An Agent-Based Library to Make the Visitor Pattern Unnecessary

The agent mechanism permits a much better solution to the problem addressed somewhat clumsily by the Visitor pattern: adding operations to existing types, without changing the supporting classes. The solution is detailed in Meyer and Arnout (2006) and available through an open source library available on the download site of the ETH Chair of Software Engineering (ETH Zurich, Chair of Software Engineering, at *http://se.ethz.ch*).

The resulting client interface is particularly simple. No change is necessary to the target classes (`PUDDING`, `CONTRACT`, and such): there is no more `accept` feature. One can reuse the classes exactly as they are, and accept their successive versions: there is no more explosion of visitor classes, but a single `VISITOR` library class, with only two features to learn for basic usage, `register` and `visit`. The client designer does not need to understand the internals of that class or to worry about implementing the Visitor pattern, but only needs to apply the basic scheme for using the API:

1. Declare a variable representing a visitor object, specifying the top target type through the generic parameter of `VISITOR`, and create the corresponding object:

   ```
   pudding_visitor: VISITOR [PUDDING]
   create pudding_visitor
   ```

2. For every operation to be executed on objects of a specific type in the target structure, register the corresponding agent with the visitor:

   ```
   pudding_visitor.register (agent fruit_salad_cost)
   ```

3. To perform the operation on a particular object—typically as part of a traversal—simply use the feature `visit` from the library class `VISITOR`, as in:

   ```
   pudding_visitor.visit (my_pudding)
   ```

That is all there is to the interface: a single visitor object, registration of applicable operations, and a single visit operation. Three properties explain this simplicity:

- The operations to be applied, such as `fruit_salad_cost`, would have to be written regardless of the architecture choice. Often they will already be available as routines, making the notation **agent** `fruit_salad_cost` possible; if not—especially if they are very simple operations—the client can avoid introducing a routine by using inline agents. In either case, there is no need for the spurious `T_visit` routines.
- It seems strange at first that a single `VISITOR` class, with a single `register` routine to add a visitor, should suffice. In the Visitor pattern solution the calls `t.accept (v)`, the target `t` identified the target type (a particular kind of pudding), but here `register` does not specify any such information. How can the mechanism find the right operation variant to apply (the cost of a fruit salad, the cost of a flan)? The answer is a consequence of the reflective properties of the agent mechanism: an agent object embodies all the information about the associated feature, including its signature. So **agent** `fruit_salad_cost` includes the