A Middleware Approach for Pervasive Grid Environments

Geoff Coulson[1], Paul Grace[1], Gordon Blair[1], David Duce[2], Chris Cooper[2], Musbah Sagar[2]

[1] Computing Department, Lancaster University, Lancaster, LA1 4YR
[2] Dept of Computing, Oxford Brookes University, UK
geoff @comp.lancs.ac.uk, daduce@brookes.ac.uk

Abstract: *Next-generation Grid applications will operate within and across many heterogeneous network types; will employ a wide range of device types ranging from supercomputers to sensor motes; and will require many more "interaction paradigms" than merely RPC and message-passing (e.g., publish-subscribe, multicast, tuple spaces etc.). In this paper, we propose a middleware approach to meeting these emerging needs. Our approach is to provide a highly flexible "overlay network framework" that underpins an extensible set of plug-in interaction paradigms. The middleware is structured using a lightweight run-time component model that enables appropriate profiles to be configured on a wide rage of device types, and facilitates run-time reconfiguration (as required for reasons of adaptation to dynamic environments). For proof of concept, we are exploring a wildfire scenario which involves mobile groups of firefighters, mobile sensors, control centres, and access to parts of the wider fixed Grid for simulation. We are also investigating the application of our approach more generally in the management of the "e-Environment".*

1. Introduction

As Grid computing evolves, there is an accelerating trend towards *diversity* in terms of both end-systems and networked infrastructures. For example, with the emergence of the "pervasive Grid" [Davies,04], we have a spectrum that ranges from cluster systems, through high-speed LAN-based systems, lower-speed WANs, infrastructure-based wireless networks, ad-hoc wireless networks (themselves ranging from relatively static to highly dynamic configurations) that employ PDA-type devices, and specialised sensor networks that employ miniature sensor devices.

In parallel, the range of types of "interaction paradigms" in use at the application level has also burgeoned. Beginning with basic point-to-point interactions (e.g. RPC and SOAP messaging), the range of interaction paradigms is expanding to include (e.g.): reliable and unreliable multicast; workflow; media streaming; publish-subscribe; tuple-space/ generative communication; and peer-to-peer based resource location or file sharing.

In the Open Overlays project [Grace,04], we are seeking to provide a Grid middleware infrastructure that can span and integrate this growing diversity at both the infrastructure level and the "interaction paradigm" level. This clearly cannot be done using standard Grid middleware such as Globus for three main reasons: *i*) current Grid middleware won't run on primitive devices because of its heavyweight and non-profilable nature; *ii*) current middleware is not network-centric—it assumes fixed TCP/IP support and deals only with end-systems; and *iii*) current middleware supports only SOAP messaging and not the range of other interaction paradigms required.

To motivate our work more clearly, consider an application scenario that is being developed by the project which involves forest or savannah fire fighting in a remote region with poor accessibility. In the scenario, *fire fighters* carry PDA-like devices that enable communication with other fire fighters and with on-site *controllers* who coordinate the work. The PDAs support: cameras to give the controllers a view of the fire; GPS to enable location tracking; screens on which text and graphics-based commands from controllers are displayed; and audio capabilities to enable group communication among fire fighters and controllers. In addition, portable environmental sensors are used to provide controllers with information such as wind speed and direction. These are placed by fire fighters and are networked wirelessly. As well as helping to directly inform the controllers, sensor output is fed into computationally-intensive real-time "fire evolution" simulations running in the fixed-infrastructure Grid. These are maintained and monitored by remotely-located *experts* who video-conference among themselves and strategically advise controllers based on longer-term projections of the progress of the fire.

Note that not only does this scenario clearly involve highly heterogeneous device and networking technologies—it also calls for a wide range of interaction paradigms (e.g. reliable ad-hoc multicast for command propagation, stream-based multicast for group audio communication, publish-subscribe for sensor data collection, SOAP-based messaging for communication with objects in the fixed Grid, etc.).

The essence of our approach to addressing the requirements of scenarios such as these is to place a flexible and configurable set of middleware frameworks over a layer of *overlay networks*, and to construct the whole architecture in terms of a lightweight component model that can be implemented on a wide range of device types, including very small devices such as sensor motes. A general definition of overlay networks is that they are virtual communication structures that are logically "laid over" one or more underlying physical networks (such as the Internet and/or a wireless ad-hoc networking environment). The benefits of the overlay approach are that *i*) it can mask the heterogeneity of the underlying networked infrastructure, providing a separation of engineering implementation from high-level functionality; *ii*) it can provide needed network services (e.g. multicast) in network environments that don't support them; and *iii*) it is inherently configurable and run-time adaptive so as to be able to address the high degree of dynamism inherent in our target environments.

The remainder of this paper is structured as follows. First, in section 2, we consider the overall architecture of our

middleware. Then, in subsequent sections we consider three key elements of the architecture: the underlying component model in section 3; the overlay framework in section 4; and the interaction paradigm framework in section 5. Finally, we discuss related work in section 6 and outline areas of future work in section 7.

2. Architecture

Our basic approach to the support of such "pervasive Grid" scenarios is to provide a highly configurable middleware framework the architecture of which is shown in Figure 1.

This architecture, called Gridkit [Grace,04], is built in terms of a component model called OpenCOM v2 [Coulson,04]. This employs a minimal runtime that supports the loading and bindings of lightweight software components at run-time. The runtime is so minimal that it can be supported even on very primitive devices. OpenCOM is used in the construction of all the layers above.

The next layer up is a distributed framework for the deployment of overlay networks as discussed in section 1.

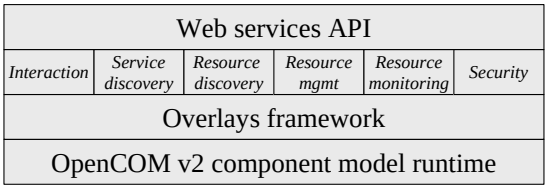

Figure 1: The Gridkit Architecture

Above this is a set of "vertical" frameworks that provide functionality in various orthogonal areas, and can optionally be included or not included on different devices. We discuss only one of these frameworks—the interaction framework—in any detail in this paper. In brief, the rest are as follows: the *service discovery framework* accepts plug-in strategies (e.g. SLP, uPnP, Salutation) to discover WSDL services in the Grid; the *resource discovery* framework accepts plug-in strategies (e.g. peer-to-peer search, Globus MDS) to discover Grid resources such as CPUs and storage; the *resource management* and *resource monitoring* frameworks are respectively responsible for managing and monitoring Grid resources; and the *security* framework provides general security services for the rest of the frameworks. These frameworks are discussed in more detail in [Grace,04].

Finally, above the vertical frameworks is an XML/ SOAP/ WSDL-based API layer that provides access to the underlying frameworks in terms that are familiar to Grid application programmers. This layer is optional and programmers can choose to use the framework APIs directly, and write their code in terms of OpenCOM components, if desired. This possibility, of course, is particularly relevant in the context of primitive resource-poor devices such as sensor elements and even PDAs.

3. The OpenCOM component model

An outline of the component model is illustrated in Figure 2. *Components* are language-independent encapsulated units of functionality and deployment that interact with other components exclusively through "interfaces" and "receptacles" (see below). *Capsules* are containing entities that offer the above-mentioned runtime API. Importantly, capsules can be implemented differently on different devices—e.g. they might be implemented as a Unix or Windows process on a PDA or PC; or directly on top of physical memory on a sensor mote with no OS. Components can be deployed at any time during run-time, and their loading can be requested from within any component within the capsule (this is called *third-party deployment*). *Interfaces* are expressed in terms of sets of operation signatures and associated datatypes; OMG IDL is used for interface specification to give language independence (note, however, that this does *not* imply the overhead of CORBA-like stubs and skeletons.) Components can support multiple interfaces: this is useful in embodying separations of concern (e.g. between base functionality and component management). *Receptacles* are "required" interfaces that are used to make explicit the dependencies of a component on other components: when deploying a component into a capsule, one knows by looking at its receptacles precisely which other components must be present to satisfy its dependencies. Finally, *bindings* are associations between a single interface and a single receptacle. Like deployment, the creation of a binding is inherently third-party in nature. That is, it can be performed by any party within the capsule, not only by the first-party components that will themselves participate in the binding.

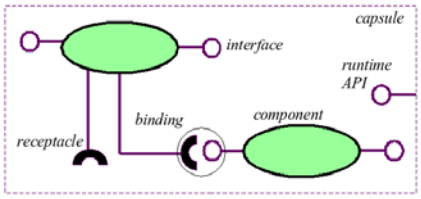

Figure 2: The OpenCOM component model

OpenCOM also supports the notions of *reflection* and *component frameworks*. Reflection is used to reason about component configurations and to dynamically alter configurations at runtime. Component frameworks are scoped compositions of components that accept plug-in components that are validated according to component framework specific constraint rules. The overlays framework and the 6 vertical frameworks discussed above are all implemented as OpenCOM component frameworks. More details are give in [Coulson,04].

The required heterogeneous realisation of the component model in various types of devices is achieved by providing different implementations of the runtime API, and by implementing components themselves in various ways. For example, on a PDA running a standard OS we might implement components as sets of Java classes or as Linux "shared objects"; whereas on a sensor mote's microcontroller, components might be implemented simply as segments of machine code. This is possible because the component model is a *local* model: distribution is built on top of this foundational layer.

4. The overlay framework

The overlay framework supports the design, deployment and management of *plug-in overlay networks* in support of pervasive Grid computing. In practice, this amounts to hosting, in a set of distributed overlay framework instances, a set of per-overlay plug-in components, each of which embodies *i*) a *control* element that cooperates with its peers on other hosts to build and maintain some virtual network topology, and *ii*) a *forwarding* element that appropriately routes messages over its virtual topology.

In terms of *deployment*, the overlay framework allows one to dynamically instantiate new overlays in a straightforward and lightweight manner. This is supported in a recursive fashion by using overlays to deploy overlays. For example, a flooding-based overlay (e.g. Gnutella) can be used to disseminate a message that (a filtered subset of) receiving hosts act upon by deploying a node of a new overlay of some desired type (e.g. an application-level multicast overlay). This is achieved by employing a *stack structure* for overlay implementations, and adopting an associated message handling regime that is inspired by the Ensemble communications framework [vanRenesse,98]. In brief, the forwarding elements of overlays are organised such that when an incoming message is not recognised, it is passed to the forwarding component of the overlay above. Given this arrangement, one can place a 'dummy' overlay at the top of the overlay stack that responds to deployment request messages.

Apart from its use in deployment, the general notion of *stacking* overlays is a powerful one, and there are numerous cases in which one overlay can usefully be employed as a substrate for another. For example, one could layer a keyword search overlay such as Gnutella over a DHT-based network such as Chord (as DHT networks do not support keyword search). Or, one could layer a content dissemination overlay such as TBCP [Mathy,01] over a resilient overlay such as RON [Andersen,01] to enhance dependability. All such scenarios can be achieved very easily using the overlay framework's stacking structure.

As well as stacking whole overlays, the overlay framework also supports *partial stacking* in which the control and forwarding elements can be separately stacked. For example, we have designed a variant of Gnutella that builds a more structured network than the completely unstructured topology constructed by standard Gnutella. This variant can be deployed simply as a control element, and an existing standard Gnutella forwarding component in the layer below can be used directly. Another example of partial stacking could be the stacking of a multicast overlay over a DHT-based overlay. Here, the multicast overlay would only need to provide a forwarding component, as the control element of the underlying DHT overlay could be used directly. Partial stacking not only saves developer effort—it also potentially conserves resources, as functionality common to a set of stacked overlays can be reused, thus saving end-system resources and potentially reducing network traffic.

As well as stacking, the overlay framework also promotes *horizontal composition* between different overlays. For example, a gossip-based overlay can be used to gossip about crashed nodes in a different overlay, and thus be used to provide a general failure detection service for other overlays. Similarly, an overlay that provides a dependability service for the nodes of other overlays could exploit a third overlay to search for suitable hosts on which overlay nodes could be redundantly checkpointed. As a third example, separate infrastructure-based and ad-hoc-based multicast overlays could cooperate side-by-side to underpin a publish-subscribe session that must simultaneously operate in both network environments.

An example of an overlay framework configuration is shown in Figure 3. This also illustrates that the framework can simultaneously support multiple overlays, some of which are related and others of which are not.

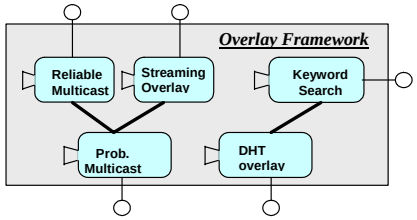

Figure 3. An example overlay configuration

Finally, in terms of the *management* of deployed overlays, the overlay framework employs plug-in 'component configurators' [Kon,00] that builds on a native reflective capability of OpenCOM (i.e. the 'architecture' meta-model [Coulson,04]). But in addition, some management functions can be carried out by overlays themselves. Within a single overlay, it is the responsibility of the control part of the implementation to manage, maintain, and repair the overlay topology. But it is also possible to use specialised overlays to manage other overlays. Examples of this relating to failure detection and dependability have already been given above.

5. The interaction framework

As argued in section 1, Grid middleware that offers only a single interaction paradigm (e.g. RPC) cannot cope with the diversity of application requirements needed by next-generation Grid applications. To address this issue, Gridkit's interaction framework provides a common environment for an extensible set of so-called plug-in interaction paradigms, or PIPs. The overall architecture and context of the interaction framework is illustrated in Figure 4.

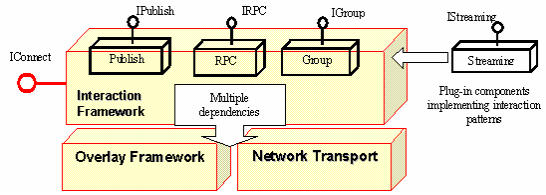

Figure 4: The interaction framework

Architecturally separating the interaction framework from the overlay framework has the effect of promoting the reuse of overlays and thus conserving resources—i.e. different interactions may re-use overlay configurations that are already in place (for example, a topic-based publish-subscribe PIP and

a reliable multicast PIP might both share a multicast tree overlay).

Because of the variety of interaction paradigms and the need to support future extensibility, it is unrealistic to define universal, fixed, interfaces to PIPs. Instead, we adopt an approach to API provision that relies on the definition of an (extensible) set of *generic APIs*. The expectation is that each generic API will be exported by a potentially large family of underlying PIPs. For example, a generic publish-subscribe API can give access to a wide range of plug-ins that implement variations on the publish-subscribe theme (e.g. channel-based, content based etc.). In cases where a PIP requires a modification of the generic API closest to its needs, the framework recommends that interface inheritance is used wherever possible to avoid a proliferation of top-level generic APIs. Avoiding a proliferation of top-level APIs is crucial in giving applications some level of stability and consistency.

The other API-oriented feature that we provide is a lightweight means of "trading" for PIP instances. Details of the "trading" scheme are provided in [Grace,05]. In brief, when a user of the interaction framework wants to create and bind to a PIP, it provides to the *IConnect* API a receptacle for the type of generic API it is looking for. Attached to this receptacle is a predicate which is matched by the framework to a suitable PIP on the basis of a match between the predicate and corresponding name-value pairs that are attached to plugged in PIPs. In addition, predicates may range over additional name-value pairs that are exported by a *context engine*. This enables PIP selection and configuration to be informed by context. For example, if the context engine reported *network_type*: *ad_hoc*, the framework could instantiate a requested publish-subscribe PIP over an ad-hoc multi-hop routing overlay rather than a tree-based multicast overlay which might be used if the context engine reported *network_type*: *infrastructure*.

Additionally, the interaction framework (optionally) supports dynamic monitoring of predicates and name-value pairs so that an exception is raised if any of these change such that the match is no longer valid. In this case either the user or the framework itself can attempt to reconfigure to meet the new circumstances. As an example, the context engine might change a name-value pair to reflect the fact that a live Ethernet MAC layer no longer exists, and the framework might on that basis change an underlying overlay from IP-based flooding to an ad-hoc network based flooding. Again, see section 4 for examples and more detail.

6. Related work

In terms of the basic component-based middleware architecture, there is a substantial body of literature on reconfigurable middleware for pervasive and 'minimal' systems. For example, *Gravity* [Cervantes,04] is a component model built on top of a Java framework for consumer devices; and *DPRS* [Roman,04] and *PCOM* [Becker,04] are other component-based designs for dynamically configurable and reconfigurable pervasive systems. *THINK* [Fassino,02] is a component-based component model that is tailored specifically at building operating system kernels. And finally, *one.world* [Grimm,00] is a system for pervasive applications that supports dynamic service composition, migration of applications and discovery of context. Our approach is related to all of these. However, by being language-independent and by separating the basic component model from the frameworks that are built in terms of it, our approach attempts to be more *generic* than the above systems (e.g. the above systems could themselves be built using OpenCOM).

In terms of overlay networks, there is, of course, considerable research in this field; but our work is largely orthogonal to this: we are primarily interesting in wrapping and composing overlays rather than in developing new ones. Researchers in Toronto have developed a generic platform called *iOverlays* [Li,04] that supports the implementation of overlays. However, it can support only one overlay at a time. The JXTA project [JXTA,05] from Sun is addressing interoperability across different peer-to-peer systems but not the dynamic composition of overlays in a general sense. It is also focused on one particular type of overlay: unstructured peer-to-peer overlays. In a more mainstream Grid context, researchers at Indiana [Pallickara,03] have developed a peer-to-peer messaging service for the Grid that incorporates both JXTA and the Java Messaging Service, but unlike our work this does not address the provision of a lightweight framework for overlay types that can transparently mediate between the range of interaction paradigms needed by applications and the range of network types that are increasingly being used.

7. Status and future work

To date we have implemented the overlay and interaction frameworks and have populated them with a substantial set of plug-ins. In the interaction framework, we have implemented publish-subscribe and group PIPs in C++ and Java respectively (this multi-language integration is straightforward thanks to OpenCOM). We have also implemented IIOP and SOAP-based RPC PIPs (in C++), and a streaming PIP (in Java). In terms of overlay plug-ins, Chord, Scribe and Application Level Multicast (i.e. TBCP [Mathy,01]) have been implemented in Java, and Gossip and Probabilistic Multicast have been implemented in C++. The two frameworks themselves, plus the context engine, are implemented in Java. Mostly, we have used the multi-language integration feature for practical reasons to accommodate more easily into the frameworks software already written.

We already have all the above software running on both PCs and PDAs, and we have just started work on porting OpenCOM to the microcontrollers that are used by Berkeley sensor motes. This builds on the Contiki mote operating system from SICS [Dunkels,04]. This work will be an interesting evaluation of our claim the OpenCOM is sufficiently lightweight to run on the full range of devices in the pervasive Grid.

We have also designed, on top of Gridkit, a *collaborative workspace application* (see Figure 5) which enables graphical communication in our fire fighting scenario between fire fighters and controllers. In more detail, we have designed an architecture involving multiple disjoint groups (e.g. all fire fighters; fire fighters in a given locality; fire fighters to controllers etc.), each of which is underpinned by a distinct PIP/overlay stack. Graphical communication is used to present map information which is overlaid with visualizations of sensor information (including positions of the relevant human actors). Controllers and field workers can sketch on the drawing surface, for example to give an estimate of the

local fire boundary, or to highlight particular features. The application is implemented using web technologies, and Scalable Vector Graphics (SVG) is used for graphical presentation. Information displayed on each display surface is considered to be an annotation of the surface which is represented using the Resource Description Framework (RDF).

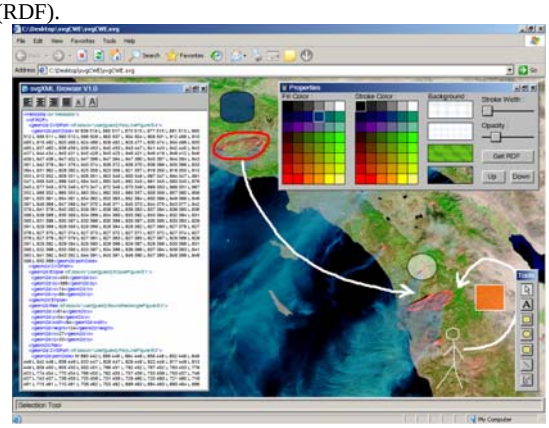

Figure 5: The collaborative workspace application

Although we have made considerable progress, a lot remains to be done. For example, there is a lot more territory to explore in the area of *distributed reconfiguration* of both overlays and PIPs. In particular, there are interesting issues in *cross-layer* distributed reconfiguration that involves intelligent cross-coordinated reconfiguration of both frameworks. For example, a publish-subscribe PIP might be adequately underpinned by an TBCP overlay while most or all of its users are situated in the fixed network; but if the situation evolves so that at some point a significant number of users are situated in ad-hoc network environments, then the optimal underpinning of the PIP needs to be reconsidered and could perhaps be better supported by a coordinated federation of horizontally-composed overlays.

Additional areas of challenge that we are addressing are the use of Model Driven Architecture concepts to configure our frameworks and also to provide formally-specified constraints on their reconfiguration; and the use of autonomic techniques so that the frameworks can not only adapt to changing environmental conditions but can also learn from prior adaptations and make better decisions on that basis.

Finally, we are looking at extending our applications work beyond the fire fighting scenario to a more general focus on managing the "e-Environment". To this end, we are forming a collaboration of leading Environmental Scientists from the Lancaster Environment Centre (LEC), the Centre for Ecology and Hydrology, and the Proudman Oceanographic Laboratory, along with leading technology providers from InfoLab21 (Lancaster University), the University of Manchester, and CCLRC Daresbury, to study the use of pervasive Grid technology in the specific area of water management. This particularly features the linkage of sensor networks and large scale environmental modelling components to provide comprehensive support for concerns such as flood forecasting and water quality control.

References

[Andersen,01] Andersen, A., Blair, G., Goebel, V., Karlsen, R., Stabell-Kulø, T., Yu, W., "Arctic Beans: Configurable and Reconfigurable Enterprise Component Architectures", IEEE Distributed Systems Online, Vol. 2, No. 7, 01.

[Becker,04] Becker, C., Handte, M., Schiele, G., Rothermel, K., "PCOM – A Component System for Pervasive Computing," Proc 2nd International Conference on Pervasive Computing and Communications, Orlando, Florida, March 04.

[Cervantes,04] Cervantes, H., Hall, R., "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model," Proc. 26th International Conference of Software Engineering (ICSE 2004), Edinburgh, Scotland: ACM Press, pp 614–623, May 04.

[Coulson, 04] Coulson, G., Blair, G.S., Grace, P., Joolia, A., Lee, K., Ueyama, J., "A Component Model for Building Systems Software", Proc. IASTED Software Engineering and Applications (SEA'04), Cambridge, MA, USA, Nov 04.

[Davies,04] Davies, N., Friday, A., Oliver Storz, O., "Exploring the Grid's Potential for Ubiquitous Computing", IEEE Pervasive Computing, Vol 3, No 2, 2004.

[Dunkels,04] Dunkels, A., Grönvall, B., Voigt, T., "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. Proc. 1st IEEE Workshop on Embedded Networked Sensors 2004 (IEEE EmNetS-I), Tampa, Nov 04.

[Fassino,02] Fassino, J.-P., Stefani, J.-B., Lawall, J., Muller, G., "THINK: a software framework for component-based operating system kernels," 2002 USENIX Annual Technical Conference. Monterey, CA: USENIX, pp 73–86, June 02.

[Grace,04] Grace, P., Coulson, G., Blair, G., Mathy, L., Yeung, W.K., Cai, W, Duce, D., Cooper, C., "GRIDKIT: Pluggable Overlay Networks for Grid Computing", In Proceedings of Distributed Objects and Applications (DOA'04), Cyprus, Oct 04.

[Grace,05] Grace, P., Coulson, G., Blair, G.S., Porter, B., "Deep Middleware for the Divergent Grid", submitted to IFIP/ACM/USENIX Middleware 2005, April 3rd 2005.

[Grimm,00] Grimm, R., Anderson, T., Bershad, B., Wetherall, D., "A system architecture for pervasive computing," Proc 9th ACM SIGOPS European workshop, ACM Press, pp. 177–182, 2000.

[JXTA,05] http://www.jxta.org/, 2005.

[Kon,00] Kon, F., Roman, M., Liu, P., Mao, J., Yamane, T., Magalhaes, L., Campbell, R., "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB", Proc. of Middleware 2000, ACM/IFIP, April 00.

[Li,04] Li, B., Guo, J., Wang, M., "iOverlays: A Lightweight Middleware Infrastructure for Overlay Application Implementations", Proc. IFIP/ACM/USENIX Middleware 2004, Toronto, Canada, 2004.

[Mathy,01] Mathy, L., Canonico, R., Hutchinson, D., "An Overlay Tree Building Control Protocol," Proc. 3rd International COST264 Workshop on Networked Group Communication, London, UK, 2001.

[Pallickara,03] Pallickara, S., Fox, G., "NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids", Proc IFIP/ACL/USENIX Middleware 03, Rio, Brazil, 2003.

[Roman,04] Roman, M., Islam, N., "Dynamically Programmable and Reconfigurable Middleware Services," Proc. Middleware '04, Toronto, Oct 04.

[van Renesse,98] van Renesse, R., Birman, K., Hayden, M., Vaysburd, A., Karr, D., "Building Adaptive Systems Using Ensemble", Software Practice and Experience. Vol 28, No 9, pp 963-979, Aug 98.