Paper#: 3025

Using Objects to Maximize Internet Application Potential

Rabah Ahmed, Ph.D.
Ptech Inc.
160 Federal Street
Boston, MA 02110
Tel: (617) 577-7100

ABSTRACT

Internet and the World Wide Web are gaining popularity in both the technical and commercial arenas. One driver of this trend is the ability to use Java to write web applications and to turn what used to be static pages into dynamic events.

The growing interest in Java, Internet, and web-enabled applications prodded software developers to roll out tool sets and applications that support the World Wide Web. While these Internet and Intranet development products do an excellent job at the software technology or systems programming level, they do not directly help develop full business solutions. The full potential of the Internet cannot be realized until it is linked into conventional business systems.

Furthermore, applications providing core business functionalities in many organizations are becoming more and more complex. This introduces a need for development tools that scale easily and help control complexity as well as supporting distributed-systems development.

Object-oriented analysis and design (OOAD) offerings are addressing these issues by providing an integrated development environment that synergizes a variety of technologies and strategies. An integrated approach helps organizations already invested in business applications react quickly to market changes and continually adapt their applications to maintain a competitive edge over time.

This paper describes the use of objects to develop Internet applications. It presents techniques for analyzing and prototyping applications using an OOAD tool and carries out a sample design of an Internet application with interdependent processes throughout an enterprise. The paper also discusses requirements for representing information, including data description, constraints, and operations as well as relationships among modules in the development process.

Modeling such an application with object technology provides the power developers need to develop and customize their environment for new technologies. With such capabilities, the model can also generate full source code in a variety of programming languages or HTML reports from a single set of repository-based specifications.

Introduction

The Internet is one of the more popular and fastest growing areas in computing today. The integration of Java with the Internet and, more specifically, the World Wide Web (WWW) has resulted in the explosive growth of Internet applications. Initially, Java was developed as a general programming language tailored to meeting the challenges involved in building software for distributed systems. The systems needed to run on multiple host architectures and to run over heterogeneous networks.

Quickly though, the importance and potential strengths of Java were recognized by the computing world. These strengths consist mainly of its object orientation combined with its network capability, portability, and security. These features combined to make Java a very compelling tool for the Web community.

Java is driving a revolution in Web page design by extending limited and static hypertext markup language (HTML) pages. Java brings dynamic and interactive Web pages to life with video, sound, and animation. With Java as the extension language, Web browser capabilities have been extended. Media-rich contents are now accessible in simple ways, without the need of many browser plug-ins.

Java support for object-oriented concepts, either inherently or by extension, made it more attractive to programmers. Java is very similar to the C and C++ programming languages. It eliminates many of the complexities that make other object-oriented languages, like C++, more difficult to learn. However, Java contains enough complexity to still make it challenging when used to solve complex business problems. This translates to an associated learning curve that can be as steep as the curves demanded by the mastering of the skills required by any other new programming language.

The popularization of Java in the application developer and business communities led to the development of Internet-related application tools that leverage the power of Java . The bulk of this application development is focused on delivering graphical user interface-based (GUI-based) applications targeted at Web site authoring and Internet access from within an application.

Shortly after the initial phase of using this new programming language, after the initial fascination with animated text and juggling graphics on Web pages wore off, people started thinking of how to leverage the new features and capabilities to achieve the full potential of the Internet. This potential can only be realized when it is linked into conventional business systems. That is, by the integration of the software assets that build the organization's business-critical systems into the new domain of the Internet.

These object-oriented Java tools come at a time when both the technical and commercial markets have selected object-oriented analysis and design (OOAD) techniques over other structured techniques. Developers and their managers now employ object-oriented techniques for commercial information-systems development because of their advanced features for modeling business processes. These techniques come at a time when the applications that provide core business functionalities are becoming more and more complex. These techniques come at a time when there is an increasing pressure to deliver more quickly in order to stay ahead of the competition. This introduces a need for development tools that scale easily, help control complexity, and support distributed-systems development.

While existing Internet and Intranet development products do an excellent job at the software technology or systems programming level, they do not directly help develop business solutions to these problems. OOAD offerings like FrameWork from Ptech address these issues by providing an integrated development environment that synergizes a variety of technologies and strategies. This approach enables organizations embarked in business applications to quickly react to market changes and continually adapt their applications to maintain a competitive edge over time. FrameWork offers a complete business solution in which applications involving distributed systems and client/server applications provide interoperability among diverse platforms.

Desired features in OO analysis and design tools

As objects gain wide acceptance and move beyond technical developers into business communities, object development tools start taking on new importance. As organizations become more process aware, more business application developers are finding that OO analysis and design techniques are

better at modeling business processes. Developers start to use more OOAD tools to capture and model technology components beyond the front end of the application.

A winning strategic approach to more complex business applications needing to be deployed not just within the department but across the enterprise will use a mix of technologies according to their strength. OOAD tools have to respond to this need by synergizing a variety of technologies and strategies in a seamless way, shifting the burden of integration with these fast changing and emerging technologies to the tool vendors. This frees the developers to concentrate on building quality business systems.

Figure 1 illustrates some of the critical capabilities needed in an OO modeling tool for it to be both comprehensive and useful.

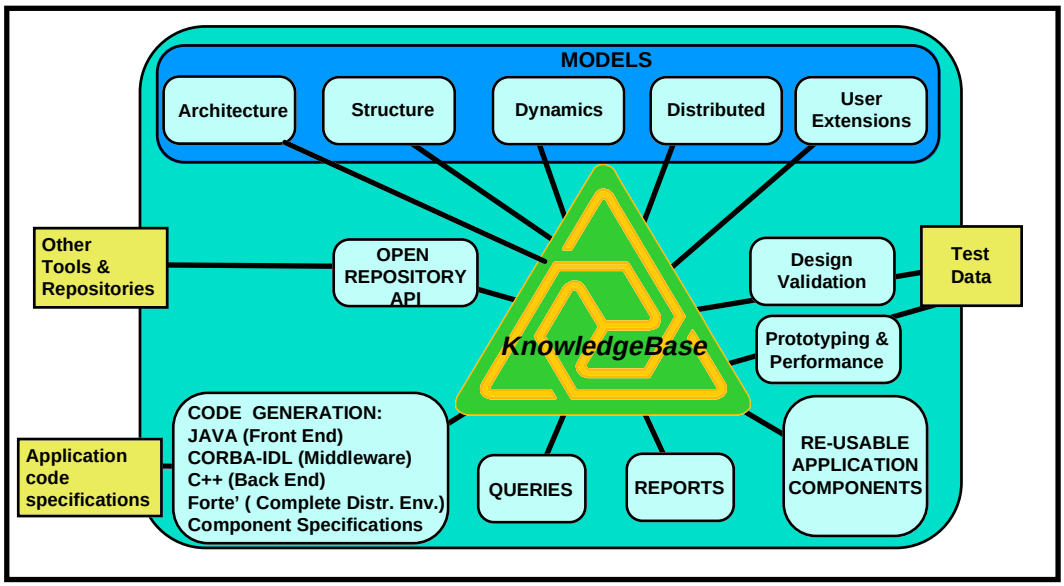

Figure 1. OOAD Tool Features

The tool should be based on a formal foundation. It should be represented by an active knowledge base that captures both the syntax and the semantics that describe the essence and reality of the business system. This knowledge base should allow browsing and verification at all levels of the business process design for consistency, correctness, and completeness. An open interface to this knowledge base is needed to ensure interoperability. Other applications must be easily interfaced to the tool to minimize any functional weaknesses.

Code generation capabilities that allow the user to evolve and reuse his specification and implementation are critically important to the implementation. The code generator should be able to bridge several technologies. These technologies range from front-end environments to middleware, such as CORBA, and to back-end environments. The support for distributed modeling is also equally important at a time where there is a rising demand and need for distributed processing environments to support both design and deployment.

Finally, the development tool should be extensible and customizable in all aspects, from the presentation to code generation. The key to extensibility is a customizable metamodeling approach. This allows the tool to be extended to support any user-defined methodology without a dependency on the tool vendor.

Approach

The big gain of using objects is the building of systems that manage complexity while being easier to modify. With Internet and Java, objects assume the role of business enablers. It is becoming almost impossible to engage in an Internet or Web application without using Java.

This paper describes the use of objects to develop an Internet application. It presents techniques for analyzing and prototyping the application using an OOAD tool. The paper describes a sample design and implementation using Java. The purpose of this paper is to illustrate through a simple example how to leverage the power of objects and Java in a common medium for business and technology professionals.

Everybody is convinced that it takes the cooperation of both domain experts and technology professionals to build a successful business application. This development environment will allow them to work together interactively to ensure that business requirements are completely and accurately reflected in the implementation. This way, processes can be defined with the appropriate concepts, captured, analyzed, designed, redesigned, implemented, and continually evolved to support ever-changing business goals.

Problem statement

Figure 2 illustrates a simple example of order processing to be synthesized. The figure shows the object model of the order processing example using Ptech/Odell notation. The object model uses various concepts to express the structural and the compositional view of the process under study as a collection of objects made useful and meaningful by virtue of their classes.

The model also captures the relationship between the objects represented by a concept called *relation*, which is composed of a pair of *functions* called *associations.* A *function* is a directional mapping from one class to another. When an evaluation is done on an object of one class (known as the domain of the function), it produces a set of zero, one, or more objects of another class (known as the range of that function). Conceptually, these relations describe the class properties with respect to other classes. The internal class properties are represented by attributes, whereas services (or responsibilities) that class instances can perform are defined using *operations*. Attributes are also referred to as functions.

The class CUSTOMER is related to class ADDRESS through the association “billing” and its inverse association “billed customer”. Similarly, the class ORDER is related to class ORDER LINE ITEM through the association pair “items” and “customer order”.

For each association (or for a function in general), constraints may be imposed on the number of objects that the functional mapping can result in. These are also known as cardinality constraints. The crow’s foot notation on the lines representing the associations show these lower and upper function bounds.

From Figure 2, both lower and upper bound on “billing” is specified to be one. This means that an instance of ADDRESS may identify one customer only, whereas a CUSTOMER instance can have more than one address. This may also translate to the fact that a customer receives billings at one address only. We also notice that an instance ORDER contains from one to many instances of ORDER LINE ITEM. This means that an order cannot be empty. These and other type of constraints will greatly help implement the business rules of the process under design.

The class CUSTOMER has many attributes, such as *customer name, customer code,* and *credit limit.* The object model illustrates the OO concept of *abstraction*. Classes can be generalized to “superclasses” or can be specialized to get “subclasses”. This concept of subdivision is referred to as a *partition.* Class CUSTOMER has a partition that includes two mutually exclusive subclasses:

CORPORATE CUSTOMER and REGIONAL CUSTOMER. This type of partition is known as *complete partition* because all the subclasses are identified. Otherwise, it would be referred to as an *incomplete partition*. All properties of the superclass CUSTOMER apply to the subclasses in the partition. This concept of partitioning supports the concepts of single and multiple inheritance found in OO design and programming.

The model shows an operation "Process Order" defined on the class ORDER. This operation needs to be refined further to include more details, such as how the order is processed and what objects are involved in the operation. This model captures only the interface and the base class upon which the operation is defined. In FrameWork, operations are triggered in a process model that portrays HOW an operation is performed by expressing methods and presenting sequences, flow of control, and serial/parallel operations. Operations can be also declared as external to the system. This mechanism is used to encapsulate functionalities and provide the flexibility for reuse whenever applicable.

Before proceeding further, it should be mentioned that one should go through a formal procedure of analysis and design. These techniques have been skipped here because they would be a subject for a separate paper.

Here is a summary of an analysis and design approach that we have found, based on our experience, very useful when building business applications using objects:

- Understand the business problem first, by conceptualizing the strategy and then building a strategic business model
- Formalize the scope and context of the problem by building the structure
- Discover objects based on the process requirements
- Detail the process specification to the implementation level
- Validate the requirements
- Deliver the complete application logic

What needs to be emphasized is that implementation strategies should be business problem-driven. The state and the infrastructure of the system should support the business decisions that implement the solutions, rather than driving them. This way, more emphasis can be placed on the conceptualized strategies without being limited by the available means and technologies.

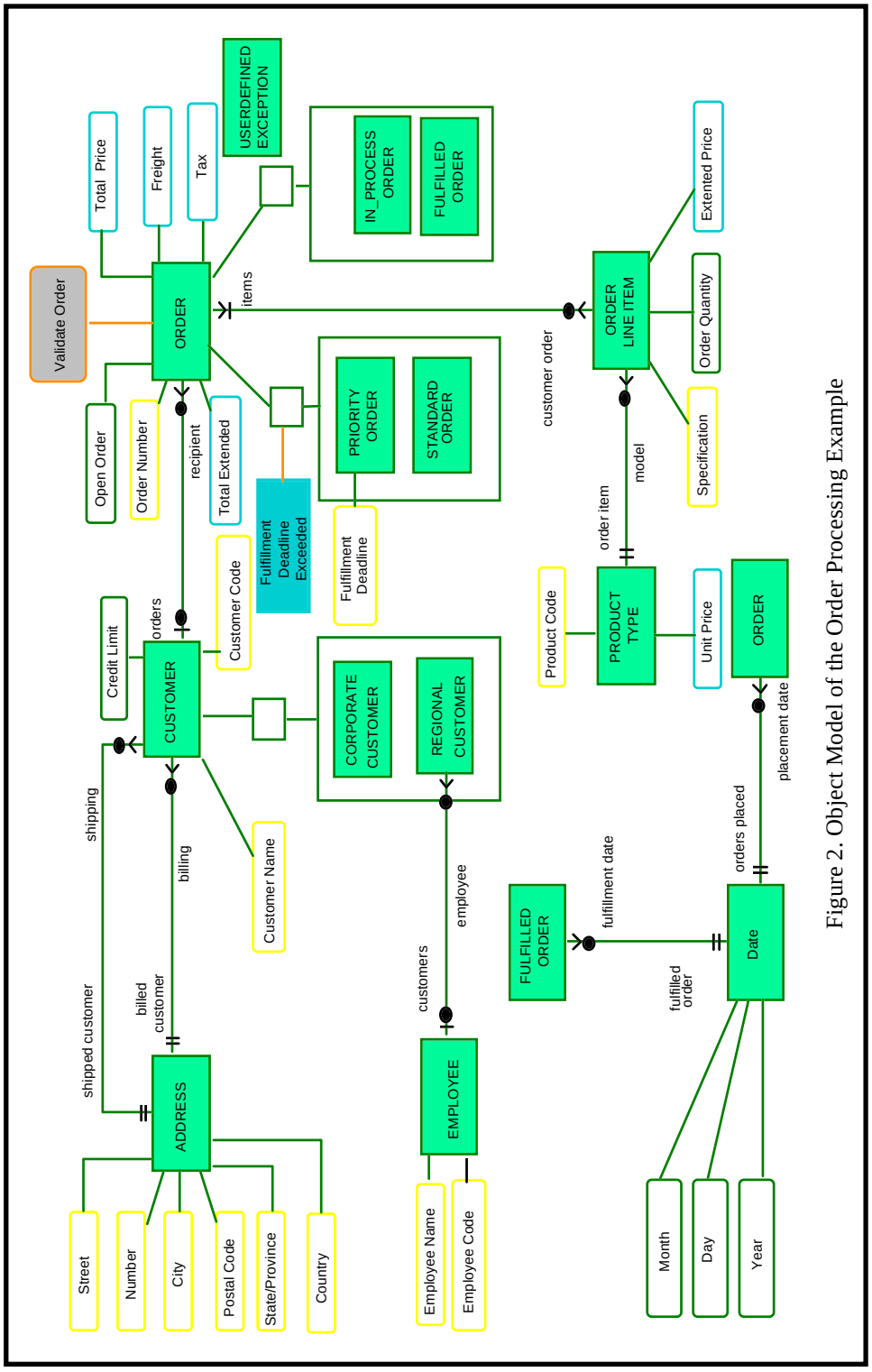

Figure 2. Object Model of the Order Processing Example

Code generation

As mentioned earlier, the design should commence with work on conceptual design to develop a rigorous and executable specification, where all the objects are captured. The next step is to map that specification into a Java implementation.

Java allows an implementation which can map a physical implementation more closely to a conceptual design without loss of design concepts. The mapping is formal in the sense that it associates each type of the modeling expression with a Java construct. The generated code captures the full semantics of the object model. Since these models fully describe the structure and behavior of the application, they provide enough information to fully implement the application with one or with a mixture of languages.

All the explanations from here forward will be closely coupled to FrameWork from Ptech Inc. Other OOAD tools may handle Java implementation in a different way to achieve a similar result.

The object models are mapped directly to Java classes representing the application classes. The mapping is natural, since FrameWork represents objects with data members and methods as does Java. Whenever a direct mapping is impossible, the code generator is responsible for adjusting the mapping so that the Java implementation is consistent with the application structure and behavior in FrameWork. The list of the OO modeling expressions and their corresponding Java implementation is shown below:

OO Modeling Expression	**Java Implementation**
Class	Class Definition
Interface	Interface Definition
Relation	Data Members for Domain/Range Class
Function	Data Members for Domain Class
Literal	Predefined Java Data Type
Static Partition	Inheritance
Functional Expression	Member Functions
Object Model	Classes Library
Class Attributes	Data Members
User-Defined Operation	Member Function (Java Method)

For each class named <class>, a file is generated <class>.java that contains the class definition and method implementations.

Listing 1 shows the code generated for the class CUSTOMER from the order object model in Figure 2. The file containing the code is named “CUSTOMER.java” as dictated by Java rules.

After some initial comments, the code starts by importing the classes that are accessed by this class using the *import* keyword. Importing a class adds that class to the current file’s name space. Java requires that the import keyword must precede all noncomment lines of code in the source file.

Then, after declaring the class name, the code starts by generating the constructors that will be used to initialize the class instances to a valid starting state. Two constructors are provided. The default constructor takes no argument and does nothing other than initialize all non-static members to their 0 states. The second constructor includes arguments that tell the constructor how to set up the objects at initialization.

Next, the accessor methods are generated. These accessors are used to make the data members accessible to members of other classes. For each single-valued function, two accessors *get()* and *set()*

are provided to return the current value of the member, or to set it to a specified value. A single-valued function maps a given object to zero or one object. That is, the cardinality of a single-valued function is equal to or less than one. For example, the generated accessor methods for the single-valued attribute "Customer_Name" are getCustomerName(), which just returns the string Customer_Name, and setCustomerName(), which assigns a new value to Customer_Name equal to the string value passed to it.

A multivalued function, on the other hand, maps a given object to zero or more objects. The number of objects, known as cardinality, resulting from evaluating a multivalued function is greater than one. Lower and upper bounds can be specified for the cardinality of a function. The number and type of accessors that are generated depend on this cardinality.

For multivalued functions of cardinalities of zero-to-many, or many-to-many, more operators are needed in addition to the previously mentioned accessors. These operators are *has()* that determines if a particular instance is in the range of the function which is implemented as a set container of objects, *num()* that returns the number of objects in the set, *add()* to add an object to the set, *remove()* to delete an object from the set, and *replace()* to replace an existing object from the set with a new one. The *replace()* is equivalent to *remove()* followed by an *add()* with the addition of properly handling the upper and lower limits. The function "orders" from Listing 1 is an example of a multivalued function. All these operators are given arguments for the range class of the function.

Next, the code generates the data members. Each data member for the function is given the model function name. By default, all functions are in the private section of the class. Strong typing is enforced; therefore a range specifier is required. In Listing 1, the range specifier for the function "billing" is "Address". For single-valued functions, the range specifier is either the class corresponding to the range class of the function or a literal type for the attributes. "Customer_Name" uses String as the range specifier.

The implementation of multivalued functions is similar to that of single-valued functions with some additional requirements. Collection of objects are used to store the values for a multivalued function. The range specifier identifies the collection class used. Sets, arrays, and lists are the available classes that implement the various collection types. The specification of the collection type is a design decision based on the type of behavior that is required. The default collection is HashSetContainer that is implemented using Java HashTables.

The data member "orders" in Listing 1 is an example of a multivalued association. Since Java does not support templates, the specifier for the multivalued functions does not indicate the type of the objects in the container. The user-defined operations are responsible for enforcing the appropriate object type. This can be easily implemented using the Java operator *instanceof* as will be shown below. Similarly, the user-defined operation has to implement the iterator to scan the list of objects in an aggregate, and to catch exceptions thrown by these accessors/operators in case of error conditions.

As discussed above, the relation between objects is bidirectional. The relation consists of an association and its inverse, also referred to as a two-way function. The generated code for a two-way function includes additional code to handle the creation and removal of objects from the range of the inverse association, when applicable. This is shown in many places in Listing 1. It is shown in replaceOrders(), removeOrders(), etc.

More on code generation

Listing 1 displays the generated code for the class "CUSTOMER", using default modifiers whenever possible. However, there are many modifications and settings that can be used during the design to tune the generated code. FrameWork provides capabilities for the designer to customize the generated code to integrate it with existing Java classes and methods.

Among these desired modification is the specification of whether the class is abstract so that it cannot be instantiated and modification of the class visibility to *public* to extend its visibility from its package to anywhere. You can also choose to make it final, so that it cannot be extended (i.e., that it cannot be subclassed). All of these settings can alter the generated code.

Similar modifiers and settings are available during the design to set the visibility, scope, and whether it's final or not for functions and operations. External operations in FrameWork are mapped to "native" methods with signatures only.

The modeling concepts of *partition* and generalization are implemented as inheritance using the *extends* keyword. The following snippet from the code implementing the class PRIORITY ORDER, shows how the class PRIORITY ORDER is declared as a subclass of the class ORDER.

```
class PrioprityOrder extends Order   {
        public  PriorityOrder() { Fulfillment_Deadline = "" ;};
  .
  .
}
```

Besides the proper handling of design constraints, such as cardinality and invariance as explained above, the code generator supports other Java concepts, such as *interfaces*, *exceptions*, and *packages*.

Figure 3 illustrates how FrameWork metamodel was extended to support these new concepts. From a modeling point of view, the concept of interface is the same as a class. They both inherit the same properties of the FrameWork meta-concept "#CLASS". However, the implementation differences between classes and interfaces are enforced by the code generator and validated before the code is generated. Interfaces can be used to achieve almost the same effect as using multiple inheritance which is not supported by Java. Classes are combined in loose affiliations known as packages. Packages can be either predefined or user-defined. The predefined packages are provided by Java, combined to form the Java class library. Similarly, exceptions are raised by operations in case of error conditions to interrupt the flow of the program. Exceptions are implemented as classes whose methods get invoked when the program executes the *throw* keyword.

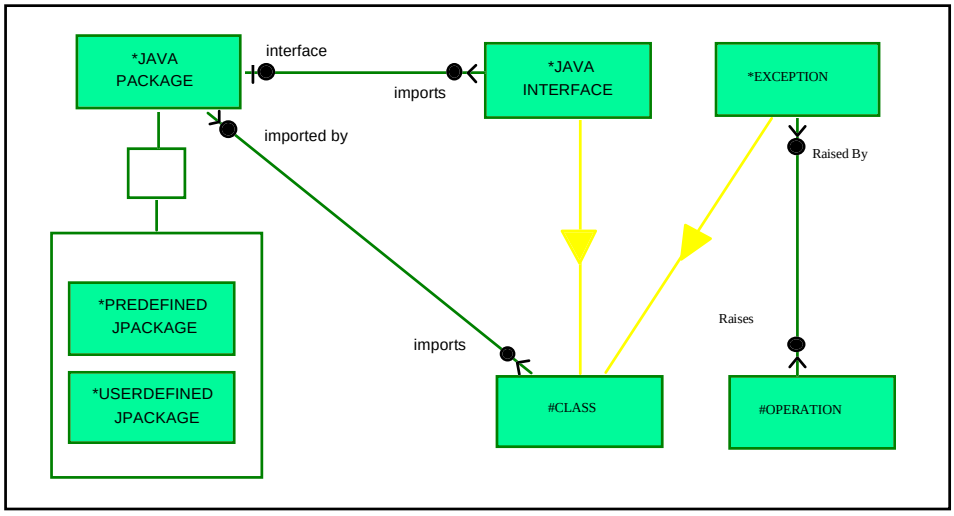

Figure 3. Java Concept Extensions

Pulling everything together

The generated code implements the general architecture of the application. It can be used as the core of the application and the rest can be developed using other techniques. For example, the code does not implement the body of the user-defined methods. However, depending on the target application, many other components have to be added.

The code generated from the object model of Figure 2 can be used to build an applet that runs from a browser or as a component in a larger application that runs as a stand-alone application. FrameWork is more helpful in the latter case where other parts of the system can be captured, analyzed, and designed in the same knowledge base. And, at the time of implementation, the designer can partition his design to specify which components should be implemented in Java and which ones are to be implemented in other languages. A typical example would be the design of a three-tier client/server application model based on Java clients and CORBA ORBs.

For the purpose of illustration, the code generated from Figure 2 has been extended to implement an applet that can be used to browse product prices and place orders over the Internet. The full code can be obtained from the author by dropping him an e-mail or written message.

Benefits of code generation

By generating code from FrameWork, the developer can eliminate errors due to incorrect implementation. This allows the developer to focus on the design itself and on improving the application. Additionally, this focus on the design eliminates logic errors in the application due to misinterpretation of the specification. The net result of these efficiencies is that time spent in the design-testing-redesign process is reduced.

The source code produced by the code generator is directly related to the design, which makes it easy to understand. This reduces the maintenance time required to make any changes or improvements to the code that may be required for optimization. Also, this means that as the design evolves, the iterative generations of code will be less costly to maintain.

The generated code supports the full expressive power of FrameWork, so that implementation concerns are truly separated from the design process. This is not true of many design and code generation tools where significant maintenance is required to synchronize design and implementation modules. As a result, design is focused on and can be driven by domain experts, reducing requirements specification efforts and cost.

Conclusion

In addition to its role on the Web, Java is also generating great degree of interest as an object-oriented language for business applications. This paper has shown how Java defines an implementation method which closely follows object-oriented design methods and techniques. This has been demonstrated by carrying a simple business example from concepts to code. Detailed explanation has been included on how Java maps the concepts defined in an object-oriented design to an implementation which adheres to the same concepts the software was designed with. This way, Java can be used to leverage the popularity of the Internet and deliver object technology in the enterprise.

What has glued everything together and eased the development strain is the use of an integrated environment where application requirements, management rules, coding guidelines, specification and code documentation, and application design and code generation are all tied together in one knowledge base. This offers the great prospect of flexibility and improved software reuse at the design, architectural, and implementation levels. It is what is needed in a time when technology as well as

business professionals are working together interactively, faced with challenges that are seriously affected by urgency, scaling, and complexity.

Finally, the rapid influx of new technologies and languages, like Java, makes it difficult to keep up with the nits and bits of these technologies, especially since they are getting more complex. One of the most important advantages of working with such an environment is the hiding of the implementation details and the shifting of the focus to the design concepts. This allows the user to concentrate on the business objects while the tool environment worries about the implementation objects.

References:

Gary Cornell, Cay S. Horstman, *Core JAVA*, Java series, Sunsoft Press.
Lugi Guadagno and Okokon Okon III, *Stepping Up the Flow of Business Information*, OBJECT magazine, July 1996.
Rabah Ahmed, *Object-Oriented Information, Analysis, Design , and Automation*, InterWorks proceedings, 1996.

// Generating Java Source class Customer
// import

```
import java.lang.*;
import java.util.*;

 class Customer   {
  public  Customer() {
   Customer_Name = "" ;
   Credit_Limit = 0 ;
   Customer_Code = "" ;    billing = new Address();    shipping = new Address();
                           orders = new HashSetContainer();
   };
  public  Customer(
       long Credit_Limit_, String Customer_Name_, String Customer_Code_, Address billing_,
        HashSetContainer orders_, Address shipping_ ) {
        billing =  billing_;
        shipping =  shipping_;
        Customer_Name =  Customer_Name_;
        Credit_Limit =  Credit_Limit_;
        Customer_Code =  Customer_Code_;
        orders =  orders_;
  }

  // Accessor methods
  // 1..1 Customer Name
  public String   getCustomerName (){return Customer_Name;}
  public void   setCustomerName(String t) {Customer_Name = t;}
  // 1..1 Credit Limit
  public long   getCreditLimit (){return Credit_Limit;}
  public void   setCreditLimit(long t) {Credit_Limit = t;}
  // 1..1 Customer Code
  public String   getCustomerCode (){return Customer_Code;}
  public void   setCustomerCode(String t) {Customer_Code = t;}
  // 1..1 billing
  public Address   getBilling() {return billing;}
  public void   setBilling(Address t) throws PtechException{  setBilling(t, true); }
  public void   setBilling(Address t, boolean setinverse) throws PtechException{
            //  If not called by t's inverse, set inverse value.
            if (setinverse == true)
                t.addBilledCustomer(this, false);
            billing = t;
  }

  // 1..1 shipping
  public Address   getShipping() {return shipping;}
  public void   setShipping(Address t) throws PtechException{ setShipping(t, true); }
  public void   setShipping(Address t, boolean setinverse) throws PtechException{
            //  If not called by t's inverse, set inverse value.
            if (setinverse == true)
                t.addShippedCustomer(this, false);
            shipping = t;
  }
```

Listing 1: CUSTOMER Java class

```
 // 0..Many orders

  public Enumeration   getOrders () throws PtechException{return orders.get();}
```

```
public void  addOrders(Order t) throws PtechException{addOrders(t, true); }
public void  addOrders(Order t, boolean addinverse) throws PtechException{
          //  If not called by t's inverse, add inverse value.
          if (addinverse == true)
             t.setRecipient(this, false);
         orders.insert(t);
}
public void   replaceOrders(Order addt, Order remt) throws PtechException{
          replaceOrders(addt, remt, true);
}
public void   replaceOrders(Order addt, Order remt, boolean inverse) throws PtechException{
          //  If not called by t's inverse, add inverse value.
          if (inverse == true) {
               remt.removeRecipient(false);
          addt.setRecipient(this, false);
}
          orders.erase(remt);
          orders.insert(addt);
}
public boolean hasOrders(Order t) { return orders.contains(t); }
public void removeOrders(Order t) throws PtechException{ removeOrders(t, true); }
public void removeOrders(Order t, boolean removeinverse) throws PtechException{
           //  If not called by t's inverse, remove inverse value.
           if (removeinverse == true)
                t.removeRecipient(false);
           orders.erase(t);
}

public int numOrders() {return orders.size();}

// User Defined Methods

// Data Members
private   String      Customer_Name;
private   long      Credit_Limit;
private   String      Customer_Code;
private   Address      billing;
private   Address      shipping;
private   HashSetContainer  orders;

}
```

Listing 1 (continued): CUSTOMER Java class