

Addressing Network Heterogeneity in Pervasive Application Environments

Paul Grace, Geoff Coulson, Gordon Blair, Barry Porter

Abstract— Pervasive computing applications typically involve rich interactions and heterogeneous network types; e.g. involving the collation of data from a sensor network into a replicated repository in a fixed network. Although the middleware approach has been highly successful in supporting application development in networked environments, current middleware technologies cannot handle the accelerating complexity in interaction types, and diversity in networks types, seen in pervasive computing environments. Therefore, we propose a middleware solution (called Gridkit), which uniformly supports an extensible set of middleware interaction types (e.g. RPC, publish-subscribe, streaming, etc.), and handles network heterogeneity by layering itself over virtual overlay networks which it manages and transparently instantiates on demand. We focus in this paper on Gridkit’s generalized architecture for the transparent deployment and management of overlay networks. We also consider the application of the Gridkit approach in two application scenarios.

Index Terms—Middleware, Overlay Networks, Pervasive Computing.

I. INTRODUCTION

Pervasive Computing applications often involve rich interactions between users and devices in highly heterogeneous networked environments involving, e.g., high-speed LANs, lower-speed WANs, ad-hoc networks, and specialized sensor networks. Consider, for example, an application that collects data from sensor networks and then stores and processes the data on powerful end-systems in the Internet; or an application involving mobile users on wireless networks (infrastructure or ad-hoc) interacting with Internet-based users and services.

Middleware is a well-established solution for providing interaction services in the face of end-system, operating system and network heterogeneity. Furthermore, middleware simplifies the task of the application developer by managing the problems of distribution. However, with respect to the types of applications considered above, current middleware technologies are limited in the following three respects:

- Middleware solutions are typically network-style centric. That is, their operation is suited to individual network types. For example: wireless network middleware transparently handle disconnection [1][2]; ad-hoc network middleware promotes abstractions suitable for ad-hoc interactions [3][4]; and enterprise middleware assumes high connectivity in fixed networks [5][6].
- Middleware technologies typically support individual

services, e.g. resource discovery, remote procedure call (RPC), publish-subscribe, etc. However, one-size does not fit all network types; for example, data sharing is well suited to ad-hoc interaction in ad-hoc and sensor networks [7][8], whereas RPC is better suited to infrastructure networks.

- Deployed middleware services are static. That is, a running service cannot evolve if the underlying network characteristics change (e.g. if it first operates in the Internet, and later extends to include a sensor network).

To address these limitations we present in this paper a network-style independent middleware platform called Gridkit [9]. Gridkit addresses two key requirements within the pervasive application domain:

- *Service Deployment.* Developers must be able to choose a suitable middleware service, which must then be deployed in an environment that potentially encompasses multiple network types.
- *Dynamic reconfiguration.* Deployed communication services must be able to adapt to changing network heterogeneity.

The key to our approach is to leverage the notion of *overlay networks* to construct middleware services (we consider an overlay network as a virtual communications structure that is logically ‘laid over’ an underlying physical network or networks). In particular, we focus on policy-based selection of overlays to deploy the best overlay type(s) in different network environments, and dynamic bridging to build services across multiple network types.

We motivate this approach by eliciting middleware requirements from two application scenarios, and evaluate the approach by documenting how effectively the middleware solution can meet such requirements in one of these scenarios.

II. EXAMPLE SCENARIOS

This section analyses two pervasive computing scenarios that are characterized by the need for multiple middleware services and the need to integrate different network types.

A. Forest Fire Fighting

This scenario is focused on forest or savannah fire fighting in remote, poorly resourced locations. There are two user roles involved: *controllers* and *fire fighters*. Controllers manage the operation: they move fire fighters, issue commands, decide where to deploy fire sensors, and investigate real-time simulations that predict the spread of the fire. Fire fighters use mobile devices, on which graphics-based commands from controllers are displayed, and deploy wind speed sensor networks.

Figure 1 illustrates a typical network configuration for this scenario: fire fighters form ad-hoc connections between

This work was funded by the EPSRC under the Open Overlays project (grant reference GR/S68521/01). The authors are with the Computing Department, Lancaster University, Lancaster, UK. (phone: +44 1524-510-393; fax: +44 1524-510-492; contact e-mail: p.grace@lancaster.ac.uk).

themselves, sensors and on-site controllers; and infrastructure networks connect all controllers. Depending on the location of the fire, the connections between on-site and remote controllers could be provided by satellite, GPRS, Wireless LAN, or other network types.

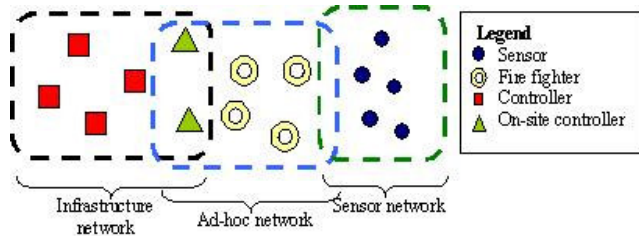


Fig. 1. Network types in the fire fighting scenario.

Middleware Requirements

- A group communication service is needed to enable controllers to disseminate commands to fire fighters (where to move, which part of fire to fight, where to put sensors, etc.).
- A publish-subscribe service is required for the collection of sensor events to be fed to the controllers' fire spread simulations.

B. Environmental informatics

In this scenario, a river, estuary and bay are instrumented with sensors to monitor temperature, water level, flow rate, pollution levels etc. Some of these sensors are networked using Ethernet (e.g. sensors in tidal-defence walls), while others employ wireless technologies (e.g. IEEE 802.15.4 or 802.11 radios; or long-wave radios for underwater use). These may be mobile in the water, and will come into the range of fixed sensors in an ad-hoc fashion. Point-to-point microwave connectivity may also be used to link individual sensor networks to strategically placed IP gateways at which sensor data is collated and cached.

Given this infrastructure, scientists in widely-dispersed locations 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. Figure 2 illustrates a typical network configuration for this scenario.

Middleware requirements

- A data retrieval service is needed to collect data from the sensor networks.
- An event service is needed to 'push' sensor data to long-running simulations.
- A video streaming service is needed to support video conferencing between fixed workstations.

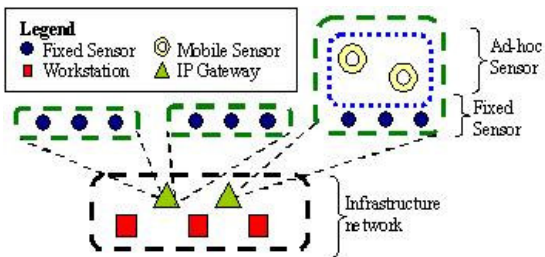


Fig. 2. Network types in the environmental informatics scenario.

III. AN OVERLAY-BASED MIDDLEWARE

Gridkit is a novel middleware framework that can support the development of scenarios of the types described above. It achieves this by deploying an extensive and extensible set of middleware services over an infrastructure of overlay networks which it itself creates and manages. The operation of Gridkit is depicted in figure 3: to deploy a middleware service involving nodes situated in different network types, Gridkit deploys in each physical network an overlay that is optimised for that network type, and then constructs an "overlay of overlays" that bridges the individual overlays. This provides a uniform network infrastructure that includes all the participating nodes and supports the required middleware service(s).

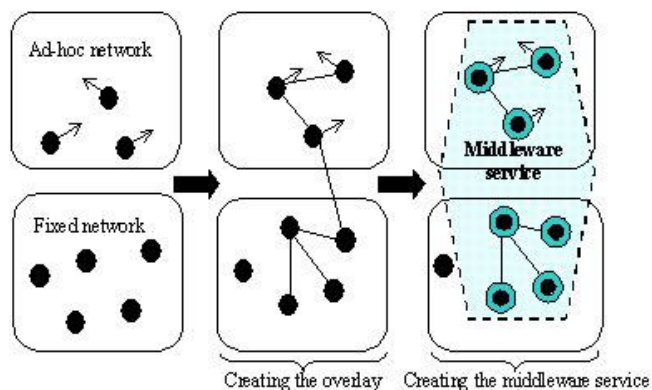


Fig. 3. Deploying a middleware service that span multiple network types

In terms of software architecture, the Gridkit framework (illustrated in figure 4) is deployed on each participating node. At the bottom level it employs a minimal runtime for the loading and binding of lightweight software components [10]. This runtime is so minimal that it can be supported even on very primitive devices. The next layer up (the *overlays framework*) is a distributed framework for the deployment of multiple overlay networks. In practice, this amounts to hosting, in a set of distributed overlay framework instances, a set of per-overlay plug-in overlay components (see figure 5). Each of these represents a single overlay network node and consists of a *control* element that cooperates with its peers on other hosts to build and maintain some virtual network topology, a *forwarding* element that appropriately routes messages over its virtual topology, and *state* element that contains per-overlay-node state such as a next-neighbours list.

Above the overlays framework is a set of further “vertical” frameworks that provide functionality in various orthogonal areas, and can optionally be included or not included on different devices. In brief, the frameworks are as follows: the *interaction framework* accepts multiple interaction type plug-ins (e.g. RPC, publish-subscribe, group communication); the *service discovery framework* accepts plug-in strategies to discover application services (e.g. SLP, UPnP, Salutation); the *resource discovery framework* accepts plug-in strategies to discover resources such as CPUs and storage (e.g. peer-to-peer search); the *resource management* and *resource monitoring* frameworks are respectively responsible for managing and monitoring resources; and the *security* framework provides general security services for the rest of the frameworks. These frameworks are discussed in more detail in [9], as is the web services layer that (optionally) sits atop the frameworks and provides a uniform web-service based API for applications.

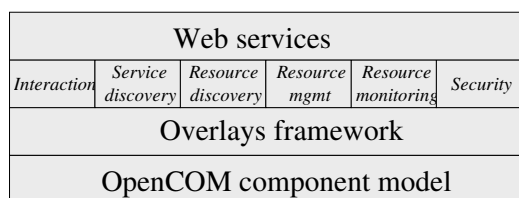


Fig. 4. The Gridkit Architecture.

Essentially, the Gridkit architecture is based upon the customised stacking of software components to achieve the required local deployment. Overlay plug-ins can be stacked on other overlay plug-ins to produce a required (virtual) network service. Then, middleware components are stacked atop these overlays to yield a required middleware service. The configuration of components is not created until run-time; hence, applications can be deployed in ‘black-box’ environments.

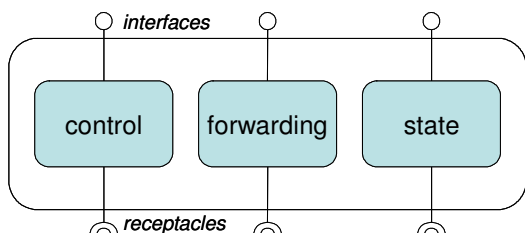


Fig. 5. Structure of an overlay plug-in

IV. DEPLOYMENT & RECONFIGURATION OF MIDDLEWARE SERVICES

In this section we present the techniques used in Gridkit to support the deployment and reconfiguration of overlay-supported middleware services across multiple hosts situated in heterogeneous networks. In particular, we focus on how suitable deployment and reconfiguration decisions are made and enacted.

A. Policy-based Deployment of Middleware Services

The task of the middleware service deployment process is to

ensure that an appropriate “stack” of software components is in place on each participating node. Participating nodes each take one of the following two roles: they are either a *member* of the middleware service (i.e. a participant that actively uses the middleware service); or a *facilitator* of the service (i.e. they do not use the service, but rather help to support its infrastructure).

Each middleware service is built on demand in an incremental fashion that is initiated when a potential member node requests a service. Such nodes present to Gridkit a *policy file* that describes how the required service can be constructed. Gridkit then creates or joins the service (as appropriate). It also takes into account the current dynamically-determined *environmental context* (e.g. available physical network types) of the node. An example policy file, described in XML, is shown in figure 6; this is part of a policy used to build a group communication service on top of a multicast overlay service. The file states that if the environmental context is ‘fixed network’, the overlay framework should deploy a Scribe [11] plug-in atop a Chord [12] plug-in (n.b. “Scribe.xml” is a software architecture description of components and connectors). An alternative policy for an ad-hoc network context (not shown) might deploy, say, an epidemic multicast overlay.

```

<ConfigurationRule>
  <Framework>OverlayFramework</Framework>
  <Interface>IMulticast</Interface>
  <attrs>
    <network>fixed</network>
  </attrs>
  <Configuration>
    <Layers>
      <Framework>Scribe.xml</Framework>
      <Framework>Chord.xml</Framework>
    </Layers>
  </Configuration>
</ConfigurationRule>

```

Fig. 6. An example policy for overlay deployment

The limitation of the above-described process is that it does not yet take into account the facilitator role. Furthermore, it will lead to disjoint middleware services when the service spans network types, i.e. we can provide a multicast service in both ad-hoc and fixed networks, but not across both. We now present refinements of the basic process described above that address both these problems.

B. Leveraging Facilitators

To involve facilitators in the deployment process we utilise two techniques: i) a stack configurator, that is inspired by the Ensemble [15] micro-protocol framework; and ii) gossip-based dissemination of deployment information to nearby Gridkit nodes.

Deployment Configurator

All messages disseminated using overlay-based middleware services in Gridkit attach information in their header about the “software stack” used to create the message. For example, if a

message was sent on a group service layer, above the Scribe and Chord overlay plug-ins, there would be three ordered identifiers in the header: `chord::scribe::gk_group`; these are added in LIFO fashion. This list is then used on the receiving host to traverse the component configuration and ensure it is processed by the same components that constructed it. Notably, we re-use this approach to support deployment. When a message cannot be ‘up-called’, because that layer hasn’t been deployed, it is sent to the stack configurator that then deploys the correct components.

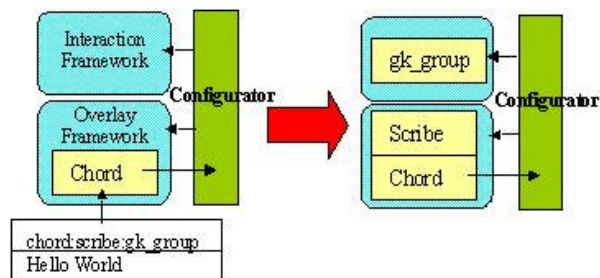


Fig. 7. Message based deployment of stacked components

Figure 7 illustrates this process; for example, a message is received on the Chord overlay pertaining to a Scribe-gk_group service, but the Scribe and gk_group layers are not in place in the architecture, so the message is sent to the configurator. The configurator then uses reflection to determine which elements required by the message are missing in the architecture, and then dynamically inserts what is needed.

Gridkit Deployment Service

The deployment configurator process works for cases where the nodes involved in the same overlay. Facilitators in a different overlay network will not receive messages sent within the other network. Hence, we leverage the concept of a common deployment service across all Gridkit nodes. In this case, the service is provided using gossip style dissemination [13]; messages sent by the frameworks (like that in figure 8) are periodically selected and sent probabilistically to local Gridkit nodes, which then gossip the message further. When the gossip service receives a message it is uploaded to the stack configurator, which decides whether to deploy the required components. At present, we make the assumption that the deployment service is available on executing Gridkit nodes, and software components are available locally to be deployed.

Gridkit may be deployed across multiple network types; hence, the service gossips information independently of the network. Figure 8 illustrates the deployment service set-up in each Gridkit framework; in this case a gossip version for each network type (ad-hoc and fixed) is plugged-in, which combine to form a single service; messages received from one domain are gossiped into the other domains. When such a message is received, it is passed to the Gridkit configurator that makes decisions on whether to deploy software to support a service, thus becoming a facilitator for that service.

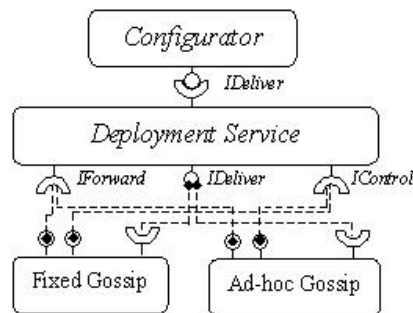


Fig. 8. Gridkit's Deployment Service Architecture

Bridging Overlays

When a middleware service is deployed across disjoint overlay networks, Gridkit employs a dynamic bridging technique to facilitate the merging of the service. For example, when building a group multicast service, the initial deployment may contain a probabilistic multicast overlay in the ad-hoc domain, and a multicast tree overlay [14] in the fixed domain. The service is then disjoint as there is no connection between the two overlays.

A ‘bridge’ in Gridkit is a generic overlay plug-in that is placed above two or more overlay plug-ins (each implementing a disjoint overlay). A bridge can be used to support any type of middleware service, and can be placed above any overlay types (albeit it is sensible to bridge only overlays providing the same service e.g. multicast). The plug-in is implemented using the control-forward approach described in section 3. The control element manages the joining and leaving of disjoint overlays to the bridge; the forward component is responsible for receiving messages from one overlay plug-in and forwarding them to the other plug-ins below (message ids are stored to ensure messages are not duplicated). Finally, the state component stores the neighbour information (bridged plug-ins). Figure 9 illustrates this architecture; here we can see a group middleware service underpinned by two overlay multicast services: probabilistic multicast (PMCast) and a multicast tree (ALM) [14]. This node then acts as a facilitator for the group communication service, merging the nodes from two different overlay networks into a single service overlay.

The final problem to address is: how and where is the bridge deployed? In our current implementations, overlays emerge based upon the policies described in the previous sections and hence nodes with multiple network types (hardware interfaces) will produce multiple stack configurations e.g. a group component over two overlay plug-ins as seen in figure 9. In these cases the configurator detects the multiple overlays and places the bridge automatically. This would occur, for example, on the detection of two disjoint multicast overlays of the same group type (we currently identify a group by its URL). Similarly, if the node has the potential to be a bridge, but isn’t participating in the service, then the deployment service for facilitators can be re-used to deploy a bridge.

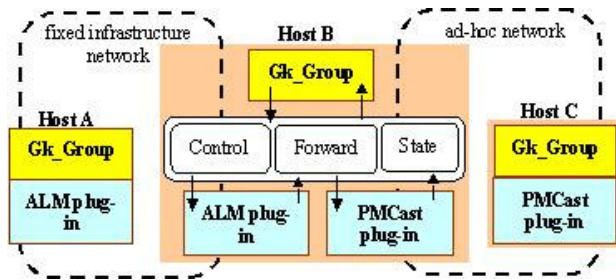


Fig. 9. Bridging overlays in Gridkit

We acknowledge that this approach is limited in that it may produce multiple unnecessary bridges, increasing the amount of duplicate messages forwarded. Similarly, some nodes may be better suited than others to act as a bridge (i.e. ones with more resources and more available bandwidth). We are currently investigating how to reuse the deployment service to disseminate information between Gridkit nodes in order to co-ordinate the deployment of bridges (e.g. potential bridge nodes co-ordinate to place one or more bridges in ideal locations in a scenario).

Reconfiguring Middleware Services

Pervasive computing scenarios do not remain constant over time. It is likely nodes will move, change their role, and importantly change from one network type to another e.g. when a mobile device moves from out of range of a wireless infrastructure network. Therefore, to deploy once and retain the same deployment is unsuitable; rather a middleware service must be dynamically reconfigurable and *self-configuring* to cope with such changes in its individual members. This could take the form of the addition of new overlay networks, or a change in overlay type.

We are currently investigating the dynamic reconfiguration of deployed services. Gridkit is reflective, i.e. it supports the inspection and dynamic reconfiguration of software components, based upon the well-established event action-adaptation procedure. Gridkit configurators subscribe to context events (e.g. change in network, change in location etc.), so when these events occur it can trigger a software reconfiguration. In particular, we are interested in the co-ordinated reconfiguration of communication services across multiple hosts, requiring a shared decision making process.

PRELIMINARY EVALUATION

In this section, we provide a qualitative evaluation of our techniques to build middleware services using overlay networks in order to overcome the problems of creating applications over different network types. For this we take the initial requirements we elicited from fire-fighting scenario in section 2 and demonstrate how these are reached using the Gridkit approach.

The forest fire scenario has two requirements: 1) a group communication service between PCs, laptops, and PDAs interconnected by both infrastructure, and ad-hoc networks; 2) a publish-subscribe service between all the devices in the

scenario, across the same diverse network types. The following describes a testbed to simulate the scenario. We connected a set of 3 PCs using a fixed network. One PC was equipped with a wireless network card to connect to an 802.11b network in infrastructure mode. In addition, 2 laptops each equipped with 2 wireless interfaces connected to the wireless infrastructure network, and also operated in ad-hoc mode. Finally, we connected PDAs and sensors using 802.11b in ad-hoc mode.

The applications were deployed on each device type, along with the middleware components. We then investigated the outcome of performing the initial request for a middleware configuration (i.e. both group and publish-subscribe), and then executing a join of that service. Table 1 illustrates the deployment results of these requests. It can be seen that the two requirements are met i.e. the communication services support the applications across the diverse network types. This is essentially achieved by the deployment of the appropriate system configuration on the devices, particularly the additional software bridges.

On Fixed Controllers (FC) in the fixed network we used the Interaction Framework to build the required group communication (GK_GRP) and publish-subscribe (GK_PS) services. These were constructed on top of an Application-Level Multicast tree overlay (TBCP [14]) within the Overlay Framework (OF). For the on-site controllers (MC) there are two network types to bridge, so the same Interaction Framework set-up is built atop bridged overlays – in this case Probabilistic Multicast for the ad-hoc network, and Scribe for the wireless infrastructure. One FC with fixed and wireless network interfaces bridges the other FCs and MCs, therefore including a bridge within the overlay framework.

TABLE 1. FOREST FIRE DEPLOYMENT RESULTS

Role	Device	Network	Software
FC	2 PCs	Fixed	OF → TBCP IF → GK_GRP, GK_PS
FC	1 PC	Fixed & Wireless	OF → TBCP:Bridge, Chord KBR:Scribe:Bridge IF → GK_GRP, GK_PS
MC	2 Laptops	Wireless & Ad-hoc	OF → PMCast:Bridge, Chord KBR:Scribe:Bridge IF → GK_GRP, GK_PS
FF	PDA	Ad-hoc	OF → PMCast:Bridge, IF → GK_GRP, GK_PS
Sensor	PDA	Ad-hoc	OF → PMCast:Bridge IF → GK_PS

Discussion of fire fighting scenario

This scenario involves spontaneous networking, but relies on a reasonable level of connectivity between participants to maintain up-to-date information. That is, fire fighters and sensors that are out of range of nodes in the other networks will hinder the application's operation. Interestingly this problem is helped in the scenario by end-user interaction for the creation of spontaneous networks i.e. controllers can direct the movement of nodes to repair communication services. We are currently investigating this approach to improve reliability and dependability in scenarios of this type.

RELATED RESEARCH

Previous middleware has considered the problem of network heterogeneity. However, this work is largely geared to specific cases of network integration, and tightly coupled to particular interaction types. For example, CORBA [6] offers a bridging solution (Inter-ORB bridges) to ensure distributed objects interoperate across different types of network. Typically these networks are of the same type, however, ALICE [1] and DOLMEN [2] investigate how CORBA bridges operate between wireless infrastructures and fixed infrastructures. Hence, these present a solution for a single interaction type (i.e. RPC) in a single deployment case (spans two specific networks). In a similar vein, publish-subscribe broker networks have been investigated [3][4] that ensure events are sent and received between endpoints in wireless and fixed networks. There are also other middleware solutions that perform specific integration tasks. However, as far as we are aware no middleware supports a general solution to this problem in order to selectively build any communication service atop any combination of network integrations.

CONCLUSIONS & FUTURE WORK

In this paper we have presented our approach to deploying communication services across diverse and heterogeneous communication networks. Fundamentally, this involves a dynamically deployable and reconfigurable middleware framework that constructs and deploys an appropriate configuration of overlay networks to ensure that the application's requirements for a communication service are met. In addition, we have introduced policy-based configuration rules for dynamic deployment that is coupled with a generalised software bridging solution to ensure that communication service messages span overlays, and hence network types.

In future work, we will focus on investigating the deployment of software and development of further real-world scenarios. In this paper, we have presented one approach to system software and application deployment. However, this may not be well suited to all application types. For example, we are also interested in deployment approaches involving super peers; for example, in the case of sensor networks, with limited resources; a more powerful node may manage deployment of components. In this special case, we would assume a base capability of the super-peer to connect to the resource-limited node, and remotely load and configure software (rather than the local node manage its capability). Furthermore, there may be cases where stricter deployment is controlled by end-user (system admin) commands.

Finally, we are also investigating the potential for automation of the deployment process. That is, rather than have fixed policies reside on hosts, these policies can change over time based upon the performance of the communication service. Hence, newer configurations that perform better will be deployed. We envisage that nodes will collude to provide information about their operation, and this data will drive rule-based learning techniques to generate new policy files.

ACKNOWLEDGMENTS

The authors would like to acknowledge their colleagues on the project: Chris Cooper, David Duce, Musbah Sagar, Wei Li, Laurent Mathy, and Wei Cai.

REFERENCES

- [1] M. Haahr, R. Cunningham and V. Cahill, "Towards a Generic Architecture for Mobile Object-Oriented Applications", Proceedings of the 2000 IEEE Workshop on Service Portability and Virtual Customer Environments, pp. 91-96, San Francisco, December 2000.
- [2] M. Liljeberg, K. Raatikainen, M. Evans, S. Furnell, K. Maumon, E. Veldkamp, B. Wind and S. Trigila, "Using CORBA to Support Terminal Mobility". Proceedings of TINA '97, 1997.
- [3] G. Cugola, E. Di Nitto and A. Fuggetta, "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS". IEEE Transactions on Software Engineering, 9(27), pp. 827-850, September 2001.
- [4] G. Mühl, L. Fiege, F. Gärtner and A. Buchmann, "Evaluating Advanced Routing Algorithms for Content-Based Publish/Subscribe Systems", Proceeding of 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, pp.167-176, Texas, October 2002.
- [5] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard, "Web Services Architecture", W3C Working Draft, <http://www.w3.org/TR/ws-arch/>, August 2003.
- [6] Object Management Group, "The common object request broker: Architecture and specification", Tech. Report. Version 2.0, July 1995.
- [7] M. Boulkenafed and V. Issarny, "A Middleware Service for Mobile Ad Hoc Data Sharing, Enhancing Data Availability", *Proceedings of ACM/IFIP International Middleware Conference*, Rio de Janeiro, Brazil, June2003, Heidelberg: Springer-Verlag, 2003, pp. 493-511.
- [8] C. Mascolo, L. Capra, S. Zachariadis, W. Emmerich, "XMIDDLE: A Data-Sharing Middleware for Mobile Computing", International Journal on Personal and Wireless Communications, 21(1), pp. 77-103, April 2002.
- [9] P. Grace, G. Coulson, G. Blair, B. Porter, "Deep Middleware for the Divergent Grid", in *Proceedings of ACM/IFIP International Middleware Conference*, Grenoble, France, December 2006, Heidelberg: Springer-Verlag, 2005, pp. 334 - 353.
- [10] G. Coulson, G. Blair, P. Grace, A. Joolia, K. Lee, J. Ueyama, "A Component Model for Building Systems Software", Proceedings of the IASTED Software Engineering and Applications (SEA'04), Cambridge, MA, USA, Nov 04.
- [11] 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.
- [12] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", Proc. ACM SIG-COMM, San Diego, 2001.
- [13] A. M. Kermarrec, L. Massoulié and A.J. Ganesh "Probabilistic Reliable Dissemination in Large-Scale Systems", IEEE Transactions on Parallel and Distributed Systems, 14(3), pp. 248-258, March 2003.
- [14] Mathy, L., Canonico, R., Hutchinson, D., "An Overlay Tree Building Control Protocol," Proc. 3rd International COST264 Workshop on Networked Group Communication, London, UK, 2001.
- [15] Hayden, M., "The Ensemble System", PhD Thesis, Cornell University, USA, 1998.