The WebShop e-commerce framework

Marcus Fontoura[1], Wolfgang Pree[2], and Bernhard Rumpe[3]

[1] Cyberspace and Web Technology Department, IBM Almaden Research Center
650 Harry Rd., San Jose, CA, 91520 , U.S.A
fontoura@almaden.ibm.com
[2] Software Research Lab, University of Constance
D-78457 Constance, Germany
pree@acm.org
[3] Software and Systems Engineering, Munich University of Technology,
D-80290 Munich, Germany
rumpe@acm.org

Abstract. This paper describes the WebShop e-commerce framework using a variant of UML that is specifically targeting at frameworks and product line architectures. Stereotypes and Tags are used to describe standard as well as application specific collaborations/pattern. This allows a compact and efficient description of the WebShop architecture suited for developers applying the WebShop framework.

1 WebShop overview

WebShop basically allows the creation of on-line stores from a description of the products that should be offered and sold on a Web site (see Figure 1).

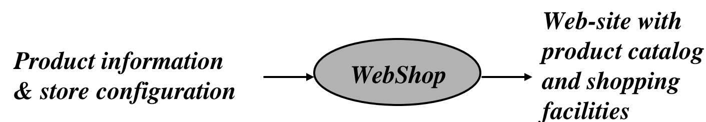

Figure 1 The goal of the WebShop framework.

As specific Web stores differ in various aspects, the WebShop framework defines the following variation points:

- Payment options: companies accept various payment options, such as credit cards, e-money, and so on. Moreover, completely new electronic payment methods may arise and the framework should be able to incorporate them.

- Promotions: promotions usually depend on parameters such as the overall shopping volume of a customer or the frequency a customer comes along. For example, a bookstore site might send a gift at the end of the year if the sales volume of a customer has surpassed a certain limit. Another example would be to freely upgrade to a faster delivery, if a customer buys goods for a greater value amount. WebShop should be easily extended in that regard.
- Reports: every organization requires different kinds of management information. Examples include rankings of the best customers, sales figures on various single products and product groups, and information regarding the preferred payment methods. Once again, WebShop should be open for any extension of the reporting subsystem.

Figure 2 sketches the configuration options of WebShop. Typical WebShop adaptations can choose among predefined payment options so that this aspect allows a black-box configuration. The promotion and report generation will quite likely be adapted to the specific requirements of each application.

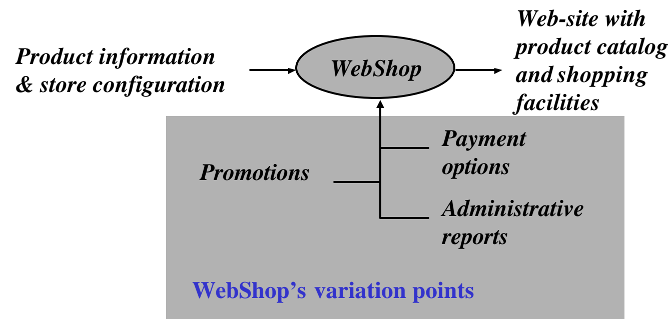

Figure 2 Variation points of WebShop.

Figure 3 represents the navigational structure of a typical WebShop application. Each single rectangle represents a web page and the arrows represent the actions that cause movement between the pages.

The “Product list” page is the application entry point. It displays the list of available products. Clients select and add them to their individual shopping cart. A client can review the shopping cart at any time for changing the products and quantities in the cart. When the client wants to checkout he or she only has to select a payment method and provide the required information. The system then verifies the information and either processes the transaction or reports an error.

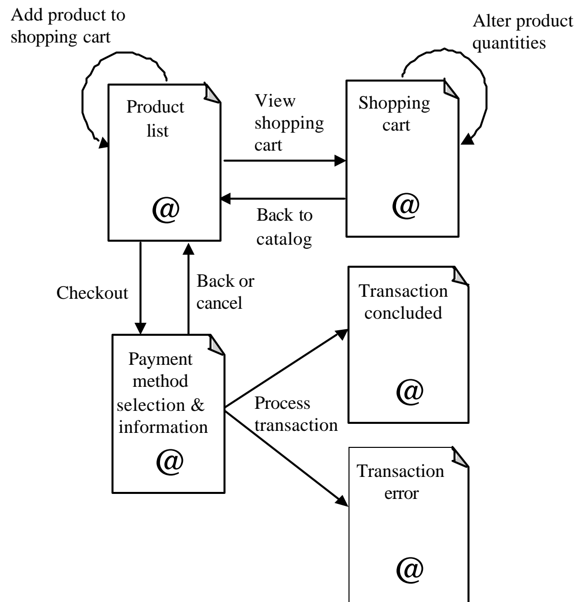

Figure 3 Navigational structure of a typical WebShop application.

WebShop allows the creation of a complementary site for displaying the administrative reports. Typically, the structure of the administrative part consists of a number of reports. The end user can select from an overview list any of the reports available (see Figure 4).

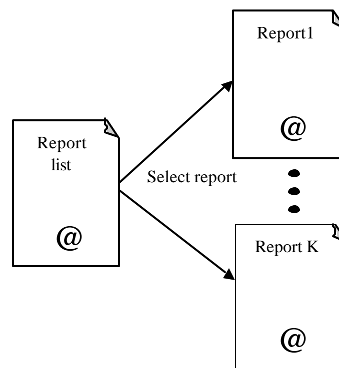

Figure 4 Report listing on a separate site.

For the rest of the paper it is assumed that the reader is familiar with basic concepts of UML and with the stereotype and tag extensions mechanisms.

2 WebShop components

The following sections present the core aspects of WebShop by means of UML-F diagrams [1]. This presentation forms the basis for identifying patterns that are useful in the context of e-commerce frameworks. Thus the UML-F tag extension mechanism comes in to define specific pattern tags.

2.1 Shopping cart

The core entity of the WebShop framework is the shopping cart. For each client access to a Web store a new shopping cart object is created. This shopping cart takes care of the connection and is responsible for controlling the user selection of products and the checkout operation. Figure 5 shows an UML diagram demonstrating what a shopping cart contains - the exact number of products and one transaction log.

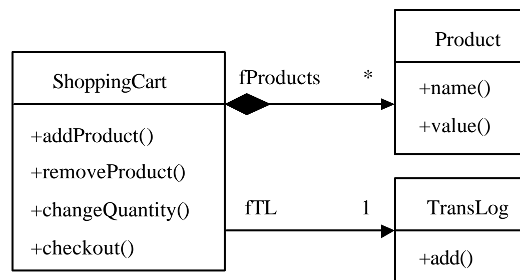

Figure 5 UML class diagram of ShoppingCart and two of its associated classes.

Methods addProduct(), removeProduct(), and changeQuantity() in class ShoppingCart modify the products already chosen accordingly. Method checkout() processes the payment transaction. It is also responsible for updating the system transaction log by invoking method add() in class TransLog. The transaction log may be used for various customer relationship management activities such as promotions. Thus, it forms the basis of various reports.

2.2 Payment options

Each application created by the framework will provide a number of payment options. In particular, the cart's checkout() method requires the information on available payment choices. To keep payment methods flexible, WebShop

applies the Separation construction principle [3] (see Figure 6). As all the Payment objects interacting with a shopping cart have to be able to process a payment, the interface Payment defines the processPayment() method. Therefore, all specific classes used for payment have to implement the Payment interface, as illustrated in Figure 6.

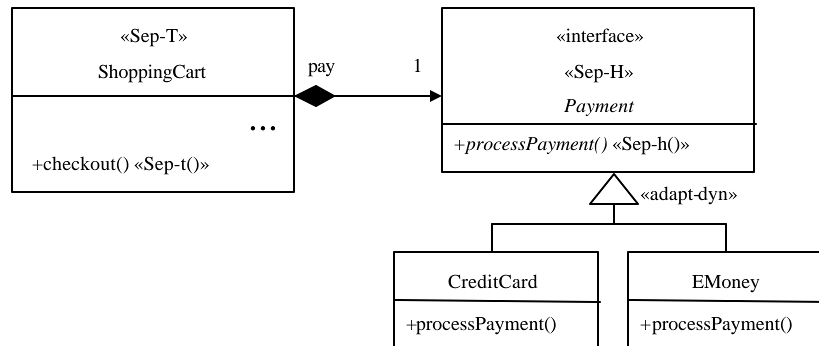

Figure 6 The Payment interface.

Figure 6 uses UML-F tags to identify explicitly the template and hook methods and classes [3]. The method checkout() is the template method, since it is responsible for invoking processPayment(). This is a hook method that varies for different classes that implement the Payment interface. The «adapt-dyn» tag indicates that the classes are dynamically loaded into the system when needed.

From the client's perspective, a Web form should present the payment options available. The client selection is then proceeded to the store that has to instantiate the appropriate payment object and plug it into the shopping cart. The sequence diagram in Figure 7 illustrates this scenario. It shows the creation of an object to process credit card transactions in Figure 7(a), and the e-money transactions in Figure 7(b).

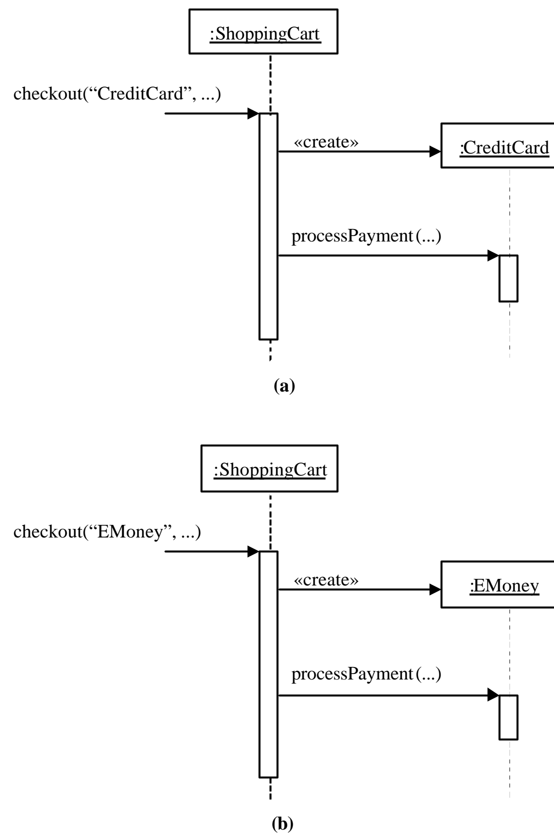

Figure 7 Creating the appropriate payment object (a) for credit cards and (b) for e-money.

Note that the parameter specifying the payment option is a string that represents the class name of the specific payment class. The checkout() method uses dynamic class loading to instantiate the appropriate class based on its name. Example 1 illustrates the code for the checkout().

```
public boolean checkout(String paymentClName, String payInfo) {

        boolean flag = false;
        Payment payment = null;

        try {  // Tries to instantiate a Payment object
               Class c = Class.forName(paymentClName);
               payment = (Payment) c.newInstance();
        }
```

```
        catch(Exception e) {
            // error!
        }
              // The method total()calculates the total value
              // of goods in the shopping cart. This method is a
              //private method in class ShoppingCart.

        if (payment.verifyPayment(payInfo, total())) {

                ... // Add transaction to log
                ... // some further processing

                flag = true;

        }
       return flag;
}
```

Example 1 Source code fragments of method checkout() in class ShoppingCart.

Parameters of the processPayment() method

The various implementations of the hook method require different arguments. As the client– supplies these arguments through a Web form, WebShop assumes that they are provided in a single string that is formatted according to simple conventions, that is, as "number = '5534453567144532'; expdate = '10/2002'; name = 'John V. Lee'". Each implementation of processPayment() parses and processes this input string. The attributes in the string are defined by the particular payment classes. The sequence diagram in Figure 8 illustrates this behavior for the CreditCard object.

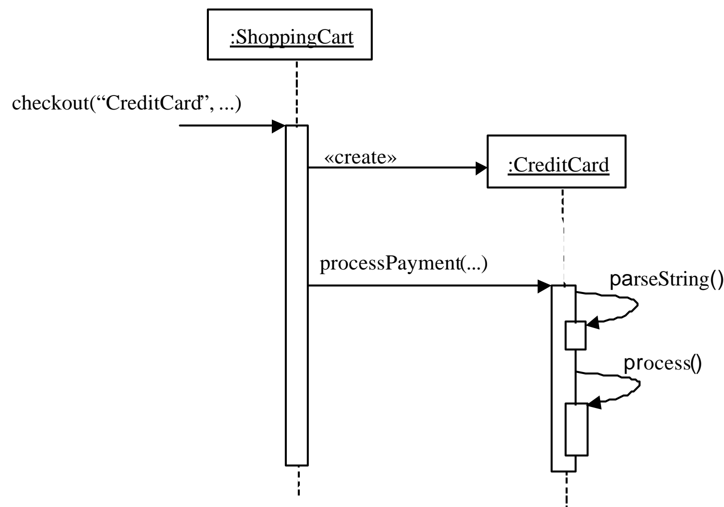

Figure 9.8 The general behavior of concrete implementations of processPayment().

2.3 The Payment pattern as UML-F tag set

With ShoppingCart as the template class and the Payment interface as the hook class, the UML-F extension mechanism supports the definition of a domain-specific tag set. This can be particularly useful if several frameworks in the e-commerce domain offer a similar design for keeping the payment choices flexible. Table 1 presents the tags of the Payment pattern. Figure 9 applies these tags in the context of the WebShop framework.

Table 1 The UML-F tag set for the Payment pattern.

Tags:	**Payment-ShoppingCart**
Introduced Names	«Payment–ShoppingCart», «Payment–Payment», «Payment–checkout()», «Payment–processPayment()» or alternatively the abbreviated versions «Paymt–Cart», «Paymt–P», «Paymt–ceckout()», «Paymt–process()»
Application examples	see Figure 9

Motivation and Purpose; informal explanation of effect, discussion	The pattern keeps the verification of the payment information flexible and applies the Separation pattern for this purpose. The «Paymt–ceckout()» method is responsible for dynamically loading the appropriate «Payment–Payment» class based on its input arguments. «Paymt–process()» must parse the string that contains the payment information before the actual verification.
Expansion	The expansion of the newly introduced tags leads to the diagram in Figure 6

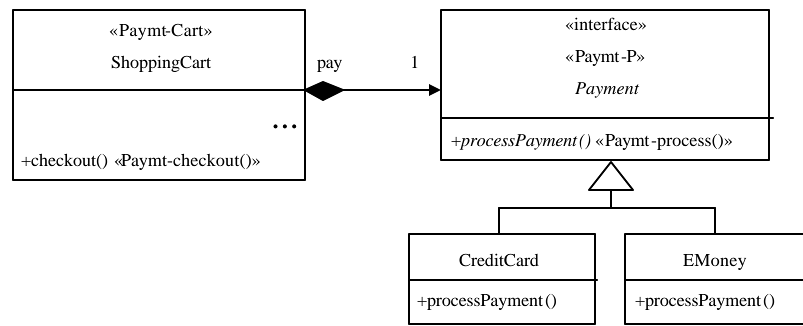

Figure 9 Annotating the payment variation point through the Payment pattern tags.

The «Payment-ShoppingCart» tag set is a typical example for a framework or application specific use to describe a pattern. The pattern relies on framework specific classes and, therefore, is less of use elsewhere. On the other hand, there are only a few basic construction principles that all true pattern rely on. The «Payment-ShoppingCart» uses the Separation principle. The use of well known higher-level construction principles allows to abstract from details of the code and to present the overall structure/architecture of a system in a compact form.

2.4 Defining promotions in WebShop

In order to deal with promotions, the checkout() method invokes a method definePromo(). This method defines a promotion that depends for example on the overall value of purchased goods.

As the WebShop framework should be able to support several promotions at the same time, WebShop applies the Chain-Of-Responsibility pattern [2]. For example, when a frequent shopper buys goods for more than $ 1000.00 he or she should receive not only a discount, but also a free delivery.

The Chain of Responsibility (COR) pattern allows each promotion object to check if the current transaction follows the conditions required by it. The object then forwards the request to the next promotion object, if any. The object diagram in Figure 10 exemplifies a combination of two such promotion objects to which a ShoppingCart object refers to.

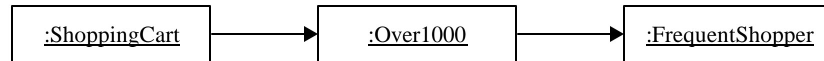

Figure 10 Composing promotion objects.

The :ShoppingCart object is responsible for invoking the definePromo() method as the first in the chain of promotion objects. These promotion objects are further responsible for forwarding the request in the chain. In the sample chain shown in Figure 10, the object of class Over1000, which gives discounts for transactions over $ 1000.00, treats the request and forwards it to the next promotion object, which issues free delivery for frequent shoppers. Figure 11 annotates the promotions variation point with the COR tags.

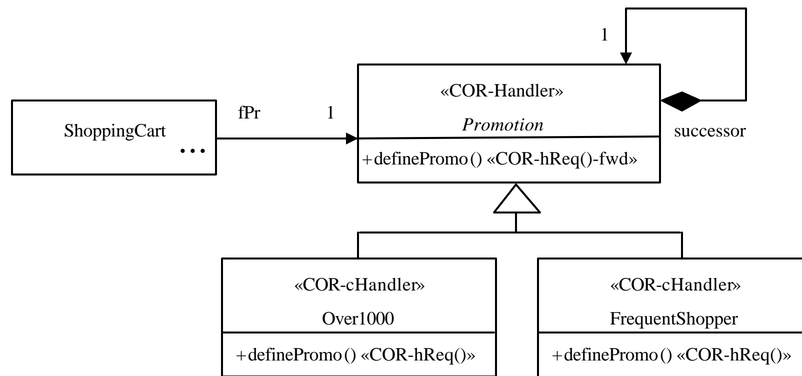

Figure 11 Annotation of the promotions variation point with the COR tags.

The implementation of definePromo() should have access to the payment and user information and to the transaction log in order to support promotions based on the user history, such as FrequentShopper. However, to process large log information on-line may have prohibitive costs. This may be avoided in the WebShop framework by a preprocessor method that is executed only once, i.e. when the promotion object is created to filter the log information. Figure 12 illustrates this situation.

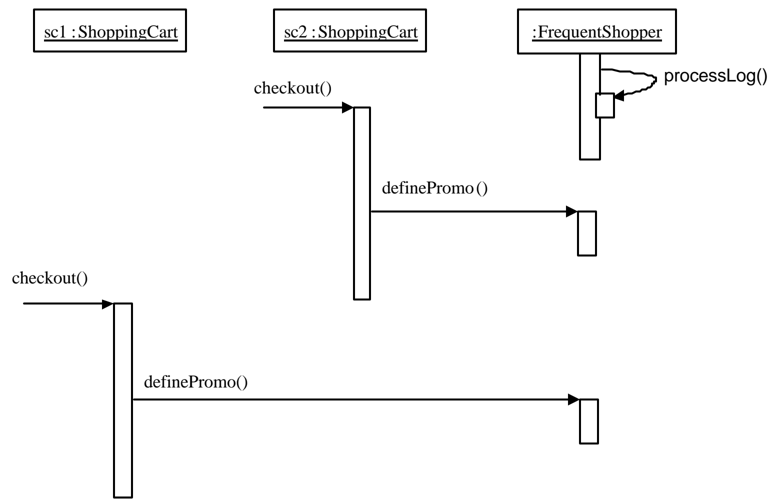

Figure 12 Typical life-cycle for promotion objects.

2.5 Reports

The report generation in WebShop relies on the Separation pattern (see Figure 13). Analogous to the Payment subclasses, a string is used to uniquely specify which class implementing Report should be loaded the system, as illustrated in Figure 14.

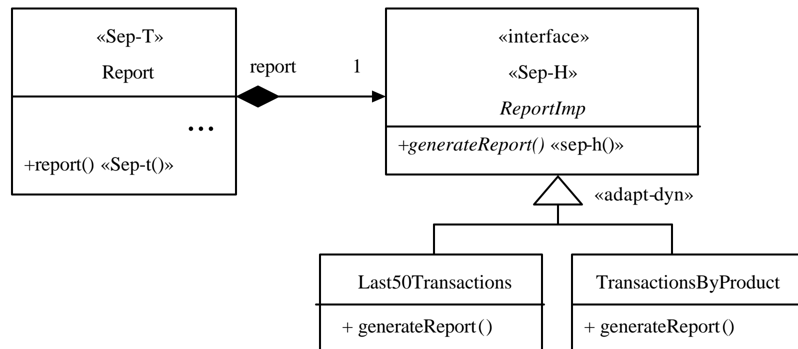

Figure 13 Annotating the report variation point through the Separation pattern tags.

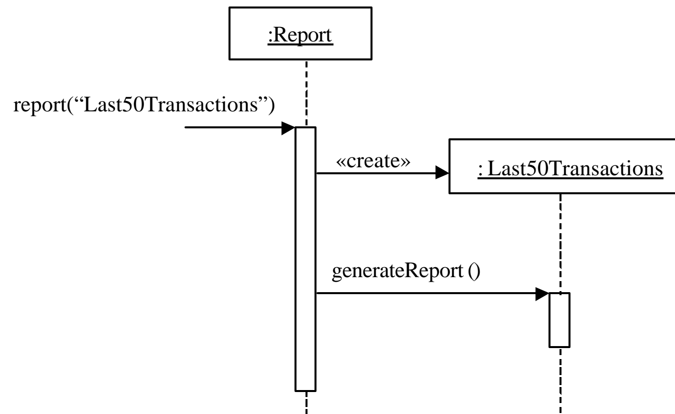

Figure 9.14 Creating the appropriate report object

The report() method is responsible for the dynamic loading of the appropriate ReportImp class and for the invocation of the generateReport() method, which returns a string containing the report written/generated in HTML. For example, the class Last50Transactions lists the last 50 transactions processed by the system. Class TransactionsByProduct, on the other hand, lists all the transactions grouped by product.

As the design is analogous to the Payment pattern and as it is quite useful to have a dynamically created object to treat end-user requests, we define another domain-specific pattern called Web Request. The payment and administrative reports variation points both use the Web Request structure. To add to it, both specialize the Web Request pattern in a simple way. The Payment pattern, for instance, is a specialization of Web Request that treats payment processing requests. Table 3 presents the tags for Web Request pattern. Figure 15 models the report variation point together with the Web Request tags.

Table 3 The tag set for the Web Request pattern.

Tags:	**WebRequest**
Names	«WebRequest–Loader», «WebRequest–Request», «WebRequest–load()», «WebRequest–request()», or alternatively the abbreviated versions «WebR–L», «WebR–R», «WebR–load()», «WebR-request()»
Applies to	See Figure 6, 13.

Motivation and Purpose; informal explanation of effect, discussion	The pattern keeps the execution of a request flexible and applies the Separation pattern for this purpose. The «WebR–load()» method is responsible for dynamic loading of the appropriate «WebR–R» class based on its input arguments. Generally, the «WebR-request()» method has to parse an input string parameter before performing the actual request.
Expansion	See Figure 6, 13.

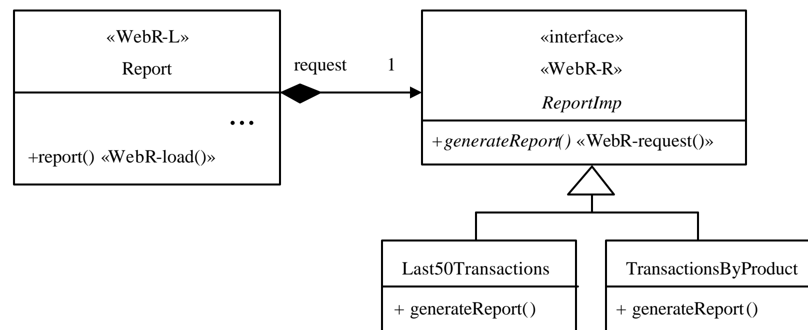

Figure 15 Using the Web Request tags to model the report generation.

3 Conclusions

The WebShop framework was developed by the authors for the purpose of demonstrating UML-F in the domain of Web applications. Thus, the framework is not regarded as a full-fledged system out of which real Web stores can be derived. For example, the framework in the presented version does not encounter security features. The UML-F Web site http://www.UML-F.net provides the Java source files and some sample adaptations of WebShop.

The UML extensions, namely the UML-F profile [1] used to describe the WebShop framework proved very effective to provide an application developer an intuitive and easy overview of the framework. The UML-F profile mainly provides a set of tags, like «adapt-static», «fixed», «hook», «template», «Separation» and so on, together with mechanisms to introduce new tags and to describe their meaning and intention in an informal yet systematic way.

4 References

[1] M. Fontoura, W. Pree, and B. Rumpe, *The UML Profile for Framework Architectures*, Addison-Wesley, (to appear 2001).

[2] E. Gamma, R. Johnson, R. Helm, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995.

[3] W. Pree, *Design Patterns for Object-Oriented Software Development,* Addison-Wesley, ACM Press, 1995.