must be able to represent both levels of information (model and meta-model) in an integrated way, in order to allow the specification of interconnections between the two different levels.

\[ \text{spec}_{\text{UML}} \rightarrow \text{semantics} \rightarrow \text{set of all well-formed UML models} \]

\[ \text{spec}_{\text{UML}} + \text{instantiation axioms} \rightarrow \text{semantics} \rightarrow \text{The UML model of a particular system} \]

\[ \text{spec}_{\text{UML}} = \text{spec}_{\text{UML}} + \text{spec}_{\text{SYS}} + \text{axioms}_{\text{UML-SYS}} \]

\[ \text{formal spec}_{\text{SYS}} \rightarrow \text{semantics} \rightarrow \text{game pieces} \]

*Figure b: A two-layers conceptual model*

Our approach is described in 'figure b'. It consists in the integration of all the information into a single conceptual model based on dynamic logic. This conceptual model combines the information proceeding from:

1. \( \Sigma_{\text{UML}} \) (the formal specification of the meta-model)
2. \( \text{axioms}_{\text{SYS}} \) (the instantiation axioms)
3. \( \Sigma_{\text{SYS}} \) (the formal specification of the model)

The meta-model specification is written once for all, the instantiation axioms and the model specification are obtained by the application of a semi-automatic transformation method that creates a modal logic specification from a UML model. In section 2.1 we describe the formalization \((\Sigma_{\text{UML}}, \phi_{\text{UML}})\) of the meta-model, and in section 2.2 we give a sample of the formal specifications \((\Sigma_{\text{SYS}}, \phi_{\text{SYS}})\) of the model level and the instantiation axioms.

### 2.1 Formal representation of the meta-model level

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 states diagrams that show the intra-object dynamics in terms of state transitions.
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 impact on other diagrams, if a modification is made to one diagram.

Object-oriented systems evolve over their life cycle of for a variety of reasons. One of the most common forms of evolution involves the extension of an existing schema by addition of new classes of objects or the addition of attributes to the original objects. Sometimes class structures are reorganized even when the set of objects is unchanged. At the other extreme, a class reorganization might reflect not only the extension and reclassification of existing objects, but also structural changes (other than addition of attributes) in the original objects. Another form of evolution involves the modification of sequence diagrams, state charts, and so on. This kind of evolution is more difficult to deal with (and are out of the scope of our current work).

In this section we give a formal specification of the elements in the meta-model level using a formal language based on Dynamic Database Logic (DDL) [Spruit et al. 93, Wieringa et al. 94, Wieringa and Broersen 98] (see appendix 1 for details about DDL). This specification consists of a signature $\Sigma_{\text{UML}} = (S_{\text{UML}}, F_{\text{UML}}, P_{\text{UML}}, E_{\text{UML}})$ and a formula $\phi_{\text{UML}}$ over $\Sigma_{\text{UML}}$. The elements of the initial algebra denoted by the specification are meta-model elements, such as classes and relationships. The transition relations between possible worlds represent modifications on the system definition, for example adding a new class, modifying an existing class, etc.

The formula $\phi_{\text{UML}}$ is the conjunction of two disjoint sets of formulas, $SE$ and $DE$ 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). The static rules are specified upon the schema elements. 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 events. Those sets can be enriched by defining new properties that developers want to verify.

For space limitations only a brief part of the specification will be shown (for a more detailed presentation, see [Pons 97]).

Let ClassName, AttrName, OperName be sets of class names, attribute names and operation names respectively.

**Specification of Attribute**

**Sorts**

- Attribute

**Taxonomy**

- Attribute $\leq$ ModelElement

**Updatable functions**

- name : Attribute $\rightarrow$ AttrName
- type : Attribute $\rightarrow$ ClassName

**Events**

- updateName: Attribute, AttrName $\rightarrow$ Event
- .................etc..............

**Axioms**

- va: Attribute
- [updateName (a,n)] name(a)=n
- .................etc..............

**End spec.**

**Specification of Class**

**Sorts**

- Class

**Taxonomy**

- Class $\leq$ ModelElement

**Updatable functions**

- name : Class $\rightarrow$ ClassName
- attributes: Class $\rightarrow$ List-of-Attribute
- operations: Class $\rightarrow$ List-of-Operation