multiplicity: Class $\rightarrow$ Interval

**Updatable predicates**

*IsAbstract*: Class

**Events**

- `addAttribute`: Class, Attribute $\rightarrow$ Event
- `addOperation`: Class, Operation $\rightarrow$ Event
- `deleteAttribute`: Class, Attribute $\rightarrow$ Event
- `deleteOperation`: Class, Operation $\rightarrow$ Event
- `changeMultiplicity`: Class, Interval $\rightarrow$ Event
- `concretize`: Class $\rightarrow$ Event

**Axioms** $\forall c$: Class

1. Formulas specifying static properties of classes:
   - There is not two attributes with the same name'
   - $\forall c$:ClassDiagram $\rightarrow$ ClassDiagramName

**Specification of ClassDiagram**

```plaintext
imports Class, Association, Aggregation, Generalization
```

**Sorts**

```plaintext
ClassDiagram
```

**Taxonomy**

```plaintext
ClassDiagram $\leq$ ModelElement
```

**Nonupdatable functions**

```plaintext
name : ClassDiagram $\rightarrow$ ClassDiagramName
```

**Updatable functions**

```plaintext
classes: ClassDiagram $\rightarrow$ List-of-Class
associations: ClassDiagram $\rightarrow$ List-of-Association
aggregations: ClassDiagram $\rightarrow$ List-of-Association
generalizations: ClassDiagram $\rightarrow$ List-of-Generalization
```

**Updatable predicates**

```plaintext
IsA : Class $\times$ Class
PartOf: Class $\times$ Class
Disjoint: Class
```

**Events**

- `addClass`: ClassDiagram, Class $\rightarrow$ Event
- `addGeneralization`: ClassDiagram, Generalization $\rightarrow$ Event
- `etc: $\ldots$`

**Axioms** $\forall d$: ClassDiagram, $\forall c_1, c_2$: Classes, etc...

1. Formulas defining the meaning of the updatable predicates.

- `Disjoint(c_1, c_2) $\iff$ $\exists g$: Generalization:
  - $(\exists v$: OptionTree: $(v \in$ options$\{g\}) \land$ genType$(v) = disjoint \land g_1 \in$ options$\{v\}) \land g_2 \in$ options$\{v\}$
  - $\land root(g_1) = name(c_1) \land root(g_2) = name(c_2)$
  - $\lor \exists c_1$: Class $(IsA(c_1, c_2) \land Disjoint(c_1, c_2))$
  - $\lor \exists c_2$: Class $(IsA(c_2, c_1) \land Disjoint(c_1, c_2))$

2. Formulas specifying static properties of class diagrams (Schema Checking):

- In an class diagram the class names are unique'
- $(c_1 \in$ classes$\{d\}) \land c_2 \in$ classes$\{d\}$ and $name(c_1) = name(c_2)$ $\rightarrow$ $c_1 = c_2$
- Classes which are involved in relations are defined in the model'
- $\forall p$: Participant $(a_i \in$ associations$\{d\} \land p \in$ participants$\{a_i\} \land n = name(p))$
  - $\rightarrow \exists c$: Class $(c \in$ classes$\{d\} \land name(c) = n)$
- Generalization is antisymmetric'
- $(IsA(c_1, c_2) \land IsA(c_2, c_1)) \rightarrow c_2 = c_1$
- Multiple inheritance does not lead to name conflicts'
- $(\exists c_1, c_2, c_3$: Class $(IsA(c_1, c_2) \land IsA(c_2, c_3) \land c_3 \neq c_2$ and
  - $\exists a$: Attribute $(a \in$ attributes$(c_2) \land a \neq$ attributes$(c_1)$)
2.2 Formal representation of the model level

The elements in the model level are specified in the context of the Metamodel specification. To do this we impose a few restrictions on signatures, and thus on the general language. The resulting language is called Model and Meta-Model Language (MMM).

An object signature $\Sigma_{sys}$=( $S$, $\leq$, $F$, $P$, $E$ ) is a Dynamic Database Logic signature with the following special features:

- It includes the signature $\Sigma_{IM}$. This means, for instance, that there is a set of distinguished sorts $S_{IM} \subseteq S$, these sorts represent the structure of the system. For example, Class represents a set whose elements are classes (it is usually called a metaclass).

- There is a sort name Object, such that Object $\in S$, not Object $\leq$ $S$ and not $S \leq$ Object for all sort $S$ different from Object. This sort represents object identifications. The following functions are defined over Object: firstId:Object $\rightarrow$ Object, lastId:Object $\rightarrow$ Object.

- Class is a universal sort such that $c$ $\in$ Class for all $c$ $\in$ $S$ different from Object.

- There is an updatable function symbol $my\_class$:Object $\rightarrow$ Class, $my\_class \in F$. This function associates each object identity to the entity denoting the class it belongs to.

- There is an updatable predicate symbol, Exists:Object. In any state of the world, the existence set of a class is the set of class instances that exist in that state. The extension of a class (the set of all possible objects in the class) is always the same set of instances, independently from the state of the world, but the existence set varies with the state of the world. The predicate $Exists$ defines the set of existing objects in each state.

- There is an event symbol, new:Class $\rightarrow$ Event. The event term $new(C)$ denotes the creation of a new instance of the class denoted by the term $C$.

- For all sort $C$ such that $C$ $\subseteq$ Class, there is a unique nonupdatable (constant) function symbol, $c$: $C$. Intuitively, the constant $c$ denotes the class $C$ itself (this kind of model is similar to the structure of Classes and Metaclasses in Smalltalk).

- There is an infix event symbol 'dot', $\cdot$: Object,Message $\rightarrow$ Event. The formula $[(obj\_term \cdot message\_term)]$ Pred_term means that immediately after the object denoted by $obj\_term$ receives the message denoted by $message\_term$, the Pred term is true.

- Let Value be the set of primitive values (such as Integer and Boolean values). There is an updatable (polymorphic) infix function symbol 'dot', $\cdot$: Object,AttributeName $\rightarrow$ (Object $+$ Value), that binds each object to the values of its attributes.

- There is an updatable predicate symbol, Enabled: Event. This predicate indicates which events are allowed to happen in each state.

A specification of an object-oriented system is a pair ($\Sigma_{sys}$, $\phi_{sys}$), where $\phi_{sys}$ is a formula over $\Sigma_{sys}$ that defines the semantics of the system.

Figure 3 shows an example of a formal specification using this version of dynamic logic (the definition of some implicit, for example Object, $\cdot$, my_class).