**REPRESENTATION OF CLASS DIAGRAM AND SEQUENCE DIAGRAM IN PROCESS MODELING: UML-BASED STUDY**

*Erwin Widodo and **Atsushi Ohnishi  
*Faculty of Science and Engineering, Ritsumeikan University, Japan  
e-mail: *gr042044@se.ritsumei.ac.jp and **ohnishi@is.ritsumei.ac.jp

**ABSTRACT**

Even the usage of UML (Unified Modeling Language) has become popular in object-oriented (OO) analysis and design stage, software-intensive system’s stakeholder still encounters some problems in rendering their process modeling. One prominent problem is inter-model inconsistency in UML models’ representation. Many approaches have been done to overcome this problem yet each of them seems to be effective only in some cases or even just in one particular circumstance.

Contribution of this paper is to present two representations analysis of class diagram and sequence diagram which are regarded as critical models in UML. Metric-Based Representation is used to analyze class diagram. This investigation focuses on a way of observing the possibility by giving a basic argument of graphical representation of this diagram. UML Formal Semantics is engaged by means of static semantics in terms of consistency checks of sequence diagram its context with class diagram.

The expected outcome is justification of how should both class diagram and sequence diagrams be represented precisely and accurately. Afterwards, one underlying contribution will be attained for further considerable substance, assuring UML inter-model validation.

**Keywords:** class diagram, sequence diagram, metric-based representation, UML formal semantics

**Introduction**

Not take a long time after invented in 1997; UML has become the acknowledged standard for object-oriented (OO) analysis and design stage in software-intensive system development. Basically, UML is a visual modeling-language and it consists of several representations by means of diagrams. These representations are categorized into two groups. First group is referred as static diagrams and used to depict the static structure of a system. This group consists of class diagram, use case diagram, component diagram and deployment diagram. Second one, dynamic diagrams, specifies how to control the flows of the system program that should behave. Belonging to this group are sequence diagram, collaboration diagram, state diagram as well as activity diagram [17].

In the OO system-development, class diagram hold a critical role. It is a key early artifact that determines the quality of the software system since the very beginning of the OO software analysis and design. Class diagram is really the internal wiring diagram for a system. It defines the static and unchanging structure of the system [12]. This diagram clearly shows the objects that make up the system and the lines of communication between them. This is the principal diagram for analysis and design.
stage, and of course, there is a great deal of preparation needed to produce a class diagram. Lots of information can be included on this diagram, including the data that is to be stored in the system and the some processes that take place.

Along with state diagram, sequence diagram describe interaction within observed system. Sequence diagram illustrate object interaction specifically. This diagram takes a flow of activity and maps it in a two-dimensional way to the internal components of an observed system. An interaction is the communication pattern performed by the object to accomplish a specific purpose, such as performing complex operation. Sequence diagram emphasize on time ordering between inter-objects messages such as signal or method calls [12], [3]. Again, this diagram is very rich. Most OO analysis and design stages use this diagram extensively. It can be used not only to identify the interaction but also as a basis for determining what objects are needed in the system to provide its functionality.

UML potentiality may overcome not only concrete things such as programming language statements, database schemas, and reusable software but also including conceptual things such as business processes and system functions as well. This role is done by providing different diagram types supporting to the development process from requirement specification to implementation. However, the outcome models do not come up without problem. The real system is presented by various diagrams view an observed system from different perspectives or abstraction levels. Thus, the various UML models of the same system are not independent specification yet overlapping. They depend on each other in some ways. For a simple example, changes in one particular model or diagram may imply multiplying changes in other diagrams. Moreover, in a non-trivial system, a broad and complex representation gives stakeholders a typical problem in interpreting UML-models.

In addition, inconsistency is the main problem in process modeling using UML-based diagrams. A process of a UML-based software development usually faces two kinds of problems. The first concerns consistency among artifacts within a given model and can be identified as an intra-consistency problem. The second concerns consistency between different models or inter-consistency problem [8]. Intra- and inter-consistency demands seem to be necessary for correctness of software development process. Without preparing some precautions for this problem, usually, software analysis and design only becomes an inferior stage in software development process.

The main goal of this paper is to give a conceptual study in preparing consistency validation of UML models. The focuses are class diagram as representation of static structure of observed system and sequence diagram as the proxy of dynamic behavior or interaction inside the system. By understanding these representation methods, we will have adequate basic justification to give more validation guarantees on visualizing UML models, especially on class diagram as well as sequence diagram.

Mainly, this paper describes two approaches in representing both class diagram and sequence diagram as the critical artifacts in OO analysis and design stage. They are metric-based representation of class diagram and UML formal semantics for sequence diagram. In the former approach, OO metric is used to analyze class diagram. This investigation focuses on a way of observing the possibility by giving a basic argument of graphical representation of this diagram. Three related steps are: to set up localizing central class by metric calculation, to select prominent element depending on metric values to determine class context and to visualize the new created sub-view of class diagram. For the latter, formal semantics of UML is engaged by means of static
semantics. This type of semantics of sequence diagram is used to check whether the sequence diagram itself is consistent with class diagram declaration.

The rest of this paper is organized as follows: Section 2 and 3 describe the UML models, class diagram in section 2 and sequence diagram in section 3. Both sections comprise of their basic definition and elucidation related to their visualizations. These sections elaborate on object-oriented metric and usage of formal semantics. Section 4 provides an abstraction of one example of software system in UML representation regarding the metric-based and semantics implementation. Then, some concluding remarks are drawn in section 5. Finally, a short reference of future work as a further expansion of this research ends this paper.

**Metric-based representation of class diagram**

When we make a representation of a complex system using UML models, we will deal with voluminous graphical as well as textual information. This appearance of information makes us difficult to read and oversee large diagrams which generated from the systems. The visualization of the models suffer from a kind of “haziness”: when zooming out far enough to see the class context (context of the basic element of the system) i.e. the related structure of currently observed classes, detail information like role names or even class names cannot always be read. On the other hand, when we do zooming in to be able to recognize details, the class context is likely to become isolated or concealed at least partially.

To solve this problem with graphical representation using the UML, object-oriented metrics can be employed to draw the attention of developers to central or critical sections of an object-oriented design [7]. In this paper, metrics are used for the creation of class diagram abstraction that show selected sub modules from the reverse engineered architecture of software systems. The metrics are used in several repetitive stages and different ways of:

- Calculate metrics for an observed set of classes.
- Select prominent elements depending on the yielded metric values and determine the class context by calculating and combining metrics.
- Visualize and inspect the newly created sub-view on the architecture.

This metric approach depicts the needed information of selected classes by highlighting dependencies extended by stereotype of classes. Moreover, this concept also provides a stripped view on the class diagram that shows only the coupling between classes to provide an overview and understanding of the connection and relation among classes.

**Basic concepts of object-oriented metrics and related terminologies**

Object-oriented metric is quantitative evaluation regarding collecting or computing numbers needed in pursuing a specific intent related to understanding, controlling or improving of software system [18]. There are two (2) types, first is *product metrics* that measure properties of the software products; second is *process metrics* that measure properties of the process used to obtained these products. *Product metrics* include two categories: *external product metrics* cover properties visible to the users of a product, for example: functionality metrics, performance metrics and usability metrics; *internal product metrics* cover properties visible only to the development team, for example: size metrics, class metrics or object metrics. *Process metrics* include cost
metrics, measuring the cost of a project, effort metrics, estimating the human part of the cost and advancement metrics, estimating the degree of completion of a product under construction. In this research, we employ internal product metric that emphasize on the usability of class diagram by mean of class metrics.

To make a selective representation of a complex class diagram, we have to localize a set of class based on their connectivity. This connectivity is usually expressed as classes’ coupling. This concept has been defined in the context of structured design methods as the degree of interdependence between modules [19]. In the context of object oriented software systems, it refers to the degree of interdependence between object classes. There are three (3) kind of coupling. First is tightly coupled, it is when one object depends implicitly on another. Object instances are tightly coupled with their classes. Second, when one object depends directly on the visibility of another, they are closely coupled. Third is loosely coupled when one object references another only indirectly to the other’s public interface [15].

In object-oriented systems, close coupling should be avoided in favor of loose coupling and high cohesion, to achieve a well modularized system that allows reuse. Localizing areas of high coupling has therefore to finding classes that require the context of their surrounding (closely coupled) classes for understanding and thus breaking the system down into smaller parts that are easier to comprehend. This is the basic of creating selective representation in class diagram using OO metrics.

**Object-oriented metrics involved**

To determine the suitable metrics for isolation of class diagram part, we have to consider about two types of class metrics, i.e. intra class metrics and inter class metrics. Intra class metrics measure characteristics of class interne and may therefore be used to select single set of classes whose respective metrics values exceed a given threshold. Inter class metrics quantify features between a set of classes and can therefore be used to select subsets of the given set of examined classes (e.g. a package). Below, we present several established OO metrics that measure characteristics about classes or their relation to other classes.

The fan-in and fan-out metrics [5] measure the information flow between classes. Fan-in counts the number of locations from which control is passed into a module (a class, in our case), plus the number of data structures from which information is retrieved by the examined class. We use the following definition for fan-in. Let $C$ be a class and $S$ the set of classes calling methods from $C$. Then $FI = |S|$. Fan-in metric can be used to find classes whose services are used by many others, i.e. who have a high reusability.

As fan-out metric can be used synonymously with CBO (coupling between objects [15]), we only use the latter term. CBO measures the number of classes to which a class is coupled. According to remarks and comments on CBO and coupling, we include coupling through inheritance into calculation but we do not consider messages that a class sends to itself in the metric calculation. We do so for reason that self messages do not appear in graphical representations on the class diagram level. Moreover, the concept of the metric is to get an idea of interconnection and interaction between distinct classes. Two classes are considered coupled, if methods declared in one class call methods or access attributes defined in the other class. We use the following definition for CBO:
Let $C$ be a class and $M = \{m_1, ..., m_n\}$ the set of methods of $C$, $R_i$ the set of methods called by $m_i$ and $A_i$ the set of attributes accessed by $m_i$. Let $\omega() : \text{Set(Feature)} \rightarrow \text{Set(Class)}$ determine from a set of features (i.e. attributes or methods) to the set of their owners (i.e. classes). Then:

$$CBO = \bigcup_{i=1}^{n} \omega(R_i) \cup \bigcup_{i=1}^{n} \omega(A_i)$$

High CBO values indicate large numbers of interconnections between classes.

Another metric considered is MPC (message passing coupling). This metric counts the number of send statements in a class [11]. This is similar to CBO but considers in addition to the number of distinct coupled classes, the number of messages, or the quantity of inter class communication. This can be used as a measure for the intensity of coupling between classes.

**Formal semantics representation**

We focus on using formal semantics of UML in this paper to check the consistency between sequence diagram and class diagram. As we present the semantics of sequence diagram in the context of class diagram, firstly, we would like to introduce the notation involved of class diagram. Then, it is followed by introduction of the notation for sequence diagram. In recouping, the static semantics of sequence diagram is elaborated in the end of this section.

**Notation of class diagram**

One particular class diagram $\Delta$ of an observed system identifies the environment in which the sequence diagrams do their operations. This operating environment consists of four groups listed below:

1. First group provides the static information on classes and also their inheritance relationships:
   - $CN$: the finite set of classes in the diagram.
   - $\text{super}$: the partial function which maps a class to its direct super class, for example: $\text{super}(C) = D$ means $D$ is the direct super class of $C$.

2. Second group describes the structure of each class and for $C \in CN$, it includes $\text{attr}(C)$:
   - The set of $\{<a_1:T_1>, \ldots ,<a_m:T_m>\}$ attributes of $C$, where $T_i$ stands for the type of attribute $a_i$ of class $C$, and will be referred by $\text{type}(C.a_i)$.
   - The type of an attribute is assumed to be a built-in simple data type.

3. Third group identifies the relationships among the classes:
   - $AN$: the finite set of associations names captured in the diagram and the set $Ass$ of associations that is a subset of $CN \times AN \times CN$.

4. Fourth group defines the set of methods for each class in the diagram: $\text{method}(C)$ is the set of all methods of class $C$ in the diagram.
   - $A$ is one particular association which is used to define $<C_1, A, C_2>$, defines a representation of association between classes, $\text{aggr}$ represent a specific association which indicates a lifetime dependency among the related parts. One end of the association is considered the `whole' part of the aggregate structure and the other ends are considered `parts' of the class at the `whole' end. The class at the whole end is referred to as the aggregate class, and the classes at the parts end are part classes.
Cardinality or multiplicity is not regarded in this elaboration since it has no relation to class diagram represented.

Notation of sequence diagram

To give bearing in mind, it is worthy to overview the context of sequence diagram with the other UML models. Starting with use case diagram, each use-case describes particular functionality of a system [10]. Generally, a use-case is defined as one basic course and several alternate courses in requirement analysis phase. A use-case course describes a sequence of interactions between actors and the system, which is an abstract template of a family of scenarios. According to each course description, a sequence diagram is presented in system design phase to realize the corresponding use case course. Sequence diagrams are used to present the dynamic behavior of system design while class diagrams are system static structure. As one of two kinds of UML interaction diagrams along with collaboration diagram, a sequence diagram shows interactions between objects arranged in a time sequence.

In applying syntax of sequence diagram, we have to consider what type of sequence involved. Since a sequence diagram is defined as a sequence of messages, we should give formal definition of messages [11].

A message is a tuple: \( \text{msg} = (\text{obi} : C_i, \text{obj} : C_j, \text{action}, \text{order}) \) where:

1. \( \text{obj} \) is the source object of the message with class type \( C_i \), of course the source of message can also be an actor.
2. \( \text{obj} \) is the target object of the message with class type \( C_j \).
3. \( \text{action} \) is a guarded method call of the form \( g \rightarrow \text{act} \), where \( g \) is a Boolean expression of attributes of source object \( \text{obj} \), and possibly public variables; \( \text{act} \) is either a command without method calls (an internal action) or a method call \( \text{obj}.m() \).
4. \( \text{order} \) is the order number of the message in the corresponding sequence diagram structure tree. All the order numbers of messages in the sequence diagram construct a partial order. The order number of a message is given according to its position in the tree. The root node is corresponding to the starting object or actor of a sequence diagram. Then the first layer branches with order number is 1, 2, 3, · · · , \( n \). From the \( u \)-th node \( \text{obj}_u \) of first layer object nodes, perhaps there are \( m \) nodes. The corresponding order numbers of branches are \( u.1, u.2, \cdots, u.m \). The \( u.v \) is the \( v \)-th branches from the node object \( \text{obj}_u \). Given the sequence diagram example in figure 5, the corresponding structure tree with ordered messages is shown in figure 6.

Static semantics of sequence diagram

Suppose, there is a class diagram \( \Delta \) and let \( CN \) be the set of class names in \( \Delta \). Then, \( AN \) the set of association names, the form of \( \langle C_1, A, C_2 \rangle \) is representation of association between classes, \( A \) is one particular association, where \( C_1, C_2 \in CN, A \in AN \) and also \( \text{method}(C) \) the set of all the methods of class \( C \) in \( \Delta \). Obviously, for a given message

\[ \text{msg} = (\text{obi} : C_i, \text{obj} : C_j, g \rightarrow \text{m}(), \text{order}) \]

we can define its static semantics \( M_s \) as follows [16]:

\[ M_s[\text{msg}] = C_i \in CN \land C_j \in CN \land \exists A \in AN \]

\[ \langle C_i, A, C_j \rangle \in AN \land m \in \text{method}(C_j) \]
The static semantics of a sequence diagram is defined as the conjunction of all its messages in the structure tree. Therefore, the static consistency of a sequence diagram is statically consistent (in the context of a class diagram $\Delta$), if the static semantics of every message is true. In particular, all classes of objects in the sequence diagram must be a class in the class diagram, every method from a class $C$ to another class $D$ must be a declared method in class $D$, and there must be an association between class $C$ and $D$ so that the method of $D$ can be called by $C$. The checking of this kind of consistency can be easily automated.

**Example of representation study: NextGenPOS**

In this section, we elaborate metric and semantics-based representation for both class and sequence diagrams. We exploit a NextGenPOS system, abbreviation of “point of sale system”, an integrated software application that records sale and handles payment [10]. This system consists of 32 class and 148 methods. To illustrate the idea of the size and structure of this system, we display associated class diagram as shown in figure 1.

![Figure 1. Hazy class-diagram representation of NextGenPOS system](image-url)

As can be seen, it is difficult to depict information about this class diagram. Even, in case, we zoom in this diagram until it reaches the readable size, we still have another kind of problem. We will have an architectural concealing or implementation...
discarding. Base on this point, this metrics-based approach is employed in providing a selective representation of this class diagram as exploited in following sub sections.

**Class diagram**

In giving a brief understanding how should one particular metric being calculated, we provide one simple example from Sale class and its makePayment method. We illustrate these exemplary CBO and MPC metric counting by means of Java code abstraction, a cutlet of related sequence diagram and Sale class below.

![Sequence Diagram and Java Code Abstraction](image)

**Figure 2. Cutlet of payment sequence diagram and Sale class**

The abstraction of Java code above can be used as an input of MPC metric counting when we are in reverse engineering condition. On the other hand, the associated sequence diagram is an alternative way in undertaking this calculation, especially when we are in system development phase. In perspective of Sale class, it can be easily measured that the value of CBO is “1+2”, since it involves 1 method invoked (makePayment) and 2 attributes involved in this invocation which are. As MPC value for this abstraction, it is 1, since there is makePayment operation solely. Base on this guideline, we can calculate CBO and MPC values for all classes in NextGenPOSSystem.

Doing the similar calculation procedure, now, we can select one prominent class from original class diagram as a central class in a new selective representation. This class is selected based on above-average metric values. For a non-trivial class diagram, the choice can be constrained further by applying one particular threshold value to create a binomial group of classes. Then, we can go further to attach this central class with its context by determining the set of classes to which the messages are sent. Since the regarded metrics are MPC and CBO, we will have a set of class classes which is focused on coupling strength as follows:
Actually, this selective representation bases on the original class diagram with the coupling line added between coupled classes. Also, each dependency is labeled with the metric value for the connected classes.

**Sequence diagram**

Static semantic of sequence diagram is defined as the conjugation of all its messages passing in its hierarchical structural tree [16]. Therefore, the static consistency of sequence diagram is capture by the static semantic condition. It means sequence diagram is consistency in the context of class diagram $\Delta$ if the static semantic of observed sequence diagram is true. In particular, all classes of objects in the sequence diagram must be a class in the class diagram, every invocation from class $I$ to another class $J$ must be related to a declared method in class $J$ and there must be an association between class $I$ and $J$ so that the method in $J$ can be called by $I$.

To elaborate the representation of sequence diagram, we start with syntax of class diagram sample below.

![Sequence Diagram](image)

**Figure 4. Conceptual class diagram of payment by credit sub system**

The notation of above class diagram is given as follows, where we omitted the methods of classes as well as the attributes:

$CN = \{\text{Cashier, NextGenPOSSystem, TaxCalculator, Account, CreditAuthorizationService}\}$

$AN = \{\text{makeSale, getTaxes, isApproved, post}\}$
\[
\text{super}(C) = \text{Object}, \text{ for all } C \in CN \\
\text{attr}(\text{Cashier}) = \{<\text{omitted}>\} \\
\]

\[
\text{attr}(\text{Account}) = \{<\text{omitted}>\} \\
\text{Ass} = \{<\text{Cashier, makeSale, NextGenPOSSystem}, <\text{TaxCalculator, calculateTax, NextGenPOSSystem}, <\text{NextGenPOSSystem, isApproved, CreditAuthorizationService}, <\text{NextGenPOSSystem, post, Account}>\}
\]

A refinement of the model allows us to add more details, such as attributes and operations [11].

In consistency checking of sequence diagram, we provide one example of sequence diagram which exhibit one flow/scenario which is named as payment by credit as a part of NextGenPOSSystem in figure 5.

![Figure 5. Sequence diagram of payment by credit thread](image)

For the first consistency, i.e. conformity between sequence diagram and class diagram, we develop static semantic for this payment by credit flow as follow:

\[
M_s[[\text{PaymentByCredit}]] \text{ is defined as:} \\
CN = \{\text{Cashier, NextGenPOSSystem, TaxCalculator, CreditAuthorizationService, Account}\} \\
\text{Ass} = \{<\text{Cashier, makeSale, NextGenPOSSystem}, <\text{TaxCalculator, calculateTax, NextGenPOSSystem}, <\text{NextGenPOSSystem, isApproved, CreditAuthorizationService}, <\text{NextGenPOSSystem, post, Account}>\} \\
\text{m} = \{\text{makeNewSale, endSale, makeCreditPayment, getTaxes, requestApproval, postReceivable, postSale}\}
\]

Second consistency for sequence diagram well formedness is checked by mean of corresponding hierarchical structure tree for this sequence diagram as illustrated in figure 6. This composition of tree is constructed based on basic activation notation, iteration notation and create/destroy notation [16].
Ordered branches present the hierarchical relationships among the messages. The execution of message (i.e. execution of the method represented by the message) in the sequence diagram must follow the traversing rule of first root then son trees from left to right. For one sequence diagram, there is only one token to denote control right of invoking message. In the beginning of an execution of the sequence diagram, the token is in the actor’s hand (Cashier). When the source of message wants to invoke a message, the source object or actor must hold the token. When the message is invoked, the token should be passed. The source of message will wait until the target return the token. The target can be invoked by another message or return the token to the previous source which invoked it before. We consider that the execution of a sequence diagram from the start to termination is the process of the token traversing from the root node of the hierarchical tree to sub trees from left to right, finally the token return back to the root.

Contrarily, the main indicator of inconsistency in sequence diagram, regarding its well formedness can be evaluated base on the message order in structure tree. If the order of the messages given in the sequence diagram with one thread is not conformed to a hierarchical structure tree, then the diagram is not well formed. For example, the sequence diagram shown in figure 7 is illegal because after the first message makeCreditPayment( ) finished, the thread control point is returned the actor Cashier. However the next message postCredit( ) in the diagram should be invoked by object TaxCalculator which does not possess the flow control (token) right at that time.

Concluding remarks

In this paper we presented two approaches for representation study of UML class diagram and sequence diagram consecutively. Using these techniques we observed that:
a. Selective representation technique of class diagram gives us more opportunity to understand programs more clearly by examining smaller subparts that eventually merge to a whole. Our motivation was to find solutions for the “haziness” often encountered when viewing large UML diagrams: because of their inherent characteristic to contain graphical as well as textual information, it is hard to see both fine detail and an overview over the complete structure at the same time. We have presented a NextGenPOS application to show two different visualizations in addition to the plain class diagram (figure 3).

b. The applied semantics for sequence diagram can be used as one alternative approach in overcoming inconsistency between UML models. As stated in Reports on teaching and using UML for software development show that the majority of inconsistencies are caused by the lack of a precise understanding of these issues [9]. Moreover, for example, the report [11] depicted from UML 2003 Workshop on Consistency of UML shows that more than 80 percent mistakes in sequence diagrams drawing was related to message passing between unlinked objects. The observed semantics in this paper can be applied to reduce this sort of mistake. However, implementing of this approach is not free of hurdles. Since creating a semantic syntax is not such a trivial task, not all users have enough intention in doing so. It requires not a small effort to build the semantics as a base for consistency checking. This is one prominent disadvantage highlighted from this paper. We will work further on this set of problems to provide more effective approach in UML inter model consistency checking.

Future work

To represent these two UML models by means of metrics and formal semantics gives us more focused visualization and appropriate information underlying inter model consistency checking. However, we still have to consider the complexity dealing with the syntax in formal semantics. As we can imagine, it is very rare to find that user has intention to check this formal meaning manually (as we stated beforehand in section 5). In addition, the role of automation system for calculating desired metrics value and checking every single instance of syntax are almost inevitable. In responding to this, we will create an application system which copes with class diagram and sequence diagram simultaneously. We will exploit the communication between classes or objects by observing their association and message passing. In short, class diagram and sequence diagram will be regarded as inputs. Then, representation application system will take part as a process or through-put. In the end, message passing and association between classes or objects visualization will be the outcome of these concurrent system. At a glance, the approach for our future work is shown as follows:

![Diagram showing the approach of message passing and association visualization system](image-url)
This approach will utilizes observed class diagram and sequence diagram as the inputs. The through put artifact is the visualization of messages passed between classes or object instantiated. To sum up, the outcome is the validated class diagram and sequence diagram. Advantages of this proposed approach are:

a. We will have a comprehensive graphical visualization so that it will be easier to undertake tracking action from displayed outcome of message passing backward to the class and sequence diagram respectively.

b. It is not necessary again to create a circumference method, such as developing formal semantic for class diagram or sequence diagram, in order to check the validity of process modeling.

To provide this automation system, we will integrate the UML tool development with the manifestation of the approach that is shown in aforementioned figure. Having this, one real contribution in UML inter model’s validation will be conveyed splendidly.

References


P.R. Reed, Jr. Developing Applications with JAVA and UML, Addison-Wesley, 2002.