GRIDKIT: Pluggable Overlay Networks for Grid Computing

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

[1] Computing Department, Lancaster University, Lancaster, LA1 4YR
[2] Dept of Computing, Oxford Brookes University, UK
{gracep, geoff, gordon, laurent, yeungwk, w.cai}@comp.lancs.ac.uk
{daduce, cscooper}@brookes.ac.uk

Abstract. A 'second generation' approach to the provision of Grid middleware is now emerging which is built on service-oriented architecture and web services standards and technologies. However, advanced Grid applications have significant demands that are not addressed by present-day web services platforms. As one prime example, current platforms do not support the rich diversity of communication 'interaction types' that are demanded by advanced applications (e.g. publish-subscribe, media streaming, peer-to-peer interaction). In the paper we describe the Gridkit middleware which augments the basic service-oriented architecture to address this particular deficiency. We particularly focus on the communications infrastructure support required to support multiple interaction types in a unified, principled and extensible manner—which we present in terms of the novel concept of pluggable overlay networks.

1. Introduction

Following initial offerings such as Legion [1] and Globus [2], a 'second generation' approach to the provision of Grid middleware is now emerging. This is taking Grid middleware forward from an era of ad-hoc platforms to a more architected approach built on service-oriented architecture and web services standards and technologies. It promises a more unified and principled approach to the support of Grid applications. However, despite these advances, the state of the art in Grid middleware is still inadequate to support advanced applications with demanding characteristics such as the following: high levels of heterogeneity in both end-systems and networking infrastructures; large scale; high complexity; real-time interactive collaboration; multiple media-types; QoS-sensitivity; and the need for dynamic reconfiguration in response to environmental change.

An example of such an application is an *environmental informatics* scenario which is being developed at Lancaster University as a large scale inter-disciplinary collaboration [3]. In this application, a river, estuary and bay are instrumented with a range of types of sensors (e.g. to monitor temperature, water level, flow rate, pollution levels, coastal erosion etc.). Some of these sensors are networked using standard

wired technologies such as Ethernet (e.g. sensors in tidal-defence walls), while other employ various wireless technologies (e.g. IEEE 802.15.4 or 802.11 radios; or long-wave radios for underwater use). Point-to-point microwave connectivity is also used to link individual sensor networks to gateways at which sensor data is collated and cached. In addition, networked, remotely controllable, surveillance cameras are distributed around the area to monitor coastal changes and near-shore waves and currents. And finally, broadband Internet access is available via a number of strategically-placed IP gateways. Given this infrastructure, scientists in widely-dispersed locations can selectively store data for future analysis, integrate and process live sensor data on their workstations, cooperatively visualise this data in real-time (supported by a video conferencing system), and use both stored and live data to computationally steer long running environmental simulations. Many of the processing and visualisation activities additionally require the dynamic acquisition of large scale processing and storage services from the wider Grid.

In terms of *communication services*, which are the focus of this paper, this application clearly requires more than merely the message-passing and request-reply-based 'interaction types' that are offered by conventional service-oriented architecture. Some examples: request-reply-with-QoS is required for time sensitive sensor network queries; reliable/ unreliable messaging services are required for streaming sensor data; publish-subscribe interaction is required to register interests in specific sensor events; tuple-space interaction may be required for cooperation between large scale processors running computationally-intensive simulations; peer-to-peer interaction may be needed for cooperation or resource discovery; media-streaming-with-QoS is needed for video monitoring and conferencing; reliable/ unreliable group interaction is needed to support real-time collaboration between scientists; and workflow interaction is needed to automate common observe-archive-visualise sequences.

Although interaction types such as the above could be provided in an ad-hoc manner alongside the 'classic' interaction types offered by the service-oriented architecture, we believe that there are clear benefits in explicitly supporting interaction types in the middleware as 'first class' services [4]. In outline, this facilitates the provision of an *extensible* set of interaction types, and also enables these to be *uniformly supported* in terms of both API abstractions (thus easing the application programmer's task), and underlying infrastructure (thus facilitating the sharing of infrastructure and optimal resource management).

We have already explored in previous work ([4]) how to present interaction types as first class services at the API level. In this paper we focus on the underlying communications infrastructure required to support an extensible set of interaction types. Our approach at both levels is to promote extensibility by wrapping added functionality (i.e. interaction types and communications infrastructure elements respectively) as *plug-in components* that are hosted by appropriate *component frameworks* in a component-based middleware environment. This approach is fully consistent with the general architecture of our component-based Grid middleware, called *Gridkit*, which is entirely composed of such component frameworks [5].

As detailed in section 2, we structure the bulk of the communications infrastructure layer as a component framework in which *overlay networks* are the unit of pluggable functionality. In outline, overlay networks [6] are virtual networks that 'overlay' the

physical network infrastructure with some value added functionality or semantic (which, in our case, are useful in supporting one or more interaction types). For example, a reliable multicast overlay [7] can be used to underpin a publish-subscribe interaction type. We believe that providing a plug-in framework for overlays is a powerful and general way of supporting a rich set of interaction types as required by advanced Grid applications.

In the remainder of this paper we first, in section 2, provide an overview of the Gridkit architecture. Then, in section 3, we discuss Gridkit's Overlays Framework in detail, and then provide an evaluation of our work to date in section 4. Finally, we survey related work in section 5 and present our conclusions in section 6.

2. An Overview of Gridkit

As illustrated in figure 1, 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 are as follows:

- *Service binding*. This hosts pluggable interaction types (as defined in the introduction) 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 or UPnP for more traditional service discovery, GRAM [8] for CPU discovery in a Grid context, and P2P protocols for more general resource discovery [9], [10].
- *Resource management*. This comprises both coarse-grained distributed resource management as currently provided by services such as GRAM [8], and fine-grained local resource management (e.g. of channels, threads, buffers etc) that is required to build end-to-end QoS [11]. Again, our approach here is based on the notion of pluggable mechanisms and policies.
- *Grid security*. This hosts pluggable services that support secure communication between participating nodes orthogonally to the interaction types in use.

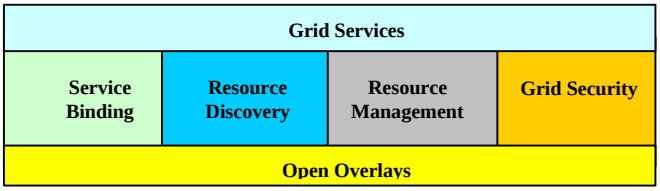

Fig. 1. The GRIDKIT Vision.

These four domains of middleware functionality are implemented in Gridkit as independent component frameworks (hereafter CFs). As well as being directly available to application developers, these can easily be combined to provide more complex middleware capabilities. For example, service bindings can integrate with Grid security to produce secure interactions. Crucially, all four domains are underpinned by the Overlays Framework. In the remainder of this paper, we focus on this, and on the service binding and resource discovery frameworks. We do not discuss resource management and security further in this paper.

In terms of its ancestry, Gridkit is an instantiation of the generic OpenORB middleware platform [12], and hence follows the philosophy of building systems in terms of *i*) components (using the OpenCOM component model [13]), *ii*) CFs and *iii*) reflection [12]. Gridkit is also strongly based, at the source code level, on an existing web-services-based mobile computing framework called ReMMoC [14]. From this, it inherits high performance and a minimal memory footprint (around 250K), thus making it deployable in almost any system environment (e.g. both workstations and PDAs).

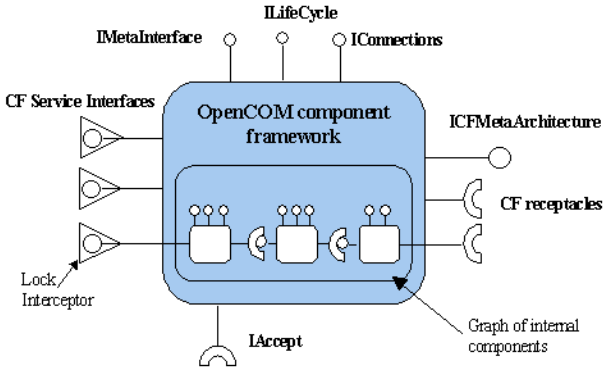

Fig. 2. The Component Framework Model

The generic architecture of a Gridkit/ OpenCOM CF is shown in figure 2. The CFs look from the outside just like a normal OpenCOM component: hence, each Gridkit CF implements an *ICFMetaArchitecture* interface which provides operations to inspect and dynamically reconfigure the CF's internal structure (maintained as a 'graph' of internal sub-components). Additionally, to ensure that dynamic changes to the framework (such as occurs when a new plug-in component is inserted) are 'valid', each CF exports a receptacle (i.e. 'required' interface) named *IAccept*; different validation strategies can be plugged into this so that once a change is made to the CF's structure, the plug-in checking strategy is executed, and if invalid the CF rolls back to its previous state. By default, the internal sub-component topology is checked against a set of XML-based architectural descriptions of valid component configurations. The *IMetaInterface*, *ILifeCycle* and *IConnections* interfaces shown in figure 2 are associated with the standard reflective meta-models provided by OpenCOM [14]. We do not discuss these further in this paper.

Turning now to the specific aspects of Gridkit emphasised in this paper (i.e. the Overlays Framework, Service binding, and Resource discovery), figure 3 depicts a

three-layer architecture that is composed of: *i*) an abstract middleware layer, *ii*) a layer of abstract-to-concrete mappings, and *iii*) a concrete middleware layer. Each of these layers in turn consists of multiple CF instances. This renders the architecture inherently configurable and extensible so that components implementing specific functions can be plugged in when and where required. In addition, applications requiring only minimal middleware functionality (e.g. client-side only) need not incur the overhead of unwanted functionality. This is especially important for execution on devices with limited resources; e.g. mobile devices.

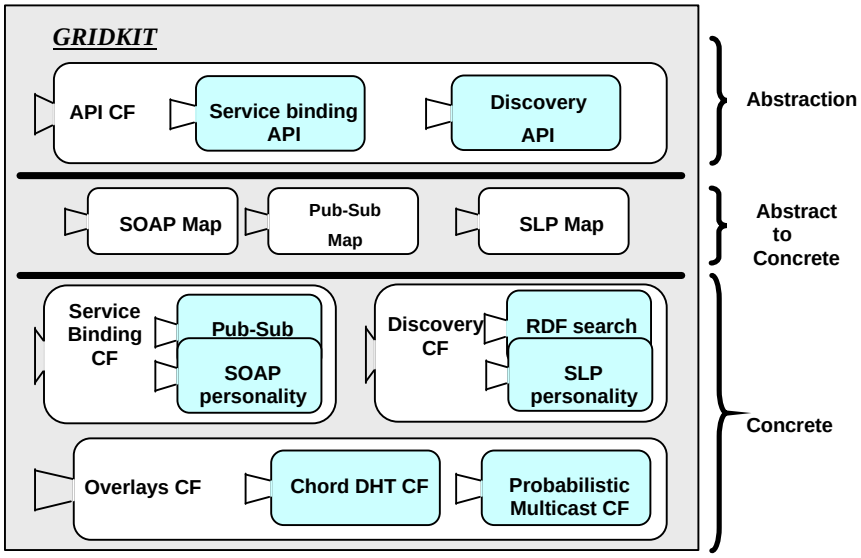

Fig. 3. The GRIDKIT Architecture

The 'abstraction' layer consists of a "Grid-oriented" API component framework that is built in terms of web services abstractions. Within this, the Service Binding API is used to create and use various sorts of user-to-service bindings, which may employ various interaction types. Abstract service interactions are described using the Web Services Definition Language (WSDL) which allows service interfaces to be uniformly specified irrespective of the underlying interaction type. This is achieved by exploiting WSDL's approach of breaking interactions down into individual one-way messages: any conceivable service operation, from the user's perspective, can be described abstractly in terms of input or output messages (e.g. an RPC service is described as an output message followed by an input message; whereas a publish-subscribe service is described, from the perspective of the subscriber, as an output message for the subscription followed by potentially many subsequent asynchronously-received input messages representing incoming publications).

Discovery—of both services and resources—also forms part of the API CF. Again, WSDL is used to abstract over different modes of interaction with service and resource discovery mechanisms. This is relatively straightforward for conventional

service discovery protocols (e.g. SLP, UPnP, Jini and Salutation) because all of these tend to be based on advertisements of service types with service attributes.

The 'abstract to concrete' mapping layer then takes the abstract information submitted through the abstract middleware layer and maps it to interfaces of the currently exposed concrete middleware implementation(s) in the layer below. This mapping is based on ReMMoC principles, a detailed discussion of which can be found in [14]. The CF allows multiple mapping components to be maintained in order for different services to be simultaneously hosted over different 'concrete layer' functionality.

Finally, the 'concrete' layer itself is composed of three CFs organised in two layers. The top layer comprises CFs that support concrete service binding and resource discovery. The Service Binding CF provides a set of available interaction type implementations that are deployed as application-layer protocols, and wrapped as components and plugged into the framework. Again, multiple interaction types can operate in parallel; for example, figure 3 illustrates component plug-ins for SOAP and publish-subscribe operating side-by-side. The binding framework exposes its network requirements to the underlying Overlays Framework using the exposed receptacle technique illustrated in figure 2 and described above. The Discovery CF similarly allows multiple discovery technologies (e.g. SLP, UDDI, Jini, P2P-based etc.) to be simultaneously plugged into the framework. Furthermore, resource discovery requests and advertisements of resources can be executed in parallel over each of the plugged-in mechanisms so that Grid applications can maximise the number of resources that are found, find them more quickly, and distribute their resources to a wider audience. Again, the discovery framework utilises the underlying Overlays Framework to enhance discovery—for example in the case of peer-to-peer based discovery plug-ins.

Underpinning the Service Binding and Discovery CFs is the Overlays Framework which is discussed next.

3. The Overlays Component Framework

3.1 Background on Overlay Networks

As mentioned, our approach to the provision of network-level communication support in Gridkit is to uniformly abstract all such support as overlay networks. The benefit of this is that it allows us to treat the diversity of communications support mechanisms in a consistent manner whether or not the underlying physical network supports the required mechanism (the special case of a standard IP network is handled by viewing it as a NULL overlay).

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). Overlays have 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) [7]; 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) [15]. 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.

At the most basic level, primitive wireless sensor network environments require specialised routing overlays to support even simple message-based interaction. At a higher level, publish-subscribe and group interaction can be underpinned in the Internet by multicast overlays such as SRM [7], or tree-based overlays [16]. Similarly, peer-to-peer interaction can be underpinned by unstructured overlays such as Gnutella [17], or by structured dynamic hashtable (DHT)-based overlays such as Chord [18]. We now examine specific overlay network technologies in more detail.

DHT-based peer-to-peer overlays are primarily used to provide reliable resource discovery in large-scale distributed systems. Prime examples are Pastry [19], Chord [18], CAN [20] and Tapestry [21]. In these systems, application-specific elements (resources) and network nodes are both assigned unique identifiers (keys) from a structured ID space. For example, Chord, Pastry and Tapestry employ a circular identifier space of N-bit integers modulo 2N, whereas CAN uses a *d*-dimensional Cartesian identifier space. Then, resources are stored, as far as possible, at the node that bears the resource's unique ID (or a node close to it). The network itself is structured in terms of the structure of the key space. For example, Chord is structured as a ring supplemented with additional links corresponding to 'chords' across the ring (each node maintains a routing table of the node IDs and IP addresses of both its neighbours in the ring and non-neighbours to which it is connected via 'chord' links). In terms of routing, messages are forwarded across the overlay to nodes whose IDs are progressively closer to the key in the identifier space. Each DHT type provides different mechanisms to establish the structure and to route messages through it, although generally, resource location is achieved in O(log N) messages where there are N nodes in the overlay.

Unstructured overlays, such as Gnutella, organize their nodes in a 'random' graph, and resources are typically named and located by arbitrary keywords rather than by N-bit IDs. Search requests are forwarded using flooding or random walks, and each node that is visited by a request message evaluates the query locally on its own content before (potentially) forwarding it. This makes it possible to search for weakly specified resources (i.e. we do not need to know the ID of a target resource as we do in DHT networks); but search is less reliable than in DHTs—unstructured overlays may fail to locate a desired resource even if it is present in the network. The greatest advantage of unstructured overlays is that they support arbitrarily complex queries, such as keyword search. However, they are relatively inefficient because queries for content that is not widely replicated must be sent (flooded) to a large percentage of the nodes. Such flooding mechanisms scale poorly, although they are better suited to small-scale ad-hoc and sensor networks.

Apart from DHT and unstructured flooding networks, application-level multicast trees have been extensively used for large-scale deployment of multi, and anycast communication [16]. Like DHTs these are structured overlays. They are quite diverse in terms of the algorithms used both to build and maintain the overlay topology, and to route messages over it. Content distribution overlays are related to application-level multicast trees. These overlays, which are primarily used to broadcast media, add multiple channels and caching support to basic multicast functionality. A final type of

overlay network is the routing overlay used in ad-hoc or wireless sensor networks. As mentioned above, these can be based on unstructured flooding principles, but in practice they are strongly driven by resource-poor nature of their environment. This implies that routing algorithms must be careful to avoid redundant message sends and to minimise message hop count.

The above technologies offer very specific services to application developers. Therefore, to increase flexibility there is now an emerging body of research which is examining mechanisms to add extra and/ or contrasting network services above existing overlay configurations i.e. *to layer overlay networks atop other overlay networks*. For example, Scribe [22] is a scalable application-level multicast overlay built on top of Pastry: it provides publish-subscribe capabilities on top of a standard DHT infrastructure. Any Scribe node may create a *group*; other nodes can then join this group, and multicast messages to all members of the group. To initiate a group, a Scribe node asks Pastry to route a CREATE message using the group ID as the key. When a Scribe node wishes to join a group, it asks Pastry to route a SUBSCRIBE message with the group's group ID as the key. At each node along the route, Pastry invokes Scribe's forward method. If it is already a forwarder it adds the child else it creates a new group and routes the join again. Other similar layering approaches have been attempted. For example, Structella [23] layers a Gnutella like overlay atop Pastry, while SplitStream [24] is a high-bandwidth content distribution system again built upon Pastry.

This work on layering various application-focused network services over DHTs indicates how middleware functionality such as our binding and discovery framework can benefit from such an approach. Indeed, a survey of these technologies [25] has presented APIs for common overlay services such as distributed key-based routing, distributed hash-tables, distributed object lookup and multicast behaviour. Such APIs offer the potential to simplify the development of distributed systems based upon re-usable overlay services. This is a novel approach that has heavily influenced the design of our Overlay Framework. However, we believe that this approach does not go far enough; it concentrates on DHT-based technologies and does not generalise to the many types of overlays that are available (as discussed above). Hence, we propose a more general approach whereby overlay networks can *arbitrarily* (albeit sensibly) depend upon one another. For example, a publish-subscribe overlay can be layered atop a DHT in one configuration, or a flooding-based overlay in another (e.g. in a small scale ad hoc or wireless sensor network). We examine this generalised architecture in the following section.

3.2 Structure of the Overlays CF

The goal of our research in this area is to develop ways of building fully customisable, extensible, and evolvable overlays by *factoring out* generic techniques and protocols (e.g., large-scale neighbour discovery, and network capability discovery techniques [16]), and enabling these to be composed, extended and dynamically reconfigured under the auspices of a well-defined CF.

The role of the Overlay CF (see figure 4) is to provide overlay network services to the higher-level CFs: to route packets through virtual networks that are tailored to support the various service-level interaction types. Sophisticated communication

services, e.g. content distribution, P2P resource discovery and reliable multicast, can be supported by appropriate overlay configurations. As with the other CFs, multiple overlay personalities can be plugged in. Multiple service bindings can then operate over their selected overlay or overlay stack. The nodes of the (overlay) network will be composed of machines hosting appropriate Gridkit CFs, which will allow autonomic management of the overlay to support the application. That is, the algorithms to maintain the required network structures will be dynamically managed by communication between the low-level component frameworks in each of the nodes.

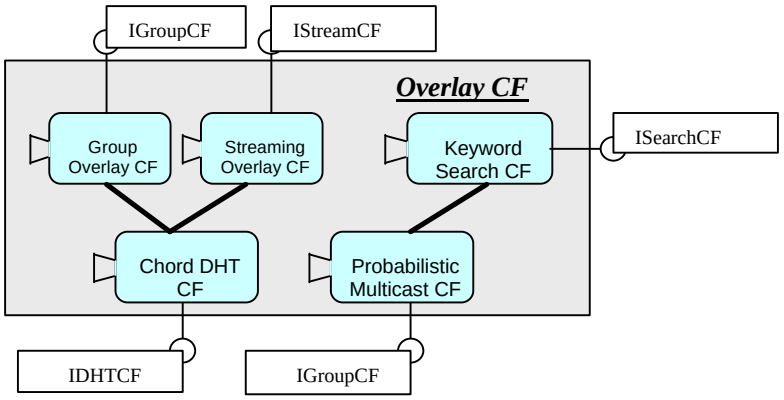

Fig. 4. The overall architecture of the Overlay framework

In turn, the contrasting interaction types, and discovery types, within Gridkit are supported by various overlay configurations. We believe that these configurations must be based both on the current environmental context (of the node) and on application requirements (e.g. operating in ad-hoc or sensor networks, requiring group communication etc.). Hence, it is these properties that drive both the initial configuration and dynamic reconfiguration of the overlay framework. Figure 4 illustrates how the Overlay CF allows for multiple overlays to be configured, and for there to be dependencies between different overlays. For example, two overlays are shown depending on the Chord DHT CF, whereas the keyword search overlay operates atop a completely separate overlay. Each overlay in the framework also exposes its interface to the outside, allowing it to be used by higher-level service bindings and discovery mechanisms. In addition, there is also scope for coarse-grained dynamic reconfiguration of the framework: for example, adding new overlays as and when the context or application requirements change. Thus a streaming overlay can be dynamically added above the DHT to supplement an existing group overlay. Alternatively, an existing implementation may be changed e.g. to a new or enhanced version.

A key feature of the Overlay CF architecture is that it does not enforce a fixed layered architecture; overlays can depend upon one another in arbitrary combinations. However, we must clearly ensure that configurations are *sensible*. The overlay CF achieves this by maintaining a set of architectural rules defined in XML, which describe sensible configurations. In addition, the CF adopts common interfaces that

can be used by multiple plug in components. For example, the *IGroupCF* interface can be offered by a range of alternative underlying implementations, e.g. featuring either the probabilistic or DHT based multicast components.

To build the individual overlay frameworks themselves, we require that plug-in overlays are internally structured into three separate elements: a *control* element, a *forwarding* element and a *state* element, (we refer to this as the "CFS" structure). The control element encapsulates the distributed algorithm used to establish and maintain the overlay structure, the forwarding element encapsulates the forwarding or routing algorithm itself, and the state element gives access to generic state such as a nearest neighbour list. Importantly, this CFS structure supports fine-grained layering and combining of overlays so that, for example, different forwarding elements can simultaneously be in operation over the same control elements.

The CFS structure has two further advantages: *i*) the use of common architectural elements ensures that we can produce generalised dependencies between overlays (as we can implement overlays to well-defined common interfaces), and *ii*) we can perform fine-grained reconfiguration of individual overlays; i.e. we can add or change the individual behaviour of an overlay as and when the environmental context changes.

Based upon the CFS structure, we develop individual overlays as small component frameworks that are composed of three plug-in components (the control component, the forwarding component and the state component). This is illustrated in figure 5. The three components interact within the individual overlay CF as shown by the bi-directional arrows in figure 5. The exported interfaces and receptacles are then used to express dependencies with other overlay implementations. We have found that state does not particularly lend itself to commonality, therefore we do not propose a standard interface for this; instead just the control and forwarding components can be directly used by other overlays.

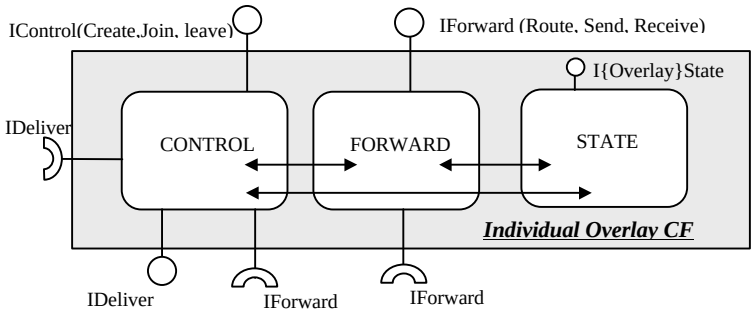

Fig. 5. The Control – Forwarding – State structure

The control component presents the *IControl* interface with common operations to create, join and leave an overlay; the implementation of the component determines how these operations are used to create the configuration between Gridkit nodes. The forwarding component has operations to route messages to nodes in the overlay, send messages to neighbour nodes, and receive any incoming messages. The control component also exports an *IForward* receptacle to allow it to forward control

messages via its own, or a different overlay's, forwarding mechanism. Similarly, the control component exports an *IDeliver* interface; this is used by lower-level overlays, which, when they receive a message, pass it to the control component atop. Particular *IDeliver* implementations determine how to deal with incoming messages (e.g., react to it, forward it etc.). Finally, the forwarding component exports an *IForward* receptacle that allows it to directly forward messages using the underlying implementation.

4. Evaluation

4.1 Publish-Subscribe Experiments

To evaluate our Overlay Framework we have performed initial experiments to determine how a publish-subscribe service binding can be supported by different overlay configurations. The publish-subscribe binding offers content-based event notification, and is influenced by the STEAM platform [26]; the use of group communication to distribute events (rather than centralised network brokers) ensures that the binding can be applied over a variety of overlay networks, e.g. flooding in ad-hoc networks, DHTs, and large scale multicast trees. Figure 6 presents its component personality. Events are XML messages wrapped in SOAP envelopes; hence, there are components to publish, subscribe and filter events of this type. Notably, the personality exports an *IGroup* receptacle into which different overlay configurations can be connected. We have explored three configurations as follows.

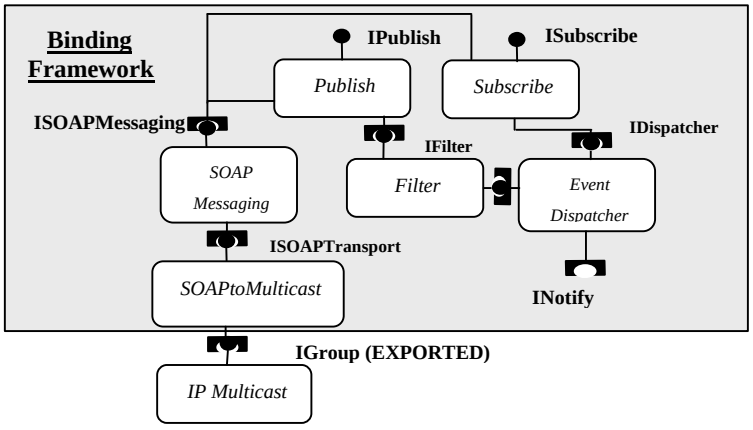

Fig. 6. The component configuration for the Publish-Subscribe Binding

Configuration 1 (base implementation). Gridkit is initiated on a mobile device connected to an infrastructure-based wireless network (with support for IP multicast). This is the base configuration of the publish-subscribe personality as used in other middleware technologies, e.g. ReMMoC [14]. In this set-up, no additional network

services are required and hence we do not plug an overlay into the personality; the binding simply functions atop standard IP multicast, as seen in Figure 6. We present this case to indicate that a binding can operate above a NULL overlay.

Configuration 2 (ad-hoc networks). In this scenario, a mobile device is operating in ad-hoc mode. Hence, this time we require an overlay network to support group based message dissemination. Figure 7 illustrates the implementation of such an overlay; notably the common interfaces of control and forwarding are combined to create the *IGroup* interface, which is then plugged into the publish-subscribe binding. 'Probabilistic multicast' is an unstructured overlay that intelligently floods messages in ad-hoc networks to support group communication. The node receives all messages, discarding the ones not from a member group, and then decides whether or not each message should be forwarded. This decision is based on the previous messages that the node has received; if a large number of duplicates of a message have already been received, the probability that the message will be forwarded reduces, i.e. 0 duplicates implies a probability of 1, whereas 2 duplicates implies a probability of 0.25. The implementation follows the CFS approach. The state component maintains two types of state, the group identifiers, and the message history of these groups. The *IPMCastState* interface provides specific operations to access this information, which is used by both the control and forwarding components. The control component has two behaviours: first it controls group participation by the node, and secondly it decides whether the received messages (from the forwarder) are passed up to the higher level binding through the *IDeliver* receptacle. Finally, the forwarding component transmits and receives messages across ad-hoc networks using a UDP-based broadcast mechanism.

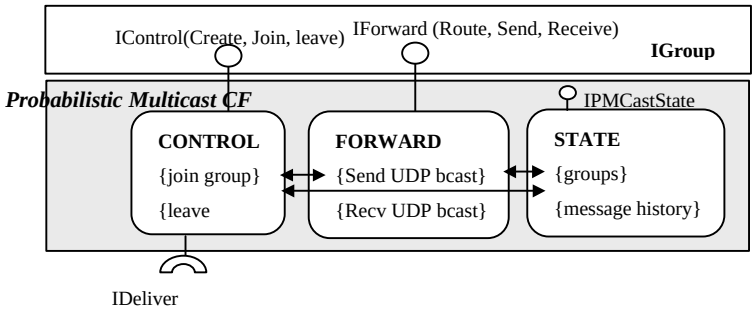

Fig. 7. The Probabilistic Multicast Layer

This experiment illustrates how our generalised approach can be used to tailor overlays to particular scenarios, so that service bindings remains correctly supported. Furthermore, the CFS implementation adds increased benefits of fine-grained reconfiguration. In particular, the forwarder component can be replaced with different damping metrics i.e. the probability can be decreased (to alleviate network traffic) based upon current network conditions.

Configuration 3 (Distributed Hash Tables). Finally, our third scenario requires large-scale publish-subscribe behaviour between elements across the Internet. For this purpose, we create a DHT-based overlay configuration. In particular, we demonstrate how our architecture implements Scribe over Pastry (as used for distributed group

communication) to underpin our own content-based publish-subscribe binding. Figure 8 illustrates the layout of the Scribe and Pastry overlays.

The Scribe overlay is implemented in the CFS mould, and its interfaces combine to form the *IGroup* interface, allowing it to be used by the publish-subscribe binding. The state component holds the individual topics (groups) that the node is participating in, along with the neighbour nodes for this topic. The forwarding component then allows messages to be published to the group (the underlying pastry forwarder is used to forward messages to the topics neighbours). The control component manages the creation of groups, joins and leaves above the existing pastry infrastructure i.e. it ensures that the topics are maintained correctly, and that the participation in the Pastry ring (by the node) is maintained. Furthermore, the control component uses Pastry forwarding to distribute control messages through the overlay (which in turn build the topic structures). Any messages received by the Pastry forwarder are passed up to the Scribe controller which decides whether they are application messages (in which case they are forwarded), or control messages (in which they are used to help build the overlay).

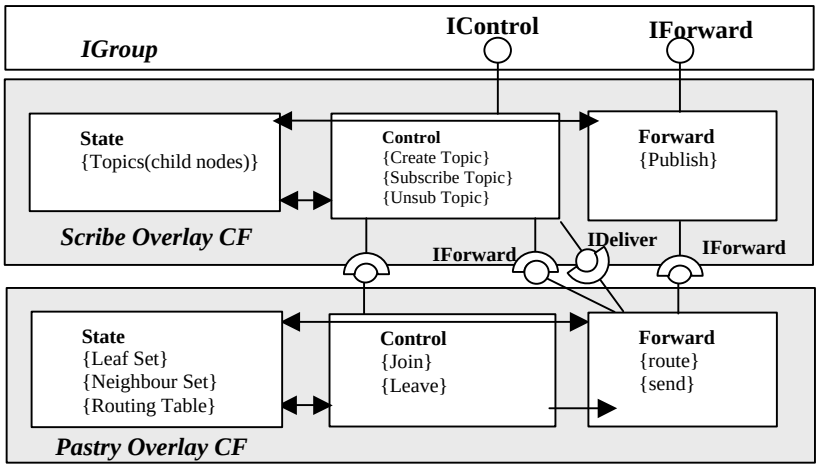

Fig. 8. Multiple overlays building the overlay service

4.2 Discovery Experiments

Our second set of experiments investigates how we could similarly use overlay mechanisms to underpin discovery services. In particular, we have investigated the use of the Service Location Protocol (SLP) and Gnutella.

The SLP protocol provides capabilities to perform traditional service discovery, i.e. service endpoints can be discovered for a particular service type. SLP is based on multicast communication: service agents listen on the SLP multicast address and respond to service requests and advertisements. Therefore, we utilise similar techniques to the publish-subscribe experiment to run SLP over a range of overlay

configurations. The configuration of components for SLP is illustrated in figure 9. It consists of components to construct SLP message headers for service and attribute lookup, and algorithms to send and receive SLP messages. It can be seen that the configuration again relies upon the exported *IGroup* receptacle to plug overlays into.

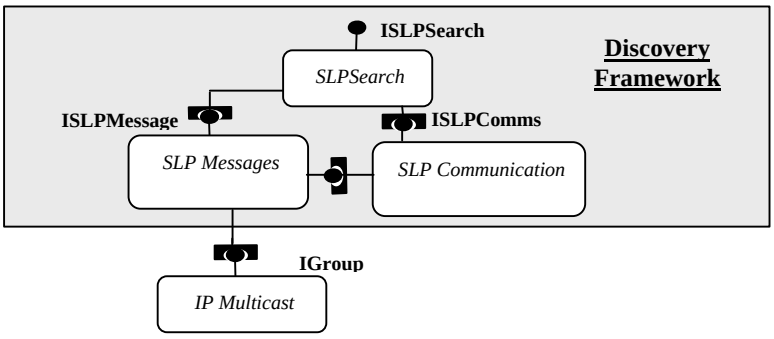

Fig. 9. The SLP discovery implementation

As with publish-subscribe, we utilise different overlays to meet discovery requirements in different settings. When we wish to discover services in an ad-hoc network, the probabilistic multicast overlay of figure 7 is plugged into the discovery framework; whereas, when performing service discovery in a large-scale environment (say the Internet) the DHT-based overlay of figure 8 is plugged in. Hence, a key feature of Gridkit is the reusability of lower level overlays to underpin both differing binding and discovery implementations.

Our final experiment investigated layering a Gnutella [17] like protocol above the Pastry DHT (in the style of Structella [23]). This configuration provides complex, keyword-based queries, not just service type, over a large-scale network.

Figure 10 shows the structure of the two overlays, with the Gnutella overlay based upon the routing properties of the underlying DHT. The *ISearch* interface extends *IForward* and *IControl* to present service and keyword lookup semantics to the higher levels of Gridkit. The Structella overlay maintains the list of neighbour nodes (those to which it forwards queries) in the state component. The control component consists of 'ping' and 'pong' operations that sends messages to find and connect to nearby "connected" members of the overlay. Finally, Structella offers operations to create content query messages (from the Forward component) that are distributed throughout the Structella overlay. Incoming responses to queries and pings are delivered to the Structella control component, which determines whether they should be forwarded, responded to, or used by this node.

Again, the use of the CFS structure allows for fine-grained changes to be made to the overlay; we can replace the Structella forwarding mechanism (flooding the Pastry ring, or random walk are possible alternatives). In addition, we are currently examining alternative layouts for Gnutella; for example, AGnuS [27] is an 'altruistic' Gnutella protocol that we are investigating how to layer above multiple overlay types.

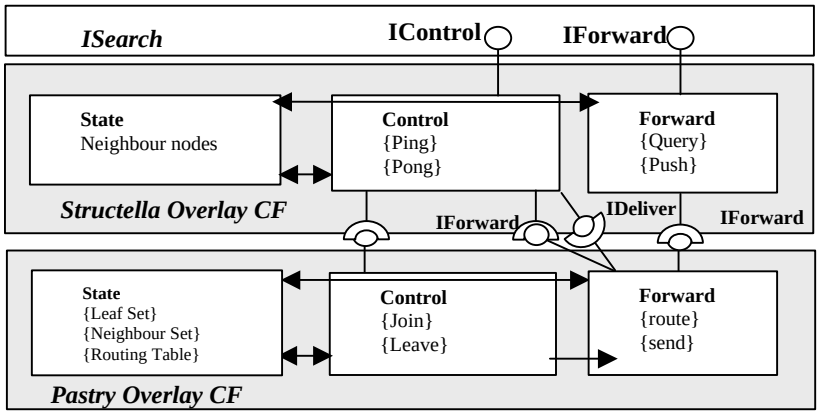

Fig. 10. A Gnutella overlay atop a Pastry overlay

5. Related Work

We are not aware of any other work that is specifically addressing the provision of integrated and pluggable overlay network support for both Grid applications and sensor network environments. However, there is a considerable amount of related work in the various sub-areas.

In terms of Grid middleware, there are platforms, notably ICENI [28], that support the notion of software components. However, these platforms, so far as we are aware, support components only at the application level: there is no infrastructure level componentisation. In terms of wider, non-Grid-specific, middleware, there are a number of platforms that take a component-oriented approach at the infrastructure level, and feature plug-ins to extend system functionality. Among these are DynamicTAO [29], UIC [30], Arctic Beans [31] and RAPIDware [32]. However, none of these support the notion of plug-in overlay networks; indeed neither do they support the range of configurable and reconfigurable interaction types provided by Gridkit.

There is, of course, considerable research in the narrower field of overlay networks themselves; but this work is largely orthogonal to our focus: we are primarily interesting in wrapping and applying overlay technologies rather than in developing new ones (see also section 3.1). In terms of generic support platforms for overlays, researchers at the University of Toronto [33] have recently developed a generic platform called *iOverlays* that supports the implementation of overlay networks. Essentially, iOverlays is a low-level software cross-connect that forwards massages according to a script that embodies the semantics of a particular overlay. Macedon [34] provides higher level support by providing a domain specific language for describing overlays which is compiled to a workable implementation. iOverlays and Macedon are both orthogonal to our interests. They provide no API (e.g. no

interaction types) and are not Grid-integrated (i.e. they have no service-oriented architecture, no WSDL etc.). Our work also differs in focusing more on co-existence of, and cooperation across, multiple overlay instances which is required to simultaneously support multiple interaction types in the same application.

Finally, in terms of sensor networks a lot of work has been done on message routing, particularly on schemes that aim to conserve scarce battery resources. A good example of early work in this is [35]. A more recent snapshot of research in the area is available in a special section of CACM on wireless sensor networks [36]. Again, however, this work is orthogonal to our main interests of wrapping and applying communications infrastructure, as opposed to developing new algorithms.

6. Conclusions and Future Work

In this paper we have described our approach to the provision of communications support in Grid middleware that must support advanced applications such as those outlined in the introduction. We have argued that our multi-level architecture, consisting of interaction types underpinned by overlay networks, has considerable power and generality in supporting such applications. The approach is easier for application programmers to learn and use as it offers a consistent programming model across different interaction types. It is also extensible in that new interaction types and new overlays can be developed and plugged into the middleware, even at runtime. And the approach also benefits from supporting multiple interaction types and overlays in the same environment. This means that interaction types/ overlays can be built in terms of value added extensions to existing plug-ins; and that both code and state (e.g. nearest neighbour state or forwarding algorithms) can be shared among multiple interaction types/ overlays.

Empirically, we have demonstrated that our generalised architectural approach for the overlay CF fits to specific binding and discovery technologies. We have successfully demonstrated the use of publish-subscribe above a range of overlay technologies (in small scale ad-hoc networks, and large-scale networks); furthermore, we have implemented SLP and Gnutella, using similar techniques to illustrate our approach is equally applicable to resource discovery mechanisms.

In our future work, we plan to further evaluate our middleware by building applications scenarios based on the environmental informatics work outlined in the introduction. Furthermore, we plan to explore the self-management of services and applications in Gridkit. This will build on the inherent openness of the (component-based) platform but will require additional CFs that deal with areas such as monitoring, recovery strategy selection, and recovery strategy deployment. We have carried out initial explorations in this area [37], but Gridkit will provide a challenging context for these ideas.

8. References

[1] Grimshaw, A., Ferrari, A., Knabe, F., Humphrey, M., “Legion: An Operating System for Wide-Area Computing”, IEEE Computer, Vol 32, No 5, pp 29-37, May 1999.

[2] Foster, I., Kesselman, C., Tuecke, S., "The Anatomy of the Grid: Enabling Virtual Organizations, International Journal of Supercomputer Applications, Vol 15, No 3, 2001.

[3] Centre for Sustainable Water Management. http://www.swm.lancs.ac.uk/

[4] Parlavantzas, N., Coulson, G., Blair, G.S., "An Extensible Binding Framework for Component-Based Middleware", Proc. EDOC 2003, Brisbane, Australia, Sept. 2003.

[5] 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.

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

[**7**] Floyd, S., Jacobson, V., Liu, C., McCanne, S., Zhang, L., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", IEEE/ACM Transactions on Networking, Volume 5, Number 6, pp 784-803, December 1997.

[8] 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.

[9] Smith, P., Simpson S., Hutchison, D., "Peer-to-Peer Networking for Discovering Programmable Resources", Proc. 4th Intl. Workshop on Networked Group Communication (NGC 2002), 23rd - 25th October 2002, Boston, USA.

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

[11] Parlavantzas, N., Blair, G.S., Coulson, G., "A Resource Adaptation Framework for Reflective Middleware", Proc. 2nd Intl. Workshop on Reflective and Adaptive Middleware (located with ACM/IFIP/USENIX Middleware 2003), Rio de Janeiro, Brazil, June, 2003.

[12] Coulson, G., Blair, G.S., Clark, M., Parlavantzas, N., "The Design of a Highly Configurable and Reconfigurable Middleware Platform", ACM Distributed Computing Journal, Vol 15, No 2, pp 109-126, April 2002.

[13] Clark, M., Blair, G.S., Coulson, G., Parlavantzas, N., "An Efficient Component Model for the Construction of Adaptive Middleware", Proc. IFIP Middleware 2001, Heidelberg, Germany, Nov. 2001.

[14] 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 2003.

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

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

[17] Gnutella Community. "Gnutella Protocol Specification v0.4", dss.clip2.com/GnutellaProtocol04.pdf.

[18] 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).

[**19**] Rowstron, A., Druschel, P., "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems", Proc. IFIP Middleware 2001, Heidelberg, Germany, Nov, 2001.

[**20**] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., "A Scalable Content-Addressable Network", Proc. ACM SIGCOMM, San Diego, 2001.
[**21**] Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., Kubiatowicz, J., "Tapestry: A Resilient Global-scale Overlay for Service Deployment", IEEE Journal on Selected Areas in Communications, Vol 22, No 1., January 2004.
[**22**] Castro, M., Druschel, P., Kermarrec, A-M., Rowstron, A., "SCRIBE: A Large-Scale and Decentralised Application-Level Multicast Infrastructure", IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), 2002.
[**23**] Castro, M., Costa, M., Rowstron. A., "Should we build Gnutella on a structured overlay?", Proc. 2nd Workshop on Hot Topics in Networks. Cambridge, MA USA. November , 2003.
[**24**] Castro, M., Druschel, P., Kermarrec, A-M., Nandi, A., Rowstron A., Singh, A., "SplitStream: High-bandwidth Content Distribution in a Cooperative Environment", Proc. IPTPS'03, Berkeley, CA, February, 2003.
[25] Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., Stoica, I., "Towards a Common API for Structured P2P Overlays", Proc. Second International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA. February 2003.
[**26**] Meier, R., Cahill, V., "STEAM: Event-Based Middleware for Wireless Ad Hoc Networks", Proc. of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02), pp. 639-644, Vienna, Austria, 2002.
[**27**] Hughes, D., Warren, I., Coulson, G., "AGnuS: The Altruistic Gnutella Server", Proc. 3rd International Conference on Peer-to-Peer Computing (P2P2003), Linkoping, Sweden, 2003.
[28] Furmento, N., Mayer, A., McGough, S., Newhouse, S., Field, T., Darlington, J., "ICENI: Optimisation of Component Applications within a Grid Environment", Parallel Computing, Vol 28, No 12, pp 1753-1772, 02.
[29] 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 2000.
[30] Roman, M., Kon, F., Campbell, R., "Reflective Middleware: From Your Desk to Your Hand", IEEE Distributed Systems Online, 2(5), August 2001.
[31] 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, 2001.
[32] Sadjadi, S., McKinley, P., Kasten, E., "Architecture and Operation of an Adaptable Communication Substrate". Proc. Ninth IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'03), San Juan, Puerto Rico, pp 46-55, May 2003.
[33] Li, B., Guo, J., Wang, M., "iOverlays: A Lightweight Middleware Infrastructure for Overlay Application Implementations", Proc. IFIP/ACM/USENIX Middleware 2004, Toronto, Canada, 2004.
[34] Rodriguez, A., Killian, C., Bhat, S., Kostic, D., Vahdat, A., "MACEDON: Methodology for Automatically Creating, Evaluating and Designing Overlay Networks, Proc. USENIX/ACM Symp. on Networked Systems Design and Implementation (NSDI 2004), 2004.
[35] Intanagonwiwat, C., Govindan, R., D. Estrin, et. al., "Directed Diffusion: A Scalable and Robust Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Communication Paradigm for Sensor Networks", Pro. 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000), 2000.
[36] Culler, D.E., Hong, W., Special Section on Wireless Sensor Networks, Communications of the ACM, June 2004.
[37] Blair, G.S., Coulson, G., Blair, L., Duran-Limon, H., Grace, P., Moreira R., Parlavantzas, N., "Reflection, Self-Awareness and Self-Healing in OpenORB", Proc. ACM Sigsoft Workshop on Self-Healing Systems (WOSS'02), Nov 02.