A Multi-protocol Framework for Ad-hoc Service Discovery

Carlos A. Flores-Cortés, Gordon S. Blair, Paul Grace
Computing Department
Lancaster University
Lancaster, LA1 4YR, UK
+44 (0)1524 510337
(c.florescortes, gordon, gracep)@comp.lancs.ac.uk

ABSTRACT

Discovering the appropriate services in ad-hoc computing environments where a great number of devices and software components collaborate discreetly and provide numerous services is an important challenge. Service discovery protocols make it possible for participating nodes in a network to locate and advertise services with minimum user intervention. However, because it is not possible to predict at design time which protocols will be used to advertise services in a given context/ environment, it is now becoming clear that dynamic discovery mechanisms are required by mobile nodes to cope with the heterogeneity of discovery platforms. Existing adaptive mobile middleware solutions such as ReMMoC and INDISS have investigated this style of dynamic discovery. However, these have yet to consider the emerging suite of protocols for discovery in ad-hoc networks. In this paper we present a component-based service discovery framework for the development of an adaptive multi-personality service discovery middleware, which will operate in diverse environments e.g. fixed and ad-hoc networks. This supports a common architecture for individual discovery protocols to enhance configurability and re-configurability of the framework, and minimize resource usage through component re-use. Finally, to evaluate this framework we investigate the development of four existing ad-hoc service discovery protocols using our approach.

Categories and Subject Descriptors
C.2.2 [**Network Protocols**]: Applications.

General Terms
Design.

Keywords
Service discovery, middleware, MANET, mobile computing, ad-hoc.

1. INTRODUCTION

Service discovery is an important aspect of mobile computing. The dynamic nature of interactions in mobile environments requires the resource or service to be found first, as there is no prior knowledge of what resources are available in the environment, or what method should be used to communicate with them. Existing adaptive mobile middleware solutions such as ReMMoC [5] and INDISS [1] have investigated this style of dynamic discovery and interaction, particularly focusing on configuring and using the appropriate discovery protocol to find services in the current environment. These systems have so far only considered infrastructure-based wireless networks in which standardised discovery protocols such as Jini[15], Service Location Protocol (SLP)[6], and Universal Plug and Play (UPnP)[17] are utilised. However, there is now emerging a suite of discovery protocols for mobile ad-hoc networks (MANETs), which have been developed to deal with the limitations of nodes in such environments, i.e.: unpredictable network topology, variable number of participating nodes, different levels of dynamicity, and nodes with restricted knowledge of their neighbours. Example protocols are Scalable Service Discovery (SSD) [14], Group Service Discovery (GSD) [2], ALLIA [12] and SLP-based [11]. Hence, we argue that a reconfigurable middleware for service discovery must additionally consider these protocol types for operation in MANETs.

However, this explosion of discovery protocol heterogeneity causes difficulties for both the operation and software development of such middleware. Configuring multiple protocols at run-time can overload the limited resources of the mobile device (e.g. memory consumption and network traffic generation), and a significant effort is involved in the development of individual discovery protocols. Hence, in this paper we propose a component framework approach for the development of a configurable and dynamically reconfigurable multi-personality discovery middleware for operation in both nomadic and MANET style operation. For this framework, we identify a common component architecture that individual discovery protocols follow. In the deployment phase this provides the following benefits: i) component re-use by multiple protocols minimizes resource usage, ii) simplified configuration and dynamic reconfiguration of multiple concurrent protocols. In the development phase, our framework promotes code re-use simplifying the development of new protocols, and allowing them to easily be plugged into a reconfigurable architecture.

The remainder of the paper is organized as follows. Section 2 presents two ad-hoc scenarios to illustrate service discovery heterogeneity. Section 3 then discusses the common interaction pattern present in different ad-hoc Service Discovery Protocols (SDPs) and our general Service Discovery Framework pattern. To evaluate our framework we implemented four ad-hoc SDPs;

section 4 describes the implementation details, and section 5 presents an initial evaluation of our approach. Section 6 then analyses related work in the field of service discovery middleware. Finally, section 7 concludes the paper and discusses our future work in this area

2. AD-HOC SCENARIO

In this section we present two scenarios to provide a motivation for our work. In our first scenario two different taxi companies in a city offer their services using a MANET based dispatch system for communication, as in [7]. Ad-hoc terminals have also been installed in the main city tourist areas to offer tourist information e.g. restaurants in the area, hotels, maps, museums, etc. Other companies have also installed their own ad-hoc servers providing a variety of services e.g. news about weather or traffic conditions, etc. These services are advertised and can be discovered via the ad-hoc network using different SDPs. For example the taxi company A is using GSD to advertise its services whereas company B is using ALLIA; the tourist office is using SSD for weather services, and traffic conditions are advertised using SLP-B. Tourists equipped with ad-hoc devices can send a direct job request to the nearest taxi, also taxi drivers can receive news about weather and traffic conditions.

The second scenario is an E-learning scenario [8], where different information services can be accessed by students using an ad-hoc network. In this example a student is preparing for her final exam. The student is working on her PDA on the patio of a coffee shop on the campus. Before leaving her home, the student has downloaded the PowerPoint slides related to her exam topic onto her PDA. After working through a few slides, she comes across an annotation referring to a paper that provides more details of a specific aspect. Using her PDA she tries to locate the paper somewhere and download it. To do so, her PDA joins to other nearby computers forming an ad-hoc network. Users of some of these computers have similar interests as her and might thus be able to provide the requested information. Fortunately, she has found another computer that not only has the requested paper, but also, has granted her permission to download it. However, the paper is in postscript format, which her PDA is not able to display. Therefore, she tries to locate a conversion service that transforms the paper into a PDF file. Because her PDA battery is low, she decides to print the paper. Finally, she prints out the paper using the library printer which was located using her PDA. Notably, several services are used in this e-learning environment e.g. files repository, file format conversion, printer, etc., and the dynamic nature of interaction means the discovery protocols advertising the services cannot be predicted in advance; in this case a number of different SDPs are used for advertising services.

We argue that fixed middleware offering fixed discovery protocols cannot support these application types. To develop applications for these scenarios middleware platforms should have the following requirements: i) discover services using different ad-hoc protocols, ii) discover services in diverse scenarios and environmental conditions; iii) provide efficient mechanisms to interact with different SDPs simultaneously (i.e. configurability, dynamic re-configurability), and iv) have low performance overhead in terms of resource usage of the device, and network bandwidth consumption.

3. SERVICE DISCOVERY FRAMEWORK

3.1 Ad-hoc Service Discovery Protocols

Existing SDPs like Jini, UPNP and SLP have been developed to advertise and discover services over infrastructure-based wired networks. However, these approaches are not suitable for MANETs environments, firstly because many of them require central directories, and secondly because these solutions were designed without considering the resource constraints typical in wireless networks, therefore they make extensive use of multicast or broadcast transmissions which are almost costless in wired networks but are power hungry in wireless networks. As a result, a suite of SDPs for MANETs has emerged.

Therefore, the design of our framework is formed from the analysis of these existing ad-hoc protocols, where we have found common patterns of interaction among them; these are discussed in detail through this section. Firstly, as a common feature, it was observed that in order to reduce traffic overhead and provide scalable solutions, these protocols group participating nodes in vicinities. Such vicinities are determined by the number of hops (diameter) that a message (i.e. advertisement, request, etc) can reach. To describe their general interaction pattern ad-hoc SDPs can be divided into two groups [3], SDPs that utilize dynamic directories and SDPs that do not.

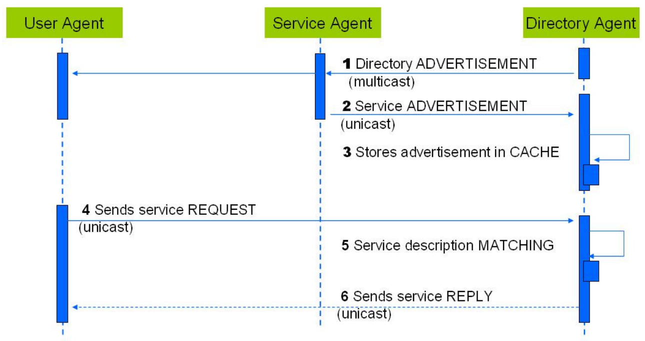

User Agent
Service Agent
Directory Agent
1 Directory ADVERTISEMENT (multicast)
2 Service ADVERTISEMENT (unicast)
3 Stores advertisement in CACHE
4 Sends service REQUEST (unicast)
5 Service description MATCHING
6 Sends service REPLY (unicast)

Figure 1. Interaction pattern of SDPs with dynamic directories.

3.1.1 SDPs with dynamic directories

In this group three different agent types interact to advertise and locate services: i) the User agent (UA) which performs service discovery on behalf of the clients, ii) the Service agent (SA) which represents and advertise services, and iii) the Directory agent (DA) which collects service advertisements and responds to service discovery queries. In contrast with central directories of fixed solutions, ad-hoc directories do not require pre-defined infrastructure and are deployed dynamically as in [14] and [9]. Starting on the right of figure 1, initially DAs (1) advertise their presence by sending multicast messages to nodes in their vicinity; since DAs are deployed dynamically more than one DA could be present in the same vicinity. Then, SAs register their services with DAs by sending unicast messages to them (2,3). UAs interested in locating a service send unicast requests to the DAs (4). DAs match requested service descriptions against descriptions that have been collected from nodes in the vicinity (5). Finally, if a service was found a unicast reply is sent to requesting UA (6).

3.1.2 Directory-less SDPs

As illustrated in figure 2, the interaction pattern of these protocols is very similar to the interaction pattern described for the former group. Notice that even when centralized DA(s) are not required by these protocols, a local DA is present on each node. Hence, to advertise and locate services, SAs (1) advertise their services by sending multicast messages to all nodes in their vicinity (push-based mode), although some protocols could opt for a pull-based approach as in [11]. Local DAs are responsible for maintaining a record of service advertisements from neighbouring nodes (2). So as to locate services DAs will first perform a search on its local cache (3). If a match was not found a multicast service request is sent to neighbouring nodes (4). Finally neighbouring nodes hosting requested service description send a unicast reply to the requesting node (5).

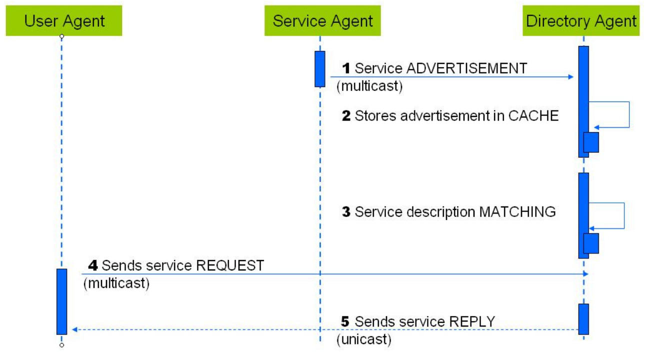

User Agent
Service Agent
Directory Agent
1 Service ADVERTISEMENT (multicast)
2 Stores advertisement in CACHE
3 Service description MATCHING
4 Sends service REQUEST (multicast)
5 Sends service REPLY (unicast)

Figure 2. Interaction pattern of directory-less SDPs.

3.2 Service Discovery Architecture

Based upon the common interaction pattern described in the previous section, we are proposing a component framework [16] approach for the development of a configurable and dynamically reconfigurable multi-personality service discovery middleware. To provide support for the UA, SA and DA functionality, six components were designed in our framework architecture. These components and their relationships are shown in figure 3 and then individually discussed in the following sections.

3.2.1 The Advertiser component (Service Agent)

This component is responsible for advertising hosted service descriptions to neighboring nodes. Service descriptions are stored in the local cache and a variety of policies can be utilized to adjust the advertisement's frequency. Advertisements can be sent in a unicast or multicast mode using the network component.

3.2.2 The Request component (User Agent)

The request component is used by applications to send and process service requests. This component is also responsible for matching requested service descriptions with descriptions stored in the local cache. On a positive match, a reply message is sent to the requesting node using the reply component. Also, it is possible to define a policy to forward service requests to neighboring nodes when a requested service was not found. Although in the interaction pattern described before, the matching process is handled by the DA, we decided to integrate this functionality to this component since the matching process is executed after the reception of a service request which is handled by this component.

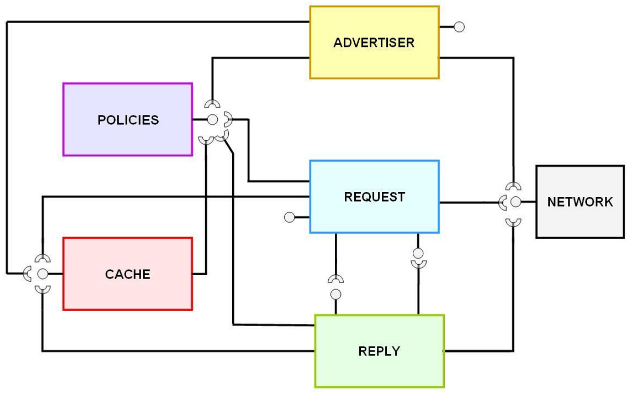


Figure 3. The core service discovery framework architecture.

3.2.3 The Reply component

The DA functionality was divided into two components: the cache and reply components. The reply component generates and sends service replies to requesting nodes when a positive service match is found by the request component. Moreover, if a service reply is received, an event notification is sent to the local application.

3.2.4 The Cache component

The primary function of the cache component is to manage temporary information required by protocols to work accurately. Key messages received and sent by nodes are stored in this cache component as entries. Different types of messages are stored as different entry types. Some protocols e.g. SSD, Allia, GSD, require to store service advertisements in the cache component with the aim of maintaining an updated directory of services available in its vicinity. Similarly other protocols like SLP-B store service requests and replies in the cache with the objective of automatically retransmitting to maintain an updated view of services available in their local domain. Also, protocols like GSD require maintaining a reverse route table for sending back replies to service requests. This reverse route table can also be defined as a cache entry and handled by the cache component.

Therefore, the cache component is used by the advertiser, request and reply components to store, update, replace, delete and retrieve entry values. Most entries are deleted automatically by the cache component after a determined TTL (specified on each entry), unless an update or replace message is received in advance. Also, cache entries can be deleted at any moment from components connected to it.

3.2.5 The Policies component

Most MANET SDPs define policies that take into consideration: user preferences, application needs and/or inclusive context requirements. Bandwidth overhead, expected number and mobility of participating nodes, and node capabilities are some of the issues important to consider when defining these policies. Policies that users define and applications require include but are not limited to: i) caching preferences, like refresh rate, activation, size etc. ii) advertisement preferences e.g. time to live, vicinity diameter, frequency etc. iii) directory preferences, like

advertisement period, type of activation, etc. and iv) forwarding preferences e.g. no forwarding, just process, or process and forward of incoming messages. This component provides functionality for: loading policies from a specified XML file; applying policies to components connected to it and changing policy values according to user preferences, application needs or context requirements.

3.2.6 The Network component

The network component is responsible for providing network level communication to components connected to it. Through this component different routing schemes e.g. unicast, multicast, bordercast, and ad-hoc routing can be provided.

4. IMPLEMENTATION ISSUES

We have evaluated our framework by implementing the GSD, SSD, ALLIA and SLP-B protocols. Our implementations were based on [2],[14],[12] and [11] respectively. For our implementations we used OpenCOM [4] as the underlying component technology. Because the policies, cache and network component required the same functionality in all protocols, they were implemented once as generic components. Hence, these components are capable to provide simultaneous support to multiple components connected to them (figure 4). However, even though the advertiser, request and reply component also follow similar patterns of interactions individual implementations were required for each protocol.

In our framework, the advertiser component is not only responsible for handling service description advertisements but also for managing other additional protocol messages. For instance, SSD protocol includes a set of messages required for the directory election process and ALLIA utilizes a heartbeat message to notify to nodes in the vicinity the presence of a platform. Therefore, since different protocols use different types of messages, different implementations were required for each protocol.

In all cases the request component is responsible for sending and handling incoming service requests. When a service request is received a matching process is executed. However, SDPs use different languages to describe their services. Also, service descriptions are not always only matched against local service descriptions, but also additional information is considered for matching a service. For instance, GSD describe its services using an ontology based on the DARPA Agent Markup Language (DAML+OIL) and services are matched based on service groups, whereas SSD uses WSDL to describe its services and service descriptions are not only matched against service descriptions in the local cache but also against other directories' summaries. Besides, in some cases additional processes should also be handled by this component. For instance GSD maintains a Reverse-Route table that is used to send a service reply back to the source of the request. This table must be updated every time that a service request is received.

Similarly to the reply component, specific processes must be executed in some protocols when a reply is received or sent. For instance, ALLIA maintains hit-rate statistics of successful service requests; therefore intermediate nodes update their statistics when a service reply is forwarded.

However, even though some components were implemented individually for each protocol, the same type of components was used in all implementations and all components have the same type and number of interfaces and receptacles for their bindings. This allows them to be plugged in easily into the framework. For example, a configuration of the multi-personality service discovery framework with all four protocols we implemented is shown in figure 4.

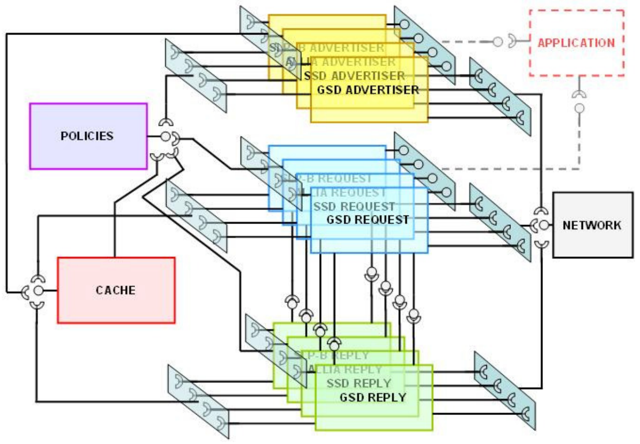


Figure 4. Multi-personality discovery configuration.

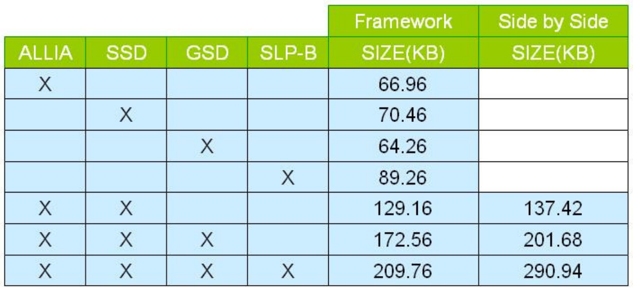
5. EVALUATION

A first benefit of our framework implementations comes from the ability to easily configure a middleware platform to provide support to different discovery protocols simultaneously. Having protocol implementations with common component architecture simplifies the configuration process since the component types and connection bindings remain the same for any protocol implementation. Therefore a common algorithm can be used to configure any personality (i.e. single, multiple protocols). A typical configuration process requires a base of three components (i.e. policies, cache and network), plus three protocol based components (i.e. advertiser, request and reply) for each supported protocol to be configured.

Re-configurability is another benefit of our framework. Fine-grained changes can be made to support environmental context changes. For instance, the network component that is supporting N discovery protocols can be replaced by a new component with a different routing scheme; this is simple single component replacement algorithm. Also, individual protocols can be changed in a fine-grained manner using simple re-configuration algorithms because of the well known configuration pattern e.g. a protocol's advertiser component can be extended with richer descriptions.

Furthermore, our framework approach supports re-use of components; components are re-used at development and deployment time, this simplifies the development effort and time, reduces resource use, and enhances configurability. In our implementations the policies, cache and network components were reused. Additionally, we have found that component implementations for different protocols have similar interaction patterns; hence we were able to re-use internal component algorithms from one discovery protocol to another.

Finally, we analyzed the overhead of our framework by measuring the size of the Java classes (that made up the component configurations) loaded into memory. These measures are illustrated in figure 5; these show the cost of each individual protocol in the framework. Then we measure the cost when multiple protocols are configured. We compare these measures against the side-by-side measurement of individual protocols (not configured in the framework). It can be seen that resource usage is reduced (due to component re-use), and that the overhead of a multiple protocol personality is not restrictive for resource-poor mobile devices.



				Framework	Side by Side
ALLIA	SSD	GSD	SLP-B	SIZE(KB)	SIZE(KB)
X				66.96	
	X			70.46	
		X		64.26	
			X	89.26	
X	X			129.16	137.42
X	X	X		172.56	201.68
X	X	X	X	209.76	290.94

Figure 5. Size of framework personalities

6. RELATED RESEARCH

In the last years, a number of middleware approaches that deal with service discovery heterogeneity have emerged. ReMMoC is a configurable and reconfigurable reflective middleware that interoperates with heterogeneous services in mobile environments by dynamically adapting both its binding and discovery protocol. ReMMoC consists of two component frameworks: (1) a binding framework that allows the interaction between services by plugging-in different binding type implementations e.g. IIOP client, publisher, SOAP client, etc., and (2) a service discovery framework which can be configured to use different service discovery protocols in order to discover services advertised by those protocols. To tackle heterogeneity of protocols, the service discovery framework continuously checks the environment, identifying discovery protocols currently in use and different discovery personalities are configured based on the discovery protocols present in the environment. However the service discovery framework was not designed to support ad-hoc SDPs. Our approach is also more configurable and re-configurable than ReMMoC; the common framework pattern of protocol implementations supports fine-grained configuration and reconfiguration of discovery protocol behaviour, and component re-use reduces the resource usage.

INDISS is another service discovery middleware designed to provide service discovery interoperability in highly dynamic networked home environments. To achieve SDP interoperability a set of parsers and composers dedicated to different SDPs must be embedded into the system. INDISS decouples components from protocols based on event-based parsing techniques. To do so, a set of events were identified based on conceptual similarities among SDPs. Therefore, because communication between parsers and composers does not depend on any syntactic detail of any protocol a parser from a determined protocol can communicate with composers from any other protocol and vice versa. Protocol translation is useful when interoperability between different protocols is required. However, there is unlikely to be a direct mapping between the functionality provided and required in heterogeneous discovery applications e.g. a UPnP application requiring service status notifications, cannot receive these from an SLP-b advertised, ad-hoc service. In addition, in ad-hoc environments the placement of bridges cannot be controlled. Therefore, we argue a configurable and re-configurable multi-protocol personality is better placed to integrate into the operation of ad-hoc applications.

Finally, uMiddle [10] and MSDA [13] both investigate bridging mechanisms to support service discovery interoperability across different communication middleware and network domains. Again, these do not consider all types of ad-hoc discovery protocols, and the bridging solutions place requirements on network infrastructure that cannot be guaranteed in all ad-hoc operating conditions.

7. CONCLUSIONS AND FUTURE WORK

This paper has introduced a component-based service discovery framework for the development of an adaptive multi-personality service discovery middleware in ad-hoc environments. We have also presented a common component pattern for the development of protocols to be plugged into the framework; this pattern is based upon common patterns within existing ad-hoc SDPs. The evaluation of our framework was based around the implementation of four ad-hoc SDPs using our common pattern, and then plugging them into the framework. We have shown that this approach enhances configurability and re-configurability, minimizes resource usage through component re-use, and additionally simplifies the development of new protocols by promoting code reuse.

Future work includes the development of a more fine-grained architecture to increase code re-usability and configurability. For instance, by identifying the commonalities present in the advertiser, request and reply component, new components can be created. For instance, a component responsible for managing directory messages could be created to separate service description advertisements from directory messages in the advertiser component. Also a component capable to handle service descriptions from different protocols could be useful to have a more dynamic request component capable for matching service descriptions from different protocols using a variety of matching algorithms. Furthermore, is necessary to investigate efficient mechanisms that allow our framework identify ad-hoc SDPs present in the environment. Also, we envisage resource optimisation e.g. bandwidth by integrating ad-hoc protocols to existing routing protocols for MANETs.

We also wish to extend our approach further to consider all network types, and then evaluate our architecture with other diverse discovery protocols including: Bluetooth, SLP, UPnP, UDDI, JXTA, and other peer-to-peer resource discovery technologies. Finally, we foresee the development of a configurable and re-configurable middleware to provide service discovery interoperability across different network styles e.g. integrating discovery across fixed and ad-hoc network types.

8. REFERENCES

[1] Bromberg, Y.D., and Issarny, V. INDISS: Interoperable Discovery System for Networked Services. In *Proceedings of Middleware 2005*. Grenoble France November 2005.

[2] Chakraborty, D. Joshi, A., Finin, T., and Yesha, Y. GSD: A Novel Group-based Service Discovery Protocol for MANETs. In *Proceedings of the 4th IEEE MWCN*. Stockholm, Sweden, September 2002.

[3] Cho, C., and Lee, D. Survey of Service Discovery Architectures for Mobile Ad hoc Networks. Term paper, Mobile Computing, CEN 5531, Department of Computer and Information Science and Engineering (CICE), University of Florida, Fall, 2005. http://www.cise.ufl.edu/class/cen5531fa05/, October, 2006.

[4] Clarke, M., Blair, G.S., Coulson, G., and Parlavantzas, N. An Efficient Component Model for the Construction of Adaptive Middleware. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware)*. Heidelberg, Germany, November 2001.

[5] Grace, P., Blair, G.S., and Samuel, S. ReMMoC: A Reflective Middleware to support Mobile Client Interoperability. In *Proceedings of International Symposium on Distributed Objects and Applications (DOA).* Sicily, Italy. November 2003.

[6] Guttman, E., Perkins, C., Veizades, J., and Day, M. Service Location Protocol, Version 2, RFC 2608 (Proposed Standard). June 1999.

[7] Huang, E., Hu, W., Crowcroft, J., and Wassell, J. Towards Commercial Mobile Ad Hoc Network Applications: A Radio Dispatch System. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*. Illinois, USA, May 2005.

[8] Konig-Ries, B., and Klein, M. Information Services to Support E-Learning in Ad-hoc Networks. *In Proceedings of 1st. International Workshop on Wireless Information Systems (WIS)*. Ciudad Real, Spain, April 2002

[9] Kozat, U.C., Tassiulas, L. Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues. In *AdHoc Networks Journal*, Vol. 2, No. 1, pp. 23-44, 2004, ELSEVIER.

[10] Nakazawa, J., Tokuda, H., Edwards, W.K., and Ramachandran, U. A Bridging Framework for Universal Interoperability in Pervasive Systems. In *Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS'06)*. Lisboa, Portugal. July 2006.

[11] Penz, S. SLP-based Service Management for Dynamic Ad-hoc Networks. In *Proceedings of the 3rd International Workshop on MPAC,* Grenoble, France, November 2005.

[12] Ratsimor, O., Chakraborty, D., Joshi, A., and Finin, T. Allia: Alliance based service discovery for ad-hoc environments. In *Proceedings of the 2nd international Workshop on Mobile Commerce.* Atlanta, Georgia, USA, September, 2002.

[13] Raverdy, P., Rive, O., de La Chapelle, A., Chibout, R., and Issarny, V. Efficient Context-aware Service Discovery in Multi-Protocol Pervasive Environments. In *Proceedings of the 7th International Conference on Mobile Data Management (MDM'06).* Nara, Japan. May 2006.

[14] Sailhan, F., and Issarny, V. Scalable Service Discovery for MANET. In *Proceedings of the 3rd IEEE PerCom.* Kauai Island, USA, March 2005.

[15] Sun Microsystems. JINI^{TM} Architecture Specification. Version 2.0, June 2003.

[16] Szyperski, C. *Component Software: Beyond Object-Oriented Programming*, Ad-dison Wesley, December 1997.

[17] UPnP Forum. UPnP^{TM} Device Architecture. Version 1.0, June 2000.