A reflective class-based system

When compiling an object, Logtalk distinguishes prototypes from instance or classes by examining the object relations. If an object instantiates and/or specializes another object, then it is compiled as an instance or class, otherwise it is compiled as a prototype. A consequence of this is that, in order to work with instance or classes, we always have to define root objects for the instantiation and specialization hierarchies (however, we are not restricted to a single hierarchy). The best solution is often to define a reflective class-based system [Maes 87], where every class is also an object and, as such, an instance of some class.

In this example, we are going to define the basis for a reflective class-based system, based on an extension of the ideas presented in [Cointe 87]. This extension provides, along with root objects for the instantiation and specialization hierarchies, explicit support for abstract classes [Moura 94].

Defining the base classes

We will start by defining three classes: `object`, `abstract_class`, and `class`. The class `object` will contain all predicates common to all objects. It will be the root of the inheritance graph:

```
:- object(object,
    instantiates(class)).

    % predicates common to all objects

:- end_object.
```

The class `abstract_class` specializes `object` by adding predicates common to all classes. It will be the default meta-class for abstract classes:

```
:- object(abstract_class,
    instantiates(class),
    specializes(object)).

    % predicates common to all classes

:- end_object.
```

The class `class` specializes `abstract_class` by adding predicates common to all instantiable classes. It will be the root of the instantiation graph and the default meta-class for instantiable classes:

```
:- object(class,
    instantiates(class),
    specializes(abstract_class)).

    % predicates common to all instantiable classes

:- end_object.
```

Note that all three objects are instances of class `class`. The instantiation and specialization relationships are chosen so that each object may use the predicates defined in itself and in the other two objects, with no danger of method lookup endless loops.

Summary

- An object that does not instantiate or specialize other objects is always compiled as a prototype.

- An instance must instantiate at least one object (its class). Similarly, a class must at least specialize or instantiate other object.

- The distinction between abstract classes and instantiable classes is an operational one, depending on the class inherited methods. A class is instantiable if inherits methods for creating instances. Conversely, a class is abstract if does not inherit any instance creation method.

XHTML + CSS Last updated on: November 16, 2005