GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing

Thirunavukkarasu Sivaharan, Gordon Blair, and Geoff Coulson

Computing Department, Lancaster University, Lancaster, LA1 4YR, UK
{t.sivaharan, gordon, geoff}@comp.lancs.ac.uk

Abstract. In this paper we present GREEN a highly configurable and re-configurable publish-subscribe middleware to support pervasive computing applications. Such applications must embrace both heterogeneous networks and heterogeneous devices: from embedded devices in wireless ad-hoc networks to high-power computers in the Internet. Publish-subscribe is a paradigm well suited to applications in this domain. However, present-day publish-subscribe middleware does not adequately address the configurability and re-configurability requirements of such heterogeneous and changing environments. As one prime example, current platforms can-not be configured to operate in diverse network types (e.g. infrastructure based fixed networks and mobile ad-hoc networks). Hence, we present the design and implementation of GREEN (Generic & Re-configurable EvEnt Notification service), a next generation publish-subscribe middleware that addresses this particular deficiency. We demonstrate the configurability and re-configurability of GREEN through a worked example: consisting of a vehicular ad-hoc network for safe driving coupled with a fixed wide area network for vehicular traffic monitoring. Finally, we evaluate the performance of this highly dynamic middleware under different environmental conditions.

1 Introduction

Recent advance in wireless network technologies (e.g IEEE 802.11) and computational devices (e.g. PDA, PC) have created opportunities for the vision of pervasive computing applications [1], which embrace both fixed infrastructure based (wired and wireless) networks and wireless ad-hoc networks. Event based communication based upon the publish-subscribe model is well-suited to pervasive computing applications, as it presents an asynchronous and decoupled communication model [2], [3],[4],[44]. Notably, pervasive computing applications operate across highly heterogeneous environments in terms of network types (e.g. WAN, MANET) and device types. However, many publish-subscribe middleware have specifically targeted fixed infrastructure based networks e.g SIENA [5], Gryphon [6], Hermes [7] and JEDI [8]. At the other extreme STEAM [9] is specifically designed for wireless ad-hoc networks. We argue that publish-subscribe middleware that operates over a single homogenous network environment (i.e. WAN or MANET) and offers a single (or fixed) interaction type (i.e topic based or content based) cannot cope with the diversity of environmental constraints and requirements presented by pervasive computing applications. Dealing with such extreme heterogeneity is a fundamental

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3760, pp. 732 – 749, 2005.

challenge for future publish-subscribe middleware and one that is demonstrably not addressed by existing platforms. To overcome this problem we believe it is necessary to build highly configurable and dynamically reconfigurable publish-subscribe middleware which can be deployed in heterogeneous network types and heterogeneous device types and meet application and environment specific requirements.

This paper presents GREEN, a deployment and run-time reconfigurable publish-subscribe middleware. GREEN follows the well established approach to the development of reflective middleware [10], [11]; it uses the marriage of OpenCOM components [12],[13], reflection [14] and component frameworks (CFs) [15] to yield a configurable, reconfigurable and evolvable publish-subscribe middleware architecture. In particular, GREEN is configurable to operate over heterogeneous network types (e.g. MANET and WAN) and supports pluggable publish-subscribe interaction types (i.e. topic based, content based, context, composite events). Further, the underlying event routing mechanisms are reconfigurable to support selected interaction type in different network types. The distributed event routing and event filtering is underpinned by pluggable distributed event broker overlays; we create overlays of event brokers to suit contrasting network types.

In the remainder of this paper we first, in section 2, describes our approach to building reconfigurable middleware. Then, in section 3, we present the GREEN architecture. In section 4, we describe the implementations of GREEN configurations based upon a case study and then in section 5, provide performance results of our work to date. Finally we survey related work in section 6, and present our conclusions in section 7.

2 Building Re-configurable Middleware: Lancaster Approach

It is clear that GREEN middleware must accommodate an increasing diverse range of requirements arising from the needs of both applications and underlying systems (e.g device types, network types). Moreover, it is clear that to achieve this accommodation GREEN must be capable of both deployment-time configurability and run-time re-configurability. Unfortunately, the current generation of mainstream middleware is, to a large extent, heavyweight, monolithic and inflexible and, thus, fails to properly address such requirements. It is important to note, the approach for achieving *re-configurability* is important in itself. Therefore, this section describes the approach taken by GREEN to address these requirements. GREEN follows Lancaster's well-founded approach to building re-configurable middleware platforms [10],[11]. GREEN is built using our well founded lightweight component model [12],[13], uses reflective techniques [10] to facilitate re-configuration, and employs the notion of component frameworks (CF) to manage and constrain the scope of reconfiguration operations.

Component technology [15] has emerged as a promising approach to the construction of configurable software systems. With component technology, one can configure and reconfigure systems by adding, removing or replacing their constituent components. Importantly, components are packages in a binary form and can be dynamically deployed within an address space. Additional benefits of component technology include increased reusability, dynamic extensibility, improved understandability and better support for long term system evolution. It should be

noted, however, that current component models (e.g. Enterprise JavaBeans, Microsoft COM) provide little or no support for integrity management; system integrity can be easily compromised if run-time reconfiguration operations are not carried out with great care. In our previous work we have addressed this problem and presented our component model known as OpenCOM [12], [13]. OpenCOM is a lightweight, non-distributed, language independent component model that is independent of any infrastructures, thereby enabling GREEN middleware itself to be built using components. Figure 1 shows the basic elements of the component model. Components interact with other components through interfaces and receptacles. Interfaces are expressed in terms of sets of operation signatures provided by the component. Receptacles are required interfaces that are used to make explicit the dependencies of a component on the other components. Bindings are associations between a single interface and a single receptacle (within an address space).

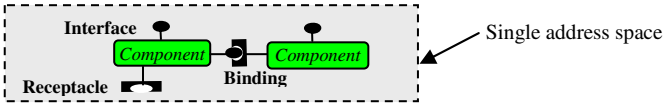

Fig. 1. Basic elements of OpenCOM component model

Furthermore, our component model is highly *reflective* [10]; in other words, the component configurations that comprise the middleware are associated with causally connected data structures (called meta-structures) that represent (or, in reflection terminology, 'reify') aspects of the component configurations, and offer *meta-interfaces* through which these reified aspects can be inspected, adapted and extended. The use of reflection facilities the management of run-time reconfiguration of the middleware, and also helps address the issue of integrity management referred above.

The second key underpinning of GREEN is the adoption of the concept of *component frameworks* (CFs) to architect and build GREEN. CF was originally defined by [15] as 'collections of rules and interfaces that govern the interaction of a set of components plugged into them'. Each CF targets a specific domain and embodies 'rules, interfaces and components' that make sense in that domain. The rules define valid configurations (or graph) of components. Crucially CFs actively police attempts to plug-in or swap new components according to these rules. It is important to note, GREEN architecture consists, a set of hierarchically composed component frameworks (more on this later). Furthermore GREEN applies our notion of *deep middleware* [11], [46] in which the middleware platform reaches down into the (heterogeneous) network to provide flexible communications services with which to support a range of publish-subscribe interaction types at the application level.

3 The GREEN Architecture

3.1 Overview

This section describes the GREEN architecture, a generic, configurable, reconfigurable and reflective publish-subscribe middleware to support pervasive computing application development. GREEN uses OpenCOM as its component

technology, is built as a set of component frameworks (CFs), and is based upon the generic middleware framework proposed in [46]. This generic middleware framework developed at Lancaster offers a two layered architecture, where the higher layer is an *interaction framework* that takes plug-in interaction types (e.g publish-subscribe, RPC, tuple-space); the lower layer is an *overlay framework* which takes plug-in overlay implementations (e.g application level multicast overlays).

Fig. 2. The overall architecture

This approach separates middleware interaction types from the underlying overlay network implementations as seen in figure 2, providing configurability, re-configurability and re-use. Importantly, GREEN concentrates on 'publish-subscribe' based interaction types and 'event broker overlays'.

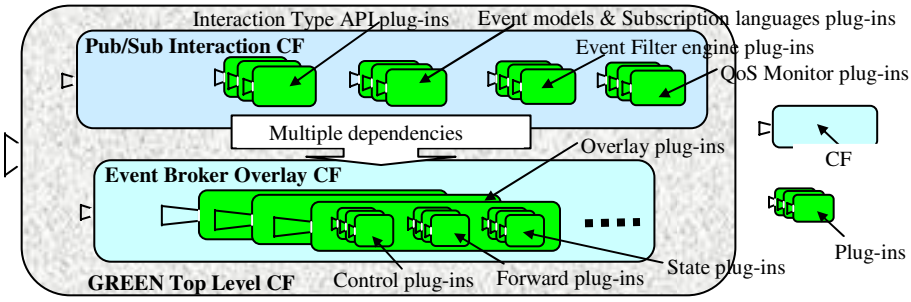

Fig. 3. GREEN Architecture

The GREEN architecture consists of two main component frameworks (CFs); 1) publish-subscribe interaction CF and 2) an event broker overlay CF (see figure 3). The publish-subscribe interaction CF is configured by plugging in different publish-subscribe interaction type implementations e.g. topic based, content based etc. The event broker overlay CF is similarly configured by plugging in different overlay protocol implementations, e.g. probabilistic multicast overlay for ad-hoc networks and the Scribe [19] overlay for WAN etc; where overlay networks are virtual communication structures that are logically laid over an underlying physical network such as Internet or ad-hoc networks [17].

Finally, the GREEN top level CF (see figure 3) which is itself composed of two layers of the above mentioned CFs; mandates the appropriate layer composition between interaction CF and overlay CF and configures the two CFs to provide distinct implementations of GREEN configuration(s). For example, 1) proximity and content based publish-subscribe interaction type underpinned by probabilistic

multicast overlay for ad-hoc networks and 2) a content based publish-subscribe underpinned by Scribe [19] overlay for wide area networks (WAN) etc.

3.2 Publish-Subscribe Interaction Component Framework

The main function of the pub-sub interaction CF is to provide various pub-sub interaction types such as topic based and content based etc. Therefore, over time the framework may be configured as a topic based pub-sub personality where subscribers can make topic based subscriptions, or change to content based pub-sub personality for highly expressive content based subscriptions, or, context based system (e.g. location and proximity based as in [9], [22]). Within the pub-sub interaction CF changes can be made at distinct levels (illustrated in figure 3). Firstly each interaction type API plug-in can be replaced; e.g. topic based interaction type API is replaced by content based interaction type API. This re-configuration is performed according to application requirements. Secondly, different subscription language plug-ins such as FEL (described later) and XPATH [45], event data models (e.g. strings, sequences of values (tuples), name-value pairs, XML based, objects) and the associated event filter engine implementations can be plugged-in. The decision on the subscription language plug-in and the event data model to configure, can be made in light of device context, e.g. resource scarce embedded devices can use simple strings as event data model instead of verbose XML. Furthermore, run-time reconfigurations can be made in light of changes in the quality of service (e.g. throughput). For example, content based interaction personality can be replaced by a topic based interaction to help obtain high event throughput in the system.

In order to test and evaluate the pub-sub interaction CF, we have implemented 1) interaction type API plug-ins (i.e topic based, content based and proximity based), 2) an extensible subscription language known as FEL and its associated event filter engine plug-ins and CLIPS -a composite event specification plug-in and 3) QoS monitor plug-ins (i.e. TCB – discussed later).

Exp : : = a **S** b | a **S** b [**T**]
S : : = { Subject names , * }
T : : = t | t & **T**
t : : = % name % operator $ value $
name = { attributes names }
operator = { = , > , < }
value = { string or numeric values }
a = // , b = / , * = Any subject
Exp - Filter expression

Exp : : = a **S** b ?#CONTEXT#operator$value$ |
a **S** b [**T**]?#CONTEXT#operator$value$
S : : = { Subject names , * }
T : : = t | t & **T**
t : : = % name % operator $ value $
name = { attributes names }
operator = { = , > , < }
value = { string or numeric values }
a = // , b = / , * = all subjects ,
CONTEXT = {DISTANCE , ...}

Fig. 4. a) Grammar of FEL b) Grammar of extended FEL for Context

Example subscriptions	FEL Filter Type plug-in
1) //stock/[]	Topic
2) //stock/[%name%=IBM&%exchange%=$NYSE$&%price%>50]	Topic+Content
3) //RoadTraffic/[%type%=$TrafficLight$]?#DISTANCE#<15	Topic+Content+Context

Fig. 5. Example subscription types in FEL

FEL (Filter Expression Language) is an 'extensible' language that we have defined and implemented, which enables the definition of topic filters, content filters and context filters depending on the configured FEL plug-in type. The grammar of the subscription language is illustrated in figure 4. A few example subscriptions in FEL are illustrated in figure 5. Notably, example three is a context based subscription, specifically a location context (i.e. proximity) and enables the subscriber (e.g. vehicle) to receive 'traffic light information generated from traffic lights which are located within 15m distance'. The context filter plug-in transparently handles adding context data (e.g. GPS location coordinates) to the original events published by the application. Importantly, the application programmer need not deal with context data.

The XML based event data model was chosen in the implementations as it's easily extensible, interoperable and it is platform and programming language independent, however it suffers from high processing overhead. The FEL subscription language plug-in is suited for individual event notifications. However, in some applications subscribers may require to specify interest in the occurrence of multiple related events. Specifying interest in composite events on top of a content based publish-subscribe system is a powerful interaction type for many distributed applications [23]. Therefore, the popular rule-based inference engine CLIPS(C Language Integrated Production System) [26] is provided as a additional plug-in in the framework for applications which require rule-based composite events specification support. When the CLIPS plug-in is configured, a subscriber can submit 'event-condition-action' based subscriptions. Some of the benefits of using CLIPS language are its platform and language independence and the high efficiency of the inference engine. The internal implementation of CLIPS is based upon RETE nets [27]. Furthermore, it is feasible for new subscription languages and event data models to be dynamically integrated into the framework at a later date.

3.3 Event Broker Overlay Component Framework

The primary function of the event broker overlay framework is to provide the underlying distributed event routing and filtering implementations for the selected interaction type plug-ins (see figure 3). The framework can be configured to provide different overlay implementations depending on the network type (e.g. MANET, WAN) and the interaction type of the personality. The overlay plug-in configured for a particular pub-sub personality heavily influences 1) the suitability of the personality to the network environment such as WAN, MANET, 2) scalability and 3) fault tolerance properties of the system. The overlay CF provides pluggable overlays for diverse environments (e.g. probabilistic multicast overlay for MANET and Scribe overlay for WAN etc). The framework is configurable based on environmental context such as the mobility model of the network, for example MAODV[31] overlay implementation can be configured for ad-hoc networks with low mobility pattern and can then be reconfigured to probabilistic multicast overlay if the ad-hoc network becomes highly mobile.

In terms of design, the overlay CFs per-host overlay plug-ins are implemented in terms of three standard component plug-ins (i.e. control, forwarding, state, see figure 3). The *control* component cooperates with its peer brokers on other hosts to build and maintain the broker network topology. It is in charge of managing the overlay event broker network. It encapsulates the distributed algorithms used to establish and maintain the broker overlay structure. The *forwarding* component routes events over

the broker network. This component enables pluggable event forwarding strategies. For example, the simplest approach is to forward the event to all other brokers along the broker tree overlay. Subscriptions are never propagated beyond the broker receiving them. An alternative strategy is subscription forwarding: when a broker receives a subscription from one of its neighbors, it stores the subscription in a subscription table and forwards the subscription to all its remaining neighboring brokers. This effectively sets event forwarding routes through the reverse path followed by subscriptions. Finally, the *state* component encapsulates key states such as nearest broker neighbours lists, connected clients lists (i.e. publishers, subscribers).

In order to test and evaluate the event broker overlay CF, we have populated the framework with three alternative overlay plug-in implementations: probabilistic multicast overlay for configuring pub-sub personality over mobile ad-hoc network, Scribe overlay implementations for wide area networks and IP multicast for local area networks (LAN) and infrastructure based wireless LAN. Furthermore, it is feasible for new overlays to be dynamically integrated into the framework at a later date, in addition to currently available overlay plug-ins.

4 Implementations of GREEN: A Case Study

In this section we consider a pervasive computing application case study and describe how GREEN implementations are configured and reconfigured to meet the requirements imposed by the application and the underlying heterogeneous environment.

4.1 Application Scenario

The scenario embraces vehicular ad-hoc networks (VANET) and wide area fixed network, to facilitate: 1) the autonomous inter-vehicle cooperation over VANET and 2) the monitoring and control of vehicular traffic over a wide area network. The experimental test-bed consists of a small number of robot vehicles augmented with wireless LAN (IEEE 802.11b), GPS, ultrasonic sensors, magnetic compass and on-board PDA. The laptops with WLAN placed on the roadside act as the bridge between the VANET and the Internet. The autonomous vehicles travel along a given path, defined by a set of GPS waypoints (a 'virtual' circuit). Every vehicle discovers and cooperates with other vehicles in its proximity to travel safely and avoid collisions. The vehicles in close proximity form a VANET. Furthermore, sensor data generated by vehicles (i.e location, speed, bearings, time stamp) are relayed via WLAN to road-side base stations placed only at strategic points on the road network. These base stations connected to the Internet form a large scale wide area sensor network, thus facilitating traffic monitoring and control. Users in the Internet may query traffic information derived from vehicular sensor data (e.g. slow speed may imply high road traffic). The vehicles approaching a base station needs to temporally reconfigure the personality to operate over from the default WLAN ad-hoc mode to infrastructure mode to relay sensor data to the base station. This application clearly embraces heterogeneous networks and heterogeneous devices and presents two main requirements on GREEN 1) QoS aware event-based middleware suited for mobile ad-hoc networks to enable inter-vehicle communication 2) event-based middleware suited for fixed wide area network to enable dissemination of vehicular sensor data to enable traffic monitoring and control. More details of the VANET test-bed can be found in [28], [29].

4.2 The GREEN Configuration for Mobile Ad-Hoc Networks

The GREEN configuration for MANET is specifically configured to address the following requirements and constraints of the VANET environment:

- support inter-vehicle events communication in a mobile ad-hoc network
- end-to-end event channel QoS monitoring in ad-hoc networks
- content , proximity based interaction and composite events specification

Publish-Subscribe Interaction CF plug-ins for MANET
The GREEN configuration for MANET is illustrated in figure 6. It shows the composition of the component plug-ins within the interaction CF. Similarly it shows the composition of the component plug-ins in the overlay CF. It can be seen the interaction CF is layered on top of the overlay CF by the GREEN top level CF, which is not shown in the figure to simplify the presentation. The publish component (i.e. the one that exports the IPublish interface) and the subscribe component (featuring the ISubscribe interface) publish XML based events and subscribe to events of interest respectively. ISubscribe interface supports a *context (i.e proximity)* based subscriptions in addition to topic and content based subscriptions in FEL. The *proximity plug-in* encapsulates the FEL context filter engine extended for location context.

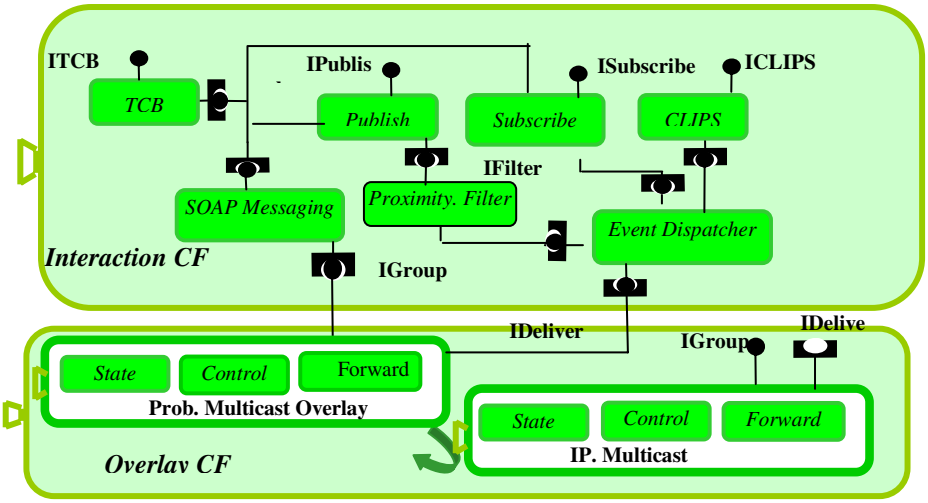

Fig. 6. GREEN configuration for MANET

Furthermore, subscribers can specify *rule-based composite events* using ICLIPS interface. Subscribers specify composite events in the CLIPS language [26] as scripts in a text file and load using the ICLIPS interface. The script files can be (un-)loaded dynamically. The CLIPS component plug-in encapsulates the implementation of CLIPS inference engine. Furthermore, clients can specify *quality-of-service* (QoS) monitoring requirements using ITCB interface, particularly event delivery deadlines and assign them to particular event types (i.e. event channels). For example, 'event

channel a: event delivery deadline 300ms' and 'event channel b: event delivery deadline 1000ms'. The callback function on the ITCB interface notifies the publisher or subscriber(s) regarding event delivery deadline failures in a guaranteed time bound. The event channel QoS monitoring functionality is implemented by University of Lisboa's *Timely Computing Base* (TCB) [30] plug-in. The TCB plug-in is configured in this personality, as end-to-end event channel QoS monitoring and fail safety is a crucial requirement in VANET for safe driving. Note though that TCB requires a predictable MAC protocol for wireless ad-hoc networks such as TBMAC [32] and also a real-time operating system. Interested readers can refer to [30] for more details on the design of TCB.

Event Broker Overlay CF Plug-ins for MANET

The event broker overlay plug-in must address the unique challenge of MANET environment (i.e. the topology of the network is highly dynamic). There are no fixed infrastructures to place event brokers in MANET. Hence, a fully distributed event broker overlay is implemented; where all mobile nodes perform partial event brokering functionality (i.e event routing, filtering). Notably, producer side and consumer side event filtering is supported. Event producers define the event type and scope of event propagation. Subscriptions (content filters) are deployed only at the consumer side unlike the common approach of *subscription forwarding* [5]. In subscription forwarding strategy, the subscriptions (content filter) form a reverse path for content based event forwarding. However forwarding subscriptions is not suitable in VANET as reverse path(s) quickly become redundant as the network topology is highly dynamic. Hence, in this overlay plug-in, event forwarding is based on event type (i.e. topic) and proximity; each event type is hashed to a underlying multicast group address. Events are forwarded from producers to consumers using the underlying multicast overlay. Scalability has been identified as a drawback with the aforementioned approach in WANs [5], [8]. We address this by allowing producers to define proximity of the event propagation (e.g the proximity radius can be set to 25m), coupled with location aware event forwarding provided top of the underlying multicast overlay. Furthermore, each consumer has to deal with small number of content filters (i.e their own) compared to producers or dedicated event brokers having to match potentially arbitrarily large number of content filters. This helps distribute the event processing overhead evenly among the resource scarce wireless mobile devices and avoids having single points of failure. The aforementioned mechanism adopted by the event broker overlay plug-in is strongly influenced by STEAM [9].

As mentioned before, a multicast overlay plug-in underpins the aforementioned event broker overlay plug-in. This leads to the requirement of designing a multi-hop multicast overlay suited for VANET. There exist numerous multicast algorithms (both proactive and reactive) for ad-hoc networks. However, most existing algorithms (MAODV, AMRoute, CAMP, MCEDAR, etc.) perform inadequately when high node mobility is present in the MANET environment [31] e.g. as in VANETs. Hence we implemented a *probabilistic multicast overlay plug-in* (see fig 6)*;* a multi-hop multicast protocol suited for MANETs with high node mobility. This is an unstructured overlay that intelligently floods events. Each node intelligently decides whether or not each message received should be forwarded to its neighbors. The decision is based on previous messages that the node has received; if a large number of duplicates of a message have already been received, the probability the message is forwarded reduces.

Furthermore, by default, the personality in each PDA (i.e. vehicle) is configured to operate over the probabilistic multicast overlay plug-in (in WLAN ad-hoc mode). The overlay CF is dynamically reconfigured to operate over an IP multicast plug-in (WLAN in infrastructure mode), as shown in fig 6, when the environmental context changes (i.e. 'if PDA is within the coverage of a fixed base station').

4.3 A GREEN Configuration for Wide Area Networks

The GREEN configuration for WAN (see figure 7) is configured to address the following requirements and constraints of the fixed WAN environment:

- support events communication in wide area fixed infrastructure based networks to enable distributed vehicular traffic monitoring and control
- support content based interaction type

Publish-Subscribe Interaction plug-ins for WAN
Similar to MANET configuration, the configuration for WAN (see fig 7) shows how the component plug-ins are composed within the interaction CF and the overlay CF to meet the requirements and constrains of WANs. Here we focus only on the differences compared to the MANET configuration. The configuration illustrated in figure 7 allows a *content* based subscription (in addition to topic) only. Assuming content filters would be adequate here, the CLIPS plug-in to specify rule based *composite events* is not configured. WAN is used for disseminating non critical traffic data; hence do not require stringent QoS requirements as in VANET. Therefore, the TCB plug-in is not configured as well.

Event Broker Overlay plug-ins for WAN
The WAN environment requires large scale publish-subscribe middleware between elements across the Internet. In the default configuration shown in fig 7; the event

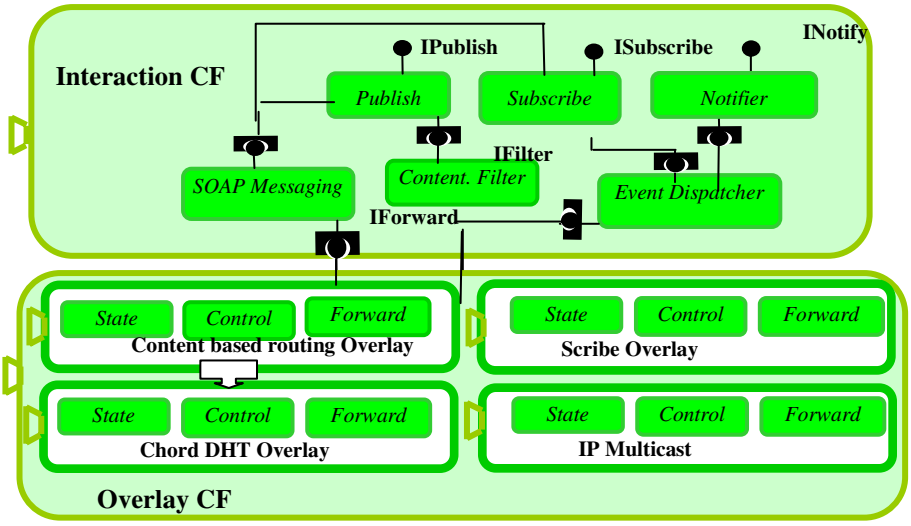

Fig. 7. GREEN configuration for WAN

broker overlay plug-in is underpinned by a Chord DHT (distributed hashtable) [21] overlay.

Here, the event broker overlay plug-in uses rendezvous nodes in the network, which are special event brokers that are known to both producers and consumers. For each event type, a rendezvous node exists in the network. An event type is hashed to a rendezvous point. When a consumer subscribes, the subscription (i.e. content filter) is forwarded towards the rendezvous node R. Every broker that forwards a subscription stores the content filter and event type. Event publications are routed to rendezvous nodes for the event type and then follow the reverse path taken by the subscriptions. This event broker overlay plug-in is similar to the basic event routing mechanisms adopted in Hermes [7]. This overlay plug-in is suited for fixed WANs where the broker topology is fixed. The advantage is the support for content based routing, which is more scalable in WANs. The alternative overlay plug-in as illustrated in fig 7 is Scribe over Chord overlay where each event type is hashed to a Scribe multicast group. This alternative overlay plug-in does not support content based event routing but supports the content based filtering at consumer side. The configuration handles situations where the broker network is subject to topology changes triggered by node and/or link failures. This is made possible as the Scribe overlay [19] manages broker topology changes [19]. Moreover IP multicast plug-in can be configured instead of the Scribe overlay in conditions where network supports IP multicast. A node which is configured to have separate IP multicast and Scribe overlay plug-ins can act as a bridge between the WAN and the VANET (e.g road side base stations mentioned in the application scenario), hence, enabling dissemination of vehicular sensor data generated from VANETs to be distributed over WANs.

Both the above configurations (i.e for MANET, WAN) have been implemented. In addition other configurations, not discussed in this paper, have been integrated to provide publish-subscribe communication infrastructure in our other integrated middleware platforms e.g. CORTEX Middleware for sentient object based, context aware applications in mobile ad-hoc networks [33], ReMMoC a service oriented middleware for mobile clients in infrastructure based wireless networks [34] and GridKit for GRID applications in large-scale networks [18]. The case study clearly demonstrates how the single flexible GREEN middleware is adaptable to various events matching schemes and underlying infrastructure.

5 Evaluation

This section provides concrete performance results of the GREEN family of configurations. It provides quantitative evaluation results on 1) the cost of personality configuration and dynamic reconfiguration and 2) memory footprint cost of GREEN middleware. The experiments utilize a combination of following base device types 1) PDA: HP iPAQ h5450 pocket PC device with a 206MHz strongARM processor, 64Mbytes of system RAM, windows CE 3.0 operating system and IEEE 802.11b wireless network at 11Mbytes/s; 2) PC: 1.7GHz processor and 256Mbytes RAM with windows XP and 100Mbytes/s fast Ethernet.

Experiment 1: Measurements of start-up, configuration, and dynamic fine-grain reconfiguration operations. This experiment evaluates the performance costs incurred by three reflective operations provided by OpenCOM run-time (i.e. loading,

binding, dynamic reconfiguration) on a PDA hosting the GREEN configuration for MANETs (see fig 6 for the configuration). The experiments measure the timing cost of loading components and configuring into configuration A (i.e. IP multicast as the overlay plug-in, see fig 6) and configuration B (i.e probabilistic multicast as overlay plug-in) and finally, the cost of dynamic fine-grain reconfiguration from configuration A to configuration B and vice versa by changing the overlay plug-ins . The results of the experiments are illustrated in fig 8 and demonstrate where the actual overheads occur.

Personality	Components Load time (ms)	Components binding time(ms)	Total start-up time(ms)	Dynamic reconfiguration time(ms)	Total components	Number bindings
Config' A	2580	176	2756	77 (A to B)	10	10
Config' B	2587	171	2758	76 (B to A)	10	10

Fig. 8. Measurements of base reflective operations

The components load time is the most expensive reflective operation and consumes a large part of the overhead incurred in personality configuration. This is because each component is a separate dynamic link library (DLL) that must be first loaded into program memory from storage memory in Windows CE operating system. The component configuration time (i.e. binding time) represents the time taken to initiate a new configuration personality by binding component interfaces to component receptacles. Configuring the middleware personality costs less compared to loading the components (i.e configuring takes 6.33% of the time compared to 93.66% time taken to load components in config' A personality). Furthermore, the time taken to do fine-grain reconfiguration is consistently lower compared to the initial startup time of the personalities (i.e. fine-grain reconfiguration took 2% of the initial startup time).

Impact of dynamic fine grain reconfiguration. Fig 9 illustrates the experiment investigating the impact of dynamic fine-grain reconfiguration, on a topic-based publish-subscribe service invocation. For this purpose, the middleware was used to invoke 1000 publish calls using both configuration A (i.e IP multicast as overlay plug-in) and configuration B (probabilistic multicast as overlay plug-in) within a host, and dynamically reconfiguring between the two with varying levels of frequency. The first test involved no dynamic reconfiguration; this is a simulated base test of the time taken to perform 500 publish invocations using configuration A and 500 publish invocations using configuration B. Subsequent tests used the architecture meta-model interface of the CFs to dynamically reconfigure the underlying overlay plug-in. In test

Test Description	Time(ms)	Publish calls/second	% Time increase from base test 1
1) 500 publish calls using config A + 500 publish calls using config B	3185	313.97	0
2) 500 config A then 500 B	3283	304.59	3.07
3) 250 config A then 250 B (x2)	3853	259.53	20.97
4) 100 config A then 100 B (x5)	5198	192.38	63.20
5) 50 config A then 50 B (x10)	6260	159.74	96.54

Fig. 9. Cost of dynamic reconfiguration

two, 500 publish calls were performed by configuration A, then dynamically reconfigured to configuration B and then 500 further publish calls were made. Similarly, test three performed 250 publish calls using configuration A then 250 publish invocations using configuration B and this was repeated again.

The results of the five tests are shown in fig 9. It can be seen, as the frequency of reconfigurations increases, the time taken to perform 1000 invocations increases.

For behavior where reconfiguration is generally *out-of-band*, i.e. infrequent compared to the number of base service calls, the additional overhead is less significant (a 3.07% increase in time). However, as fine grain reconfiguration becomes more frequent, e.g. 10 reconfigurations in 1000 base invocations, the overhead becomes significantly greater (a 96.54% increase in time). An example out-of-band scenario which requires the above reconfiguration is, where a PDA is required to reconfigure from configuration A to configuration B when the PDA loses the coverage of a base station. Then the PDA have to use an ad-hoc multicast protocol such as probabilistic multicast overlay instead of IP multicast (i.e. as no support in MANET) to communicate with its peers. Overall the experiment shows dynamic reconfiguration does not necessarily result in high performance cost.

Experiment 2: Evaluation of the memory footprint cost of GREEN

At present mobile and embedded devices have a limited amount of system memory, which can quickly be consumed by the applications. Therefore it is important to minimize the amount of memory needed to store the middleware implementations in a device.

Config-No	Descriptions	Environment
1	Topic based P/S over IP Multicast	WinCE, WLAN
2	Topic based P/S over Prob. Multicast overlay	WinCE, WLAN
3	Content based P/S over IP Multicast	WinCE, WLAN
4	Content based P/S over Prob. Multicast overlay	WinCE, WLAN
5	Proximity based P/S over IP Multicast	WinCE, WLAN , GPS
6	Proximity based P/S over Prob. Multicast overlay	WinCE, WLAN, GPS
7	Config' 3 + QoS (TCB)	WinCE, WLAN
8	Config' 6 + Composite events spec(CLIPS)	WinCE, WLAN , GPS

Fig. 10. Test configurations for memory footprint measurements on PDA

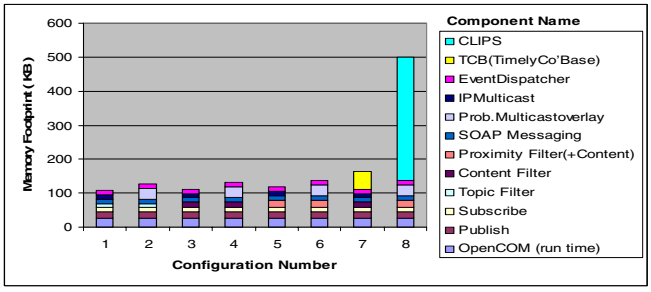

Fig. 11. Memory footprint of GREEN configurations for PDA (MANET)

Config- No	Descriptions	Environment
1	Topic based P/S over IP Multicast	WinXP, LAN
2	Topic based P/S over Scribe overlay	WinXP, WAN
3	Content based P/S over IP Multicast	WinXP, LAN
4	Content based P/S over Scribe overlay	WinXP, WAN
5	Config4+ Composite events spec(CLIPS)	WinXP, WAN

Fig. 12. Test configurations for memory footprint measurements on PC (WAN)

This section examines the resource costs in terms of the static memory footprint of diverse GREEN configurations. Fig 10 documents some valid configurations currently supported for MANETs and fig 11 illustrates the memory costs of the corresponding configurations. Similarly fig 12 documents some valid configurations currently supported for WANs and fig 13 illustrate the memory costs of the corresponding configurations. All the implemented and listed components (in fig 11,13) are OpenCOM components and are implemented in C/C++, except the Scribe overlay component which is implemented in Java. Fig 11 and 13 also illustrate the constituent components of the respective configurations and show how different configurations are composed. The configurations are suited for mobile devices with limited memory, as most configurations for PDAs (WinCE) are around 100Kbytes (e.g. configurations 1, 3, 5 in fig 11). The most expensive configuration in terms of memory for PDAs is configuration 8 and this is mainly due to the high footprint of the CLIPS component. Furthermore, GREEN conserves memory in two distinct levels. Firstly, by only storing the components required by the personality in the storage memory of the device (i.e savings on the storage memory of the PDA). Secondly, only the components that are currently used by configuration are loaded (the OpenCOM run-time provides operations to load and unload components at run-time) into program memory from storage memory (i.e. savings in program memory usage) and components are unloaded from program memory if they are no longer required by the new configuration.

Overall, this experiment shows how GREEN family of configurations achieves low memory footprint despite its generality and flexibility.

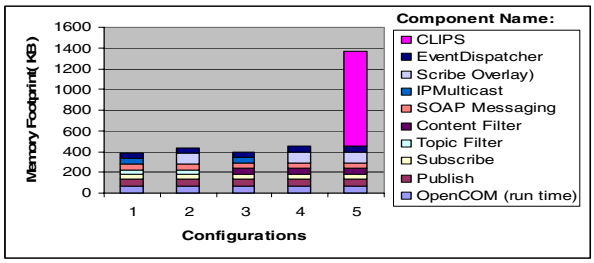

Fig. 13. Memory footprint of GREEN configurations for PC (WAN)

6 Related Work

Today there exists many research and commercial publish-subscribe systems such as SIENA [5], JEDI [8], Gryphon [6], Elvin [36], MSMQ [37], SonicMQ [38]. This

research has primarily focused on the support of various non-functional properties such as scalability, reliability, etc, largely on fixed network environments. Furthermore, some have attempted to address the emergence of mobile computing: JEDI [8] extends support for client mobility within an infrastructure based wireless networks where event brokers are fixed; and STEAM [9] is specifically designed for mobile ad-hoc networks where broker topology constantly changes. From the functional point of view, existing systems implement a fixed publish-subscribe interaction type (i.e. topic based or content based etc). In general, less emphasis has been placed upon publish-subscribe systems which are open, configurable and re-configurable to support changeable publish-subscribe interactions types which embraces diverse network types and device types.

The only work we know, of constructing highly configurable publish-subscribe middleware are DREAM [39], REDS [40] and the work of Filho et.al [41]. DREAM [39] provides a component framework for configurable and dynamic message-oriented middleware. However, DREAM does not explicitly support distributed network of event brokers. A configurable and dynamic notification service is provided by [41]. However, it does not explicitly support MANETs. REDS [40] provides a configurable, distributed event dispatching system. However it is configurable only within the scope of content based systems in WANs and lacks support for dynamic re-configuration of middleware. Furthermore, these systems do not explicitly support pluggable interaction types and they do not explicitly embrace different network types and device types and hence fall short of providing the level of re-configurability of GREEN.

Finally, there are number of middleware platforms that take a reflective approach to provide configurable and reconfigurable system. However, their focus has largely been on synchronous interaction, e.g. DynamicTAO [42] and UIC [43] are CORBA ORBs offering remote object invocations. Furthermore, there is considerable research in the narrower field of overlay networks themselves but this work is largely orthogonal to our focus. In particular, there are numerous multicast and routing protocols for ad-hoc networks such as MAODV, AMRoute, CAMP, MCEDAR which can underpin different overlays in MANET [31].

7 Conclusions

In this paper we have described our approach to the provision of open, highly configurable and re-configurable publish-subscribe middleware that embraces different network types and interaction types. We have empirically demonstrated using an evaluation scenario: that our architecture, has considerable generality and flexibility in supporting pervasive computing applications. The pluggable event broker overlay structure enables us to embrace different network types such as mobile ad-hoc networks and large scale networks. The pluggable interaction types provide a powerful programming model for the application developers. The architecture is extensible in that new publish-subscribe interaction types and event broker overlays can be developed and plugged into the middleware, even at run-time. Furthermore, performance evaluations of the middleware have demonstrated that flexibility is not necessarily at the expense of performance. Furthermore, the performance figures provide a clear insight into the relative performance tradeoffs for different

configurations. Ongoing work is investigating the impact on GREEN in sensor networks based on motes, tacking the issues of memory size and reconfiguration in such areas.

Acknowledgments

This work is partly supported by the IST-FET-2000-26031, (CORTEX- CO-operating Real-time senTient objects: architecture and EXperimental evaluation) project and FP6-IST-004536 (RUNES-Reconfigurable Ubiquitous Networked Embedded Systems) project.

References

[1] M. Weiser. Ubiquitous computing. IEEE Hot Topics, 26(10):71--72, 1993.
[2] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri. Generic support for distributed applications, IEEE Computer, 33(3):68--76, 2000.
[3] Blair, G.S., Campbell, A.J., Schmidt, D.C., "Middleware Technologies for Future Communication Networks", IEEE Network, Vol. 18, No. 1, January 2004.
[4] C. Mascolo, L. Capra, Emmerich,w. "Middleware for Mobile Computing (A Survey)". In Advanced Lectures on Networking - Networking 2002 Tutorials, Pisa, Italy. volume 2497 of LNCS, pages 20-58.
[5] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf "Achieving Expressiveness and Scalability in an Internet-Scale Event Notification Service". Nineteenth ACM Symposium on Principles of Distributed Computing (PODC2000), Portland OR. July, 2000
[6] G.Banavar et al. An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems. In Proc. of the 19th Int. Conf. on Distributed Computing Systems, 1999.
[7] P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In Proc. of the 1st Int. Workshop on Distributed Event-Based Systems, July 2002.
[8] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. IEEE Trans. on Software Engineering, 27(9):827–850, September 2001.
[9] Rene Meier, V. C. "Steam: Event-based Middleware for Wireless Ad Hoc Networks.". In Proceeding of the International Workshop on Distributed Event-Based Systems (DEBS'02), Austria. 2002.
[10] 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.
[11] Blair, G., Coulson, G., Grace, P., "Research Directions in Reflective Middleware: the Lancaster Experience", Proceedings of the 3rd Workshop on Reflective and Adaptive Middleware (RM2004) co-located with Middleware 2004, Toronto, Ontario, Canada, October 2004
[12] 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.
[13] Coulson, G., Blair, G.S., Grace, P., Joolia, A., Lee, K., Ueyama, J., "OpenCOM v2: A Component Model for Building Systems Software", Proceedings of IASTED Software Engineering and Applications (SEA'04), Cambridge, MA, ESA, Nov 2004.

[14] 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, pp 33-38, 2002.
[15] Szyperski, C.,Component Software: Beyond Object-Oriented Programming. Addison Wesley, 1998.
[16] Coulson, G., Grace, P., Blair, G.S., Cai, W., Cooper, C., Duce, D., Mathy, L., Yeung, W.K., Porter, B., Sagar, M., Li, J., "A Component-based Middleware Framework for Configurable and Reconfigurable Grid Computing" to appear in Concurrency and Computation: Practice and Experience, 2005.
[17] Doval, D ,O'Mahony, D, "Overlay Networks: A scalable alternative for P2P", IEEE Internet computing, jul-aug 2003
[18] Grace, P., Coulson, G., Blair, G., Mathy, L., Duce, D., Cooper, C., Yeung, W., Cai, W., "GRIDKIT: Pluggable Overlay Networks for Grid Computing", Proceedings of International Symposium on Distributed Objects and Applications (DOA), Larnaca, Cyprus, October 2004
[19] 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.
[20] 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.
[21] 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.
[22] X. Chen, Y. Chen, and F. Rao, "An Efficient Spatial Publish Subscribe System for Intelligent Location-Based Services," Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS '03), June 2003
[23] S. Schwiderski. Monitoring the behaviour of distributed systems. PhD thesis, University of Cambridge, April 1996.
[24] A. P. Buchmann. Architecture of active database systems. In N. W. Paton, editor, Active Rules in Database Systems, 2: 29–48. Springer-Verlag, 1999.
[25] S. Chakravarthy and D. Mishra. Snoop: An Expressive Event Specification Language for Active Databases. Data and Knowledge Engineering, 14(1):1–26, November 1994.
[26] Gary Riley. CLIPS homepage. http://www.ghg.net/clips/CLIPS.html, 2002.
[27] Charles Lanny Forgy. RETE: A Fast Algorithm for the Many Patterns/Many Objects Pattern Match Problem. Artificial Intelligence, 19(1):17–37, September 1982.
[28] Sivaharan, T., Blair, G.S., Friday, A., Wu, M., Duran-Limon, H., Okanda, P., Sørensen, C.F., "Cooperating Sentient Vehicles for Next Generation Automobiles", Proc of the MobiSys, 1st ACM Workshop on Applications of Mobile Embedded Systems (WAMES 2004), Boston, USA, June 6, 2004
[29] Collaborative Robotics Research at Lancaster university http://www.comp.lancs.ac.uk/computing/users/angie/rendezvous/robotics.html
[30] Antonio Casimiro, Paulo Verissimo. Using the Timely Computing Base for Dependable QoS Adaptation. In Pro of the 20th IEEE Symposium on Reliable Distributed Systems, pages 208–217. IEEE Computer Society Press, 2001.
[31] Royer, E. M., Toh, C-K., A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks, IEEE Personal Communications Magazine, pp 46-55, April 1999.
[32] R.Cunningham and V. Cahill, "Time Bounded Medium Access Control for Ad Hoc Networks", in Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC'02). Toulouse, France: ACM Press, 2002, pp. 1-8.

[33] Sørensen, C.F., Wu, M., Sivaharan, T., Blair, G. S., Okanda, P., Friday, A., Duran-Limon, H., "A Context-Aware Middleware for Applications in Mobile Ad Hoc Environments", Proc' of the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC'2004) at Middleware 2004, Toronto, Canada, October 2004.
[34] Grace, P., Blair, G. S, Samuel, S., "ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability". In Proceedings of International Symposium on Distributed Objects and Applications (DOA), Catania, Sicily, Italy, November 2003.
[35] S.Chen, p. Greenfield: QoS evaluation of JMS: an empirical approach, In Proc. of the 37th Hawwaii International Conference on System Sciences, Hawaii, USA, 2004
[36] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelpsothers. Content Based Routing with Elvin4. In Proc. of the 2000 Australian UNIX and Open Systems Users Group Annnual Conf., Canberra, Australia, June 2000.
[37] Microsoft Message Queuing (MSMQ),2002. Microsoft, http://www.microsoft.com/msmq/
[38] SonicMQ, 2002. Sonic software, http://www.sonicsoftware.com
[39] M. Leclercq, V. Quema, and J.-B. Stefani. Dream: a component framework for the construction of resource-aware, reconfigurable moms. In Proc. of the 3rd Workshop on Adaptive and Reflective Middleware, pages 250–255. ACM Press, 2004.
[40] Gianpaolo Cugola, Gian Pietro Picco. REDS: A Reconfigurable Dispatching System" technical report , Politecnico di Milano (submitted for publications) ,2005
[41] Silva Filho R. S., De Souza C. R. B., Redmiles D. F. The Design of a Configurable, programmable and Dynamic Notification Service. in Proc. Second International Workshop on Distributed Event-Based Systems (DEBS'03), USA, June 8th, 2003.
[42] 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.
[43] Roman, M., Kon, F., Campbell, R., "Reflective Middleware: From Your Desk to Your Hand", IEEE Distributed Systems Online, 2(5), August 2001.)
[44] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, The many faces of publish/subscribe, ACM Computing Surveys, (2):114--131, 2003.
[45] XML Path Language (http://www.w3.org/TR/xpath20/)
[46] Grace, P., Coulson, G., Blair, G.S., Porter, B., "Deep Middleware for the Divergent Grid", Proc. IFIP/ACM/USENIX Middleware 2005, Grenoble, France, November 2005.