A Component-based Middleware Framework for Configurable and Reconfigurable Grid Computing

Geoff Coulson[1], Paul Grace[1], Gordon Blair[1], Wei Cai[1], Chris Cooper[2], David Duce[2], Laurent Mathy[1], Wai Kit Yeung[1], Barry Porter[1], Musbah Sagar[2], Jason Li[2]

[1] Computing Dept., Lancaster University, Lancaster LA1 4YR, UK
[2] Dept of Computing, Oxford Brookes University, UK

e-mail: 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 web services technologies. However, we argue that present-day Grid middleware is severely limited in supporting projected next-generation applications which will involve pervasive and heterogeneous networked infrastructures, and advanced services such as collaborative distributed visualisation. In this paper we discuss a new Grid middleware framework that features *i*) support for advanced network services based on the novel concept of pluggable overlay networks, *ii*) an architectural framework for constructing bespoke Grid middleware platforms in terms of 'middleware domains' such as extensible interaction types and resource discovery. We believe that such features will become increasingly essential with the emergence of next-generation e-Science applications.

Keywords: Grid middleware, components, reflection, overlay networks.

1. Introduction

The Open Grid Services Architecture (OGSA) [1] has recently emerged as a 'second generation' distributed computing approach to Grid middleware. 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, Grid middleware is still deficient in many areas of distributed computing support that, we believe, are key to the successful hosting of large-scale, next generation, Grid applications. We are particularly concerned with next-generation applications that exhibit such properties 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 [2].

We contend that such applications fundamentally over-stretch state-of-the-art

Grid middleware. Next-generation applications will require sophisticated communications services beyond standard SOAP messaging 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).

In this paper we provide an overview of *Gridkit*, a configurable and dynamically reconfigurable middleware framework. which consists of an underlying pluggable overlays framework that supports an extensible set of higher-level middleware service domains. In particular, we examine how we apply a lightweight component-based technology to construct an extensible family of open, programmable and ‘pluggable’ overlay networks that underpin various domains of middleware functionality. For example, we support an extensible range of interaction types, such as those listed above, which can be made available and selected according to both the application domain and/or execution context. In addition, we facilitate 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).

In the remainder of the paper we first, in section 2 we present the overall architecture of Gridkit. Subsequently, we discuss the overlays framework in detail in section 3; and then in section 4 we discuss three key higher-level middleware service domains: interaction services, resource discovery and resource management. Finally, section 5 draws conclusions about the presented framework and indicates areas of future work.

2. Gridkit: a Configurable and Reconfigurable Grid Middleware Framework

As illustrated in figure 1, Gridkit is based on an ‘open overlays’ layer which abstracts the diversity of underlying communications support mechanisms in a consistent manner whether or not the underlying physical network supports a given communications service (e.g. multicast or QoS). On top of this is an extensible set of orthogonal domains of generic middleware support. These in turn support a Grid Services layer which, for reasons of compatibility with current Grid middleware practice, presents the Gridkit facilities to applications in terms of a web services API.

We have so far identified the following middleware domains as central:

1 *Interaction Services*. This domain provides sophisticated application-level communication services beyond SOAP: i.e., support for QoS management, and for different types of interaction such as those listed in section 1.

2 *Resource discovery*. This provides for *service*, and more generally, *resource*, discovery. To maximise the flexibility available to applications, it supports the parallel use of multiple discovery technologies. Examples of alternative technologies are SLP or UPnP for more traditional service discovery, Globus MDS for CPU discovery in a Grid context, and P2P protocols for more general resource discovery.

3 *Resource management*. This comprises both coarse-grained distributed resource management as currently provided by infrastructures such as GRAM, *and* the fine-grained local resource management (e.g. of channels, threads, buffers etc) that is required to build end-to-end QoS.

4 *Grid security*. This supports secure communication between participating nodes

orthogonally to the interaction types in use.

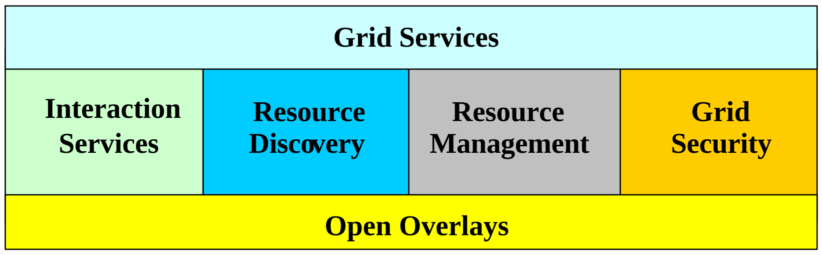

Figure 1: Overall Gridkit Architecture.

Gridkit follows Lancaster's established approach [3] of building systems in terms of *components* (using our OpenCOM component model), *component frameworks* and *reflection*. In particular, each of the areas in figure 1 is implemented as a (reflective) component framework (hereafter, CF) that is configurable and dynamically reconfigurable by means of 'plug-in' components. Each CF can be made directly available to applications; or, alternatively, multiple CFs can be combined to provide more complex facilities. For example, Interaction Services combines with Grid Security to produce secure communication channels; and Resource Management combines with Resource Discovery to support the discovery and allocation of computational nodes. We do not discuss the Grid Security domain further in this paper: this will be the topic of a separate forthcoming paper.

3. The Overlay CF

Overlay networks are virtual communications structures that are 'laid over' an underlying physical network such as the Internet. They are typically implemented by deploying appropriate application-level routing functionality at strategic places in the physical network (in principle both the core and the edges). Overlays are primarily used for two reasons: *i*) to alleviate the effects of slow or sporadic deployment of new services in the Internet (e.g. application-level multicast) [4]; and *ii*) to directly provide application-level functionality that is out-of-scope for the underlying network (e.g. large-scale peer-to-peer file sharing) [5]. Examples of overlays are reliable multicast services (e.g. SRM), content dissemination networks, unstructured peer-to-peer search (e.g. Gnutella), distributed hash table-based routing (e.g. Chord), and routing in ad-hoc and sensor networks.

In current practice, overlays are usually deployed individually and used to support a single fixed application. However, we believe that in order to support the diverse communications requirements of next generation Grid applications in highly heterogeneous environments (from sensor networks to wireless networks to large scale fixed networks), a more flexible and dynamic approach is required. In particular, we see the need *i*) to dynamically instantiate new overlays on demand to support newly instantiated higher-layer software (middleware services and applications), *ii*) to operate multiple cooperating overlays in a coordinated manner (see examples below), and *iii*) to be able to configure and dynamically reconfigure overlays to meet the evolving needs of executing applications.

It is the role of the Overlay CF (see figure 2) to address these issues. Overlays, which are themselves realised as small CFs, can be dynamically plugged into the framework and bound into existing topologies of overlays. Figure 2 illustrates how the CF supports multiple overlay configurations, and how different overlays can depend on each other. In particular, two overlays are shown depending on a Chord key-based routing (KBR) CF, whereas a keyword search overlay operates atop a separate overlay. Each overlay in the framework can expose its interface to the outside, allowing it to be used by higher-level services. The current configuration is driven by current application requirements (e.g. operating in ad-hoc or sensor networks, requiring group communication etc.). In addition, there is scope for the environmental context of the node to drive dynamic reconfiguration of overlays structures. Thus, for example, a streaming overlay can be dynamically added above the KBR to supplement an existing group overlay, or an existing implementation may be changed to a new or enhanced version.

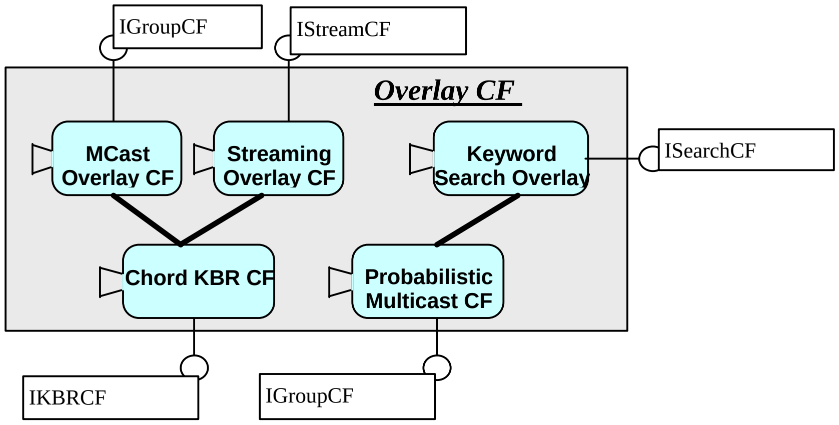

Figure 2: Architecture of the Overlay CF.

The structure of each individual overlay plug-in is in terms of three sub-components [6]. These are: *i*) a ‘control’ part which cooperates with its peers to build and maintain a virtual network topology, *ii*) a ‘forwarding’ part that routes messages over the virtual topology, and *iii*) a ‘state’ part that encapsulates state such as nearest neighbours. Given this pattern, overlay implementations can quickly be developed by building on the individual sub-components of existing overlays. For example, we have an content-based routing overlay that can use the ‘control’ part of a number of different overlay types, but provides its own forwarding sub-component [7].

More detail on the Overlay CF is given in [6].

4. Three Example Middleware Domains

4.1 The Interaction Services CF

The Interaction Services CF facilitates the deployment and dynamic reconfiguration of multiple, simultaneously available, interaction or ‘binding’ types. These are used to support various kinds of interaction between distributed application components. some examples of these were given in section 1. Further examples are SQL links between applications and databases; FTP links (including GridFTP); and bindings that encapsulate a workflow process involving multiple processing entities. These diverse forms of interaction are uniformly implemented as 'pluggable' binding types within the CF. This promotes the reuse of recurring interaction patterns and mechanisms, and allows the multitude of binding types found in Grid application scenarios to be uniformly handled.

To illustrate the functioning of the CF we demonstrate how a publish-subscribe binding type is realised and underpinned by a particular overlay configuration (provided by the Overlay CF). The binding type (see figure 3) offers content-based event notification: events of a particular subject are disseminated in groups. Events are realised as XML messages wrapped in SOAP envelopes, and there are components to publish, subscribe and filter events of this structure.

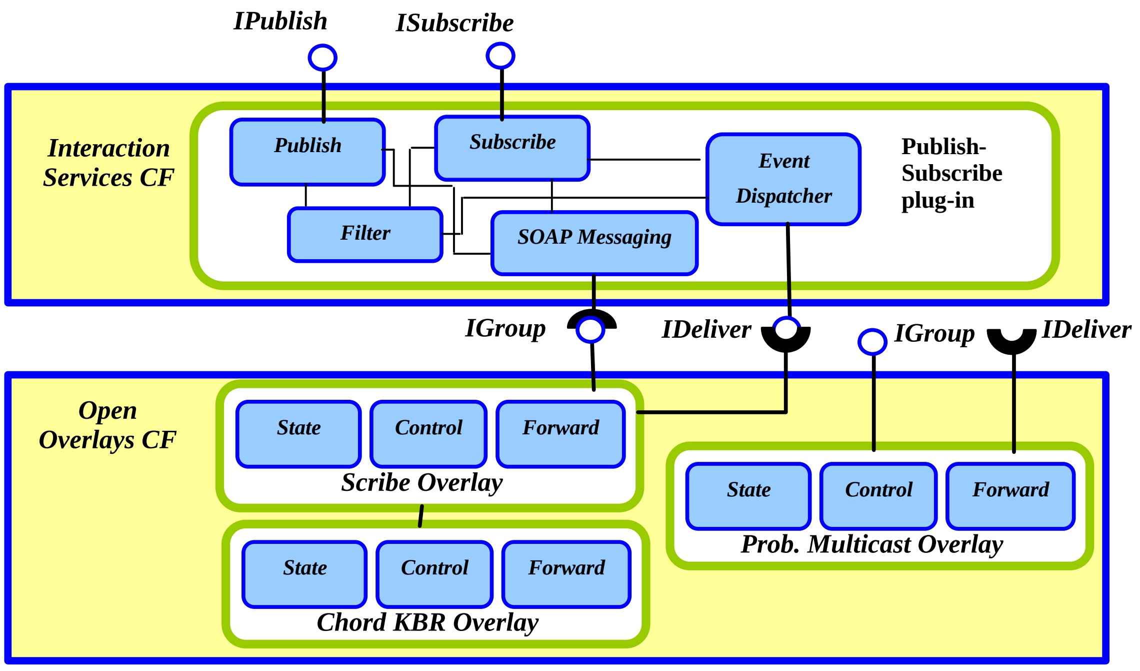

Figure 3: Component configuration for the Publish-Subscribe Binding Type.

Figure 3 also demonstrates how the CF can adapt to meet changing environmental information. The publish-subscribe binding type exports a receptacle stating a requirement for group based message dissemination (*IGroup*). When the node is operating in an ad-hoc network environment an overlay implementing 'probabilistic multicast' is used; this is an unstructured overlay that intelligently floods messages. Each node receives all messages, discarding messages that are not from a member group, and then decides whether or not each message should be forwarded. The decision is based on previous messages that the node has received: if a large number of duplicates of a message have already been received, the probability that the message will be forwarded reduces; e.g., zero duplicates implies a probability of 1, whereas two duplicates implies a probability of 0.25. Alternatively, when the node is operating in a large-scale fixed network, a Scribe-based [8] overlay is plugged in. Scribe builds a multicast overlay for events of a particular topic on top of the key based routing mechanism provided by an implementation of the Chord algorithm [9].

The Interaction Services CF illustrates how our generalised overlays-based approach can be used to tailor application execution to particular networking contexts, so that bindings remain correctly supported in conditions of dynamic change. Furthermore, the CF adds the further capability of fine-grained reconfiguration. For example, the probabilistic multicast forwarder component can be replaced with different damping metrics; i.e. the probability can be decreased (to alleviate network traffic) based upon current network conditions.

4.2 The Resource Discovery CF

The purpose of the resource discovery CF is to provide a reconfigurable service and resource discovery facility that can uniformly perform ‘lookup’ operations using an extensible set of discovery protocols. An application developer or Grid service can uniformly discover resources that match their requirements, based upon matching resource type and attributes, irrespective of the discovery mechanism that is advertising it. Hence, services can be discovered if they have been advertised through traditional service discovery mechanisms such as UPnP, SLP and Jini; through P2P discovery technologies like JXTA and Gnutella; or through standard Grid resource discovery technologies like Globus MDS.

To exemplify this we present a particular discovery technology integrated within the CF, namely *RDFPeers* [10]. In this discovery plug-in, RDF [11] is used to describe resources that are uniquely identified by URIs: each RDF description is a set of triples each of which consists of a subject, predicate and object. The subject is the resource being described and the predicate and object are a name-value description of a property of the resource. Our RDFPeers plug-in extracts these triples from a given description and stores the information in a distributed hash table overlay in such a way as to facilitate future lookup requests. More specifically, the plug-in hashes the resource three times for each of the triples: i.e. once for the subject key, once for the object key, and once for the predicate key. This in turn allows a comprehensive range of queries to be performed. For example, “given an object O_i and predicate P_i find a subject that matches this property”, and “given subject S_i find its object and predicates”.

To integrate this into the framework, the RDFPeers plug-in exports a requirement for distributed hash table behaviour (the IDHT receptacle in figure 4). For this purpose, we plug-in a DHT implementation operating in the overlay framework. In this case we have implemented a DHT overlay on top of the Chord KBR overlay that adds local hash table storage of data, and replication on top of the Chord KBR mechanism.

The Resource Discovery CF illustrates a key benefit of our pluggable overlay approach, namely the re-use of existing overlay services to underpin multiple binding and discovery services that are operating concurrently. Similarly to the Interaction Services CF, multiple discovery protocols operate in parallel, and we have implementations of SLP, UPnP and Jini protocols that depend on one or more pluggable multicast overlays; hence, they require plug-ins similar to those provided in figure 3.

More detail on the Resource Discovery CF is given in [12].

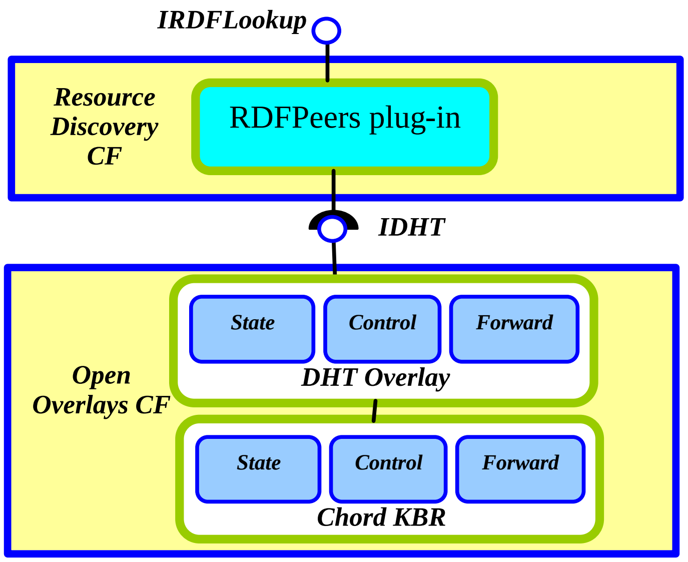

Figure 4: Gridkit's RDFPeers implementation.

4.3 The Resource Management CF

The function of Grid resource management middleware is to appropriately map applications to nodes according to their resource requirements, and to dynamically manage the resourcing of applications as they execute. Although successful, existing systems (e.g. Globus [13]) have a number of limitations. In particular, they are *coarse-grained* in the sense that resource specifications tend to deal with whole machines (or at best processes) rather than with fine-grained resources such as threads, buffer pools, connections etc. In addition, existing systems don't offer any consistent notion of an *abstract* resource: they deal exclusively with concrete entities such as CPUs, memory bytes etc. This makes it difficult to map from application-centric notions of resource (e.g. "I need 3 matrix containers of type X, 1 buffer pool of type Y, a scheduler for EDF threads, and a Java virtual machine") to the notion of 'resource' that the system understands. And finally, existing systems lack support for *run-time adaptation*—the resources allocated at application launch-time cannot be adjusted during runtime.

The goal of our Resource Management CF is to appropriately place the application's constituent components on some specific set of physical computational nodes. It is the framework's job *i*) to map components to nodes, *ii*) to ensure that each component's tasks are adequately resourced by its supporting nodes, and *iii*) to maintain the resourcing of the application at runtime as resource needs and resource provision fluctuate. At the most abstract level, there are two parts to the CF: *i*) *global resource management*, which coordinates resource management over multiple computational nodes, and *ii*) *local resource management*, which manages resource allocation and usage in individual computational nodes. The local resource management part is heavily based on earlier work [14]. The CF is driven by an *application description* which is submitted for execution. This consists of the following: *i*) a set of top-level OpenCOM components that comprise the application, *ii*) a set of bindings (expressed in terms of specifications understood by the Interaction Services CF) between interfaces and receptacles of these components that capture the abstract topology of the application, *iii*) a set of so-called *tasks* which, among other things, express the required QoS of different parts of the application, and *iv*) a *mapping* of tasks to components.

Together, the set of top-level components and bindings comprise the compositional structure of the application. Both tasks (see below) and bindings are decorated with QoS annotations which are used later when the various components of the application are mapped to physical computational nodes. QoS annotations are expressed in terms of pluggable *QoS ontologies* which are defined by domain experts. Each QoS ontology is associated with a plug-in QoS mapper that understands how to derive resource requirements from a QoS specification expressed in the associated ontology. For example, a QoS ontology that was concerned with the application domain of media transcoding applications might define QoS parameters such as “throughput in frames per second”, “latency”, and “acceptable frame degradation”, together with mappings from these parameters to a resource ontology that comprehends concepts such as “buffer pool size”, “number of high-priority threads” etc.

Tasks are abstractions of ‘activities’ or ‘units of work’ which are meaningful at the application level as a subject of QoS annotation. Crucially, the definition of the tasks that comprise an application is orthogonal to the structure of the application itself in terms of components. Thus, in some cases a single task may span a set of (cooperating) components, while in others a single component may host multiple independent tasks (also, tasks may overlap, as shown). Examples: i) a ‘transcode stream’ task could be realised as a set of components that cooperate to transcode a media stream (e.g. buffering, compressing, encoding etc.); ii) multiple instances of an ‘access database’ task could be encapsulated within a single component that deals with concurrent database access.

The orthogonality of tasks and component structure facilitates QoS specification that is meaningful at the application level. Thus, in the ‘transcode stream’ case, the QoS specification is attached to the entire user-visible task rather than to microcosmic aspects such as buffering etc. This orthogonality also offers a useful separation of concerns between writing an application and specifying its QoS. Note that tasks, as well as serving as units of QoS specification, also have a runtime representation that is used in ongoing resource management during application execution.

More detail on the Resource Management CF can be found in [15].

5. Conclusions and Future Work

We have argued that existing Grid middleware is poorly equipped to support next-generation Grid applications that are built on heterogeneous and complex networking infrastructures and which involve complex application level services (such as distributed collaborative visualisation). The Gridkit approach, which integrates middleware and overlay networking functionality, explicitly addresses the needs of such applications. In particular, its Interaction Services CF and Resource Discovery/ Management CFs allow multiple interaction types, discovery types, and resource management policies to be simultaneously hosted over multiple overlay network configurations.

Our Gridkit implementation currently consists of three binding types (publish-subscribe, and two RPC binding implemenations: SOAP and IIOP) and two discovery technologies (Service Location Protocol – SLP, and Universal Plug and Play - UPnP).

We also have a collection of overlay networks implemenations as described previously (Chord key-based routing, Chord Distributed Hash Table, Scribe and Probabilistic Multicast). This involves a total of 47 OpenCOM components and component frameworks. We are currently in the process of extending this set 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 [16] as an alternative multicast 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 distributed visualisation scenario that has 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) framework 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 [17], but Gridkit will provide a challenging context for such ideas.

6. References

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

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

[3] 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.

[4] 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.

[5] Balakrishnan, H., Kaashoek, F., Karger, D., Morris, R., Stoica, I., “Looking Up Data in P2P Systems”, CACM, Vol 46, No 2, pp43-48, Feb 03.

[6] Grace, P., Coulson, G., Blair, G., Mathy, L., Yeung, W.K., Cai, W, Duce, D., Cooper, C., “GRIDKIT: Pluggable Overlay Networks for Grid Computing”, Proc. International Symposium of Distributed Objects and Applications (DOA’04), Larnaca, Cyprus, October 2004.

[7] Hughes, D., Coulson, G., Warren, I., “A Framework for Developing Reflective and Dynamic P2P Networks (RaDP2P)”, *Proc. 4th IEEE International Conference on Peer-to-Peer Computing*, Zurich, Switzerland, August 2004.

[**8**] Castro, M., Druschel, P., Kermarrec, A-M., Rowstron, A., “SCRIBE: A Large-Scale and Decentralised Application-Level Multicast Infrastructure”, IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), 2002.

[9] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakarishnan, H., “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”, Proc. ACM SIG-COMM, San Diego, 2001).

[**10**] Cai M., Frank, M., “RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network”, Proc 13th ACM *International World Wide Web Conference (WWW)*, New York, USA, May 2004.

[11] World Wide Web Consortium, “Resource Description Framework”, http://www.w3.org/RDF/, 2004.
[12] Grace, P. Blair, G. S., Samuel, S., “ReMMoC: ”, Proceedings of International Symposium on Distributed Objects and Applications (DOA 2003), Catania, Sicily, November 2003.

[13] Foster, I., Kesselman, C., Tuecke, S., “The Anatomy of the Grid: Enabling Virtual Organizations, International Journal of Supercomputer Applications, Vol 15, No 3, 2001.**[14]** Duran-Limon, H., Blair G.S., “Reconfiguration of Resources in Middleware”, 7th IEEE International Symposium on Object-oriented Real-time Dependable Systems (WORDS 2002), San Diego, CA, January 2002

[15] Cai, W., Coulson, G., Grace, P., Blair, G., Mathy, L., Yeung, W., “The Gridkit Distributed Resource Management Framework”, Submitted to European Grid Conference, Amsterdam, Netherlands, February 2005.
[16] 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.

[17] 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.