found right away their proper place in the object-oriented arsenal; designers have no trouble deciding when they are applicable and when not.

In practice, all nontrivial uses of agents—in particular, the cited pattern replacements—also rely on genericity, inheritance, polymorphism, dynamic binding, and other advanced OO mechanisms. This reinforces the conviction that the mechanism is a necessary component of successful object technology.

> NOTE
>
> **For a differing opinion, see the Sun white paper explaining why Java does not need an agent- or delegate-like facility (Sun Microsystems 1997). It shows how to emulate the mechanism using Java's "inner classes." Although interesting and well-argued, it mostly succeeds, in our view, at demonstrating the contrary of its thesis. Inner classes do manage to do the job, but one can readily see, as in the elimination of the Visitor pattern with its proliferation of puny classes, the improvement in simplicity, elegance, and modularity brought by an agent-based solution.**

Agents, it was noted above, allow object-oriented design to provide the same expressive power of functional programming through a general mechanism for defining higher-order functionals (operations that can use operations—themselves recursively enjoying the same property—as their inputs and outputs). Even lambda expressions find their counterpart in inline agents. These mechanisms were openly influenced by functional programming and should in principle attract the enthusiasm of its proponents, although one fears that some will view this debt acknowledgment as an homage that vice pays to virtue (La Rochefoucauld 1665).

Setting aside issues of syntax, the only major difference is that agents can wrap not only pure functions (queries without side effects) but commands. Ensuring full purity does not, however, seem particularly relevant to discussions of architecture, at least as long as we enforce the command-query separation principle, retaining the principal practical benefit of purity—referential transparency of expressions—without forcing a stateful model into the artificial stranglehold of stateless models.

Agents bring the final touch to object technology's contribution to modularity, but they are only one of its elements, together with those sketched in this discussion and a few more. The combination of these elements, going beyond what the functional approach can offer, makes object-oriented design the best available approach to ensure beautiful architecture.

Acknowledgments

I am grateful to several people who made important comments on drafts of this contribution; obviously, acknowledgment implies no hint of endorsement (as is obvious in the case of the functional programming grandees who were kind enough to share constructive reactions without, I fear, being entirely swayed yet by my argument). Particularly relevant were observations by Simon Peyton Jones, Erik Meijer, and Diomidis Spinellis. John Hughes's