METAMODELS FOR THE OBJECT-ORIENTED APPROACHES OF BOOCH AND OMT* **

F. Losavio, A. Matteo†
Centro de Ingeniería de Software Y Sistemas ISYS,
Escuela de Computación Universidad Central de Venezuela,
Apdo. 47567, Caracas, Venezuela
flsavio@conicit.ve, amatteo@conicit.ve

F. Schlienger
Laboratoire de Recherche en Informatique L.R.I.
Université de Paris-Sud, Orsay
Bat. 490, 91405 Orsay, Cedex, France
fs@lri.lix.fr

Abstract

The main goal of this work is the definition of metamodels for two well known object-oriented methodologies for software development: Booch and Rumbaugh et al. (OMT). The semantics of the underlying concepts is discussed in order to construct the metamodels. They are an important issue towards the definition of the abstraction for the tool supporting the methodology (e.g. the metamodel constitutes the abstraction underlying a graphical editor or a template for a class specification). Moreover, the metamodels facilitate to switch easily from one methodology to another, in the context of an application development environment.

Keywords: metamodel, object-oriented models, object-oriented methodologies, software development

* Object Modeling Technique, by James Rumbaugh, Michel Blaha, William Premerlani, Frederick Eddy, William Lorensen
** This research is supported by the New Technology Program of the BID-CONICIT and the Postgraduated Cooperation Program in Informatics of the CEFI-CONICIT
† This work was initiated when the authors were visiting the Laboratoire de Recherche en Informatique, L.R.I., Orsay, France
1. INTRODUCTION

The framework considered for this work is an integrated environment for the development of object-oriented (OO) applications\(^1\) providing tools for software development using analysis and design methodologies such as Booch\(^2\) and Rumbaugh et al. (OMT)\(^3\). Important features of these tools are the generation of class specifications and the graphical editors, supporting the graphical notations of the mentioned methodologies.

The main goal of this paper is the definition of the two metamodels corresponding to the OO models supporting Booch and OMT methodologies, discussing fundamental OO concepts holding a similar and different semantics in each methodology. These differences or particularities are captured by each metamodel as classes. Among the great number of OO methodologies for software development, whose comparison may be seen in\(^4\)\(^5\)\(^6\)\(^7\), we have chosen to consider the approaches of Booch and OMT because, on one hand they are supported by well known tools and they are fairly well documented. Moreover, according to their authors, they have been used to implement real applications. On the other hand they show a complete OO modeling process including static, dynamic and functional aspects.

In general, a metamodel enlights the basic concepts of the underlying model. Particularly, the two metamodels proposed here would permit to:

- Establish equivalencies among the methodologies. A process can be defined for each methodology, indicating the step up to which the processes are similar. Notice that the main aspects considered in a methodology are the concepts on which it is based, the notation used and a development process. So, within the proposed environment, a user may switch from one methodology to another, for developing an application. For example\(^8\), the analysis phase may start using OMT methodology, continuing the design using Booch method.

- Define a pattern for the specification of a class which is common to the two methodologies; this pattern facilitates the automatic code generation for applications built within the environment.

- Represent the model of the problem domain component for the corresponding edition tool.

2. SEMANTICS OF THE OBJECT-ORIENTED CONCEPTS USED BY EACH METHODOLOGY

In the following section, the semantics of the OO notions supported by each methodology, are discussed considering objects' static and dynamic aspects.
The class and object concepts, which are fundamental within an OO context, are classically considered within the two OO models. An object is something (an abstraction) that make sense in the application domain. It has an identity, differentiating it from other objects, which are distinguished by their inherent existence and not by their descriptive properties. Moreover, an object is an instance of a class. A class is a description of a group of objects with common attributes (or data values), operations and semantics. An attribute is a property of the objects in a class; notice that unlike objects, attributes have no identity. An operation is a function or transformation that may be applied to or by objects in a class.

2.1 STATIC ASPECTS

The notions underlying the static aspects of the OO model supported by both methodologies, are described in what follows.

2.1.1 RELATIONSHIP NOTIONS

Static links are used for characterizing relationships among objects, which are permanent in time. They are presented in what follows for both methodologies, pointing out differences and similarities:

i. Booch association notion.
Booch considers the association relationship, which is used early in the analysis development process, for establishing a semantical connection among the classes that have been identified. Later, this relationship is refined into the inheritance, has or use relations. The association denotes just a semantical dependence and establishes in general a bidirectional connection. A meaning can be given to the association using the role played by each one of the classes involved. An association may be related with itself and it is called reflexive association. Moreover, more than one association may be established between the same pair of classes. Associations may also be adorned with cardinalities, applied to the target end of the association and denoting the number of links between each instance of the class source and instances of the target class.

ii. Booch use notion.
The use relation has recently appeared in OO models, and it has been included in several OO methodologies, Booch's for example. This relation establishes a path for message exchange among objects. Its semantics establishes the fact that a client object "uses" the services of a server object. Moreover, even if it is considered as a static relationship, since it defines a permanent connection among objects of two different classes, it is possible to associate a dynamic meaning to this client-server link. Henderson-Sellers\textsuperscript{9}, considers that the OMT association and aggregation relations, used during the analysis phase, are transformed into a use relation during the design phase. In Booch, if an association denotes a bidirectional
instance connection, a use relation is a possible refinement of an association, signaling which are the client and the server providing certain services. A class may use another in the description of its interface and/or in its implementation.

iii. OMT association notion.
The association concept is carefully treated in the OMT model. The only refinement considered is the aggregation relationship. An association or link (instance of an association for relating objects as class instances) is the way for establishing relationships among classes or objects, respectively. Associations have multiplicity, for specifying how many instances of a class may relate to a single instance of an associated class. Multiplicity constrains the number of related objects. Associations may be named, and may have roles, related to each end class involved in the association. Notice that this semantics is similar to the Booch use notion. Moreover, associations may be binary, ternary or of higher order; Booch instead, considers at most binary relationships. An OMT association may be also modeled as a class, called associative class, holding attributes and operations. This feature is absent in the Booch association notion.

iv. Aggregation notion.
The aggregation concept is similarly treated in both models. An aggregation defines an ownership relation between an assembly object and one or more component objects. The particular semantics between two objects $x$ and $y$ is the following: $x$ has-a $y$ or reciprocally, $y$ is-a-part-of $x$, where $x$ is called aggregate or composite object and $y$ is called a component object. In Booch model, it is considered by value or by reference. The "by value" aggregation means that the objects of the component class depend from the objects of the aggregate class, e.i. that the object life cycle of the component class is conditioned by the existence of the object of the aggregate class. The "by reference" notion, instead, means that the objects of the component class are independent from the existence of the objects in the aggregate class. In OMT, an aggregation is an association with some extra semantics (part-whole or a-part-of relationship), in which objects representing the components of something are associated with an object representing the entire assembly. Component objects are not independent from the assembly, in the sense that the component life cycle depends on the assembly life cycle (similar to the Booch "by value" notion; the "by reference" notion is not considered in OMT).

In both methodologies, aggregations hold transitivity and anti symmetric properties (tree organization) and they may have explicit multiplicity (called cardinality in Booch), allowing a fixed, variable or recursive aggregation.

v. Inheritance notion.
The inheritance (also called generalization/specialization) notion used by Booch and OMT is similar to the notions currently used in other OO models. Simple and multiple inheritance are considered. A subclass
extends or constrains the structure and behavior of its superclass (inheritance by extension or restriction). Notice that within the OMT model, the term inheritance is reserved for the implementation language mechanism, whereas the *generalization* term stands for the conceptual relationship; however, the term *multiple inheritance* is also used. The OMT model distinguishes the inheritance involving a class partition from the inheritance with non-disjoint classes.

In both methods, a generalization/specialization class can be considered as an abstract class, i.e. a class without instances. Booch provides a notation to distinguish it, OMT does not have a particular one, but instead, it allows to indicate when a class has instances.

### 2.1.2 CONSTRAINTS

Static constraints, also called *invariants*, must be verified at every moment by the objects of a class. They restrict the possible states of these objects. Constraints may be included in the definition of the concepts used in the modeling process. For example in OMT, the notation for the generalization mechanism distinguishes whether the subclasses are mutually exclusive (disjoint) classes or not. Another example is the number of objects that can be related in an association relationship between two classes, which is constrained by Booch's cardinality or OMT's multiplicity. In OMT, these restrictions are called *constraints on links*. Other kinds of constraints, such as attribute values, are not directly specified using the model's concepts.

Both Booch and OMT models consider constraints expressed as declarations, on objects, attributes, classes and associations. In general, declarative constraints are converted into conditions and/or procedures of a class in the target language implementing the application model.

### 2.2 DYNAMIC ASPECTS

A *state transition graph* is defined on a class and describes the possible life cycle of the objects in the class, i.e. the sequence of events that may affect the objects during their life course. The nodes of the graph represent the *states* of each object in the class. The arcs correspond to the transition of an object from one state to another, caused by an event on the object. The state transition semantics involves the following aspects:

* An object enters the state $E_1$; the occurrence of event $E_{1'}$ causes the object to enter the state $E_2$.

* The situation in which an object is in a given state, and there is no transition to pass into another state for the occurrence of a certain event, may be handled in two ways:
- the event is ignored (in OMT)
- the event cannot occur. In this case, constraints are imposed by the state transition graph on the possible event occurrences and not only on the object evolution.

The OMT dynamic model concepts are based on the works of Harel\textsuperscript{10, 11}, with an extended semantics. The following considerations may be established:

- The operation notion is used to describe the reaction of the object with respect to the events. Two types of operations are distinguished:

  - an activity, whose execution lasts a certain time. It is associated to a state and it ends after a predefined time or when an event causes a state change.
  - an action is an instantaneous operation, which is associated to a state transition. An action may send an event to an object.

- The structuring of the state transition graphs is allowed:

  - nesting facilitates to detail an activity or an event by means of a graph transition graph involving higher level activities or events.
  - generalization permits to show different states as a unique state.
  - aggregation means to define an aggregate object state as the state transition graph aggregate of its components.

The description of eventual dependencies among the state transition graphs is achieved by conditions interchange. Booch dynamic model follows the main guidelines mentioned for OMT, so it will not be discussed here.

3. METAMODELS FOR THE OBJECT MODEL OF EACH METHODOLOGY

In Section 2, the concepts underlying the Booch and OMT OO models have been presented. In the following sections the respective metamodels and their notation will be described.
3. I BOOCH METAMODEL

Figures 1 below shows the Booch metamodel, expressed using Booch notation.

![Diagram of the Booch metamodel](image)

**Figure 1.** Metamodel for Booch OO model.
3.2 NOTATION USED IN THE CLASS DIAGRAM FOR BOOCH METAMODEL.

In what follows, the notations used for expressing the class diagram of Booch metamodel in Figure 1 will be given. For a detailed description of Booch notation, the reader is referred to\(^2\).

A cloud with dotted borders denotes a class

A cloud with an inscribed triangle denotes an abstract class

Let \( X, Y, Z \) be classes. The following icons denote relation among classes:

\[
\begin{array}{c}
\begin{array}{c}
X \\
\bigcirc \\
Y
\end{array} \\
\end{array}
\]

Denotes a use relation. Class \( X \) uses class \( Y \)

\[
\begin{array}{c}
\begin{array}{c}
X \\
\bigcirc \\
Y
\end{array} \\
\end{array}
\]

Denotes an aggregation or has relation. Class \( X \) has \( Y \) as one of its parts

\[
\begin{array}{c}
\begin{array}{c}
X \\
\longrightarrow \\
Y
\end{array} \\
\end{array}
\]

Denotes an inheritance relation. Class \( X \) inherits from class \( Y \)

The following icons, called adornments, express the cardinality of the relations:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Exactly one</td>
</tr>
<tr>
<td>N</td>
<td>Zero or more (a limited number)</td>
</tr>
<tr>
<td>0..N</td>
<td>Zero or more</td>
</tr>
<tr>
<td>1..N</td>
<td>One or more</td>
</tr>
<tr>
<td>0..1</td>
<td>Zero or one</td>
</tr>
<tr>
<td>2..7</td>
<td>Specific range</td>
</tr>
<tr>
<td>2..7,9</td>
<td>Specific range or exact number</td>
</tr>
</tbody>
</table>
For example, for adornments 1 and 1..N and a has relation between classes X and Y, saying:

```
X ---- Y
   ^
  1   1..N
```

we say that, for each instance of X we may have 1 to N instances of Y and for each instance of Y there is exactly one instance of X.

### 3.2 DESCRIPTION OF BOOCH METAMODEL

The class modeling the application in Figure 1, called application model, is constituted by several category classes, which in turn are conformed by several classes, called Booch class. A Booch class is an abstract class characterizing the different kinds of classes that may be defined in the methodology, through an inheritance relation. The gen/spec abstract class models the inheritance relation. We find the abstract class and the concrete class having as subclasses leaf and non-leaf classes. Moreover, inheritance is also used to model non gen/spec classes which are parameterized, instantiated, gen/spec parameterized, utility and metaclass classes. Notice that an utility class may also be a parameterized class. In order to define the different kinds of class relations in the Booch object model, we have established the association superclass, from which use, inheritance and has subclasses are derived for modelling these notions. Each Booch class is constituted by operations, and eventually by attributes and constraints classes, modelling the corresponding concepts. Finally, object diagram and state transition diagram classes may be associated to each Booch class.

Notice that in Booch metamodel, the parameterized, instantiated, utility and gen/spec parameterized classes are not directly related to concepts in the OO paradigm, nevertheless, they are included in Booch OO model. These classes are actually more relevant during the design phase.

Notice also that an object diagram can be associated to each Booch class. This feature models the different kinds of client/server communications among the objects of a class and the other related objects of the system. Moreover, it can be considered as a refinement within the analysis phase. Nevertheless, it does not add any additional semantics to the fundamental OO concepts.
3.3 OMT METAMODEL

Figures 2 below shows OMT metamodel, expressed using OMT notation.

![OMT Metamodel Diagram]

Figure 2. Metamodel for the OMT model.

3.4 NOTATION USED IN THE CLASS DIAGRAM FOR OMT METAMODEL

The notations used for expressing the class diagram of OMT metamodel in Figure 2 will be given below. For a detailed description of OMT notation, the reader is referred to³.
3.5 DESCRIPTION OF OMT METAMODEL

OMT and Booch metamodels are similar with respect to the basic concepts of the OO model. Nevertheless, on one hand, in the OMT model, the Booch non-gen/spec specialized classes are not present. On the other hand, with respect to the inheritance hierarchy established for the relations among classes, they reflect the differences we have stated in Section 2. Notice also that an explicit notation for the abstract class is absent in OMT, as also the concept for grouping classes. Finally, another notion, that is present in OMT and absent in Booch, is that an association may be modeled as an associative class.

Notice that states that it is useful to model an association as a class when links can participate in associations with other objects or when links are subject to operations; so this concept is included in the OMT model and it is shown in Figure 2 as the associative class. Nevertheless the associative class notion is not a concept of the OO paradigm.

3. CONCLUSIONS

We have presented here two interpretations of the basic concepts underlying the OO models, for Booch and OMT methodologies respectively. The corresponding metamodels have been established on these basis.
These metamodels enhance the construction of a model integrating differences and similarities of the OO concepts which have been presented here. The definition of this model by means of a pattern or template for class specification is actually an undergoing work. We believe that this is an important goal towards the construction of a common underlying abstraction for graphical edition tools supporting the graphical notations and class specifications corresponding to the two methodologies, within an integrated environment for developing object-oriented applications. This model could be also extended to include other methodologies.

Moreover, this approach facilitates the fusion of different methods, establishing the basis for an equivalence between concepts and processes and allowing to switch easily among the methodologies.

4. REFERENCES


2 BOOCH G. Object-Oriented Analysis and Design with Applications Benjamin/Cummings Company (1994)


4 CHAMPEAUX D., FAURE P. A comparative study of object-oriented analysis methods JOOP, (March/April 1992) pp 21-33


8 LOSAVIO, F., MATTEO A. An Object-oriented process for combining object-oriented methodologies To appear

