mark suffices, as in **agent** x.f (?, v). This also indicates that the original forms with all arguments open, **agent** f and **agent** x.f, are abbreviations for **agent** f (?, ?) and **agent** x.f (?, ?).

The call mechanism applies dynamic binding: the version of f to be applied will, as in non-agent calls, depend on the dynamic type of the target.

If f represents a query rather than a command, you can get from the corresponding agent the result of a call to f by using item instead of call, as in a.item ([x, u, v]) (which performs a call and returns the value of its result); or you can call call and then access a.last_result, which, in accordance with the command-query separation principle, will return the same value, with no further call, in successive invocations.

For more advanced uses, rather than basing an agent on an existing feature f, it is also possible to write agents inline, as in editor_window.set_mouse_enter_action (**agent do** text.highlight **end**), illustrating a typical use for graphical user interfaces, the basic style for event-driven programming in EiffelVision library. Inline agents provide the same mechanism as lambda expressions in functional languages: to write operations and make them directly available to the software as values to be manipulated like any other "first-class citizens."

More generally, agents enable the object-oriented framework to define higher-level functionals just as in functional languages, with the same power of expression.

Scope of Agents

Agents have turned out to be an essential and natural complement to the basic object-oriented mechanisms. They are widely used in particular for:

- Iteration: applying a variable operation, naturally represented as an agent, to all elements in a container structure.
- GUI programming, as just noted.
- Mathematical computations, as in the example of integrating a certain function, represented by an agent, over a certain interval.
- Reflection, where an agent provides properties of features (not just the ability to call them through call and item) and, beyond them, classes.

Agents have proved essential to our investigation of how to replace design patterns by reusable components (Arnout 2004; Arnout and Meyer 2006; Meyer 2004; Meyer and Arnout 2006). The incentive is that while the designer of any application needing a pattern must learn it in detail—including architecture and implementation—and build it from scratch into the application, a reusable component can be used directly through its API. Success stories include the Observer design pattern (Meyer 2004; Meyer 2008), which no one having seen the agent-based solution will ever be tempted to use again, Factory (Arnout and Meyer 2006), and Visitor, as will be discussed next.