UML as a Heterogeneous Multiview Notation Strategies for a Formal Foundation

Egidio Astesiano – Gianna Reggio
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova – Italy
email {astes, reggio}@disi.unige.it

On UML

UML is a semi-formal notation, in the sense that it has a rather precise syntax (including well-formedness conditions) but an informal semantics (just a natural language description). The task of its formalization is not trivial and poses new problems, since UML has some relevant and novel features with respect to the existing specification formalisms. Let us single out our personal view of some of those features.

Multiview The UML model of a system consists of many different diagrams, each one describing a view either of the system or of some of its parts. For example, a class diagram describes the static structure of the system, while a state diagram describes the local reactive behaviour of a system component (an object of some class), a sequence or collaboration diagram describes some mutual interactions between some components of the system, and so on. It may happen that the functionality and some constraints on an operation of a class are in the class diagram, its local behaviour is in a state diagram, its interactions with the objects of another class are in a collaboration diagram, and those with the objects of a third different class are in a sequence diagram.

Heterogeneous The various kinds of UML diagrams use very different notations, as a variant of entity relationships, state charts, message sequence charts, a kind of Petri nets (activity diagrams), et cetera.

Extendable UML offers some ways to the users to extend the kinds of the elements used in the diagrams, as stereotypes, tagged values and constraints. Moreover the language used for the constraints is not fixed; a particular one, OCL, has been defined but its use is not mandatory and can be replaced by other languages, including also natural language text.

Notation UML is only a notation and not a method (see in [?] a detailed discussion on the difference between notation/formalism and method); thus it can be used in different ways by different methods, and a method may consider only parts of UML or different parts in different phases of the development process (see, e.g., [?]).

The UML documentation makes explicit the first two points; indeed such notes treat each kind of diagrams separately and do not consider their mutual relationships. For example well-formedness conditions and informal semantics do not cover groups of diagrams and just consider single diagrams in isolation.

Why a Formal Foundation for UML

For us to give UML a "formal foundation" means to provide a formal semantics for any of the UML diagrams in an integrated way, taking into account also their mutual relationships; clearly it may happen that only a part of UML can be formalized, or better can be formalized in a sensible way.

Below we present a detailed list of reasons for establishing a formal foundation for UML.

- To make precise, complete, consistent the informal semantics of UML; thus also to extend the well-formedness conditions and the informal semantics to cover groups of related diagrams. In this way UML would be really a standard, in the sense that there would be an unambiguous description allowing to say whether an interpretation of UML (by a software development method, by a tool) is correct or not.
- To find which parts of UML cannot be formalized in a sensible way, so that their meaning cannot be really fixed; and then discuss if they are useful even if informal and still cannot originate any misunderstanding (perhaps the Use Case Diagrams falls in this category).
- Perhaps to simplify the various UML notations by finding parts that can be dropped, because they are redundant, not relevant or constructs whose precise informal definition is too complex to be effectively presented and used.
- To offer the basis for building and validating software tools for UML, also allowing to know what can be done automatically and what not.
- To offer a background for extending UML with some formal specification language for the constraints (e.g., Z or the new common algebraic language CASL [**?**]).

It is important to stress that in our opinion the formalization of UML should remain fully transparent to the users; they do not need to be aware of the formal semantics, but they should have a revised version of the official UML documentation encompassing what expressed by the formal semantics.

Clearly the formal semantics of UML may also help build and assess software development methods using such notation.

For us the above reasons are all valid, but we want to report, for the sake of discussion, a quite different opinion. Since UML can be seen as "a group of heterogeneous semi-formal notations", different methods could use the same diagrams intended with different meanings; in this case each method should establish the relationships among them and say how, when and for what to use the various diagrams. For example a method may enforce that all the executions of the objects of a class should be those described by sequence diagrams, another one may just say that they may have at least such executions, and a third one may use some sequence diagrams to describe prohibited executions; a method may decide that each use case corresponds to an operation of a special main object in the system, and so on.

For someone this great freedom is a reason of the success of UML (and of similar semi-formal notations), because each one can tailor the notation as she/he prefers and software tools may just support a very particular variant. Only the graphic syntax is really standard.

Some Strategies

A UML description of the model of a system (specification form now on) consists of a set of diagrams of different types, where each diagram provides a view of either a part of the system or of the whole system (static structure, internal activity of a component in isolation, ...).

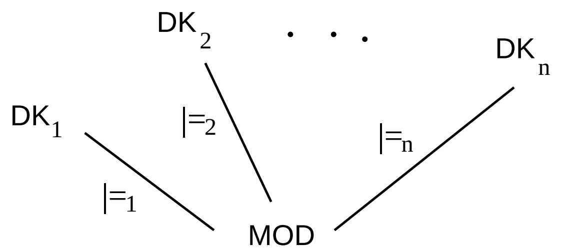
Let us first group the UML diagrams into a family of sets, DK_i $(i = 1, \ldots, n)$, each one correspoding to a kind of diagrams.

Assume to have a UML specification of a system
$\mathsf{SPEC} = \mathsf{D}_1, \ldots, \mathsf{D}_k$;
its semantics is roughly
$\mathrm{Sem}(\mathsf{SPEC}) = \{M \mid M$ agrees with the view represented by $\mathsf{D}_j, j = 1, \ldots, k\}$
where M is a mathematical structure formally representing a system.

To precisely define this semantics we need

- a class MOD of structures formally representing the systems which can be modelled with UML;
- for each kind of UML diagrams DK_i $(i = 1, \ldots, n)$, a binary relation
  $\models_i \subseteq \mathsf{DK}_i \times \mathsf{MOD}$.

Then
$\mathrm{Sem}(\mathsf{SPEC}) = \{M \in \mathsf{MOD} \mid M \models_j \mathsf{D}_j \; j = 1, \ldots, k\}$
where for each j, $\models_j$ is the validity relation for the type of D_j.



If it happens that $\mathrm{Sem}(\mathsf{SPEC}) = \emptyset$, then SPEC is inconsistent, and so it contains some errors; the most probable errors would be inconsistencies between state and sequence/collaboration diagrams, or among different sequence/collaboration diagrams.

On the basis of this general schema, let us discuss at least two different strategies for tackling this big task.

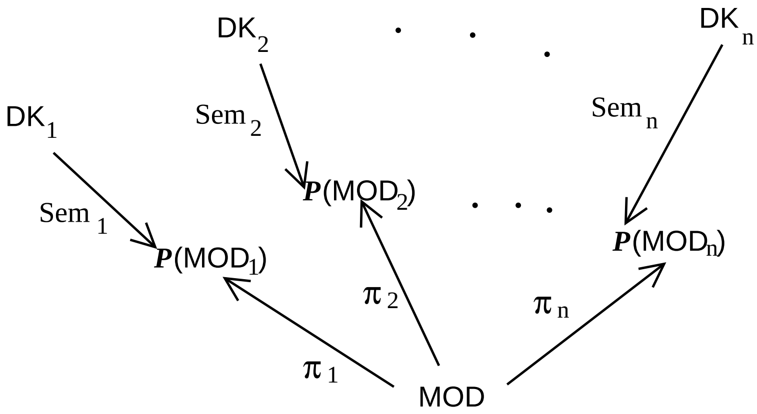
Combinatorial approach We give for each kind of diagrams, say DK_i, a semantics using formal models able to express the system views that they represent,
$\mathrm{Sem}_i \colon \mathsf{DK}_i \to \mathbf{P}(\mathsf{MOD}_i)$.

This initial step is reasonably easy, relying on results present in the literature; if we cannot do that for some kind of diagrams in a sensible way, then they should remain only informal.

Then we define the elements of MOD by joining together elements of MOD_1, ..., MOD_n respectively, and give a family of projection operations from MOD to MOD_i, say π_i

Finally, given $M \in \mathsf{MOD}$

$$M \models_i \mathsf{D}_i \ \text{ iff } \ \pi_i(M) \in \mathrm{Sem}_i(\mathsf{D}_i).$$



Translational approach We give for each kind of diagrams, say DK_i, a translation into some existing specification formalism,

$\mathbf{TR}_i$: $\mathsf{DK}_i \to \mathsf{FORM}_i$.

As before, if we cannot do that for some kind of diagrams, they should remain only informal.

Then we find a basic specification formalism FORM powerful enough with its semantics

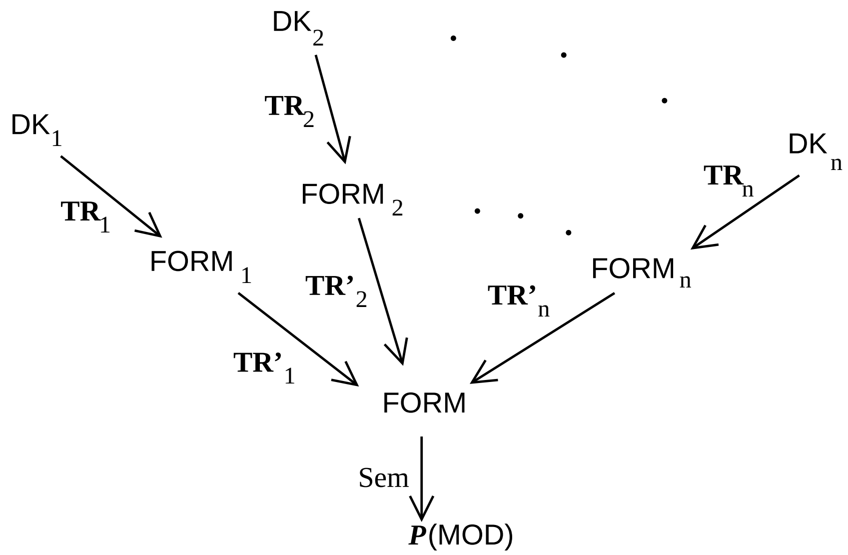
Sem: $\mathsf{FORM} \to \mathbf{P}(\mathsf{MOD})$

(e.g., many-sorted second-order logic), and for each $i = 1, \ldots, n$ we give a translation of FORM_i into FORM

$$\mathbf{TR'}_i\colon \mathsf{FORM}_i \to \mathsf{FORM}.$$

Finally, given $M \in \mathsf{MOD}$

$$M \models_i \mathsf{D}_i \ \text{ iff } \ M \in \mathrm{Sem}(\mathbf{TR'}_i(\mathbf{TR}_i(\mathsf{D}_i))).$$



The above two last diagrams show also how to decompose modularly the big task of formalizing UML; however that does not mean that we can just start by working on formalizing the diagrams in DK_i, forgetting that their semantics/translation will be part of a more complex framework: the models/formalisms used in the various cases cannot have an extremely different nature and we cannot choose them by looking only at one kind of diagrams.

Acknowledgement Some ideas illustrated here have been presented at a meeting of the "Reactive Systems Group" of the COFI Initiative [**?**], last July at Dagsthull; we acknowledge the benefit of the discussions during that meeting.