TOWARDS A COMPONENT-BASED MIDDLEWARE FRAMEWORK FOR CONFIGURABLE AND RECONFIGURABLE GRID COMPUTING

Geoff Coulson[1], Paul Grace[1], Gordon Blair[1], Laurent Mathy[1],
David Duce[2], Chris Cooper[2], Wai Kit Yeung[1], Wei Cai[1]

[1] Computing Dept., Lancaster University, Lancaster LA1 4YR, UK
+44 1524 593054

[2] Dept of Computing, Oxford Brookes University, UK

contact: geoff@comp.lancs.ac.uk

ABSTRACT

Significant progress has been made in the design and development of Grid middleware which, in its present form, is founded on service-oriented architecture and web services technologies. Nevertheless, Grid middleware is still severely limited in key areas. In this paper we discuss work that aims to address some of these limitations. First, we consider how ideas and principles from the wider middleware research community can usefully be applied in a Grid middleware context. Then we focus on our own current work on integration of the Grid middleware platform with an extensible set of interaction types and advanced network services, and on an architectural framework for Grid middleware internals. We believe that these areas, along with complexity management, will become increasingly important as sophisticated e-Science applications start to exploit the potential of service-oriented architecture-based middleware.

Keywords

Grid middleware, components, reflection, overlay networks.

1. INTRODUCTION

Following initial offerings such as Legion [Grimshaw,99] and Globus 2 [Foster,01], the *Open Grid Services Architecture* (OGSA) [OGSA,03] has recently emerged as a 'second generation' distributed computing approach to Grid middleware that is taking Grid support forward from an era of ad-hoc platforms to a more architected approach built on service-orientation and web services technologies. This new approach promises a more unified and principled approach to the support of Grid applications. It augments generic web services standards by defining a specific abstract notion of 'Grid service'; and also defines Grid-specific architectural elements 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.

However, despite these advances, OGSA, and indeed the web services technologies on which it is based, are still deficient in many areas of distributed computing support which, we believe, are key to the successful hosting of large-scale, next generation, Grid applications.

We are particularly concerned with applications that exhibit properties such as: high levels of heterogeneity in terms of both networking and end-systems; real-time interactive collaboration employing multiple media-types; large scale, complexity and dynamic (re-)configuration; QoS-sensitivity, and adaptability to changes in environmental conditions. An illustrative example of such an application is a world-wide collaborative visualization session involving large numbers of scientists who join and leave the session dynamically and are connected by a variety of access networks and end-systems (including wireless networks/PDAs), and involving multiple media such as visualization data, live sensor output, vector graphics and video [VESC,03].

We contend that such applications fundamentally over-stretch the state-of-the-art in existing Grid support. More specifically, our analysis is that current platforms have *three* major areas of deficiency in terms of advanced application support:

- *Integration with advanced network services*. One of the attractions of OGSA is its simple SOAP-based model of interaction. However, advanced applications often require more sophisticated communications services in terms of, for example, QoS management, and, especially, different 'interaction types' (e.g. RPC, asynchronous RPC, reliable/ unreliable messaging, publish-subscribe, tuple-space-based interaction, peer-to-peer based interaction, media-streaming, reliable/ unreliable group interaction, workflow interaction, distributed voting or auction protocols, and various transactional styles); note that much of the low-level supporting research on interaction types is now subsumed under the topic of *overlay networks* [El-Sayed,03]).
- *Architectural framework.* OGSA focuses on interoperability through the use of 'ubiquitous' Web protocols and associated abstractions (e.g. WSDL). However, this focus on interoperability needs to be

complemented with a strong internal platform architecture that supports the integration of diverse system elements in terms of both *breadth* (e.g. generic, 'horizontal' distributed services such as persistence, visualization, conferencing) and *depth* (e.g. underlying 'vertical' communication services in intimate contact with the network; and with other, end-system based, resources).

- *Complexity management*. As Grid applications become increasingly large, complex, and long-lived, there emerges a strong need for their sophisticated management. It is becoming recognised that the scale and complexity of such systems demand a self-managing or autonomic approach. Linking back to the previous two points, it is crucial that such self-management is applicable to the architecture of the *whole* system including communication services. We argue that this implies an open and programmable approach to system construction.

In our current research we are addressing these deficiencies through a pervasive component-based approach that integrates middleware and (overlay) networking functionality. Component technologies have already been adopted successfully in Grid research to promote structure and re-use at the *application* level (see, e.g., [Furmento,02]). But here we propose the use of component technology not only for applications but *throughout the platform architecture* in terms of both breadth and depth as outlined above. It is important to stress that we are *not* proposing the use of traditional heavyweight component technologies like Enterprise JavaBeans (EJB). Rather, a major aim of the research is to develop and apply a lightweight component model that imposes minimal overhead, and can be used to build even low-level, system-oriented, functionality. The component model should also be system and language independent, and API-neutral, so that it can be used to construct arbitrary application-level distributed programming environments as required (e.g. OGSA, web services, or, indeed, EJB).

A particular goal is to apply lightweight component-based technology to construct an extensible family of open and programmable overlay networks, thus providing an approach that is network-centric, offers a strong architecture for the system infrastructure, and facilitates self-management through the inherent openness of component-based structures [Blair,02]. This approach also promises other important benefits: *i*) a extensible range of interaction types, such as those listed above, can be made available and selected according to the application domain and/or context; and *ii*) it facilitates dynamic re-configuration of communications (and other services) as context changes (e.g. to maintain a visualization session when an end user roams to a wireless network).

To validate our approach and provide focus for practical experimentation we are using selected collaborative visualization-based applications and scenarios. Collaborative visualization is highly appropriate for this purpose because of its inherent properties as outlined above. It is also data and compute intensive which makes it an ideal case study for an infrastructure that aspires to manage both network and end-system resources in an integrated manner.

In the remainder of this paper we first, in section 2, discuss opportunities for applying research results from the wider middleware research community to specifically Grid-oriented middleware. This includes existing object-based middleware standards and products, and recent results from the advanced middleware research community. This work mainly contributes in the breadth dimension. Following this, section 3 discusses our research and results to date. Our direction here accommodates the breadth dimension but also opens up the depth dimension. Finally, section 4 draws conclusions and indicates areas of planned future work.

2. OPPORTUNITIES FOR APPLYING WIDER MIDDLEWARE RESEARCH RESULTS IN GRID MIDDLEWARE

The implementation approach currently favoured by Grid middleware developers (e.g. OGSA) is to layer the Grid environment on top of existing web services platforms. A good example of such a platform is Apache Axis [Axis,02], which provides a Java-based environment for web service deployment and invocation, and offers 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.

However, although these platforms are a useful starting point, they have significant limitations as a Grid middleware support infrastructure. *Firstly*, they are extremely limited, in comparison to object-based middleware platforms (e.g. RM-ODP and CORBA, and industry-developed platforms like Java RMI, Enterprise JavaBeans, DCOM, and the .NET remoting architecture) in the following areas:

- provision of generic (horizontal, or breadth-oriented) services—for example, CORBA supports generic reusable services like fault tolerance, persistent state, automated logging, load-balancing, transactional object invocation, event distribution, and many others;

- scalability and performance—for example, EJB and the CORBA Component Model have sophisticated support for the automated activation/ passivation of stateful services, and natively support services that span multiple machines/ networks; in addition performance engineering has been the subject of intensive research in the object-based middleware community over the last 10 years (see, e.g., [Coulson,02]).

In contrast, horizontal services are conspicuously lacking in current Grid environments and there is great potential here for reuse from the wider field. And in terms of performance, the application focus of web services-derived middleware has traditionally been on e-Commerce where dependability and security are far more important than performance (indeed, a single threaded server and an asynchronous SMTP-based transport is often all that is required). Therefore, web services platform developers have not focused on performance optimisation to anything like the extent of, say, CORBA-platform developers.

Secondly, web services-derived 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 such as those outlined in the introduction start to exploit the potential of service-oriented architectures. We also believe that a prime cause of this deficiency is an over-reliance by web services platforms on SOAP as a communications engine. Although very flexible and general, SOAP clearly shows its limitations when relied on exclusively:

- 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 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 some flexibility in terms of support for different interaction types (e.g., choice of request-reply or one-way messages), it can never support the comprehensive range of interaction types provided by object-based middleware platforms (which covers a significant subset of those listed in section 1).

The OGSA design nominally recognises the limitations of exclusive reliance on SOAP, and (theoretically, at least) leaves room for non-SOAP bindings (e.g. using CORBA IIOP and, potentially, other bindings that do have some support for QoS). However, OGSA does not currently specify any particular framework whereby such bindings can be integrated into the distributed programming model, and it similarly does not provide any framework for generic QoS specification/ enforcement.

Thirdly and finally, current web services platforms do not include recent results from advanced middleware research which is investigating highly configurable (and run-time reconfigurable) reflective and component-based middleware technologies (e.g., see the proceedings of the Second International Workshop on Reflective and Adaptive Middleware [RM,03]). A prime motivator for this research is to facilitate the custom-building of middleware platform instances that can be applied in a 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 web services, or APIs for media-streaming, or message-oriented middleware). A prime, and highly successful, example of such a platform is the open source JBoss application server [Fleury,03]. The basic philosophy of advanced middleware 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 in or out at deploy-time, and reconfigured at run-time (e.g. on the basis of dynamically evolving conditions).

Overall, our view is that next generation Grid middleware can and should leverage the results of the wider middleware field as discussed above. In doing so, it can retain key web services-derived characteristics (loose coupling, XML-based data structuring, reliance only on ubiquitous Internet standards) while additionally folding in some of the key benefits of the wider field—in particular, the availability of generic services, and scalability and performance engineering know-how offered by 'standard' middleware; and the increased flexibility and configurability promised by the advanced middleware research.

3. OUR CURRENT RESEARCH

3.1 GRIDKIT

In this section we discuss our current activities in pursuit of the above goals. Our work is subsumed under the umbrella of an architecture called GRIDKIT. As illustrated in figure 1, the aim of GRIDKIT is to provide support in each of four 'domains' that we identify as key in underlying the provision of Grid services. These domains are as follows:

- *Service binding*. This area provides sophisticated communication services beyond SOAP: i.e., support for QoS management, and for different interaction types such as those listed in section 1.

- *Resource discovery*. This provides service, and more generally, resource, discovery services, allowing for the use of multiple discovery technologies to maximise the flexibility available to applications. Examples of alternative technologies are SLP or UPnP for more traditional service discovery, GRAM for CPU discovery in a Grid context, and P2P protocols for more general resource discovery.
- *Resource management*. This comprises both coarse-grained distributed resource management as currently provided by services such as GRAM, *and* fine-grained local resource management (e.g. of channels, threads, buffers etc) that is required to build end-to-end QoS.
- *Grid security*. This supports secure communication between participating nodes orthogonally to the interaction types in use.

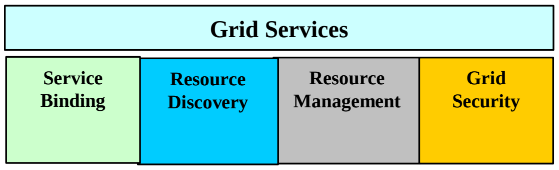

Figure 1: The GRIDKIT Vision

These four domains of middleware functionality are implemented in GRIDKIT as independent, horizontal, frameworks each of which is highly configurable and reconfigurable. As such, they are directly available to application services, and can also be combined to provide more complex middleware capabilities. For example, service bindings can integrate with Grid security to produce secure communication channels. In the remainder of this paper, we examine in detail the service binding and resource discovery frameworks (we do not discuss resource management and security further in this paper).

3.2 The Ancestry of GRIDKIT

GRIDKIT is an instantiation of the generic OpenORB middleware platform [Coulson,02], and hence follows the philosophy of building systems using *components* (with the OpenCOM component model), *component frameworks* and *reflection*. In particular, the four domains discussed above are each implemented in terms of (reflective) component frameworks (CFs) that are configurable and dynamically reconfigurable by means of 'plug-in' components.

The generic architecture of GRIDKIT is also strongly influenced an OpenORB-based, web services-based, mobile computing framework called ReMMoC [Grace,03]. ReMMoC provides inspiration and a code base for specific aspects of GRIDKIT. In particular, it contributes a prototype service binding CF.

ReMMoC was originally designed to tackle the middleware heterogeneity problem inherent to the domain of web services-based mobile computing. Typically, mobile users encounter application services implemented on different middleware standards as they move from one location to another. For example, at one location, a jukebox service may be implemented as a SOAP service and advertised using UPnP; whereas at the next location the same service may be implemented on a publish-subscribe implementation and advertised using SLP. ReMMoC addresses heterogeneity of this type, allowing mobile client applications to be developed independently of both binding styles and service discovery styles. Hence, these applications are able to continue operating as the user moves to new, unknown locations.

We believe that ReMMoC particularly recommends itself as the basis of GRIDKIT because it demonstrates how a range of interaction types can be abstracted using a web services API, thus providing the basis of an extensibility framework in which to accommodate the more complex range of interaction types required by future Grid applications. In addition, ReMMoC, thanks to its OpenCOM base, has been shown [Grace,03] to be both highly performant and to incur only a minimal memory footprint (around 27K), making it deployable in almost any system environment. Nevertheless, ReMMoC per se is limited in its applicability to the Grid because *i*) it is only a client side system *ii*) it does not consider the 'depth' dimension of integration with network services, and *iii*) it does not consider resource management or security.

3.3 The GRIDKIT Architecture

GRIDKIT is built in terms of OpenCOM/ ReMMoC derived CFs, the architecture of which is shown in figure 2.

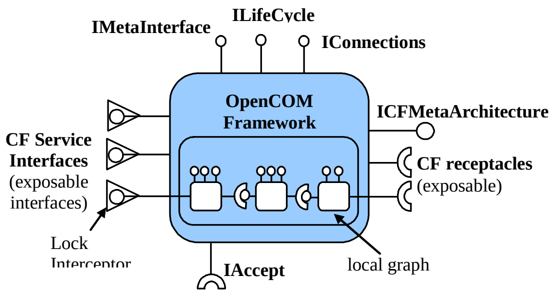

Figure 2: The Component Framework Model

The CFs behave as standard OpenCOM components; but, in addition, each implements the *ICFMetaArchitecture* interface which provides operations to inspect and dynamically reconfigure the CF's internal structure (maintained as a 'graph' of internal sub-components). To ensure that dynamic changes to the framework are 'valid', each CF exports a receptacle named *IAccept*; from here different validation strategies can be plugged into the framework so that once a change is made, the plug-in checking strategy is executed, and if invalid the framework rolls back to its previous state. By default, the local graph is checked against a set of XML-based architectural descriptions of valid component configurations. Alternatively, more or less complex strategies can be plugged in (e.g. architectural style rules [Moreira,01]).

Turning now to the wider picture, the subset of the GRIDKIT architecture that deals with service binding and resource discovery is illustrated in figure 3. This depicts a three-layer architecture that is composed of: *i*) abstract middleware, *ii*) abstract to concrete mappings, and *iii*) concrete middleware. Each of these layers in turn consist of multiple CFs. This renders the architecture inherently configurable and extensible so that components implementing specific functions can be plugged in when and where required. In addition, applications requiring only minimal middleware functionality need only utilise parts of GRIDKIT. This is especially important for execution on devices with limited resources e.g. mobile devices.

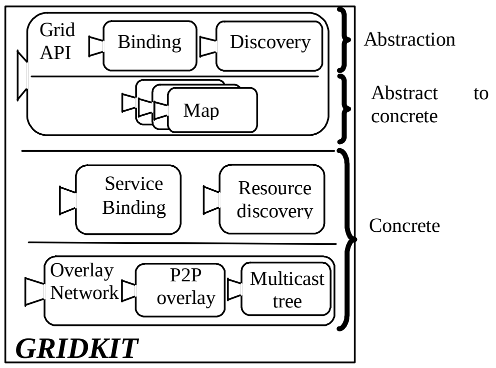

Figure 3: The GRIDKIT Architecture

The *abstract middleware* layer consists of a "Grid Service API" CF that is built in terms of web services abstractions. In particular, abstract service interactions are described in terms of WSDL so that services can be invoked irrespective of the interaction type underlying the service. This is achieved by exploiting WSDL's approach of breaking interactions down into individual messages: any conceivable service operation, from the user's perspective, can be described abstractly in terms of input or output messages (e.g. an RPC-based service might be described as an output message followed by an input message, whereas a publish-subscribe-based service can be described, from the perspective of the subscriber, as an output message for the subscription followed by potentially many subsequent asynchronously-received input messages representing incoming publications).

Discovery (of both services and resources) also forms part of the abstract middleware layer. Again, WSDL is used to abstract over different modes of interaction with service and resource discovery mechanisms. This is relatively straightforward for service discovery protocols (e.g. SLP and UPnP) because all of these tend to be based on advertisement of service types with service attributes.

The *abstract to concrete mapping* layer then takes the abstract information submitted through the abstract middleware layer and maps it to the interfaces of the currently exposed concrete middleware implementation(s) in the layer below. This mapping is based on ReMMoC principles, a detailed discussion of which can be found in [Grace,03]. Unlike ReMMoC, the Grid API framework allows multiple mapping components to be maintained in order for different services to be simultaneously hosted, each of which can use multiple service bindings. This was not necessary in ReMMoC as it was exclusively a client-side framework that did not itself support remotely accessible services.

Finally, the *concrete middleware layer* is composed of three CFs, organised in two layers. The top layer is composed of CFs to support concrete service binding and resource discovery. The *Service Binding CF* provides a set of available interaction type implementations. These are implemented as component personalities and plugged into the framework. Multiple personalities can operate in parallel to support the required level of persistence in hosting services. That is, when a service is hosted over one binding type it need not be shut down if an alternative binding must be used by another service. The binding framework exposes its network requirements to the underlying overlay framework using the exposed receptacle technique illustrated in figure 2.

The *Discovery CF* similarly allows multiple discovery technologies to be plugged into the framework (e.g. SLP, UDDI, Jini, P2P-based etc.) at any one time. Resource discovery requests and advertisement of resources can be executed in parallel over each of the plugged-in personalities so that Grid applications can maximise the number of resources that are found, find them more quickly, and can distribute their resources to a greater audience. The discovery framework utilises the underlying overlay framework to enhance discovery, and we intend to investigate the addition of alternative resource discovery technologies (based, e.g., on peer-to-peer overlays) into the framework.

Underpinning the Service Binding and Discovery CFs, the role of the *Overlay CF* is to provide overlay network services to the higher-level CFs: to route packets through virtual networks that are tailored to support the various service interaction types. Sophisticated communication services e.g. data streaming, P2P resource discovery and application-level multicast can be supported by appropriate overlay configurations. As with the other CFs, multiple overlay personalities can be plugged in.

Multiple service bindings can then operate over their selected overlay. We anticipate that the nodes of the (overlay) network will be composed of machines hosting appropriate GRIDKIT CFs, which will allow autonomic management of the overlay to support the application. That is, the algorithms to maintain the required network structures will be dynamically managed by communication between the low-level component frameworks in each of the nodes.

The inherent openness of GRIDKIT enables both coarse and fine-grained dynamic reconfiguration. To support application requirements, a new binding or discovery plug-in can be configured; or similarly, the implementation of an individual binding type can be changed e.g. a change from SOAP to IIOP for a remote method invocation plug-in. Furthermore, fine grained changes, i.e. within the component personality, can be made to better support QoS requirements; e.g. changing media filters when the performance of the network degrades.

4. CONCLUSIONS AND FUTURE WORK

We have argued that existing Grid middleware does not provide the necessary level of support for complex Grid applications such as distributed collaborative visualisation. We believe that an open component-based platform, which integrates middleware and (overlay) networking functionality, is needed to support the sophisticated communication requirements of applications of this type. For this purpose, the service binding and resource discovery architectures of GRIDKIT allow multiple interaction and discovery types to be simultaneously hosted over multiple overlay network configurations.

Our ReMMoC-derived GRIDKIT implementation initially provided us with a base of three binding protocols and two discovery technologies (consisting collectively of 35 OpenCOM components). We are currently in the process of extending this with data streaming and OGSA-DAI-based data sharing as additional bindings; and UDDI, Jini and JXTA as additional resource discovery protocols. Finally, we are wrapping an existing tree-based multicast overlay [Mathy,01] as an initial overlay plug-in.

Future work is planned on two fronts: first we will exercise and evaluate our frameworks and plug-ins by using them to support a range of distributed visualisation scenarios that have been developed at Oxford Brookes University. Second, we plan to explore the self-management of services and applications in GRIDKIT. This will build on the inherent openness of the (component-based) platform but will require additional CFs that deal with areas such as monitoring, recovery strategy selection, and recovery strategy deployment. We have carried out initial explorations in this area [Blair,02], but GRIDKIT will provide a challenging context for these ideas.

5. REFERENCES

[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.

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

[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.

[Fleury,03] Fleury, M., Reverbel, F., "The JBoss Extensible Server", Proc. IFIP/ACM Middleware 2003, Rio de Janeiro, Brazil, Springer Verlag LNCS, pp 344-354, June 2003.

[RM,03] The 2nd Workshop on Reflective and Adaptive Middleware, IFIP/ACM Middleware 2003, Rio de Janeiro; http://www.cs.wustl.edu/~corsaro/RM2003/index.html.

[Blair,02] Blair, G.S., Coulson, G., Blair, L., Duran-Limon, H., Grace, P., Moreira R., Parlavantzas, N., "Reflection, Self-Awareness and Self-Healing in OpenORB", Proc. ACM Sigsoft Workshop on Self-Healing Systems (WOSS'02), Nov 02.

[Furmento,02] Furmento, N., Mayer, A., McGough, S., Newhouse, S., Field, T., Darlington, J., "ICENI: Optimisation of Component Applications within a Grid Environment", Parallel Computing, Vol 28, No 12, pp1753-1772, 02.

[El-Sayed,03] El-Sayed, A., Roca, V., Mathy, L., "A Survey of Proposals for an Alternative Group Communication Service", IEEE Network, Vol 17, No 1, pp46-51, Jan 03.

[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.

[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.

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

[VESC,03] e-Science Visualization, NeSC, Edinburgh, Jan 03, http://umbriel.dcs.gla.ac.uk/NeSC/general/esi/events/130/workshop_report.pdf.

[Moreira,01] Moreira, R., Blair, G., Carrapatoso, G., "Reflective Component-Based & Architecture Aware Framework to Manage Architecture Composition". Proc. 3rd International Symposium on Distributed Objects & Applications, Rome, Italy, Sept, 2001.

[Mathy,01] Mathy, L., Roberto Canonico, R., Hutchison, D., "An Overlay Tree Building Control Protocol", Proc. Networked Group Communication (NGC 2001), London, LNCS 2233, Crowcroft and Hofmann (Eds), pp76-87, Nov 01.

[Grace,03] Grace, P., Blair, G.S., Samuel, S., "ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability", Proc. International Symposium of Distributed Objects and Applications (DOA'03), Catania, Italy, November 2000.