Formal Foundations of Object-Oriented Modeling Notations

Claudia Pons Gabriel Baum

Lifia, Universidad Nacional de La Plata
Calle 50 esq.115, 1er.Piso, (1900) La Plata, Buenos Aires, Argentina
e-mail: {cpons,gbaum}@sol.info.unlp.edu.ar

Abstract

In this paper we describe and classify the different solutions that have been proposed to realize the integration of graphic modeling languages, known and accepted by the software developers, with formal modeling languages having analysis and verification tools.

Inspired from that classification we define a new integration proposal, based on first-order dynamic logic. The principal benefits of the proposed formalization can be summarized as follows: - The different views on a system are integrated in a single formal model; this allows us to define rules of compatibility between the separate views, on syntactical and semantic level. -Using formal manipulation, it is possible to deduce further knowledge from the specification. -The faults of specifications expressed using a user-friendly notation can be revealed using analysis and verification techniques based on the formal kernel model.

The principal difference between this model and other object-oriented formal models is that it integrates both of the levels in the architecture of modeling notations into a single conceptual framework. The integration of modeling entities and modeled entities into a single formalism allows us to express both static aspects and dynamic aspects of either the model or the modeled system within a first-order formalism.

1: Introduction

The increasing complexity of software systems makes their development complicated and error prone. A widely used and generally accepted technique in software engineering is the combination of different models (or views) for the description of software systems. The primary benefit of this approach is to model only related aspects (like structure or behavior). Using different models clarifies different important aspects of the system, but it has to be taken into consideration that these models are not independent and they are semantically overlapping.

The models constitute the fundamental base of information upon which the problem domain experts, the analysts and the software developers interact. Thus, it is of a fundamental importance that it clearly and accurately expresses the essence of the problem. On the other hand, the model construction activity is a critical part in the development process. Since models are the result of a complex and creative activity, they tend to contain errors, omissions and inconsistencies. Model verification is very important, since errors in this stage have an expensive impact on the following stages of the software development process.

Models are constructed using a modeling language (which may vary from natural language to diagrams and even to mathematical formulas).

The success of software development methods, such as Object Oriented Analysis [6], Object Oriented System Analysis [34], Object Modeling Technique [33], Booch's design method [4] and the *Rational Unified Process* [17] are mainly based on their use of intuitively appealing modeling constructs and rich structuring mechanisms, which are easy to understand, apply and transmit to the customers. However, the lack of precise semantics for the modeling notations used by these methods can lead to inconsistencies and ambiguities.

On the other hand, formal languages for modeling, such as Z [35], VDM [19], F-Logic [21], DS-Logic [39] has a well-defined syntax and semantics. However, its use in industry is not frequent. This is due to the complexity of its mathematical formalisms, which are difficult to understand and communicate. In most cases, experts on system domain who decide to use a formal notation, center their effort upon the managing of formalism instead of focusing on the model itself. This leads to the creation of formal models that do not properly reflect the real system.

As a consequence, it has been proposed to combine the advantages of both approaches (see section 2), intuitive graphical notations on the one hand and mathematically precise formalisms on the other hand, in development tools. The basic idea for this combination is to use mathematical

0-7695-0822-7/00 $10.00

notation in a transparent way, hiding it as much as possible under the hood of graphical notations. This approach has advantages over a purely graphical specification development as well as over a purely mathematical development because it introduces precision of specification into a software development practice while still ensuring acceptance and usability by current developers.

2: Combining modeling techniques

Basically, four different proposals to carry out the integration of graphical notations and mathematically precise formalisms have been identified:

Supplemental: the supplemental proposal consists in enriching an informal model with formal concepts. Good examples of it are Syntropy [7], and Lano's [15] and Weber's [38] works, which propose the use of the formal Z notation [35] to enrich semi-formal notations.

Extension: the extension proposal consists in extending an existing formal notation, with concepts that are closer to the application domain as for instance, the concepts adopted by the object-oriented paradigm. In this way, formal notation becomes easier to be understood and managed by software developers. The most relevant examples of this proposals are the language Z extensions, such as Z++ [23] and Object-Z [8]. The languages TROLL [18] OOZE [1] and MAUDE [25], inspired on algebraic specification languages are also part of this group.

Interface: given a formal modeling language, this proposal consists in developing an alternative graphic notation to facilitate the creation and visualization of models. Examples of this proposal are the graphic interfaces provided by formal languages such as OASIS [28] and LTL [32].

Semantics: this proposal consists in formally defining the semantics of a modeling language, known and accepted by the community. Its main components are rules to associate syntactic structures of the modeling language with elements within a formally defined semantic domain.

In our opinion, "semantics" constitutes the most adequate proposal, since it allows that the specifications expressed in a notation known and accepted by the software developers acquire an accurate meaning through its "translation" into a formal domain. Our point of view is based on the fact that both the supplement integration proposal and the extension integration proposal require that the developers know the formal notation, since this constitutes a visible part of the specification. On the other hand, the disadvantage of the interface integration proposals is that the user is compelled to adopt a new graphic language, which is usually influenced by the formalism, thus making it scarcely intuitive.

The main advantage of the semantic proposal regarding the others, resides in that graphic language is turned into a formal language hence, thus the specifications expressed in a graphic language can be formally analyzed to early find out contradictions and ambiguities in the software development process. One of the keys to the success of this proposal resides in hiding the mathematical notation, as much as possible, behind the graphic notation. For example, it should be possible to use formal semantics to develop CASE tools. Only language developers should use formalism to build the CASE tools and justify their correction, while application software developers could handle graphic models avoiding the underlying mathematical formalism.

The Unified Modeling Language UML [36] is a standard graphic language for modeling and specifying object-oriented systems. The language consists of a set of constructs common to most object-oriented languages. From the standardization of the UML active discussions have risen about the semantics accuracy of its constructions. While the Object Management Group OMG was responsible for the standardizing of the UML as notation, the semantics of the UML is still a research issue.

There are an important number of theoretical works - see, for instance [27] and [12] - that deal with different parts of UML, formally defining its syntax and semantics. However, there is still a long way to run regarding this matter. It is particularly hard to compare the results of the respective articles, and it is even harder to combine such results with the aim of obtaining a semantic standard for UML. This difficulty arises because of the different works that use diverse formal methods (or languages), or cover a notation subset, or assume a particular system subclass to be specified. However, an important amount of the proposals can be classified in two groups: formalizations based on the model and formalizations based on the metamodel. We explain this classification in the following section.

2.1: Formalizing modeling languages

A number of approaches for giving semantics to modeling languages (specially the UML) can be classified in two different groups: model-based and *meta-model-based* approaches. This classification is inspired from the four levels in the architecture of modeling notations [36]. The main difference between these approaches is the focus of the formalization. Formalizations in the first group concentrate their attention on the model level (see figure 1a), while formalizations in the second group focus on the metamodel level (see figure 1b):

- In the *model-based* approaches (see [26]; [11]; [37]; [39]; [24]; [22](a)), the individuals in the semantic domain are

the business objects, for example accounts and clients of a bank. (i.e. the formalization focuses on the particular system that is being described).

- In the *meta-model-based* approaches (see [10]; [5]; [36]; [9]; [22](b)) the objective is to give a precise description of core concepts of the graphical modeling notation and provide rules for analyzing their properties. The individuals appearing in the semantic domain are modeling elements, such as classes, attributes, operations, associations, generalizations, etc. (i.e. the formalization is focused on the language itself instead of on any particular system described by the language).

The principal advantages and disadvantages of each approach are summarized in figure 2.

3: The M&D-theory

We introduce the M&D-theory, a proposal for giving formal semantics to the UML. The basic idea behind this formalization is the definition of a single semantics domain integrating both the model level and the data level. In this way, both static and dynamic aspects of either the model and the modeled system, can be described within a first order formal framework. The entities defined by the M&D-theory are classified in two disjoint sets:

- Modeling entities
- Modeled entities

Figure 3 shows this dichotomy of entities. Modeling entities correspond to concrete syntax of the UML, such as Classes or StateMachine. In contrast, modeled entities, such as Object or Link represent run-time information, i.e. instances of classes and processes running on a concrete system.

The M&D-theory provides two different kinds of instantiation relations (see figure 3):

- *Horizontal instantiation*: this relationship connects a modeling entity with its modeled entities, for example Objects are instances of a Class (or we can say that Objects are modeled by a Class) and Links are instances of an Association, S2 is an instance of Savings.

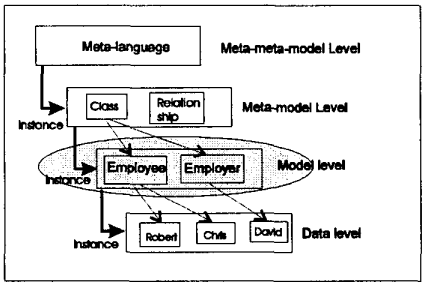

Figure 1a: model-based formalization

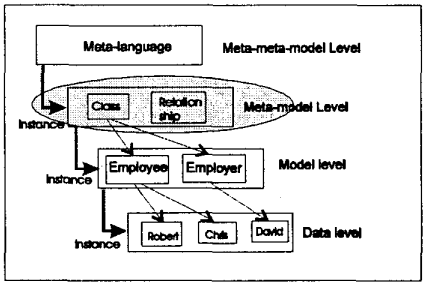

Figure 1b: meta-model-based formalization

Approaches	Advantages	Disadvantages
model-based	▪ It is appropriate for the specification of the information that is inherent to the application. ▪ It allows the detection of inconsistencies and errors in the specifications expressed in the modeling language.	▪ It is not suited for expressing consistency constraints between metaentities (e.g. structural relationships between classes). ▪ It is not possible to represent model evolution in a first order formalism.
meta-model-based	▪ It allows the representation of constraints over the modeling elements in an adequate way (for example between Class and StateMachine) ▪ It allows for the detection of errors and inconsistencies in the modeling language itself. ▪ It gives a first order framework to represent model evolution.	▪ It is difficult to represent constraints over the business objects.

***Figure 2:* advantages and disadvantages of each group**

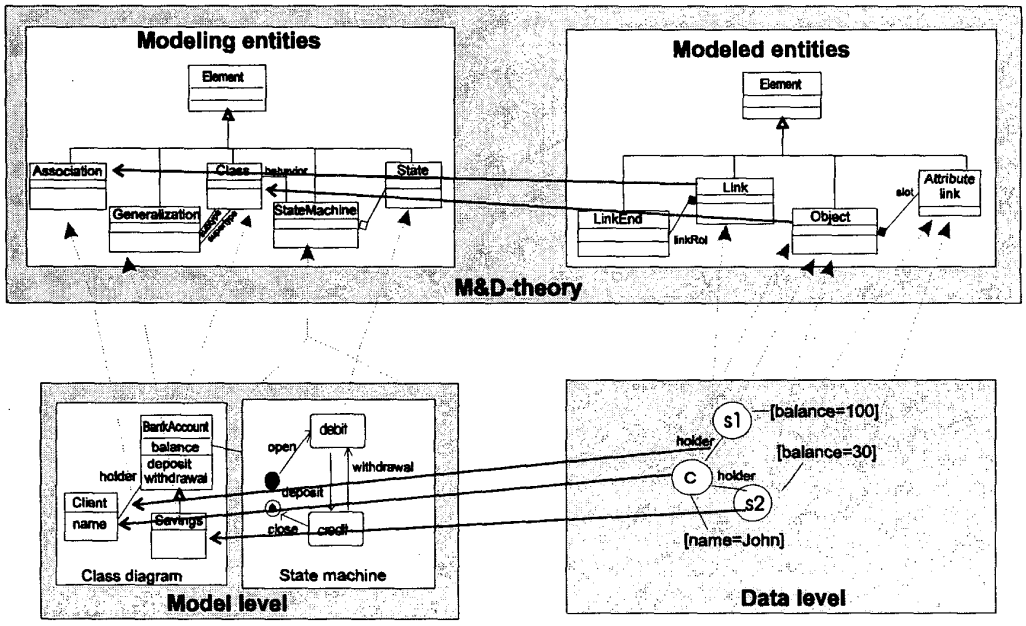

Figure 3: dichotomy of entities

- *Vertical instantiation:* this relationship represents the instantiation mechanism of the metalanguage (dynamic logic in our case). For example, BankAccount is an instance of the metaclass Class, holder is an instance of the metaclass Association, C, S1 and S2 are instance of Object.

It is interesting to highlight that horizontal instantiation preserves vertical instantiation. For example, S2 is an instance (horizontal) of Savings, and there exist M1 and M2 such that S2 is an instance (vertical) of M2 and Savings is an instance (vertical) of M1, whereas M2 is an instance (horizontal) of M1. In the example in figure 3, M1 is the metaentity Class and M2 is the metaentity Object. The formal proof of this property is in [14].

3.1: Structure of the theory

The M&D-theory is a first-order order-sorted dynamic logic theory[1] [16] [39], consisting of three sub-theories:

M&D-theory= UML-theory + SYS-theory + JOINT-theory

The sub theory UML-theory:

The theory describes modeling entities (i.e. models). In the UML, Class Diagrams model the structural aspects of the system. Classes and relationships between them, such as Generalizations, Aggregations and Associations constitute Class Diagrams. On the other hand, the dynamic part of the system is modeled by Sequence and Collaboration diagrams that describe the behavior of a group of instances in terms of message sendings, and by State Machines that show the intra-object dynamics in terms of state transitions.

Modeling entities are related to other modeling entities Consider for example the association between Class and StateMachine by the relation labeled 'behavior'. This association indicates that StateMachines can be used for the definition of the behavior of the instances of a Class. Other example is given by the relation existing between StateMachine and State, that specifies that a StateMachine is composed by a set of States. It is important to formally define how the different UML diagrams are related to one another, to be able to maintain the consistency of the model. Moreover, it is important to specify the effect of modifications of these diagrams, showing what is the

[1] A first-order order-sorted dynamic logic therory **Th** = (Σ, ϕ) consists of a signature Σ that defines the language of the theory, and a set ϕ of Σ-axiomas .A signature $\Sigma = ((S, \leq), F, P, A)$ consists of a a set of sort symbols S, a partial order relation between sorts $\leq$, function symbols F, predicate symbols P, and Action symbols A.

impact on other diagrams, if a modification is made to one diagram.

The theory consists of a signature Σ_{UML}= $((S_{UML},\leq), F_{UML}, P_{UML}, A_{UML})$ and a formula ϕ_{UML} over Σ_{UML}:

$$\text{UML-theory} = (\Sigma_{UML}, \phi_{UML})$$

The set S_{UML} contains sort symbols representing modeling elements, such as Class and StateMachine. The order relation between sorts allows for the hierarchical specification of the elements.

The sets of symbols F_{UML} and P_{UML} define functions and predicates over modeling entities.

The set A_{UML} consists of action symbols representing evolution of specifications over their life cycle. One of the most common forms of evolution involves structural changes such as the extension of an existing specification by addition of new classes of objects or the addition of attributes to the original classes of objects. At the other extreme, evolution at this level might reflect not only structural changes but also behavioral changes of the specified objects. Behavioral changes are reflected for example in the modification of sequence diagrams and state machines.

The formula ϕ_{UML} is the conjunction of two disjoint sets of formulas, ϕ_S and ϕ_D of static and dynamic formulas respectively. The former consists of first-order formulas which have to be valid in every state the system goes through (they are invariants or static properties or well-formedness rules of models). These rules are used to perform schema analysis and to report possible schema design errors. The latter consists of modal formulas defining the semantics of actions, that is to say the evolution of models.

The sub theory SYS-theory:

This theory describes the modeled entities (i.e. data and process). The elements in the data level are basically instances (data value and objects) and messages. At the data level a system is viewed as a set of related objects collaborating concurrently. Objects communicate each other through messages that are stored in semi-public places called mailboxes. Each object has a mailbox where other objects can leave messages.

Modeled entities are related to other modeled entities. For example the relationship named 'slot' between Object and AttributeLink, denotes the connection between an Object and the values of its attributes.

The theory consists of a signature Σ_{SYS}= $((S_{SYS},\leq), F_{SYS}, P_{SYS}, A_{SYS})$ and a formula γ_{SYS} over Σ_{SYS}:

$$\text{SYS-theory} = (\Sigma_{SYS}, \gamma_{SYS})$$

The set S_{SYS} contains sort symbols representing the data in the system and its relationships, such as objects, links, messages, etc.

The sets of symbols F_{SYS} and P_{SYS} define functions and predicates over data.

The set A_{SYS} consists of action symbols representing evolution of data at run time, such as object state changes.

The formula γ_{SYS} is the conjunction of two disjoint sets of formulas, γ_S and γ_D of static and dynamic formulas respectively. The former consists of first-order formulas which have to be valid in every state the system goes through (they are invariants or static properties or well-formedness rules of data). Whereas, the latter consists of modal formulas defining the semantics of actions, that is to say the possible evolution of the data.

The sub theory JOINT-theory:

This part of the theory describes the connection between model and data levels. Modeling entities are related to modeled entities. There is a special relationship among some modeled entities with their corresponding modeling entity, This relationship denotes 'instantiation', for example an Object is an instance of a Class, whereas Links are instances of Associations.

Finally, ϕ_{JOINT} is a formula constructed over the extended language $\Sigma_{M\&D}$, and thus it can express at the same time data properties (e.g. behavioral properties of objects), model properties (e.g. properties about the system specification) and properties relating both aspects. Details of the theory can be found in [29], [30] and [31].

3.2: Static and dynamic aspects of metaentities

Within the M&D-theory, we are able to express all aspects relevant to modeling entities, as follows:

Syntax and Semantics: In conventional textual notation, the syntax of a language is described by the set of characters (alphabet) and the valid sequences of symbols (words and phrases). The language is the set of all valid symbols. If the notation includes diagrams, the syntax seem to be more complex since it is not limited to a linear sequence of characters.

On the other hand, the semantics of a language talks about the meaning of each construction of the language. Usually semantics is given by explaining the constructions of a new language in terms of well-known concepts. This set of well-known concepts is called the semantics domain.

Static concepts and dynamic concepts: In the description of a language there are two dimensions that are orthogonal to both syntax and semantics, they are static concepts and dynamic concepts.

The differences between static semantics and dynamic semantics is well-recognized. While the static semantics characterizes static properties (invariant in time) of the

elements described for the language, the dynamic semantics describes the evolution or behavior of such elements over the time. But, talking about syntax, in general only well-formedness rules of the language are defined, but the problem of evolution or dynamism is neglected. That is to say, syntax is analyzed just from the static point of view. The lost dimension (i.e syntax-dynamic) appears when the constructions of the language (i.e the models) are likely to change during their lifecycle. It is important to make clear that we are not talking about modifications on the syntax of the language itself (i.e evolution on the metamodel level), but about modifications on particular UML models (i.e evolution on the model level).

In figure 4 we show the relations between both dimension. The most remarkable difference is observable for the dynamic semantics. While dynamic semantics on the data-level means run-time behavior, dynamic semantics on model-level describes model evolution in the development process.

	Model (Syntactic domain)	Modeled system (semantics domain)
Static aspects	well-formedness rules for models	well-formedness rules for states of a system.
Dynamic aspects	evolution of the model during the system life cycle.	dynamic behavior of the objects of the system at run time.

***Figure 4:* static and dynamic aspects of syntax and semantics**

3.3: Advantages of the integration

The integration of modeling entities and modeled entities into a single formalism allows us to express both static aspects and dynamic aspects of either the model or the modeled system within a first order framework. In figure 5 we summarize how the M&D-theory deal with each of the four dimension that we discussed previously. The validity problem (i.e. for given a sentence ϕ of the logic, to decide whether ϕ is valid) is less complex for first-order formalisms than for higher order formalisms.

	Model	Modeled system
Static aspects	First-order axioms on modeling entities	First-order axioms on modeled entities
Dynamic aspects	Actions and modal axioms on modeling entities	Actions and modal axioms on modeled entities

Figure 5: the M&D-theory

Although first order logic is undecidable (and as a consequence, nor do dynamic first order logic), computer systems satisfy certain properties (e.g. systems are interpreted over arithmetic structures, the state of a program is given by a finite set of values) that allow us to calculate the validity of formulas in an effective way.

4: Using the formal model

In this section we describe the principal applications of the M&D-theory.

4.1: Formalizing the UML

The M&D-theory introduce precision in the specification of object-oriented systems. Basically the theory formally defines the syntax and semantics of the standard modeling notation Unified Modeling Language (UML). Specifications expressed in a notation known and accepted by the software developers acquire an accurate meaning.

Example 1: Abstract syntax

The UML specification document [36] defines the abstract syntax of UML by class diagrams and well-formedness rules in OCL [36]. In the M&D-theory, the abstract syntax is expressed by means of sorts and functions in the model-level. The possible language constructs are restricted by well-formedness rules on models, such as:

[1] In a Classifier, attribute names are unique:

$\forall c$:Classifier$\bullet\forall f,g \in$ attributes(c) $\bullet$ name(f)=name(g)$\rightarrow$f=g

[2] Cyclic inheritance is not allowed:

$\forall c1,c2$:Classifier$\bullet$IsA(c1,c2) $\wedge$IsA(c2 ,c1) $\rightarrow$ c2 = c1

[3] the trigger of the initial transition of the top level of a state machine should be a creation. In all the other cases initial transitions do not have trigger:

$\forall t$:Transition $\bullet$ (kind(source(t))=#initial $\rightarrow$
(trigger(t)=nullElement $\vee$ (isTop(parent(source(t))) $\wedge$ trigger(t)=create)))

Furthermore, the building of a formal model allowed us to find out and correct ambiguities and inconsistencies in the UML Language. For examples of this subject readers are referred to [40].

Example 2: semantics of structural diagrams

The M&D-theory specifies the semantics of the structural constructs of the UML (i.e Class diagrams) by static axioms describing the well-formedness rules of data, and axioms specifying the inter-level connections, such as the connection between Class and Object . For example:

[1] Values of attributes match the type defined in their Classifier: ∀a:AttributeLink•
IsA(classifier(value(a)) , type(attribute(a)))
[2] An instance cannot belong to more that one composite instance: ∀i:Instance•
∃e1,e2∈ oppositeLinkEnds(i) •
((aggregation(associationEnd(e1))=#composite ∧
aggregation(associationEnd(e2))=#composite) → e1=e2)
[3] Constraints associated to a Class are satisfy for all the instances of that Class:
∀i:Instance• ∀c∈ allConstraints(classifier(i))•
(eval(c)[self/i] = true)

Example 3: semantics of behavioral diagrams

The M&D-theory describes the semantics of behavioral constructs of the UML (e.g. State Machine) by means of dynamic axioms explaining the behavior of objects in terms of transitions in the state machine associated to its class. In the UML specification document [36] this dynamic semantics is explained in an informal way using natural language.

The M&D-theory formally describes the semantics of state machines through the axioms specifying the relationships between evolution actions in the data level (in particular CallActions) and the state machine linked to class of the receiver of the action.

4.2: Specifying compatibility between sub-models

The different views of a system (Class diagrams, Statecharts, constraints, etc.) are integrated in a single formal model. This integration allows us to define compatibility rules among the different views, both at the syntactic and the semantic levels, since it provides a single formal frame where the different views of the system coexist. We give examples of compatibility rules:

Example 1: Pre/post conditions vs. State Machines

Model elements may be associated with a constraint expressing some properties. There are problems when the constrained element has also a behavior that is precisely defined else where in the model. For example, a constraint on an operation (as a pre-post condition) may be inconsistent with the effects of the transitions triggered by its calls in the associated state machine. As a consequence, it is necessary to integrate both views of the system guaranteeing that they are consistent with each other. The following rule formalizes this requirement:

∀<op,s,r,p>:Message• (classifier(r)=owner(op) →
(eval(precondition(op)[self / r, parameters / p])=true
→[r.<op,s,r,p>] eval(postcondition(op)[self / r, parameters / p])=true))

This dynamic formula states that if the preconditions of an operation hold before the execution of the operation, so the post-conditions hold after the execution. Since the effect of the operation are determined by the state machine associated to the class of the object performing the operation, this rule guarantee consistency between both specifications.

Example 2: Generalizations vs. other elements

Generalization diagrams have a strong influence on other diagrams in the model of the system. For example, if two classes c_1 and c_2 are connected by a generalization relation (e.g. c_1 is a subclass of c_2), the behavior of instances of c_1 should be a refinement of the behavior of instances of c_1. This requirement is defined by the following formula: $\forall c_1,c_2$:Classifier • ($IsA(c_1,c_2)$ → refinement(behavior(c_1) , behavior(c_2)))

4.3: Deduction of non-explicit information

One of the fundamental elements of any logic is its deductive apparatus that consists in a collection of rules which can be applied to certain initial information to derive additional information, in an purely mechanical way. Through the formal deduction mechanisms of the logic, it is possible to obtain information that is not explicitly presented in a specification. Examples of deductions in the logic can be found in [31].

4.4: Verifying system correctness

Graphical specifications can be formally analyzed (by using verification tools available in the formal model) in order to detect contradictions and ambiguities early in the software development process. The formal language allows us to express well-formedness rules of both the model and the data of the system. Given a system, it is possible to decide if it satisfies the rules or not. In this section we give examples of verification of well-formedness rules.

Example 1: Design mistakes

Let spec be the UML model in figure 6. Notice that the model has a problem of cyclic inheritance. It is possible to prove that sem(spec)=∅, that is to say that spec is inconsistent with the well-formedness rules of the theory.

In the M&D-theory, the formula φ below specifies the well-formedness rule of inheritance hierarchies:

$\varphi = \forall c_1,c_2$:Classifier $IsA(c_1,c_2) \wedge IsA(c_2,c_1) \rightarrow c_2 = c_1$

And, the IsA predicate is defined as follows:

∀c,c1:Classifier• (IsA(c,c1) ↔

(c=c1 ∨ c1∈allSupertypes(c)))

Let ϕ_{INST} be the instantiation axiom corresponding to the model in figure 6, as follows:

ϕ_{INST} = ∃a,b,c:Classifier•
(name(a)=A ∧ name(b)=B ∧ name(c)=C
∧ ∃g1,g2,g3:Generalization •supertype(g1)=a ∧
subtype(g1)=b ∧ supertype(g2)=c
∧ subtype(g2)=a ∧ supertype(g3)=b ∧ subype(g3)=c
∧g1∈specializations(a)∧ g1∈generalizations(b)∧
g2∈specializations(c)∧ g2∈generalizations(a)∧
g3∈specializations(b)∧ g3∈generalizations(c))

Theorem: the model specified by ϕ_{INST} is inconsistent with the well-formedness rule φ, that is to say:

$(\varphi \wedge \phi_{INST}) \models$ ***false***

Proof: The proof is straightforward. It can be read in [31].

4.5: Formalizing Evolution

Most works on evolution of the system specification, such as ([3], [20], [2]), deal with the problem of structural evolution for example modifying inheritance hierarchy of adding a new class, but they do not deal with the behavior evolution problem, as for instance, changing the way in which an object reacts when receiving a certain message. The evolution mechanism proposed in the M&D-theory deals with both typos of evolution.

In the M&D-theory, each evolution action is defined by means of two formulas:

- **Necessary preconditions** to describe the applicability conditions of operations. The formula $(\langle op \rangle true \rightarrow cond)$ states that the operation *op* is applicable only if the condition *cond* is true.

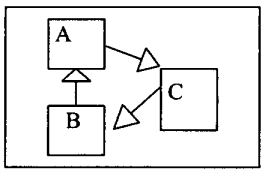

Figure 6: a UML model with a cyclic inheritance hierarchy

- **Sufficient postconditions** to describe the effect (direct effect and change propagation) of the operations. The formula $([op]cond)$ states that after the application of the operation *op* the condition *cond* is true.

These formulas may refer to either modeling entities, or modeled entities or both. This feature allows us to define:

- **intra-level change propagation**: how a modification over a modeled entity impacts on other modeled entities, and how a modification over a modeling entity impacts on other modeling entities. For example, the deletion of a feature of a Class impacts on other models such as StateMachines or Constraints referring the deleted feature. On the other hand the propagation of the deletion of an object representing a whole to the deletion of other objects (i.e its parts) is an example of propagation between modeling entities.

- **inter-level change propagation**: how a modification over an modeled entity impacts on the modeling entities, and how a modification over a modeling entity impacts on the modeled entities. For example the deletion of an attribute of a Class should be propagated to all the instances of that Class.

The specification of each evolution action consists of four sections: ***Action*** act, ***Precondition*** τ, ***Effect*** γ ***Propagation*** δ The schema above represents the following dynamic formula:

$(\langle act \rangle true \rightarrow \tau) \wedge ([act](\gamma \wedge \delta)$

Preconditions are applicability conditions, that is to say conditions under which an evolution action is semantically correct. The clause effect specifies the direct impact of the action, whereas the clause propagation specifies the side effects of the action on other related entities. The following is an example of an evolution action:

Action deleteFeature(c,f)
Precondition
[1] The Feature must exist in the Classifier: f∈attributes(c)
[2] The deleted Feature cannot be referenced from other elements in the package:
∀m∈allContents(p) f∉referencedElements(m)
Effect
[1] The Feature is deleted: ¬Exists(f)∧f∉features(c)
Propagation
[1] The corresponding slot must be deleted from all the existing instances of c (and subclasses):
∀i:Instance• ((Exists(i)∧classifier(i)∈ c∪allSubclasses(c))
→ ¬∃l∈slots(i) • atribute(l)=f)

It is possible to observe how model evolution combines with modeled data evolution. The M&D-logic allows expressing consistency rules among both different UML diagrams and these diagrams and the modeled data. Then, using the deduction mechanism of the logic, it is possible to validate these rules through the evolution.

4.6: Specifying Design Patterns

A pattern is a particular design that appears in certain situations, and that has been recognized as "good design",

that is, it leads to obtaining more flexible and elegant systems that are consequently more reusable. There are catalogs, for instance [13] , where numerous design patterns using natural language complemented by graphic languages such as UML are described. It would be desirable to have a more formal description of the patterns.

The M&D-logic can assist in the task of expressing and recognizing the design patterns. A pattern can be formally expressed by a formula in the logic stating structural obligation (i.e. hierarchy of classes, association between classes and operation signatures), responsabilities and collaborations between objects. Then, this formula allows us to detect the pattern into a specific design, in the following way:

Let:

- M be the UML specification of an object-oriented system.
- $Spec_M$ the instance of the M&D-theory formalizing M, i.e. ***translation***$(M)=Spec_M$.
- U a model for $Spec_M$, i.e. $U \in$ ***semantics***$(Spec_M)$.
- $\Phi_{pattern}$ the formula expressing the pattern.

Intuitively, M conforms the pattern if and only if it satisfies both structure and collaboration obligations required by the pattern. That is to say, the formula $\Phi_{pattern}$ is true in the model M: $U \models \Phi_{pattern}$

4.7: Hiding the formal model

To gain acceptance of the proposed formal model by typical engineers, we are developing a semi-automatic transformation method. This transformation method defines a set of rules to systematically create a single integrated dynamic logic model from the several separate elements that constitute a description of an object-oriented system expressed in Unified Modeling Language (UML). The key components of the transformation method are rules for mapping the graphic notation onto the formal kernel model.

5: Conclusion

Due to the missing formal foundation of the Unified Modeling Language UML the syntax and the semantics of a number of UML constructs are not precisely defined. We have described an object-oriented conceptual model representing the information acquired during analysis and design. We propose this conceptual model as a formal foundation for the UML.

The principal benefits of the proposed formalization can be summarized as follows: the different views on a system are integrated in a single formal model. This allows us to define rules of compatibility between the separate views, on syntactical and semantic level. Using formal manipulation, it is possible to deduce further knowledge from the specification. The faults of specifications expressed using a user-friendly notation can be revealed using analysis and verification techniques based on the formal kernel model.

The principal difference between this model and other object-oriented formal models is that it integrates both of the levels in the architecture of modeling notations into a single conceptual framework. The integration of modeling entities and modeled entities into a single formalism allows us to express both static aspects and dynamic aspects of either the model or the modeled system within a first order formalization. The validity problem (i.e. for given a sentence ϕ of the logic, to decide whether ϕ is valid) is less complex for first-order formalisms than for higher order formalisms.

Furthermore the two-level model is particularly useful for description of system evolution, and formal description of design patterns.

References

[1] Alencar, A. and Goguen, J., OOZE: an object-oriented Z environment, ECOOP91 Proc., Lecture Notes in Computer Science vol.512, Springer-Verlag, (1991).

[2] Bertino, E., Ferrari, E., Guerrini G. and Merlo, I., Extending the ODMG Object Model with time, proceedings of ECOOP98, Lecture Notes in Computer Science 1445, (1998).

[3] Bergstein, Paul, Maintenance of object-oriented systems during structural evolution

[4] Booch, G., Object Oriented Analysis and Design with Applications, Second Edition, Addison-Wesley Publishing Company, Inc, (1994).

[5] Breu,R., Hinkel,U., Hofmann,C., Klein,C., Paech,B., Rumpe,B. and Thurner,V., Towards a formalization of the unified modeling language. ECOOP97 procs., Lecture Notes in Computer Science vol.1241, Springer, (1997).

[6] Coad,,P. and Yourdon,E., Object Oriented Analysis, Yourdon Press, Englewood Cliffs,NJ, (1991).

[7] Cook,S. and Daniels,J., Let's get formal, Journal of Object-Oriented Programming(JOOP), July-August, (1994).

[8] Duke,R., King,P., Rose,G. y. Smith,G., The Object-Z specification language, T.Korson, V.Vaishnavi and B.Meyer, editors, Technology of Object-Oriented Languages and Systems:TOOLS 5. Prentice Hall, (1991).

[9] Evans,A., France,R., Lano,K. and Rumpe, B., Developing the UML as a formal modeling notation, UML98 Beyond the notation, Muller and Bezivin editors, Lecture Notes in Computer Science 1618, Springer-Verlag, (1998).

[10] Evans,A., France,R., Lano,K. and Rumpe,B., Towards a core metamodelling semantics of UML, Behavioral specifications of businesses and systems, H,Kilov editor, , Kluwer Academic Publishers, (1999).

[11] France,R., Bruel,J. and Larrondo-Petrie. An integrated

object-oriented and formal modeling environment, Journal of Object Oriented Programming (JOOP), 10(7), (1997).

[12] France, R. and Rumpe, B. editors, Proceedings of the UML'99 conference, Beyond the Standar, Colorado, USA, Lecture Notes in Computer Science 1723, Springer-Verlag (1999).

[13] Gamma, Helm, Johnson and Vlissides, Design Patterns, Addison –Wesley Publishing Company (1994).

[14] Geisler, R., Klar M. and Pons, C., Dimensions and Dichotomy in Metamodeling, In proc. of Third BCS-FACS Northern Formal Methods Workshop, Ilkley, UK, September 1998, Series in Computing, Springer-Verlag. (1998).

[15] Goldsack,S. and Kent,S., Formal Methods and Object Technology", Chapter 3: LOTOS in the Object-oriented analysis process. Editors S.J. Goldsack, S.J.H. Kent. Serie FACIT, Springer-Verlag, (1996).

[16] Harel, David, Kozen, Dexter and Tiuryn, Jerzy. Dynamic Logic. To appear. (2000)

[17] Jacobson, I, Booch, G, Rumbaugh, J. The Unified Software Development Process, Addison Wesley. ISBN 0-201-57169-2 (1999)

[18] Jungclaus,R., Saake,G., Hartmann,T., Sernadas,C., TROLL- a language for o-o specifications of information systems, ACM Transactions on IS, vol.14 no.2. (1996).

[19] Jones,C., Systematic software construction using VDM. Prentice Hall, (1990).

[20] Kesim, F. and Sergot, M.. A logic programming framework for modeling temporal objects, IEEE Transactions on knowledge and data engineering, vol.8,no.5, (1996).

[21] Kifer,M. and Lausen,G., F-Logic: a higher order language for reasoning about objects, inheritance and scheme. Proceedings of the ACM SIGMOD symposium on principles of database systems, SIGMOD RECORD, Vol.18, No.6, (1990).

[22] Kim, S. and Carrington,D., Formalizing the UML Class Diagrams using Object-Z, proceedings UML'99 Conference, Lecture Notes in Computer Sciencie 1723, (a) second part of the paper, (b) first part of the paper. (1999).

[23] Lano,K., Z++, An object-oriented extension to Z. In John Nicholls, editor, Z user workshop, Oxford 1990, Workshops in Computing, Springer Verlag, (1991).

[24] Lano,K., and Biccaregui,J., Formalizing the UML in Structured Temporal Theories, Second ECOOP Workshop on Precise Behavioral Semantics, TUM-I9813, Technische Universitat Munchen, (1998).

[25] Meseguer,J., Winkler,T., Parallel Programming in Maude. Proceedings of Research Directions in High Level Parallel Programming Languages. France, (1991).

[26] Moreira,A. and Clark,R., Combining Object_Oriented Analysis and Formal Description Techniques, In 8th ECOOP Conference, Procs. Lecture Notes in Computer Science 821, Springer, (1994).

[27] Muller. P. and Bezivin. J. editors, Proceedings of the UML'98 conference, Beyond the notation, Mulhouse, France, Lecture Notes in Computer Science 1618, Springer-Verlag (1998).

[28] Pastor,O. and Ramos,I., Oasis 2.2 : A Class-Definition Language to Model Information System Using an Object-Oriented Approach". SPUPV-95.788, Universitat P. de Valencia. (1996).

[29] Pons,C., Baum,G., Felder,M., Integrating object-oriented model with object-oriented meta-model into a single formalism, Second ECOOP Workshop on Precise Behavioral Semantics, European Conference on Object-oriented Programming, Brussels, Belgium, LNCS, (1998).

[30] Pons,C., Baum,G., Felder,M., Foundations of Object-oriented modeling notations in a dynamic logic framework, Fundamentals of Information Systems, Chapter 1, T.Polle,T.Ripke,K.Schewe Editors, Kluwer Academic Publisher, (1999).

[31] Pons,C., Ph.D Thesis, Faculty of Science, University of La Plata, Buenos Aires, Argentina, http://www-lifia.info.unlp.edu.ar/~cpons/ (1999)

[32] Reggio,G. and Larosa,M., A graphic notation for formal specification of dynamic systems, proceedings of FME97, Lecture Notes in Computer Science 1313, Springer.(1997).

[33] Rumbaugh,J., Blaha,M., Premerlani,W., Object Oriented Modeling and Design, Prentice Hall, (1991).

[34] Shlaer,S. and Mellor,J., Object Oriented Systems Analysis: Modeling the World in Data, Yourdon Press Computing Series, Yourdon Press, Englewood Cliffs, NJ, (1988).

[35] Spivey,M., The Z notation: a reference manual. Prentice Hall, Englewood Cliffs, NJ, Second edition, (1992).

[36] UML 1..3, Object Management Group, The Unified Modeling Language (UML) Specification – Version 1.3, in www.omg.org, (1999).

[37] Waldoke, S., Pons,C., Paz Mezzano,C. and Felder,M., A Formal Approach to Practical Object Oriented Analysis and Design, Procs of Argentinean Symposium on Object Orientation, Buenos Aires, (1998).

[38] Weber,M. "Combining Statecharts and Z for the Design of Safety-Critical Control Systems", Proceedings of Third International Symposium of FME96. Oxford (1996).

[39] Wieringa,R. and Broersen,J., Minimal Transition System Semantics for Lightweight Class and Behavior Diagrams, In PSMT Workshop on Precise Semantics for Software Modeling Techniques, Technische Universitat Munchen, Report TUM-I9803, (1998).

[40] Cibrán,M., Mola,V., Pons,C.,Russo,W., Building a bridge between the syntax and semantics of UML Collaborations ECOOP'2000 Workshop on Defining Precise Semantics for UML. Cannes/Sophia-Antipolis, France, 12-16 June 2000.