Performance and Integrity in the OpenORB Reflective Middleware

Gordon S. Blair[1,2], Geoff Coulson[2], Michael Clarke[2] and Nikos Parlavantzas[2]

[1]Dept. of Computer Science, University of Tromsø, N-9037 Tromsø, Norway.
[2]Distributed Multimedia Research Group, Dept. of Computing, Lancaster University, LA1 4YR, U.K.
E-mail: gordon@cs.uit.no, {geoff, mwc, parlavan}@comp.lancs.ac.uk

Middleware is playing an increasingly central role in the design of modern computer systems and will, we believe, continue to enjoy this prominence in the future. There is, however, a demonstrable need for more *openness* and *flexibility* in middleware [1]. We believe strongly that *reflective middleware* is the right technology to meet these demands. Indeed, there is strong evidence that such platforms are not only significantly more configurable and reconfigurable than conventional platforms, but that they offer better support for software evolution generally [2]. The main goals of OpenORB v2, the system discussed in this extended abstract, are to address what we perceive as the most pressing shortcomings of current reflective middleware platforms. First, *performance*: in the worst case, this needs to be on a par with that of conventional platforms, and in the best case (e.g. in cut-down configurations) it should be significantly *better*. Second, *integrity*: while permitting maximal reconfigurability, it should be possible to control and constrain reconfigurations so that damaging changes are discouraged and/ or disallowed.

The OpenORB v2 architecture is built in terms of a reflective component model. More specifically, we deploy this component model [3] not just at the application level, but also *for the construction of the middleware platform itself*. The component model is language independent, lightweight and efficient, and forms the basis of our goal of high performance. In addition, to address the issue of integrity, we rely heavily on the concept of *component frameworks* (see below). Thus, an instance of OpenORB v2 is some particular configuration of component frameworks/ components; these are selectable at build-time and reconfigurable at run-time (via reflection).

Our component model, called OpenCOM [2], is based on the core of Microsoft's COM (it avoids dependencies on non-core features of COM such as distribution, persistence, security and transactions), but it enhances COM with richer reflective facilities. Most fundamentally, OpenCOM offers a mechanism for run-time *dependency tracking* between components. To this end, we introduce the notion of 'required' interfaces to express the dependency of a component on an external interface, and then define *receptacles* as first class run-time entities that maintain pointer and type information to represent an explicit *connection* between a component and a 'required' interface. We also deploy a standard *run-time*, available in every OpenCOM address space, that maintains a *system graph* of current connections in the address space.

In addition, OpenCOM offers support for introspection and adaptation of component internals through a number of low-level meta-interfaces supported by each component:

1. The *IMetaArchitecture* interface provides access to the component's structure in terms of its internally nested components and their connections (assuming the target component is not primitive);
2. The *IMetaInterface* interface provides meta-information relating to the interface and receptacle types of the component (this can also be used to support dynamic invocation; cf. Java core reflection);
3. The *IMetaInterception* interface enables the dynamic attachment or detachment of interceptors.

The second key technology underpinning OpenORB is an instantiation of the concept of *component frameworks* (CFs) [3]. Each CF focuses on a particular area of functionality; e.g., there are CFs for protocol composition, CFs for thread schedulers and for binding types, and takes responsibility for the maintenance of the *integrity* of that area of the system. CFs exploit domain-specific knowledge and built-in constraints to enforce a desired level of integrity across reconfiguration operations (in terms of both functional and non-functional concerns), and also perform domain specific trade-offs between flexibility and consistency. In OpenORB, CFs are not merely a design concept; rather, they are reified as run-time software entities (packages of components) that support and police components 'plugged into' the CF to ensure that they conform to CF-specific rules and contracts.

In our implementation work, we have confirmed that OpenCOM-plus-CFs supports the construction of ORB functionality that is at least as efficient as conventional object-based ORBs. For example, [2] shows that an OpenORB v2 configuration featuring a CORBA based binding type implementation performs on a par with the popular Orbacus ORB. Furthermore, we have confirmed that the component model scales well in terms of its explicit enumeration of per-component dependencies. This is primarily due to the use of CFs that reduce dependencies by forbidding connections between plug-in components and components outside the CF. In our current implementation, the maximum number of dependencies in any single component is just seven and the average figure is just four. This leaves considerable scope for further reducing the granularity of componentisation that, if carried out with care, should correspondingly increase the ORB's potential for reconfigurability.

In conclusion, we believe that the combination of a reflective component model and the CF-based structuring principle represents a highly promising basis for the construction of configurable and reconfigurable ORBs. While a reflective component model provides a powerful basis for maximal flexibility and reconfigurability, its unconstrained use can easily lead to chaos. The presence of CF-based structuring tempers this expressiveness by imposing domain specific constraints on the reconfiguration process.

References

1. Roman, M., Kon, F., Campbell, R.H., "Reflective Middleware: From the Desk to your Hand", *To appear in* IEEE DS Online, Special Issue on Reflective Middleware, 2001.
2. Blair, G.S., Coulson, G., Andersen, A., Blair, L., Clarke, M., Costa, F., Duran-Limon, H., Fitzpatrick, T., Johnston, L., Moreira, R., Parlavantzas, N., Saikoski, K., "The Design and Implementation of OpenORB v2", *To appear in* IEEE DS Online, Special Issue on Reflective Middleware, 2001.
3. Szyperski, C., "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, 1998.