

Research Directions in Reflective Middleware: the Lancaster Experience

Gordon S. Blair
Lancaster University
Computing Department,
Bailrigg, Lancaster LA1 4YR, UK
+44 1524 65201
gordon@comp.lancs.ac.uk

Geoff Coulson
Lancaster University
Computing Department,
Bailrigg, Lancaster LA1 4YR, UK
+44 1524 65201
geoff@comp.lancs.ac.uk

Paul Grace
Lancaster University
Computing Department,
Bailrigg, Lancaster LA1 4YR, UK
+44 1524 65201
p.grace@lancaster.ac.uk

ABSTRACT

In this paper, we survey three generations of reflective middleware research carried out at Lancaster University, present experiences gained from this research, and highlight a number of important areas of future research. In particular, we discuss the extension of our reflective middleware ideas in terms of both depth and breadth. The depth extension applies reflective middleware principles to systems that lie beneath the traditional middleware domain: e.g. operating systems and networks. The breadth extension then applies the principles in a much broader range of application areas than those traditionally considered in reflective middleware research. These include reflective middleware for Grid computing and for sentient-object-based real-time control systems. We also briefly consider future work in applying our approach to the development of self-managing systems.

Keywords

Reflective middleware, multi-model approach, deep middleware, case studies, autonomic computing.

1. INTRODUCTION

Reflection [Kiczales,91] has now emerged as an important technique in the support of more configurable and re-configurable middleware. A number of experimental reflective middleware platforms have been developed and discussed in the literature [Kon,02]. The results of this research are also being incorporated in industrial-strength middleware; for example the design of JBoss 4.0 [JBoss,04] is strongly based on this work.

The authors of this paper have been heavily involved in this research over the last few years. In this paper, we chart the various generations of reflective middleware developed at Lancaster, and highlight a number of important ongoing research areas for the subject.

2. REFLECTIVE MIDDLEWARE AT LANCASTER UNIVERSITY

2.1 Three Generations of Systems

To date, three distinct generations of reflective middleware have been developed at Lancaster:

- *Generation 1: Early Prototypes.* In this early phase, the Python language was used to construct rapid prototypes of reflective middleware platforms. This work involved the

development of our multi-model approach and the definition of four orthogonal reflective meta-models (interface, architecture, interception and resources) [Costa,00].

- *Generation 2: Open ORB.* This phase involved the design and implementation of an experimental reflective CORBA platform with the goals of *i)* demonstrating the benefits of configurability and reconfigurability in the middleware domain, and *ii)* investigating techniques for the efficient implementation of reflective middleware. The end product was a CORBA platform that supported a radical binding framework in which individual ‘interaction types’ (e.g. streaming, messaging, transactions etc.) could be specialised for different classes of application and which also supported openness and adaptation in terms of its internal structure (e.g. in terms of protocol and resource frameworks). This platform was shown to have performance characteristics similar to commercial ORBs (but with the added benefits mentioned above), demonstrating that reflective middleware platforms can indeed be efficiently implemented [Blair,01].
- *Generation 3. Towards Middleware Independence.* In this third phase, we have investigated the role of reflection in supporting independence from any particular middleware platform or paradigm. In other words, we do not constrain applications to use (for example) CORBA; rather, we use reflection to dynamically discover the styles of middleware required in a given context and then automatically configure the middleware framework to support such styles of interaction. This approach is particularly applicable in highly heterogeneous and/ or dynamic environments and has been primarily demonstrated in a mobile setting [Grace,03].

2.2 A Summary of the Current Approach

Our current approach (as used in the third of the above-mentioned generations) is based on three key concepts: *components*, *component frameworks* and *reflection* as described below.

Component-based approaches are currently very popular in distributed systems at the application level to support properties such as third party composition, deployment and re-use. However, in our approach, we also adopt components at the level of the middleware platform itself. In other words, both the middleware platform and the application are uniformly constructed in terms of a set of interconnected components.

This basic structure is then supplemented by the coarser-grained structuring of *component frameworks*, such that a middleware

platform is composed of a set of frameworks each of which represents some aspect of the required functionality or structure (e.g., protocol frameworks, dispatching of incoming calls, resource management and scheduling, etc). Typically, these component frameworks accept ‘plug-in’ components that add or extend behaviour (e.g. in terms of pluggable protocols). Component frameworks are themselves components, thus facilitating the construction of nested structures. Component frameworks also embody domain-specific knowledge about the architecture of a given sub-structure—this is a crucial factor in maintaining the integrity of the platform during periods of change.

Reflection is then used to support introspection and adaptation of the underlying component/ component framework structures. In the spirit of our multi-model approach, three reflective meta-models are now supported (the resources meta-model that we defined in generation 1 now no longer exists—this is simply modeled as a particular middleware component framework):

1. The *interface meta-model* supports the dynamic discovery of the set of interfaces defined on a component; support is also provided for the dynamic invocation of methods defined on these interfaces.
2. The *architecture meta-model* enables the programmer to both discover and adapt the architecture of an underlying component framework. A meta-level representation of the architecture is provided in terms of components, inter-connections and architectural style rules.
3. The *interception meta-model* supports the dynamic interception of incoming method calls on interfaces and also the association of pre- and post-method-call behaviour.

We also provide a lightweight and efficient component technology supporting the above design, i.e. OpenCOM. This was initially based on a minimal subset of COM (particularly the *vtable* structure) but has now evolved to be independent of this software so that it also runs in Unix-based environments.

Further details of this design can be found in the literature [Coulson,02]. The software is also available on-line (<http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/software.php>) for experimentation by other research groups (in particular, we provide access to the core reflective component technology and an ever-expanding set of component frameworks).

2.3 Experiences from this Research

In general, this work has been very successful and in particular we have demonstrated:

1. The key role of reflection in providing a more open and flexible approach to the construction of middleware platforms (as demanded by many contemporary application

domains);

2. The fact that reflection does not necessarily incur a performance overhead when compared to standard technologies [Coulson,04a];
3. That reflective middleware can deliver on platform independence [Grace,03].

We have also demonstrated the utility of the approach in a number of areas, most notably mobile computing [Grace,03].

3. FUTURE RESEARCH DIRECTIONS

3.1 Going Deep

The essence of reflective middleware is to provide open access to key aspects of the internals of the middleware platform. In some platforms, this is restricted to fairly minimal (and shallow) aspects such as the arrival or dispatching of incoming calls. In others, more comprehensive access is provided to this underlying engineering. At Lancaster, we are currently experimenting with pushing this concept further with what we refer to as *deep middleware*. In this approach, we attempt to open up low level aspects of the system including key functionality normally considered as being located in the underlying operating system or indeed in the network itself. In some ways, our resources meta-model was an early instantiation of this concept, providing the ability to introspect and adapt resource allocation to both application-level and middleware-level tasks, and also the management of these resources (a similar approach has also been investigated by the Think project at INRIA [Fassino,02]). We are now investigating the application of such ideas in the complementary area of networking.

In the *Open Overlays* project, we are investigating the implementation of overlay networks as component frameworks within the middleware platform. The motivation here is to enable configurability and reconfigurability of core networking functionality without having to make any particular assumptions about the underlying IP network.

In terms of background, overlay networks are virtual communications structures that are logically ‘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 network (in principle both in the core and at the network edge). Overlays have to date mainly been used in two areas: *i*) to alleviate the effects of slow or sporadic deployment of new services in the Internet (e.g. application-level multicast); 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). Examples of overlay types are: reliable multicast overlays such as SRM; content dissemination networks; unstructured peer-to-peer overlays such as Gnutella; structured dynamic hashtable (DHT)-based peer-to-peer overlays such as Chord; resilient overlay networks (RONs); and the routing overlays used in ad-hoc or wireless sensor networks. See [Grace,04] for a survey.

In our design, we offer a generic component framework for overlay network deployment as shown in figure 1 below.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference '04, Month 1–2, 2004, City, State, Country.
Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

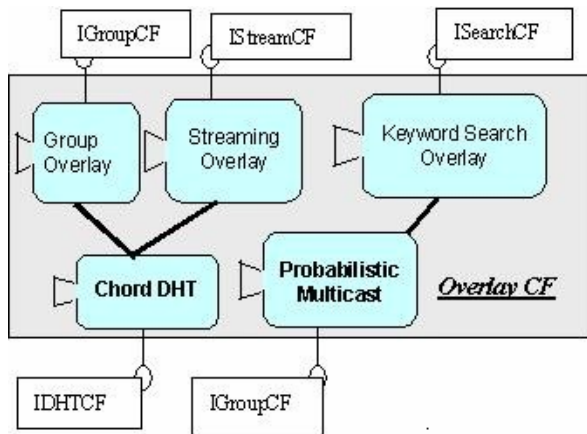


Figure 1. Generic Structure of the Overlay CF

Figure 1 shows an Open Overlays component framework configuration that involves a two multi-layered overlay instantiations: a group and streaming overlay supported by an instance of the Chord DHT overlay [Stoica,01]; and a keyword search overlay supported by probabilistic multicast. As can be seen, multiple overlay networks can co-exist within a single middleware platform instance; and it is also possible to layer overlays on top of overlays to construct higher-level, more application-specific semantics.

The component framework additionally supports finer-grained compositions of overlays by requiring that overlay plug-ins are structured in terms of three sub-components. 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 structure, 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 [Hughes,04].

Essentially, the Open Overlays work continues the direction begun in Open ORB of providing an extensible set of interaction types; but the availability of network-level support considerably extends the richness and scope of the interaction types that can be made available (e.g. into areas of resource discovery, peer-to-peer file sharing, efficient wide area publish-subscribe, wide area multicast etc).

But the ‘deep’ approach also offers the potential for intimate access to sub-overlay networking functionality around and even below the IP level. In this area we are investigating, in a second project, a more radical application of our approach. In this project, called *Netkit*, we are building component frameworks for generic network elements, including routers, in support of programmable networking functionality. Here we are not only addressing out-of-band control and management concerns: we are deploying OpenCOM components and component frameworks even in the ‘fast path’ of low level packet scheduling, routing and forwarding. Furthermore, we are not only addressing commodity PC-based routers but also dedicated Network Processors [NP,04] that are

intended to provide line-speed routing functionality in the core network.

Clearly, performance and memory footprint are crucial issues in this area. Because of these constraints we have recently completely re-engineered our OpenCOM runtime in terms of a microkernel structure in which the use of previously ‘core’ OpenCOM functionality is made optional. For example, the reflective meta-models themselves become optional and loadable-on-demand. We have also designed component frameworks that encapsulate certain fundamental low-level functionality offered by the OpenCOM runtime—e.g. loading components and creating bindings between interfaces—so that we can provide such functionality as plug-in components. Doing this allows us to directly leverage low-level mechanisms in the underlying ‘deployment environment’ (e.g. a Network Processor) without incurring any OpenCOM related overhead. For example, on the Intel IXP1200 Network Processor, we provide a plug-in component binder that leverages dedicated bus hardware to enable communication between components. Using this approach we can build packet forwarders that perform as well as dedicated software—while additionally offering all the usual benefits of configuration, reconfiguration etc. More detail on this work is available in the literature [Coulson,03], [Ueyama,03].

3.2 Going Wide

Many claims have been made about the benefits of reflective middleware but, in truth, these claims have not yet been fully substantiated. In practice, experiments have focused on rather narrow and in some ways obvious areas (most notably mobile computing). While this is beneficial, it is also important for the subject to validate the ideas more generally. Therefore, we are currently investigating the applicability of the reflective middleware approach in a wide range of additional application domains.

Most notably, we are investigating the role of reflective middleware in Grid computing. The middleware that has so far emerged to support Grid applications can be viewed as falling into two generations. The first of these was exemplified by the Globus 2 toolkit which provided a loosely-coupled set of tools (e.g. for discovering Grid resources, providing security, initiating remote job execution etc.). The second generation then attempted to structure these tools in a more ‘architected’ way by subsuming them under a Web Services-derived ‘service oriented architecture’ (see [Coulson,04b] for more detail).

From the perspective of the broader middleware community, however, these platforms—even the second generation ones—still appear quite primitive. First, they are extremely limited, in comparison to object-based middleware platforms, in terms of *i*) the provision of generic services (cf. CORBA’s fault tolerance, persistent state, automated logging, load-balancing etc., services), and *ii*) scalability and performance (cf. EJB and CCM). Second, they have little or no support for QoS specification and realisation as required by sophisticated Grid applications [Coulson,04b]. We believe that a prime cause of this deficiency is an over-reliance by these platforms on SOAP as a communications engine. Although very flexible and general, SOAP clearly shows its limitations when relied on exclusively to support large-volume scientific datasets; and its lack of support for interaction types other than messaging and request-reply interactions is also a severe

limitation. Third, the current platforms are monolithic: they have zero support for flexible configuration and reconfiguration as found in modern reflective middleware platforms and thus (for example) cannot realistically be deployed on ‘limited’ hardware platforms such as PDAs.

In the *GridKit* project [Coulson,04b], we are applying the architectural ideas discussed in this paper to provide a “third generation” Grid middleware technology.

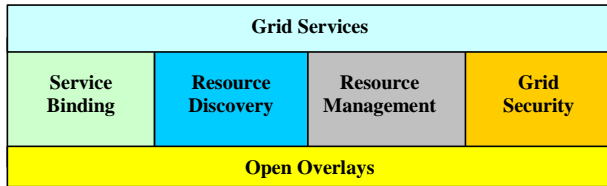


Figure 2. The Scope of Gridkit

As illustrated in figure 2, the vision of Gridkit is to provide middleware support in each of four ‘domains’ which we identify as key in supporting Grid applications. These domains, which are, of course, captured as component frameworks, are as follows:

- Service binding. This hosts pluggable interaction types and also provides generic APIs that allow the application programmer to uniformly create and use instances of selected interaction types.
- Resource discovery. This provides service discovery and, more generally, resource discovery services. It supports the use of multiple pluggable discovery technologies to maximise the flexibility available to applications. Examples of alternative technologies are SLP, UPnP, Jini or Salutation for more traditional service discovery, GRAM [Czajkowski,98] for CPU discovery in a Grid context, and peer-to-peer 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 the fine-grained local resource management (e.g. of channels, threads, buffers etc) that is required to build end-to-end QoS.
- Grid security. This hosts pluggable services that support secure communication between participating nodes orthogonally to the interaction types in use.

As well as being directly available to application developers, these frameworks can easily be combined to provide more complex middleware capabilities. For example, service bindings can integrate with Grid security to produce secure interactions. Note that all four domains are underpinned by the Open Overlays framework discussed above (which itself can be underpinned by the Netkit services where required).

We are also investigating the applicability of the reflective middleware ideas in embedded control systems. The vision of the *Cortex* project [Sivaharan,04] is that future mission-critical computer systems will be comprised of so called *sentient objects*. In broad terms, sentient objects consume events from a variety of different sources including sensors and event channels, fuse these to derive higher-level contexts, reason about these using expert system logic (based on a CLIPS inference engine), and produce output events whereby they actuate the environment or interact

with other objects. In more detail, the project is exploring the area of autonomous vehicle navigation in which vehicles, represented as mobile sentient objects, have the objective of traveling along a given path, defined by a set of GPS waypoints. Every vehicle acts as a sentient objects that cooperates with other vehicles (sentient objects) by inter-vehicle communication mechanisms and with other infrastructure objects (e.g. traffic lights or speed signals).

As expected, the Cortex work partitions the concerns involved in this application area into different component frameworks. More specifically, the Cortex platform consists of an inter-sentient-object-communication component framework (this in turn supports publish-subscribe and group interaction types), and a context component framework. The latter provides the facility for supporting a range of inference engines and sensor fusion algorithms that may be selected at runtime. For example, one fusion component provides algorithms to fuse sensor data from GPS, ultrasonic, compass and context events received via event channels and derive higher-level contexts. The fusion component algorithms include Gaussian modeling and dead-reckoning, together with home grown algorithms to fuse noisy sensor data and to help build a more accurate real time ‘image’ of the environment.

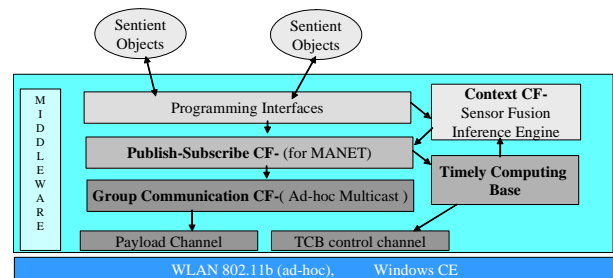


Figure 3. The Cortex Middleware

In more detail, the Cortex middleware, which is illustrated in figure 3, consists of a ‘timely computing base’ for real-time interaction on which are layered component frameworks for ad-hoc multicast and publish-subscribe (these are notionally an instantiation of the Open Overlays framework). Alongside these are component frameworks for resource and QoS management, and the sensor fusion inference engine.

Finally, we are interested in the role of reflective middleware in emerging areas such as ubiquitous computing and also environmental informatics (including wireless sensor networks).

3.3 Self-management

There is growing interest in the distributed systems community in the general area of self-repairing, self-healing or self-organizing software systems [Schmerl,02]. Self-management is clearly attractive for application domains such as mobility and ubiquitous computing but it is also potentially applicable to other areas such as the Cortex work referred to above.

Our contention is that a major prerequisite for self-management is the *openness* of systems. In other words, to support self-management, it is necessary to have access to various aspects of the system infrastructure and to be able to reconfigure such aspects at run-time. It is also important that such changes do not endanger the overall integrity of the (running) system. Clearly,

these properties are provided, at least at a basic level, by our general architectural philosophy of components, component frameworks, and reflection.

Building on this, we have developed a generic ‘self-management’ component framework [Blair,02] that supports the injection of monitoring and adaptation behaviour into the meta-space of a system. In this framework, policies for *monitoring* and *adaptation strategy selection* are expressed as timed automata, which then map directly on to *management components* which act as timed automata interpreters at run-time. These then interface to other components in the system using event notification, i.e. they register for events of interest, receive events, react to them and then emit events to interested parties. The use of timed automata has the benefit of allowing us to carry out formal analysis of the self-adaptation behaviour. Furthermore the inherent composability of timed automata allows us to build larger self-managing systems by aggregating multiple smaller systems.

In our current work on self-management we are applying the timed automata approach to several of the areas discussed in this paper. In particular, self-management is playing a large part in the automatic (re)configuration of overlay networks in the Open Overlays project.

4. CONCLUDING REMARKS

This paper has surveyed and evaluated three generations of reflective middleware research at Lancaster and has discussed a number of ways in which we are carrying this research forward into the future. In the ‘depth’ dimension, we believe that there is great potential in developing future systems that are ‘vertically integrated’ and can be seamlessly inspected and adapted as a unified ‘pool’ of component-based functionality. Already in our existing prototypes applications merge into the middleware; in the future, we see the middleware similarly merging into the underlying operating systems and network infrastructures. This promises great benefits in terms of optimally configuring systems (e.g. minimising memory footprints in resource-poor deployment environments like PDAs and wireless sensor network elements) and providing maximal scope for run-time adaptation and self-management. Of course it should also be mentioned there are numerous integrity-related issues here that must be addressed by future research.

We have also outlined our activities in the ‘breadth’ dimension. These activities are proving to be an interesting testing ground for reflective component-based approach to systems building. In particular, the Gridkit work is demanding in terms of interoperability, security and maximal flexibility, while the Cortex work is demanding in terms of the complementary areas of real-time behaviour, reliability and embedded environments. In the future we plan to extend our activities here in to the areas of wireless sensor networks. This area will simultaneously stretch us in the depth and breadth dimensions as well as providing interesting challenges in the areas of self-management.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of the large team of researchers who are currently working in the areas touched on by this paper. In particular, we single out Wei Cai, Danny Hughes, Ackbar Joolia, Kevin Lee, Nikos Parlavantzas,

Georgios Samartzidis, Thirunavukkarasu Sivaharan, Jo Ueyama, and Wai-Kit Yeung. We would also like to thank the UK EPSRC for their kind support in funding this research.

6. REFERENCES

- [Blair,02] Blair, G.S., Coulson, G., Blair, L., Duran-Limon, H., Grace, P., Moreira, R., Parlavantzas, N., “Reflection, Self-Awareness and Self-Healing”, Proc. First ACM Workshop on Self-healing Systems (WOSS’02), held in conjunction with ACM SIGSOFT ’02, Charleston, South Carolina, USA, November 18th 2002.
- [Kiczales,91] Kiczales, G., J. des Rivières, D.G. Bobrow, “The Art of the Metaobject Protocol”, MIT Press, 1991.
- [JBoss,04] The JBoss Project: <http://www.jboss.org/index.html>.
- [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.
- [Costa,00] Costa, F. Duran, H., Parlavantzas, N., Saikoski, K., Blair, G.S., Coulson, G., “The Role of Reflective Middleware in Supporting the Engineering of Dynamic Applications”, In Reflection and Software Engineering, Cazzola, W., Stroud, R. and Tisato, F. (Eds), Springer-Verlag, LNCS Vol. 1826, pp 79-98, 2000.
- [Blair,01] Blair, G.S., Coulson, G., Andersen, A., Blair, L., Clarke, M., Costa, F., Duran-Limon, H., Fitzpatrick, T., Johnston, L., Moreira, R., Parlavantzas, N., Saikoski, K., “The Design and Implementation of OpenORB v2”, IEEE DS Online, Special Issue on Reflective Middleware, Vol. 2, No. 6, 2001.
- [Coulson,03] Coulson, G., Blair, G.S., Hutchison, D., Joolia, A., Lee, K., Ueyama, J., Gomes, A.T., Ye, Y., “NETKIT: A Software Component-Based Approach to Programmable Networking”, ACM SIGCOMM Computer Communications Review (CCR), Vol 33, No 5, October 2003.
- [Coulson,04a] Coulson, G., Blair, G.S., Grace, P., “On the Performance of Reflective Systems Software”, Proc. International Workshop on Middleware Performance (MP 2004), April, 2004, Phoenix, Arizona; Satellite workshop of the IEEE International Performance, Computing and Communications Conference (IPCCC 2004), 2004.
- [Coulson,04b] Coulson, G., Grace, P., Blair, G.S., Mathy, L., Duce, D., Cooper, C., Yeung, W.K., Cai, W., “Towards a Component-based Middleware Framework for Configurable and Reconfigurable Grid Computing”, Proc. Workshop on Emerging Technologies for Next Generation Grid (ETNGRID-2004), associated with 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004), Modena, Italy, June 2004.
- [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.
- [Coulson,02] Coulson, G., Blair, G.S., Clark, M., Parlavantzas, N., “The Design of a Configurable and Reconfigurable Middleware Platform”, ACM Distributed Computing Journal, Vol 15, No 2, pp 109-126, April 2002.

[Fassino,02] Fassino, J.-P., Stefani, J.-B., Lawall, J., Muller, G., "THINK: A Software Framework for Component-based Operating System Kernels", Usenix Annual Technical Conference, Monterey (USA), June 10th-15th, 2002.

[NP,04] The Network Processing Forum:
<http://www.npforum.org/>.

[Sivaharan,04] Sivaharan, T., Blair, G., Friday, A., Wu, M., Duran-Limon, H., Okanda, P., Sorensen, C., "Cooperating Sentient Vehicles for Next Generation Automobiles", Proc. 1st International Workshop on Applications of Mobile Embedded Systems (WAMES'04), Boston, June 2004.

[Hughes,04] 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.

[Stoica,01] 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).

[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", submitted to Distributed Objects and Applications (DOA'04), June 2004.

[Schmerl,02] Schmerl, B., Garlan, D., "Exploiting Architectural Design Knowledge to Support Self-repairing Systems", Proc. 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July 2002.

[Czajkowski,98] Czajkowski, K., Foster, I., Karonis, N., Carl Kesselman, C., Martin, S., and Smith, W., and Tuecke, S., "A Resource Management Architecture for Metacomputing Systems", Proc. Workshop on Job Scheduling Strategies for Parallel Processing, pp 62-82, Springer-Verlag, ISBN 3-540-64825-9, 1998.

[Ueyama,03] Ueyama, J., Schmid, S., Coulson, G., Blair, G.S., Gomes, A.T., Joolia A., Lee, K., "A Globally-Applied Component Model for Programmable Networking", Proc. International Workshop on Active Networks (IWAN 2003), Kyoto Japan, 10-12 Dec 2003.