Forms of Containment and Inheritance in Programming Languages and Databases

Kenneth Baclawski
College of Computer Science
Northeastern University
Boston, MA 02115

Outline

- Examples.
- The `MU` Model.
- Components.
- Instance and Object Inheritance.
- Properties of Inheritance.
- Uniform Treatment of Inheritance.

The Many Forms of Inheritance and Containment

- Component: field, attribute, instance variable, member,. . .
- Instance of a type or class.
- Taxonomy: classification hierarchy.
- Single inheritance or simple public derivation.
- Multiple inheritance or virtual public derivation.
- Multiple independent inheritance.
- Prototype or extension inheritance.
- Private inheritance.

The MU Model

1. THE UNIVERSE. A *database* is a set of triples (o, a, p), where o and p are *objects* and a is an *attribute*. The universe of all possible objects is denoted $\mathbf{O}$ and the universe of all possible attributes is denoted $\mathbf{A}$.

2. TYPES. The organization of objects into types is represented using a distinguished attribute called `instance_of` or $a_1 \in \mathbf{A}$. An object o is an instance of a type t when (o, a_1, t) is in the database.

3. INHERITANCE. Inheritance is represented with a distinguished attribute called `is_a` or $a_0 \in \mathbf{A}$.
   When (o, a_0, p) is in the database, one says that o is *derived* from p.

4. SCHEMA. There is a function

$$\texttt{level} : \mathbf{O} \longrightarrow \{0, 1, 2, \ldots\}$$

that stratifies the universe of all objects into "ordinary objects," "types," "meta-types," and so on. The attribute a_1 relates objects on adjacent levels. All other attributes relate objects on the same level.

5. TYPE CONSTRAINT. If o is an instance of t, $t \xrightarrow{a} u$ and $o \xrightarrow{a} p$, then p must be an instance of u.

6. TYPE SPECIFICATION. If $o \xrightarrow{a} p$, then o is an instance of a type t and p is an instance of a type u for which $t \xrightarrow{a} u$.

7. ACYCLICITY. The directed graph determined by a_0 has no cycles.

Components

Proper attributes (i.e., those other than a_0 or a_1) represent the concept of a *component*. This includes, for example, the C++ concepts of data member, function member, operator, conversion, constructor, destructor, and so on.

Although attributes of the `mu` model are multivalued by default, they can be constrained to be single-valued, or to satisfy cardinality constraints (also called *functionality constraints*). The most important of these are:

- MANY-TO-ONE (SINGLE-VALUED). Every instance of t has at most one a-value.
- ONE-TO-MANY (CONTAINMENT). No two instances of t have the same a-value.
- ONE-TO-ONE. Both one-to-many and many-to-one.
- MANY-TO-MANY. No constraint is imposed.
- MANDATORY. Every instance of t has at least one a-value.

Instance and Object Inheritance

There are two distinct ways of defining inheritance in the `mu` model.

8. Instance inheritance. Every instance of a subtype is also an instance of any supertype.
9. Object inheritance. An instance of a subtype is derived from a unique instance of each base type.

Instance inheritance is the intuitive concept of inheritance: this is how one conceptualizes the meaning of the term. Object inheritance is the actual concept of inheritance as defined and used in object-oriented systems.

The axioms for the `mu` model were given for the case of instance inheritance. They have to be modified somewhat for object inheritance. Other important constraints also depend on which inheritance concept is used.

9. UNIQUELY TYPED: INSTANCE INHERITANCE. Every object on level 0 is an instance of exactly one "most specific type."

10. UNIQUELY TYPED: OBJECT INHERITANCE. The attribute a_1 is single-valued from level 0 to level 1.

Properties of Inheritance

- Substitutability. An instance of a subtype can be regarded as an instance of any supertype.
- Late Binding. One of the unique features of `nu&` is that late binding is split into two independent steps: recollection and name resolution.
  - Recollection. The mechanism whereby a substitution (or succession of substitutions) is undone is called *recollection*.
  - Name Resolution. A name server is required for managing the many-to-many relationship between attributes and their names. This relationship also depends on context.

- Performance Independence. If a feature is not used, then it has no effect on performance.

- Transitivity. If s is derived from t and t is derived from u, then s is derived from u.

- Independence of Base Objects. A base object of a derived object can be regarded accessed as an object on its own, not just as an object obtained by substitution from the derived object.

Uniform Treatment of Inheritance

The various forms of inheritance can be characterized within the `mu` model using cardinality and other constraints.

- MULTIPLE INHERITANCE. The attribute a_0 is transitive and is one-to-many on level 0.
- SINGLE INHERITANCE. The attribute a_0 is one-to-one on level 0 (and many-to-one on level 1).
- PROTOTYPE INHERITANCE. The attribute a_0 is transitive and is many-to-many on level 0.
- MULTIPLE INDEPENDENT INHERITANCE. The attribute a_0 has no transitivity and is one-to-many on level 0.
- TAXONOMY. The attribute a_0 is one-to-one on level 0, and every type that has subtypes is covered by them.
- PRIVATE INHERITANCE. This is represented using an ordinary attribute since private inheritance does not support substitutability and late binding. Those features that are supported can be handled entirely within the name server.

Conclusion

We have given a single framework for the many diverse forms of containment and inheritance. This supports research in this area by allowing one to compare these forms directly with one another in a single system. The `mu` model is a simple, yet powerful data model within which one can give natural representations for not only inheritance mechanisms but also the many other mechanisms that are required by an object-oriented system.