A Distributed Architecture Meta-Model for Self-Managed Middleware

Paul Grace
Computing Department
Lancaster University
Lancaster, UK

p.grace@lancaster.ac.uk

Geoff Coulson
Computing Department
Lancaster University
Lancaster, UK

geoff@comp.lancs.ac.uk

Gordon S. Blair
Computing Department
Lancaster University
Lancaster, UK

gordon@comp.lancs.ac.uk

Barry Porter
Computing Department
Lancaster University
Lancaster, UK

porterbf@comp.lancs.ac.uk

ABSTRACT

Openness and adaptation are the fundamental properties of reflective middleware platforms. Self-managed or autonomic systems require this behaviour, and therefore, reflective middleware platforms are ideally suited to the support of such systems. However, new classes of self-managed applications increasingly require support for co-coordinated, *distributed* reconfiguration, and there is limited provision for this in current reflective middleware approaches. In this paper, we document a general, flexible architecture meta-model for the safe and valid adaptation of components that make up the implementation of co-ordinating middleware nodes distributed across peer devices. We also investigate the flexibility of this approach in supporting different reconfiguration types in different environmental conditions.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Patterns (Reflection).

General Terms

Management, Design.

Keywords

Reflection, middleware, self-managing systems.

1. INTRODUCTION

There is growing interest in the distributed systems community in the general area of self-repairing, self-healing or self-organizing software systems [12] (often referred to as self* systems). Our contention is that a major prerequisite for self-

management is the *openness* of systems [2]. 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. Hence, reflective middleware is ideally suited to support self-managed systems. Here, we investigate improvements in reflective middleware approaches to better support new classes of self-managed applications.

Reflective middleware solutions now follow a well-established design approach that typically combines components, component frameworks, and reflection [1,11]. *Software components* are third-party deployable units of composition that form the building blocks of middleware implementations. *Component frameworks* (CFs) then manage a set of components related to a particular domain of middleware operation; and *reflection* supports introspection and adaptation of the software architecture. So far reflective middleware solutions have generally focused on *node-local adaptation*; i.e., where each node makes local decisions to adapt based upon environmental context inputs and locally maintained policies. However, these approaches do not fully support next generation self-managing applications: e.g., autonomic computing, peer-to-peer computing, ubiquitous computing and ad-hoc mobile computing, all fundamentally require the co-ordinated adaptation of middleware behaviour across distributed nodes. To meet this requirement, reflective middleware solutions must support the following:

- *Open access.* Required to inspect the structure and behaviour of co-ordinating distributed nodes. The information about the current structure of distributed nodes can help make decisions about the appropriate adaptation to employ.
- *Consensus.* Coordinating middleware nodes require mechanisms to make decisions about what actions to take; particularly what reconfigurations are required.
- *Safe, valid, distributed reconfiguration.* Adaptations must be made to distributed topologies of components. These adaptations must be made when the distributed system is in a safe state, and changes must be validated once the update is complete.
- *Flexibility.* Coordinated adaptation implementations must be tailored to the deployment domain. For example, some consensus and reconfiguration protocols may be too

resource-intensive for sensor networks. Similarly, centralized management of reconfiguration is not suited to p2p applications.

In this paper we investigate a flexible, distributed meta-architecture protocol that supports open access to distributed middleware component topologies, and provides mechanisms for safe, valid, distributed adaptation. This work is an extension of the OpenORB/OpenCOM [1] approach to reflective middleware. Furthermore, we demonstrate the flexibility of the approach in diverse application types and environmental conditions.

The remainder of this paper is structured as follows. Section 2 illustrates examples of different reconfigurations from a set of self-managed application domains. Section 3 investigates how local reconfiguration is currently managed. Section 4 then examines the proposed flexible meta-architecture protocol, and how it extends the node-local approach to better support distributed middleware adaptation. Finally, in section 5 we describe related research in the field of distributed reconfiguration, and draw conclusions in section 6.

2. RECONFIGURATION SCENARIOS

2.1 Mobile Ad-hoc Computing

A large class of mobile applications involve collaboration between groups of mobile users. Examples include peer-to-peer data sharing, shared workspaces and multimedia conference applications. Here are two concrete examples of distributed reconfigurations that are applicable to such applications:

- *A multimedia conferencing application.* The application employs ad-hoc devices receiving a shared multicast video stream. A change in the current network bandwidth requires a change in how the data is encoded. Therefore, a change in the common media filter needs to be agreed and then applied so all members of the multicast can receive and view video frames when the sender changes the filter.
- *A p2p messaging application.* A set of mobile nodes are participating in a group messaging application based upon a group multicast service. If a message sender changes from sending text messages, to picture messages, or video messages, then the members of the group must change to be able to receive streaming data. That is, the interaction type has changed from message based to streaming based. A local change at the sender only would affect the remainder of the multicast group who would be unable to receive the new messages.

The first is a fine-grained reconfiguration within a middleware service. The second is a more coarse-grained change of the properties of the middleware service itself.

2.2 Overlay networks

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 the core and edges). They basically consist of two parts: one part builds and maintains some kind of virtual network topology, and the other part routes messages over this virtual topology.

We have investigated the development of reconfigurable overlay networks and have also explored different types of reconfiguration, the possibility of which have emerged from the implementation concept of splitting each overlay's per node implementation into three software components [8]: i) a control component that maintains the virtual topology, ii) a forwarding component that handles the message routing algorithm, and iii) a separated state component that stores data for the other two (allowing them to be reconfigured without state transfer). We now discuss three types of overlay reconfiguration. All involve the replacement of either the control component, or forwarder component (typically on every member of the overlay).

- Topology reconfiguration: changing the control component on every node to create a different network topology. This may optimize a network's performance as the available resources increase or decrease.
- Dependability reconfiguration. The control component can also be reconfigured on selected nodes of the overlay network to increase dependability; this can take the form of replacing the repair algorithms to better cope with increasing network node failure.
- Routing reconfiguration. The routing algorithm of the network can be updated by replacing every forwarder component instance in the network. For example, adapting the forwarding element of an overlay network for ad-hoc routing for changing mobility models.

2.3 Summary of Requirements

Applications in different domains such as those discussed above require *coordinated adaptation* of middleware level implementations in order to provide consistent performance levels to members of multi-party applications. Typically, these take the form of reconfiguring software components that reside on distributed nodes. Therefore, we require a principled, generally applicable approach to support adaptation of component-based middleware in a distributed environment. We now investigate if the general, well-understood methods within reflective middleware can be extended to better support the development of self* middleware behaviour.

3. LOCAL RECONFIGURATION

3.1 Background on Reflection

In middleware platforms, two (complementary) styles of reflection have emerged, namely structural and behavioural reflection.

- *Structural reflection* is concerned with the underlying structure of objects or components, e.g., in terms of interfaces supported. Some systems provide architectural reflection, whereby the software architecture of the system can be reified and altered [1, 4], e.g. in terms of components and connectors.
- *Behavioural reflection* is concerned with activity in the underlying system, e.g. in terms of the arrival and dispatching of invocations. Typical mechanisms provided include the use of interceptors that support the reification of the process of invocation and the subsequent insertion of pre- or post- actions. Other systems provide similar capabilities through dynamic proxies [14].
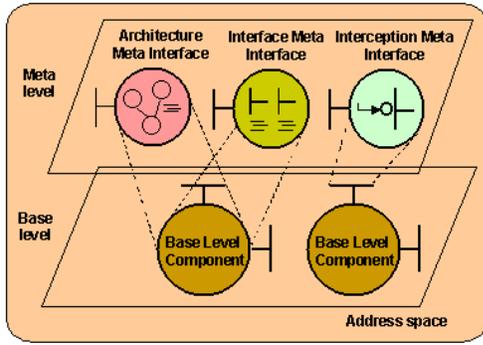
Figure 1: The meta-space structure of Open ORB

The Lancaster meta-space model [1] (illustrated in figure 1) is inspired by these styles. This model is made available to middleware developers through the OpenCOM runtime [5]. Three distinct meta-models represent the meta-space: *interface*, *architecture, and interception*. The interface and architecture meta-models provide structural reflection in terms of inspecting the interfaces of components, and the topology of components in terms of connected elements; the interception meta-model supports behavioural reflection by enabling the dynamic insertion of interceptors, which support the insertion of pre- and post-behaviour on to interfaces.

3.2 Local OpenCOM component frameworks

3.2.1 Overview

The local CF model (illustrated in figure 2) is based upon the concept of composite components as proposed by OpenORB [1]. Each CF is an OpenCOM component that has internal architecture. Additionally, each framework supports the following dimensions for performing safe, valid reconfigurations in the local address space: i) an architecture meta-object protocol, ii) validated reconfigurations, iii) quiescence management, and iv) policy configurators. Each of these dimensions is now examined in turn.

3.2.2 Architecture MOP

The architecture meta-model is fundamental when developing dynamic middleware solutions. To be subject to introspection and dynamic reconfiguration, each CF maintains a local 'graph' representing its internal structure. To reduce data duplication, this is simply a view of the information held in the OpenCOM system graph. Therefore, each CF maintains a vector of component identifiers that point to their corresponding position in the system graph. A set of operations that make up the architecture meta-model are provided through the *ICFMetaInterface* seen in figure 2.

3.2.3 Validation of reconfigurations

Providing open access to the structure of a system, and the ability to make run-time changes, increases the likelihood of system failure and opens it to third party attack. To guard against this, each framework exports a 'health check' mechanism (illustrated in figure 2 as the required interface called *IAccept*); components encapsulating knowledge about valid dynamic reconfigurations for this particular framework are then plugged into this interface. Each reconfiguration is applied as a local transaction; hence once committed initially a reconfiguration is validated such that invalid attempts are rolled back to the previous safe state. The previous meta-architecture state is stored before reconfiguration commences to support this.

3.2.4 Quiescence Management

Reconfiguration operations must only be carried when a framework is in a safe quiescent state. If a change to the configuration is made while one or more service calls on the CF are executing, then the results of these invocations could be compromised or lost. Therefore, each CF provides a readers/writers lock for access to the local CF graph. Each service call through any of the interfaces other than *ICFMetaInterface* accesses the lock as a reader (there can be n readers using the lock at any time). Any call to change the configuration of the CF, accesses the lock as a writer (a single writer can access the lock when there are no readers). The algorithm to implement this property is a standard readers/writers solution with priority for readers.
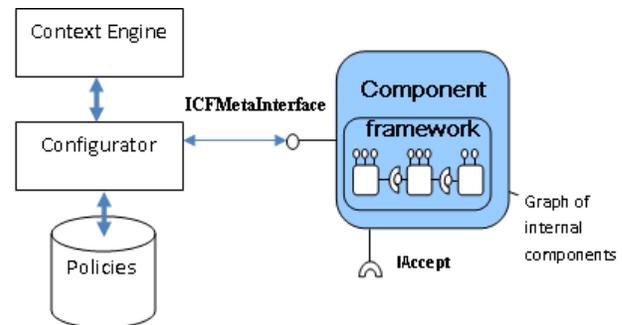

Figure 2: Local Component Frameworks

Interceptors are used to ensure that all exposed configurations access the lock as a reader before a service call is executed. Each interface exposed by a CF automatically has an interceptor attached with pre- and post- method behaviour to implement the reader role of a readers/writers solution. That is, the pre method accesses the lock and increments the reader count, while the post method decrements the count and if it is the last reader the lock is released for writers.

3.2.5 Configurators

The final aspect of the local framework model is the use of the configurator pattern [10] as illustrated in figure 2. A configurator is assigned to each framework instance, and acts as a unit of autonomy for making decisions about when and how to change the framework. Each configurator maintains a set of local policies for its framework; it is connected with a context engine to receive relevant environmental events; it communicates only with its own framework through the meta-interface. This separation of the configurator allows different configurators and policies to be used for different framework types; for example, a protocol framework may require stacking policies, as opposed to arbitrary component connections. Typical configurator policies use the Event-Condition-Action style. When an event is detected, it applies the action (reconfiguration script) – which is of the form of a set of component insert, delete, disconnect, connect, or replace operations.

4. DISTRIBUTED RECONFIGURATION

4.1 Overview

We contend that the model for local reconfiguration is equally applicable to distributed component topologies. However, tools such as OpenCOM lack the support to realise this. Hence, we discuss how the same basic CF model that has been established to support local reconfigurations is extended to provide developer support for principled, co-ordinated reconfiguration of component-based middleware behaviours across nodes. For this purpose, we introduce the concept of a *distributed component framework*, each distributed framework contains a set of local frameworks of the same type. They can be composed of hierarchical local component frameworks, and more than one distributed framework type can be created per self-managed system. The design of the distributed framework model follows the same basic themes as for local frameworks—i.e. architecture MOP, validation, quiescence and policy-driven configurators. However, these are extended with additional capabilities to increase the flexibility of the approach for operation in a distributed environment.
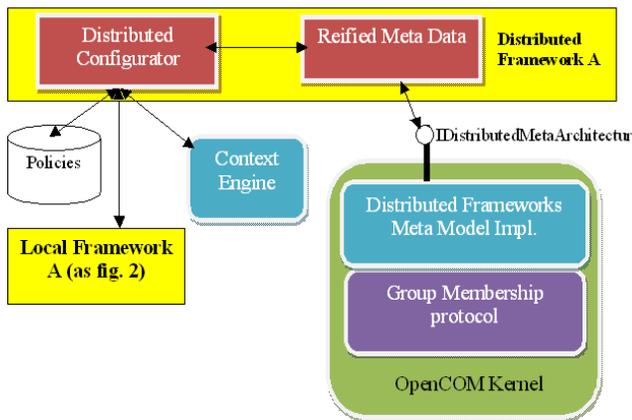
Figure 3: Distributed Frameworks

4.2 Architecture MOP & Reification

4.2.1 Lightweight Base Protocol

Each distributed framework maintains a basic architecture MOP that reifies information about the contents of the framework in terms of the node members only i.e., it identifies the instances of the local CF on each host, but does not store additional local topology information. Hence, this minimises the amount of meta-data maintained per framework.

For the implementation of this meta-object protocol we use a lightweight group membership service as the base mechanism for distributing meta-data (illustrated in figure 3); this data then builds the view of the system wide architecture. This group protocol is customizable in its implementation: typically different group membership overlays will suit different domains e.g. a p2p network in the fixed Internet can use a different group membership protocol from a sensor network application. So far, we have implemented the scalable membership protocol SCAMP [7] to maintain framework meta-data between members of a distributed framework.

IDistributedMetaInterface is exported from the component runtime (seen in figure 3) in the same fashion as IMetaArchitecture is available from the OpenCOM kernel [5]; this interface allows developers to create, and interact with instances of distributed framework architectures. A set of meta-operations allows the insertion and deletion of local framework elements into/from a given distributed framework, and the enumeration of all members of the framework. Hence, distributed frameworks are essentially dynamic, virtual architectures that are composed at run-time after local elements are inserted into them.

4.2.2 Flexibility

We also contend that to support fluctuating resource requirements of diverse environments, developers must be able to select from a set of strategies for the reification of the meta-architecture data. That is: i) flexibility in the richness of meta-data, and ii) flexibility in the locations where the meta-data is stored.

The basic meta-information maintained for each framework is shown in figure 4(a); this is essentially just the local framework members of the distributed framework. However, this is enough to subsequently discover all relevant information about the framework architecture as the meta-data contains references to the local frameworks' reflective interfaces (ICFMetaInterface), which can then be used to reflect all the component information in the distributed framework. However, a 'push'-based reification model with richer meta-data may also be employed. This will utilize more resources, but in some cases may potentially reduce the network traffic and time to make reconfiguration decisions, as the data can be stored locally. A richer meta-data model (built atop the base) is shown in figure 4(b); this adds individual component and connection information in the same format as provided in a local framework. To distribute the data in a 'push' fashion the local meta-data is gossiped to all other members using the lightweight group membership service. The selected storage points use the information (component, connections, etc.) to build a distributed view of the network wide framework. Hence, alternative, richer meta-data models can be created by third-party developers atop the basic mechanism.
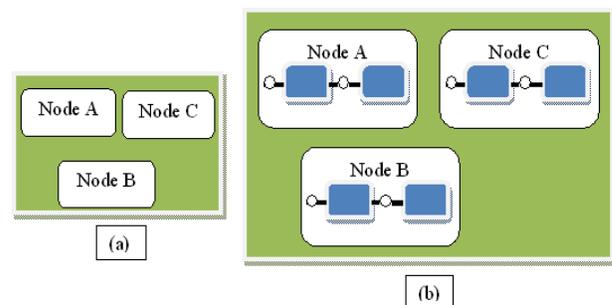
Figure 4: Alternative architecture meta-data approaches

Every member of the framework need not store the meta-data (especially, for example, in resource-constrained sensor networks); but enough nodes must store it for the data to be resilient for the lifetime of the distributed framework. Therefore, depending on the precise requirements, the meta-information could be stored at a central node, or at a subset of the nodes, or at all of them. This depends on a flexible reification strategy employed by the meta-protocol; i.e. the placement of components to collect and manage meta-data; this is shown as the "Reified

Meta Data" component in figure 3. For when there is more than one instance of this component in the framework, consistency protocols are utilized to ensure that the same view is maintained across nodes.

4.3 Validation

Validation of a distributed framework is important to ensure that the collaborating nodes maintain a correct implementation of the middleware across nodes. Hence, in a similar manner to local validation i.e. the *IAccept* plug-in, after a distributed adaptation has taken place this update is checked through inspection of the meta-data. Designated nodes in the framework have a set of plug-in rules that are used to validate the integrity of component updates across multiple nodes. An invalid reconfiguration can thereby be detected and repaired.

4.4 Quiescence

4.4.1 Centralised Quiescence

For safe dynamic reconfiguration it is important to ensure that updates do not impact the integrity of the system. Hence, the distributed framework must be made safe to adapt, i.e. placing it in a quiescent state. Here we examine a single approach we have developed within the component runtime; we then describe the need for flexible solutions.

We have so far developed a single, centralised implementation for deriving a safe state in the distributed framework that is based upon the local host approach described in section 3.2.4. A request to reconfigure the distributed framework from a central node generates a request message asking each local framework instance to be placed in a quiescent state; this message is propagated via gossiping through the meta-group service. Once a local framework is in a quiescent state it returns a notification to the configurator node. Upon the condition that all members are in a quiescent state the reconfiguration can take place.

4.4.2 Flexibility

The disadvantage of the centralised approach is that it may be too resource intensive, and may not scale suitably for large numbers of nodes. Additionally, it may not be necessary to place all nodes in a safe-state at the same time, or have a single node managing the transition to a safe state. Hence, the frameworks should support selectable approaches to safe-state management that can be tailored to the particular style of reconfiguration to be performed and the environment that the framework is deployed. Therefore, we are investigating replaceable, and decentralised strategies for safely updating components.

4.5 Policy-based Configurators

Distributed configurators (as seen in figure 3) again follow the same pattern as in local frameworks (see section 3.2.5). They receive events about changing environmental conditions, select policies, and then perform distributed reconfigurations. However, individual frameworks may have more than one configurator (e.g. there could be one on every node). Therefore, consensus protocols must be used to ensure that all members of the framework agree on the action to perform. Our development of the reconfiguration mechanisms has so far concentrated on centralised configurators; however, we are also investigating the introduction of selectable and replaceable consensus algorithms into the distributed frameworks.

5. RELATED WORK

There are a number of related areas of research to this work. These consist of reflective component models, reflective middleware, and alternative approaches to distributed adaptation of network protocols and middleware. We now analyse these in turn, examining how they differ from our approach.

Fractal [3] is a component model for the development of open, adaptive applications and systems software. Like OpenCOM, a key capability of Fractal is the use of meta-protocols to inspect and adapt frameworks of related components. These frameworks can be composed locally, or be composed from distributed Fractal components. Notably, the meta-protocols are inherently flexible; as they can be plugged into frameworks on demand, i.e. only the required meta-protocols are tailored for each framework. At present, the currently available Fractal tools provide only limited support for co-ordinated, distributed dynamic reconfigurations. However, the Fractal specification is sufficiently flexible and straightforward to introduce these as pluggable meta-protocol 'controllers'. Hence, we believe many of the aspects introduced in this paper are equally applicable in the Fractal component model. A potential avenue of future research is to investigate if this is the case.

There are now a number of established reflective middlewares e.g. OpenORB [1] and DynamicTAO [11]. Generally, these adapt their behaviour locally according to a local policy. Although potentially suitable for supporting some classes of self-managed systems, the dimensions of co-ordinated, distributed adaptations have not been addressed; therefore, we believe utilization of the approaches described in this paper will allow reflective middleware to better support more decentralized classes of self-managed systems. We have so far demonstrated this to be the case for Open ORB.

An alternative component approach that has investigated the coordinated reconfiguration of decentralized, self-managed systems is k-Components [6]. Here, a k-Component is a component with local architecture and a reflective meta protocol to inspect and adapt this architecture. Each k-Component is then related to a management agent; this is responsible for monitoring the environment and making decisions about when to adapt the component structure. In the co-ordination dimension, distributed agents can communicate with one another, although decisions to adapt are made locally. Hence, the approach is suited to only decentralized reconfigurations, with no guarantee that behaviour is changed across a system. Our approach, is in general more flexible allowing the mechanism for co-ordinated adaptation to be tailored to the requirements e.g. centralized or decentralized.

NecoMan [9] offers an alternative approach to dynamic reconfiguration, whose capabilities have inspired many of the features of our approach. It supports safe, co-ordinated updates of distributed services, typically related to network protocols. However, it has not yet been applied in diverse application environments to illustrate its full flexibility; however, we believe it presents many interesting mechanisms that could be applied within our frameworks; especially our points of flexibility in terms of consensus and quiescence.

Finally, Silva et al. [13] present a framework to support the automatic self-adaptation of distributed application components. Our approach follows some of their key ideas: monitoring the current system state, supporting flexible algorithms for diverse conditions, and using the configurator pattern. However, the approaches differ in that our approach is targeted at a more fine-

grained level. We specifically target frameworks of self-managing middleware elements, as opposed to application components. In addition, our approach is novel in that it considers an architectural view of distributed frameworks, with principled reflection mechanisms to further support adaptation decisions. Hence, self-adaptation can be applied on demand at different levels of the distributed system, from the network protocols, to the communication middleware, to the applications themselves.

6. CONCLUSIONS & FUTURE WORK

In this paper we have demonstrated the need to consider distributed dynamic reconfigurations to better support new classes of self-managing middleware. We have illustrated how the approach to local address space reconfiguration proposed in the Open ORB philosophy is equally applicable to co-ordinated distributed reconfigurations. Finally, we have shown that our approach is flexible to support many styles of distributed dynamic adaptation, to allow it to be applied in diverse application domains and environmental conditions.

There are a number of interesting future areas of research inspired by this work. Firstly, the creation of higher-level declarative languages that can be used by both middleware and application developers to describe dynamic reconfigurations both locally and globally. This may hide the developers from the inherent complexity of learning and using reflective protocols. However, such open policies may potentially cause conflicts, as multiple reconfigurations may be defined that cause the system to never stabilise (i.e. thrash from one configuration to another), or disagree with one another. Hence, mechanisms to detect and resolve such issues must be provided. Secondly, the introduction of security measures to the distributed framework is required to ensure only authentic nodes can join a framework, and only members of the framework can make reconfigurations. Again, this security measures must be lightweight and flexible in order to reduce the resource and performance cost of distributed frameworks.

7. REFERENCES

[1] 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.

[2] 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'02Charleston, South Carolina, USA, November 2002.

[3] Bruneton, E., Coupaye, T., and Stefani, J., " Recursive and dynamic software composition with sharing". In Proc. of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02), Malaga (Spain), June 2002

[4] Cazzola, W., Savigni, W., Sosio, A., Tisato, F., "Rule-based Strategic Reflection: Observing and Modifying Behaviour at the Architectural Level", Proc. of the 14th IEEE International Conference on Automated Software Engineering (ASE'99), pp.287-290, Cocoa Beach, Florida, USA, October 1999.

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

[6] Dowling, J., "The Decentralised Coordination of Self-Adaptive Components for Autonomic Distributed Systems". PhD Thesis. Trinity College, Dublin, 2004.

[7] Ganesh, A., Kermarrec, A., Massoulie, L." SCAMP: Peer-to-peer lightweight membership service for large-scale group communication". In Proc. of the 3rd Int.Workshop on Networked Group Communication, London, UK (2001).

[8] 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.

[9] Janssens, N., Michiels, S., Holvoet, T. and Verbaeten, P. "NeCoMan: Middleware for Safe Distributed Service Deployment in Programmable Networks". In Proc. of Middleware 2004, Toronto, Canada, 2004

[10] Kon. F., "Automatic Configuration of Component-Based Distributed Systems". PhD Thesis. University of Illinois at Urbana-Champaign, May, 2000.

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

[12] 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.

[13] Silva, J., Endler, M., and Kon. F., "Developing Adaptive Distributed Applications: a Framework Overview and Experimental Results", Proceedings of the International Symposium on Distributed Objects and Applications (DOA). LNCS 2888, pp.1275-1291. Catania, Sicily, Italy, November, 2003.

[14] Sun Microsystems, "Java Reflection API", 2002, http://java.sun.com/j2se/1.3/docs/guide/reflection/index.htm