.

Every facility has a location which, for the purposes of the distribution planning process consists of latitude and longitude to support calculation of the distance between two facilities. A refinery, a tank farm, and a staging area are all instances of a facility, as shown in the facility model in Figure 8.
Figure 8. Facility models

Several items of equipment such as pumps, storage tanks, tractors, tanker trailers, rail tankers, and loading docks are involved in the creation of a distribution plan. When creating a distribution plan, the status of the equipment that is used to perform the distribution is an essential planning element. All equipment has a maintenance schedule, which will make select items of equipment unavailable for some period of time while the equipment is maintained. In addition, equipment may become unavailable as a result of unscheduled maintenance.

Figure 9. Equipment models

Modeling products

The refinery has four products that are composed of gasoline with three different octane levels and one grade of diesel fuel.
**Creating the executable object model (XOM)**

The executable object model (XOM) is the model against which the rules are executed. ODM can use a Java™-based XOM or an XML Schema Definition-based XOM, also called a dynamic XOM. Using a tool like Rational® Software Architect (RSA), creating the XOM is a relatively simple task.

Figure 11 shows the Rational Software Architect wizard that is used to create the UML to Java transformation file. In this case, the file is acmeUMLToJavaXform.tc. Using this wizard, you can convert the UML classes to Java source code, including generating the setter and getter methods for all attributes. The only work that a developer needs to do is to implement constructors for the classes, which can be done using a simple feature of Eclipse that auto-generates a no argument constructor. The mechanics of XOM creation are not as important as the creation of the XOM itself.
The Acme Refining Company XOM

Selecting the Run button in the wizard results in the creation of the .java files listed in Figure 12.

Figure 12. Class files of the Acme XOM

![Image showing class files]

Example code from the XOM

The code fragments below are two examples of the Java code that is created when generating code with Rational Software Architect.

Listing 2. Pump.java

```java
package acme.foundation;

import acme.utility.Rate;

/**
 * <!-- begin-UML-doc -->
 * <!-- end-UML-doc -->
 * @author Oscar Chappel
 */
public class Pump extends Equipment {
    public Pump() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @return the pumpingRate
     * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
     */
    private Rate pumpingRate;

    /**
     * @return the pumpingRate
     * @generated "UML to Java (com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
     */
    public Rate getPumpingRate() {
        // begin-user-code
        return pumpingRate;
        // end-user-code
    }
}
```
Listing 3. Refinery.java

```java
package acme.facility;

import java.util.Set;
import acme.distributionPlan.DistributionPlan;
import acme.utility.Location;

public class Refinery extends Facility {
    public Refinery() {
        super();
        // TODO Auto-generated constructor stub
    }

    private Set<TankFarm> tankFarms;

    public Set<TankFarm> getTankFarms() {
        return tankFarms;
    }

    public void setTankFarms(Set<TankFarm> tankFarms) {
        this.tankFarms = tankFarms;
    }
}
```

Listing 3. Refinery.java

```java
package acme.facility;

import java.util.Set;
import acme.distributionPlan.DistributionPlan;
import acme.utility.Location;

public class Refinery extends Facility {
    public Refinery() {
        super();
        // TODO Auto-generated constructor stub
    }

    private Set<TankFarm> tankFarms;

    public Set<TankFarm> getTankFarms() {
        return tankFarms;
    }

    public void setTankFarms(Set<TankFarm> tankFarms) {
        this.tankFarms = tankFarms;
    }
}
```
Creating the business object model (BOM)

The BOM forms the vocabulary and grammar that are used to write the business rules. The BOM is created using the Rule Designer component of IBM Operational Decision Manager. Rule Designer is an Eclipse-based integrated development environment (IDE). As with Rational Software Architect code generation, the mechanics of BOM creation are not as important as actually having the BOM created, since the rules cannot be written without the BOM. Figure 13 depicts the BOM that was created to support rule authoring in the ODM Business Rule Management System (BRMS).

**Figure 13. The Acme Refinery BOM**

![Image of the Acme Refinery BOM](image)

Figure 14 depicts the Pump class as it appears in the Rule Designer BOM editor.

```java
private Set<DistributionPlan> distributionPlans;

/**
 * @return the distributionPlans
 */
public Set<DistributionPlan> getDistributionPlans() {
    // begin-user-code
    return distributionPlans;
    // end-user-code
}

/**
 * @param distributionPlans the distributionPlans to set
 */
public void setDistributionPlans(Set<DistributionPlan> distributionPlans) {
    // begin-user-code
    this.distributionPlans = distributionPlans;
    // end-user-code
}
```
The BOM editor indicates that the Pump class is verbalized as \texttt{pump\\rangle} and that it has one member (attribute) called the \texttt{pumpingRate}. Figure 15 indicates that the \texttt{pumpingRate} member is verbalized as "the pumping rate of a pump". This is the default verbalization.

Figure 15. Pumping rate verbalization in the BOM editor

Figure 16 shows the \texttt{pumpingRate} verbalization after it has been modified to "a pump's pumping rate".

Figure 14. The Pump class in the BOM editor
**Figure 16. Modified pumping rate verbalization in the BOM editor**

The Refinery class in the BOM editor

Figure 17 shows the Refinery class in the Rule Designer BOM editor. Note that the class is verbalized as refinery. The refinery has two members labeled distributionPlans and tankFarms.

**Figure 17. The Refinery class in the BOM editor**

Figure 18 depicts the distributionPlans> member as it appears in the Rule Designer BOM editor verbalized as "the distribution plans of a refinery".
Figure 18. DistributionPlans member in the BOM editor

![DistributionPlans member in the BOM editor](image)

Figure 19 depicts the tankFarms member in the Rule Designer BOM editor verbalized as "the tank farms of a refinery".

Figure 19. Tank Farms member in BOM editor

![Tank Farms member in BOM editor](image)

Once the BOM is completed, you can start writing rules using the verbiage provided in the BOM.

Conclusion

In this article, you saw how to create the initial conceptual and logical domain model in conjunction with the business subject matter experts. The domain model consists of a number of UML class diagrams that provide a visual representation of the domain and can serve as the basis for creating individual object models that represent especially complex rules. The initial executable object model (XOM) is implemented and ready to support execution of the rules that will be written using the business object model (BOM) that was created from the XOM. The entire
model development process is iterative. That is, the initial domain model, XOM, and BOM were created as a result of term–fact analysis workshops and early rule discovery workshops. As the project moves forward, there will be additional rule discovery workshops and term–fact analysis sessions that will result in additions to and modifications of the domain model. Those additions and modifications will be reflected in the XOM and BOM, ultimately improving the final business rule solution.
Related topics

• **IBM Software Services for WebSphere**: Find out how IBM expertise in cutting-edge and proven technologies can help you achieve your business and IT goals.

• **IBM BPM Journal**: Get the latest articles and columns on BPM solutions in this quarterly journal.

© Copyright IBM Corporation 2013

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)