A Reflective Middleware Approach to the Provision of Grid Middleware

Geoff Coulson, Gordon Blair, Nikos Parlavantzas, Wai Kit Yeung, Wei Cai

Computing Department, Lancaster University (UK)

[geoff, gordon, parlavan, yeungwk, w.cai]@comp.lancs.ac.uk

1. Introduction

In the past few years significant progress has been made in the design and implementation of reflective middleware platforms [Kon,02]—i.e., platforms that, through reflection [Kiczales,91], can be flexibly configured, and run-time adapted/ reconfigured, especially in terms of non-functional properties like timeliness, resourcing, transactional behaviour, and security. Recently, we have initiated a project that investigates applying our previous reflective middleware work to the demanding and novel—for reflective middleware—area of *Grid middleware environments*. In particular, our focus is on the web-services-based approach being adopted in the Open Grid Services Architecture (OGSA) initiative [OGSA,03]. Within this approach we are focusing on communications and resource management issues (we are specifically *not* focusing, for example, on data-Grid or semantic-Grid aspects).

The specific aims of our work are as follows:

- To develop a prototype platform based on our generic middleware technologies, but with OGSA-related functionality and an OGSA-based API (which is, however, extensible using reflection).
- To investigate the integration of generic service offered by non-OGSA environments (e.g. CORBA, .NET, Java RMI/ EJB), into the OGSA programming environment in as transparent as manner as possible.
- To evaluate the prototype platform in the context of ongoing Grid application development at Lancaster.

In this position paper we outline salient characteristics of the Grid middleware environment from our perspective, and discuss how the reflective middleware approach offers the potential for more flexible, and more evolvable Grid infrastructures. The remainder of the paper is structured as follows. First, §2 briefly surveys our previous work on reflective middleware, and §3 surveys OGSA together with the wider field of middleware research. Next, §4 outlines our approach to applying reflective middleware technology in Grid environments. Finally, §5 discusses our results to date and indicates areas of planned future work.

2. OpenOrb and OpenCOM

Lancaster's component-based reflective middleware approach has already been presented in some detail in the literature (e.g., see [Clarke,01], [Coulson,02]); here, we provide as brief an overview as is necessary to make sense of the rest of this paper. Our middleware, called *OpenORB*, is actually more of a framework than an ORB per se. That is, it can be used to define/ configure a range of types of *middleware instances* (e.g. a Grid platform, a web-services platform, a standard CORBA emvironment, real-time CORBA, a pub-sub middleware platform, etc.). Subsequently, middleware instances can be reconfigured (e.g. extended and adapted) at runtime, using reflection. For example, we can accomplish on-line software updates in 7x24 systems, or migrate CPU-intensive functions from a PDA to a dedicated application server as memory availability and connectivity vary. Additionally, OpenORB supports media streams as first-class objects, and, thanks to aggressive application of recent research results on performance optimisation in middleware platforms, performs as well or better than other state of the art commercial or research ORBs [Coulson,02].

OpenORB's internal architecture is notable for the following features:

- A consistent use of *component technology* [Szyperski,98] as the basis of configuration and runtime reconfiguration. Uniquely, we took the approach of building the *core middleware framework itself* in terms of components (using a home-grown component model called *OpenCOM* [Clarke,01]). This approach is a generalisation of the more conventional practice—adopted, e.g., by COM+, Sun's Enterprise JavaBeans, and the CORBA Component Model—in which component technology is only exploited for the construction of applications *on top of* a standard monolithic middleware platform. Our OpenCOM component model is language independent, and efficient enough to support rather fine-grained componentisation.
- The use of *component frameworks* to give structure to component configurations and help maintain system integrity in the face of reconfiguration. Each component framework provides a 'life

support environment' for specialised 'plug-in' component types in a localised domain of middleware functionality (e.g. security policy, concurrency support, message demultiplexing strategies, or a pluggable protocol framework). Furthermore, each component framework imposes domain-specific constraints on the use of the standard OpenORB/ OpenCOM reflective facilities (see below).

OpenCOM's / OpenORB's reflective facilities are provided in terms of three orthogonal meta-models as follows. First, the *architecture* meta-model supports architectural/ structural reflection—it allows the programmer to view the structure of a system/ application as a topological graph of components, and to alter the structure by manipulating this graph. Second, the *introspection* meta-model allows the programmer to discover, at runtime, the types of interfaces a component supports and to dynamically invoke operations on these interfaces (to maintain language-independence, discovery is in terms of IDL interfaces). Third, the *interception* meta-model allows the programmer to add/remove code that is to be transparently executed before or after invocations are made on a particular interface.

3. OGSA and Wider Middleware Research

3.1 The Emergence of OGSA

Following initial offerings such as Globus [Foster,01] and Legion [Grimshaw,99], the *Open Grid Services Architecture* (OGSA) [OGSA,03] is emerging as a 'second generation' distributed computing approach to Grid middleware. OGSA borrows heavily from web-services standards (especially XML, SOAP, and WSDL), and promises a more unified and principled approach to the support of Grid applications. It augments generic web-services specifications by defining a specific abstract notion of 'Grid service', and also defines Grid-specific 'patterns' such as: service factories and registries; naming and referencing conventions for service instances; support for stateful services; soft-state-based garbage collection of service instances; event notification from services; and version management. The international Grid research community is strongly committed to the OGSA initiative (see e.g. [Atkinson,02]).

Despite its perceived centrality, OGSA is still the subject of ongoing development and standardisation, and is far from having crystallised into its final form. In terms of implementation, it is still less developed. Reference implementations are underway (in particular, Argonne Labs are developing a range of Java-based implementations; and C/ Unix, and .NET implementations are planned or underway [Atkinson,02,]). However, none of these implementations have yet been convincingly exercised, optimised or validated in the 'real-world'.

3.2 Wider Middleware Research

In contrast, the wider field of research in distributed systems platforms (middleware) has been evolving over at least 10 years, and has achieved a degree of maturity in the form of standards like RM-ODP and CORBA, and in industry-developed platforms like Java RMI, Enterprise JavaBeans (EJB), DCOM, and the .NET remoting architecture. It is, however, clear that such middleware in its present form is *not* especially well-suited to the support of Grid applications: it tends to encourage brittle, tightly-coupled, systems that are inappropiate in a loosely-federated Grid environment, and its support for XML-based data structuring is inferior to that of web-service platforms.

Nevertheless, such middleware has a lot to offer in terms of principles and experience, e.g. in terms of *generic services* (for example, CORBA supports fault tolerance through replication, persistent state, logging, load-balancing, and many others), *server-side scalability* (for example, EJB and the CORBA Component Model (CCM) have sophisticated support for the automated activation/ passivation of services on demand, and natively support services that span multiple machines/ networks), and *performance engineering* (this has been the subject of intensive research in the object-based middleware community over the last 5 years).

Furthermore, cutting-edge research in distributed systems platforms [Blair,00] is now investigating the provision of highly configurable (and run-time reconfigurable) *reflective middleware* technologies (our approach in these areas was described in §2). A prime motivator for this research is to be able to custom-build middleware platform instances so that they can be applied in an very wide range of environments (e.g., from large-scale servers, to real-time embedded systems, to mobile PDAs), and can support a range of programming APIs (e.g. CORBA, or APIs for media-streaming or message-oriented middleware). The basic philosophy is to support configurability, extensibility and adaptability as fundamental system properties. In particular, the approach enables alternative policies (e.g. security policies, replication policies, service (de)activation policies, priority-assigned invocation paths, thread scheduling) and components (e.g. protocols, buffer managers, loggers, debuggers, demultiplexers) to be configured at deploy-time, and reconfigured at run-time (e.g. on the basis of dynamically evolving conditions).

4. Applying Middleware Research Results in OGSA

The implementation approach currently favoured by OGSA developers is to layer OGSA on top of existing web-services platforms. A good example of such a platform is Apache Axis [Axis,02]. This provides a Java-based environment for web-service deployment and invocation, and provides sophisticated support for messaging in terms of SOAP's extension headers, intermediaries, and multiple transport capability. Other examples of web-services platforms are Sun's ONE and IBM's WebSphere. In general, these platforms provide a useful starting point for OGSA implementation because they directly support the central web-services-derived concepts—like SOAP and WSDL—that underlie OGSA's computational model.

Nevertheless, current web-service platforms have significant *limitations* as an OGSA hosting environment. *First*, they are extremely limited, in comparison to object-based middleware platforms, in terms of the above-mentioned aspects of generic service provision, server-side scalability, and performance engineering. In terms of performance, for example, their application focus has traditionally been on e-Commerce where dependability and security are far more important than performance (indeed, an asynchronous SMTP-based transport is often all that is required). Therefore web-service platform developers have not focused on performance optimisation to anything like the extent of, say, CORBA-platform developers.

Second, these platforms have little or no support for *QoS specification and realisation*. We believe that such facilities will be increasingly demanded as sophisticated e-Science applications start to exploit the potential of OGSA's service-based architecture. A closely related limitation is the over-reliance by web-services platforms on SOAP as a communications protocol. Although very flexible and general, SOAP shows its limitations when relied on exclusively as a communications protocol:

- It is inappropriate for Grid applications involving large-volume scientific datasets [Govindaraju,00]—mainly due to its use of XML as an on-the-wire data representation. This is highly demanding in terms of bandwidth, memory and processing cycles (especially compared to earlier standards like ASN.1 and CORBA's CDR).
- It is not as transparent from the perspective of the application programmer as other application-level protocols—programmers often have to explicitly build and extract SOAP envelopes and message bodies and perform manual marshaling and unmarshaling.
- Although it offers flexibility in terms of support for various interaction patterns (e.g., choice of request-reply or one-way messages), underlying transport support (HTTP, SMTP, HTTP/S, etc.), and extension header management, SOAP does not support a comprehensive and/ or extensible range of interaction patterns (e.g. RPC, asynchronous RPC, (un)reliable messaging, publish-subscribe, blackboard systems, media-streaming, reliable/ unreliable group interaction, workflow interaction, distributed voting or auction protocols, and various transactional styles).

OGSA somewhat recognises the limitations of exclusive reliance on SOAP, and (theoretically, at least) leaves room for non-SOAP bindings (e.g. using CORBA IIOP). However, OGSA does not currently specify any particular framework whereby such bindings can be properly integrated into an OGSA-based distributed programming environment, and it similarly does not provide any framework for generic QoS specification/ enforcement.

The starting point of our research is a generalisation of the above observations: *neither OGSA nor web-service platforms support a general extensibility framework for binding-types*. Furthermore, they have no framework to specify and enforce QoS requirements apart from the relatively crude expedient of layering SOAP over alternative transport protocols.

In conclusion, our position is that OGSA implementation can and should leverage the results of the wider middleware research discussed above. In doing so, OGSA can retain its key characteristics (loose coupling, XML-based data structuring, reliance on Internet standards) while additionally folding in some of the key benefits of wider middleware research (in particular, the availability of generic services, server-side scalability, and performance engineering know-how offered by 'standard' middleware; and the increased flexibility and configurability—e.g. in terms of a framework for extensible binding-types—made possible by the newer reflective middleware approaches).

5. Our Current Research

5.1 Overall Goal

Overall, our goal is to design and develop a backwardly OGSA-compatible Grid services platform using our OpenCOM/ OpenORB technology as a hosting environment. The platform will incorporate key results and techniques from the last several years of research in

object-based middleware. It will also feature an programming model that integrates OGSA with the facilities and services found in non-OGSA middleware environments so that application developers can leverage these from OGSA without having to learn multiple APIs. Furthermore, we will exploit the inherent extensibility of the OpenCOM-based hosting environment to yield an OGSA platform that can naturally evolve to incorporate new binding-types and exploit useful generic services that are available in a number of specific environments (web-services, CORBA services, Jini services, etc.). We will also exploit the reconfigurability/ adaptability of the hosting environment to support predictable resourcing of bindings to enable e-Science applications to be able to specify QoS levels and have such specifications meaningfully supported.

In our current research we are focusing specifically *i*) on the provision of a framework for *extensible binding-types* as appropriate for Grid computing (as discussed above), *ii*) on *reflective resource management* that underpins binding-types with predictable QoS, and *iii*) on *performance optimisation*. These areas are discussed in more detail below.

5.2 The Extensible Binding-Type Framework

The goals of our extensible binding-type framework are as follows:

- To explicitly support the specification, documentation, development and integration of new binding-types.
- To support the composition of existing binding-types into new 'composite' types (e.g. a media-stream binding with encapsulated RPC bindings for control).
- To offer a generically extensible API for the use of bindings in applications.

Furthermore, to support QoS-aware e-Science applications the framework should include means for the specification of QoS, and support for the (adaptive) allocation of resources to bindings so that they can meet their given QoS specifications (see §4.3 below). Note that binding-types in our conception can be arbitrarily distributed entities; for example, one can imagine a media-streaming binding-type that wraps a compression service that resides on a different node to either the producer of consumer of the media-stream, and is transparent to both. This implies that the binding-type framework must support the notion of per-node remotely-accessible factories to enable the instantiation of such bindings.

In our framework, binding-types are represented as first-class components, and the framework is based on a small set of generic concepts. These are essentially roles (e.g. binder, referencer, resolver, controller, user) that comprise a generic 'meta-pattern' for the development of binding-types. We specify binding-types in terms of these roles using UML collaborations, and are currently investigating the use of automated code generation techniques (along the lines of the OMG's Model Driven Architecture [OMG,01]) to map from UML specifications to implementations. As well as being useful for the development of binding-types such as those mentioned above, the binding framework also enables us to 'wrap' non-web-service-derived binding mechanisms like CORBA IIOP or Java RMI so that these can be transparently exploited by application developers working in the OGSA environment. Furthermore, the framework's generically extensible API enables application programmers to transparently interact with generic services (e.g. fault tolerance, transactions, etc.) defined in these non-web-service-based environments.

5.3 Reflective Resource Management

To support the binding-type framework, we are developing a reflective *resource meta-model* that underpins the binding-type framework by allowing low-level system resources to be flexibly associated with individual bindings. This will build on initial research at Lancaster [Duran-Limon,00] on the notion of *tasks*—these are scoped execution paths that logically carry out a single 'job' but may arbitrarily span component boundaries. The proposed approach is to use reflective interfaces to associate resources (and resource factories) with tasks, and to be able to reassign these associations as conditions (e.g. specific resource availability, general system loading, or user-defined priorities) change. For example, if the server side of a SOAP binding were designated as a task, it might be given a thread, a socket, and a DOM parser factory. We plan to integrate this fine-grained resource management model with the coarser-grained, distributed, resource management services that are already in use in the Grid environment (e.g. GRAM, Condor-G [GRAM,02]). We also plan to integrate it with the work in our NETKIT project [NETKIT,02] on programmable networking to enable us to support QoS-aware binding-types supported by resource allocation *in the network* as well as merely in the end-system. This is likely to be of increasing importance to highly distributed and data-intensive e-Science applications.

5.4 Performance Optimisation

Finally, we are paying close attention to the optimisation of performance in our Grid platform. This again is building on our previous experience in

developing middleware platforms. For example, we are applying and developing techniques such as optimised request demultiplexing (at both the service and operation levels), tailoring threading strategies to current request patterns, marshaling/ unmarshaling with minimal/ zero copying, efficient buffer management, intelligent connection management, and exploitation of protocol optimisation techniques like ALF/ ILP and header caching/ reuse, etc [Coulson,01]. In addition, for very heavily used services, the use of OpenCOM as a hosting environment allows us to layer platform instances directly on hardware without an intervening OS—we are already exploring this in the NETKIT project in the context of programmable routers. We are also exploring the notion of just-in-time activation of service instances to aid in scalability.

6. Current status and Future Work

Although we are at a relatively early stage in our research. we are making rapid progress due to the fact that we are building on an established software base. At the moment, we have implemented a variety of protocols to help populate the binding-type framework; these include SOAP, OMG IIOP, a home grown media-streaming protocol, uPnP, and SLP, all wrapped as OpenCOM components. We have also implemented a range of binding-types including standard remote method invocation, publish-subscribe, reliable group interaction, group streaming, and an auction protocol. We have found that the binding-type framework does indeed speed up the implementation of binding-types and also makes them easy to use thanks to a consistent use of common API concepts.

We have also used the reflective resources framework to provide a level of QoS for certain binding-types, especially the group streaming binding-type. This is so far limited in scope in that it addresses only end-system resource management (primarily control over thread priorities), but we expect to incorporate network level support (from our NETKIT project) in the near future.

Finally, in terms of applications, we plan to investigate a range of Grid-oriented scenarios in cooperation with various science departments at Lancaster University. For example, we have plans to develop, with our Applied Statistics Dept., a set of distributed services for the processing of statistical functions on population data. This will feature a binding-type that abstract over the fact that multiple servers may process the population data in parallel.

REFERENCES

[Atkinson,02] Atkinson, M. et al., "UK Role in Open Grid Services Architecture", UK e-Science Architecture Task Force document, http://esc.dl.ac.uk/WebServices/UK_Roadmap.pdf .

[Axis,02] Apache Axis Project, http://xml.apache.org/axis/index.html.

[Blair,00] RM 2000, Workshop on Reflective Middleware at IFIP/ACM Middleware 2000, New York; see http://www.comp.lancs.ac.uk/computing/RM2000/.

[Clarke,01] Clark, M., Blair, G.S., Coulson, G., Parlavantzas, N., "An Efficient Component Model for the Construction of Adaptive Middleware", Proc. IFIP Middleware 2001, Heidelberg, Germany, November 2001.

[Coulson,01] Coulson G., Baichoo, S., "Implementing the CORBA GIOP in a High-Performance Object Request Broker Environment", ACM Distributed Computing Journal, Vol. 14, No. 2, pp 113-126, Springer Verlag Press, April 2001.

[Coulson,02] Coulson, G., Blair, G.S., Clark, M., Parlavantzas, N., "The Design of a Highly Configurable and Reconfigurable Middleware Platform", ACM Distributed Computing Journal, Vol 15, No 2, pp 109-126, April 2002.

[Duran-Limon,00] Duran-Limon, H., Blair, G.S., "The Importance of Resource Management in Engineering Distributed Objects", Proc. 2nd International Workshop on Engineering Distributed Objects (EDO'2000), California, USA, Nov. 2000.

[Foster,01] Foster, I., Kesselman, C., Tuecke, S., "The Anatomy of the Grid: Enabling Virtual Organizations, International Journal of Supercomputer Applications, Vol 15, No 3, 2001.

[Govindaraju,00] Govindaraju, M., Slominski, A., Chopella, V., Bramley, R., Gannon, D., "Requirements for and evaluation of RMI protocols for Scientific Computing", Proc. Supercomputing (SC '00), Dallas, Texas, Nov 2000.

[GRAM,02] Globus Resource Allocation Manager, www.globus.org, 2002.

[Grimshaw,99] Grimshaw, A., Ferrari, A., Knabe, F., Humphrey, M., "Legion: An Operating System for Wide-Area Computing", IEEE Computer, Vol 32, No 5, pp 29-37, May 1999.

[Kiczales,91] Kiczales, G., J. des Rivières, D.G. Bobrow, "The Art of the Metaobject Protocol", MIT Press, 1991.

[Kon,02] Kon, F., Costa, F., Blair, G.S., Campbell, R., "The Case for Reflective Middleware: Building middleware that is flexible, reconfigurable, and yet simple to use", CACM, Vol 45, No 6, 2002.

[NETKIT,02] The NETKIT Project: A Reflective Component-Based Infrastructure for Programmable Networks, UK EPSRC Grant GR/S01818/01.

[OGSA,03] Tuecke, S. et al., Grid Service Specification, draft 3, http://www.gridforum.org/ogsi-wg/drafts/GS_Spec_draft.pdf.

[OMG,01] "Model-Driven Architecture", OMG ormsc/2001-07-01, July 09, 2001.

[Szyperski,98] Szyperski, C., "Component Software: Beyond Object -Oriented Programming", Addison-Wesley, 1998.