

Resource Discovery in Pervasive Computing Environments

Ricky Robinson*
Queensland Research Laboratory
National ICT Australia†

Jadwiga Indulska‡
School of Information Technology and Electrical Engineering
The University of Queensland and
Queensland Research Laboratory
National ICT Australia†
(Dated: July 4, 2006)

Contents

I. Introduction	1	C. OWL-based Service Discovery	9
A. What is Resource Discovery?	2	D. SSDS	10
B. Why Pervasive Computing Systems Need Resource Discovery	2	E. VIA	10
C. Chapter Overview	2	F. Splendor	11
		G. INS/Twine	11
II. Resource Discovery Requirements for Pervasive Computing Environments	3	H. NEVRLATE	11
A. Traditional Computing Environments	3	I. Superstring	11
B. Emerging Computing Environments	3	J. Summary	12
C. Resource Discovery Considerations	3	V. Conclusion	12
D. The Anatomy of a Resource Discovery Protocol	4	References	12
III. Current Resource Discovery Protocols	4		
A. Simple Service Discovery Protocol	4		
B. Service Discovery Protocol	5		
C. Salutation	5		
D. Domain Name System - Service Discovery	5		
E. JXTA	5		
F. Jini	6		
G. RDP	6		
H. LDAP	6		
I. CORBA Trader	7		
J. UDDI	7		
K. Schwartz et al.	7		
L. Service Location Protocol	8		
M. Summary	8		
IV. Emerging Approaches to Resource Discovery	8		
A. Konark	9		
B. DEAPspace	9		

I. INTRODUCTION

In January of 1946, the Electronic Numerical Integrator And Calculator, or ENIAC, was switched on for the first time at the University of Pennsylvania. ENIAC, the world's first electronic computer, was over thirty metres wide and more than two metres high. It could perform five thousand addition operations per second. Today, there are computing devices a hundred thousand times more powerful than ENIAC that are small enough to be carried on one's person! While the users of ENIAC interacted with it by rewiring accumulators, today's computer users can interact with their devices by typing, pointing and clicking, and to some extent by writing, speaking and gesturing. The ability to pack computing power in one's pocket, together with advances in telecommunications technologies, has brought about a new computing paradigm called *pervasive computing*, also known as *ubiquitous computing*. In pervasive computing, one's home, office, local shopping centre and other environments host a multifarious set of embedded and mobile computing devices, which can collaborate to aid one in one's everyday tasks. However, this way of computing does not *replace* traditional desktop computing. Workstations and supercomputers, some almost as large and immobile though millions of times more powerful than ENIAC, maintain their places in the modern world. These fixed infrastructure components and the wired communication networks that link them will remain a part of the modern computing environment. Indeed, these fixed components will

*Ricky.Robinson@nicta.com.au

†National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs.

‡jaga@itee.uq.edu.au

often be required to support the mobile and computationally inferior elements. In its broadest sense, then, pervasive computing is the interaction among mobile computing devices, and the interaction between mobile devices and the static infrastructure, in aid of the human user. Thus, pervasive computing not only embodies a new way of computing, but subsumes other computing paradigms.

The mobility of devices and users coupled with dynamic application requirements mean that the set of resources with which an application may interact is in a constant state of flux. A consequence of this is the need for applications and users to locate resources and services on demand and on the fly. This mechanism is known as *resource discovery*, and it is this process with which this chapter is concerned.

A. What is Resource Discovery?

Resource discovery is the term given to the process of locating a resource that matches a provided description. Usually, resource discovery consists of advertising and querying phases, but some protocols utilise only one of these two steps. When the resources being discovered perform some task, such as printing or file storage, then resource discovery is sometimes given the more specialised name of *service discovery*. In this chapter, no distinction is made between these two terms. Thus, resource discovery is analogous to searching a library catalogue for a book that meets a particular description, or looking through a service directory to find a company that will perform a certain task.

Given the assumption that interaction among remote entities is necessary, resource discovery is required in the modern computing environment, firstly because it is impossible for each application to have knowledge of all the resources it could potentially interact with, and secondly because the set of resources with which it is possible for an application to interact changes constantly due to mobility and disconnection. However, resource discovery also provides *choice* to users and applications. In the event that several existing resources can fulfil a particular task or requirement, a service discovery protocol can be used to differentiate between these resources to return the most appropriate instances according to a set of preferences or a ranking function. When viewed in this light, resource discovery is a personalisation tool [1].

A range of resource discovery protocols are in use or have been proposed for use in various computing environments [2–10]. Typically these solutions are focused at specific kinds of environments, such as local dynamic environments [11] or structured wide-area environments [5]. In the increasingly connected world, these narrow application domains are becoming less relevant. This chapter examines some of the resource discovery protocols that can span multiple environments. Pervasive computing environments are characterised by heterogeneity, and they can be thought of as the aggregation of multiple

kinds of computing environments.

B. Why Pervasive Computing Systems Need Resource Discovery

The modern computing environment brings with it many challenges. Among these challenges are heterogeneity of computing devices and networks, scalability and changes in the environment due to user or device mobility. Yet, these same factors bring with them the opportunity to define new applications. The abundance of resources enables users to harness computational power, storage, tools and applications that are not available on their local devices. Furthermore, these devices are no longer constrained to a desk; they have been unshackled, potentially allowing users to continue working and interacting remotely with people and applications even when they are mobile. The interactions between components within the modern computing environment are therefore often of an ad hoc nature. The bindings between a resource and a resource user are performed *just-in-time*, regardless of the permanency or transience of the underlying network structures. This mode of computation has bred and will continue to breed a novel range of applications, which incorporate concepts such as location-awareness and, more generally, context-awareness. Not only is context-awareness necessary to enable applications to continue working in the face of environmental change, it can endow applications with new behaviours resulting from their new found freedom. Such behaviours include intelligent call redirection, context-aware recommendations and context-aware reminders. Thus, if necessity is the mother of invention, liberation is surely its father. In pervasive computing, invention is sustained by human imagination and retarded only by hostile computing environments, human interface requirements and complexities for which there is no current technological solution. The goal of pervasive computing research, then, is to find solutions for these problems such that invention may continue unfettered. Resource discovery is a solution to some of these problems [12]. However, these obstacles must first be hurdled by the resource discovery protocols themselves.

C. Chapter Overview

This chapter is structured as follows. Section II provides a detailed list of requirements that resource discovery protocols for pervasive computing environments must satisfy. A range of currently available resource discovery protocols will be surveyed with respect to these requirements in Section III. Section IV will survey newer resource discovery protocols that aim to overcome some of the shortcomings of current resource discovery protocols. Finally, Section V summarises our conclusions and highlights the remaining research challenges in this field.

II. RESOURCE DISCOVERY REQUIREMENTS FOR PERVASIVE COMPUTING ENVIRONMENTS

The introduction provided an overview of what resource discovery is, and why modern computing environments need resource discovery protocols. But what constitutes a resource discovery protocol? What makes one resource discovery protocol different from another, and what are the factors that led to these differences? In this section, we provide a guide to the elements that compose resource discovery protocols in general, and the requirements that various computing environments place on them. A brief survey of computing environments provides a good platform from which to launch a discussion about resource discovery protocols, and helps to explain why resource discovery protocols have evolved the way they have.

A. Traditional Computing Environments

Traditional computing environments encompass those with which we are currently most familiar. They include the Internet, workplace networks and home offices.

In general, these environments are fairly stable and static. They share common network and transport protocols (TCP/IP). Networks in the workplace and at home are often used to access the Internet, which contains documents, services and other resources. Home and office networks contain resources such as file servers, printers, proxies and mail servers. Typically, devices that offer services and other resources are permanently online; they go offline only when they fail or when they are taken down for maintenance.

New classes of applications are arising in traditional computing environments such as the Internet. For example, web services are becoming a popular means of facilitating interactions between businesses, and as a method of invoking remote operations in client-server applications. Some web services are composed from multiple other web services, and sometimes this composition must take place dynamically.

B. Emerging Computing Environments

The advent of mobile computers and the observation that greater efficiency can be obtained by pooling computing resources has led to the emergence of a range of different computing environments.

Grid computing environments offer a way to harness unused CPU cycles and storage. Present-day grids, such as NASA's Information Power Grid [13] and Australia's GrangeNet [14], consist of relatively small numbers of very high performance computers connected on a more-or-less permanent basis. This situation may change in

the future as new protocols make it feasible for more dynamic nodes to join the grid and offer resources for short periods. When this happens, the differences between grid computing and peer-to-peer computing will all but disappear [15].

In peer-to-peer (P2P) environments, each node plays the role of server *and* client. Often, nodes in these environments must also route messages between other nodes. Usually, P2P protocols are implemented as application layer overlays to lower layer protocols [16, 17]. Arguably the most salient feature of P2P is its decentralised nature: the lack of a central point of control makes it resistant to many hostile contingencies, such as node failures, topology changes and direct efforts to close it down. The utility of P2P applications hinges upon the aggregate resources of all the nodes in the network.

Ad hoc networks are composed of nodes that are often mobile and communicate via wireless links. As in peer-to-peer networks, each node must be prepared to route messages between two other nodes. Ad hoc networks have been recently popularised by Bluetooth [18] and IEEE 802.11 [19]. Often, each node in an ad hoc network is very limited as to its capabilities. Therefore, devices rely on their neighbours in order to complete their tasks. As ad hoc networks materialise under a wide variety of circumstances, their resource discovery requirements differ. Typical uses of ad hoc networks are to facilitate communication and co-ordination in military and emergency response situations, the docking of handheld devices to workstations and the creation of mesh networks.

In pervasive computing environments, standalone and embedded computing devices will cooperate to aid users in their tasks. In such systems, the devices, and the services executing on them, need to operate with a greater level of autonomy than is usual with today's applications. This allows the user to concentrate on the task at hand, and means that the computing devices can fade to the background while providing a seamless computing environment for the user. Pervasive computing environments are associated with extreme heterogeneity in terms of the types of devices, networks and applications that can be found within them. This trait means that pervasive computing environments can be thought of as the aggregation of all of the other environments discussed above.

C. Resource Discovery Considerations

The above characterisation of computing environments provides some insight as to the capabilities required of a resource discovery protocol. The environments differ as to the number and capabilities of the devices that constitute them. The underlying networks and protocols have different properties. The applications that execute within these environments utilise resources in varied ways, and therefore they expect different functionality from the resource discovery protocols that they use. Some applications might require the resource discovery protocol to

be context-aware, some might need a resource discovery protocol with advanced resource ranking features. The requirements can be summarised as follows:

- scales to networks of few and many nodes;
- scales to powerful and resource constrained devices;
- operates in structured (traditional infrastructure-based) and unstructured (ad hoc, dynamic and mobile) networks;
- operates in heterogeneous networks;
- defines a rich yet lightweight description language, and a simple API;
- is context-sensitive; and
- defines result-ranking facilities and integrates user preferences

D. The Anatomy of a Resource Discovery Protocol

At an abstract level, all resource discovery protocols share some common components:

1. resource description language;
2. routing protocols;
3. message formats; and
4. application programming interface (although some protocols are defined purely by their message formats).

The details of these basic elements, and the way they are combined and utilised, is what differentiates one resource discovery protocol from another. Some resource discovery protocols include extra features such as security and context-sensitivity; however, these are not core components of a resource discovery protocol.

Resource discovery protocols differ in the way that messages are used to discover resources. The following diagrams illustrate these differences.

Figure 1(a) shows a protocol that distributes resource descriptions to all nodes on the network. In this model, there are no queries, since every node is aware of all resources in the network. Such a model is not used in reality because it has poor scalability characteristics.

Figure 1(b) demonstrates the opposite approach to that used in Figure 1(a). In this model, there are no advertisements. Rather, each node must query other nodes to find appropriate resources. This model is used by some existing protocols.

A combination of Figures 1(a) and 1(b) leads to a model in which advertisements are issued to a subset of nodes in the network so that querying nodes can locate the desired resources more efficiently and with higher probability. This model is also widely adopted.

Finally, Figure 1(c) shows a commonly used model in which one or more nodes provide a special service called a directory or registry. Services register themselves with directories, and clients issue queries to the directories. This model entails that clients and services must first locate the directory node. Some protocols take this model a step further and organise directory nodes into a hierarchy, thereby increasing the scalability of the protocol.

These abstract models suffice to cover most of the modes of operation found among actual resource discovery protocols. The next sections examine some real resource discovery protocols.

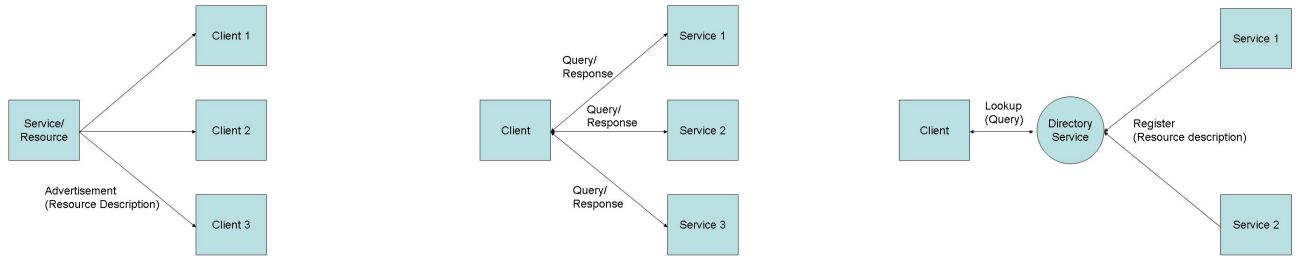
III. CURRENT RESOURCE DISCOVERY PROTOCOLS

This section examines a range of current resource discovery protocols, many of which are commercially available.

A. Simple Service Discovery Protocol

The Simple Service Discovery Protocol (SSDP) [3] is utilised by Universal Plug and Play (UPnP) to discover the capabilities of services in a community of devices. SSDP utilises a modified form of HTTP for communication. This modified form operates on top of UDP and is capable of unicast (HTTPU) and multicast (HTTPMU) communication. If a proxy (a service registry) is available on the network, then services can unicast an advertisement to the proxy, and clients can unicast queries to the proxy. If not, then multicast communication is used between clients and services. A service advertisement consists of minimal information about the service, such as its type, and a URL from which the complete description of the service can be downloaded. This description is formatted as XML [20]. Due to its extensive use of multicasting and its proxy election mechanism, SSDP is not suitable for large networks or environments containing a high proportion of mobile nodes. The requirement for service descriptions to be downloaded and interpreted by the client device may also prohibit its use on small, resource-poor devices.

SSDP is suitable only for environments on the scale of a home or office. Due to its simplistic advertising and querying mechanism, it does not scale well to large numbers of devices. Within such a constrained environment, the appearance and disappearance of new devices at frequent intervals does not pose a problem unless those devices trigger proxy elections, which results in much bandwidth being consumed.



(a) A protocol that pushes advertisements to all clients. (b) A protocol in which clients query all other nodes. (c) A protocol that utilises a directory service.

FIG. 1: The general modes of resource discovery.

B. Service Discovery Protocol

The Service Discovery Protocol (SDP) [4] is Bluetooth’s mechanism for locating services on Bluetooth enabled devices. Each Bluetooth device manages the descriptions of the services it hosts. Hence, there is no concept of service advertisement. Queries, known as service searches, are directed at a particular device. The queries sent by a client contain a list of attributes such as the service class and the protocol via which the client intends to invoke the service. Only attributes that have a corresponding Universally Unique Identifier (UUID) may appear in queries. SDP also supports browsing, which lists all the services on a particular device. Browsing is used by the client when nothing at all is known of the services on a device. This way the client becomes aware of the *types* of services a device offers.

SDP was designed with a very specific purpose in mind, and under the assumption it would be layered over the Bluetooth L2CAP [21] link layer protocol. SDP is therefore suitable only for very small ad hoc groups of wireless devices.

C. Salutation

Service discovery in Salutation [22] is accomplished by the Salutation Manager (SLM). In general, each Salutation enabled device is equipped with its own SLM. Services local to the device register themselves with the local SLM. The SLM also issues service discovery queries on behalf of local applications. If a device is multi-featured, each feature or service is advertised as a separate Functional Unit. Descriptions and queries are formatted using ASN.1 notation [23]. When an application asks its local SLM to perform a query, the local SLM may forward the query to remote SLMs. The local SLM will return to the application the ID of the SLM that has a service registered which can satisfy the application’s request.

Like SSDP, Salutation is aimed at the home and office environment. The SLM on a Salutation client may

query only those remote SLMs that it is directly aware of, reducing the level to which Salutation can scale.

D. Domain Name System - Service Discovery

DNS-SD (Domain Name System - Service Discovery) [24] provides service discovery capabilities for Apple’s Rendezvous technology [25]. A Rendezvous enabled device first assigns itself an IP address from the link-local range using the ZeroConf draft IETF standard [26]. Once a node has an IP address that does not conflict with any of its peer nodes, it may utilise DNS-SD to locate services on the network. Each device hosts a lightweight DNS server. Clients use multicast messages (mDNS-SD) [27] to register their services with other devices on the network. Clients can also use mDNS-SD to locate services on the network.

Although Rendezvous provides the best known implementation of DNS-SD, it need not be constrained to use within Rendezvous. Since DNS-SD builds upon the DNS [28] standard and does not define any new constructs or messages, it can scale to the Internet (DNS has long provided name resolution for the Internet). However, its ability to scale to the wide-area is gained due to the constraint that a search must be directed at a *particular* domain, rendering DNS-SD incapable of providing service discovery in a large number of use cases. Furthermore, DNS-SD does not provide the ability to relax queries when exact matches are not found. More importantly, because it relies on the standard messages defined by DNS, it does not support rich queries containing expressions. Finally, its reliance on DNS means that it is bound to the Internet Protocol, making it unsuitable for use in environments where IP is not supported.

E. JXTA

The rise of peer-to-peer computing can largely be attributed to applications such as Napster [29] and

Gnutella [30]. JXTA [31] from Sun Microsystems is another peer-to-peer technology, designed to provide a generic base upon which peer-to-peer applications can be built. JXTA is comprised of six main protocols. One of these is the Peer Discovery Protocol (PDP), which provides a means of discovering any JXTA resource for which there is an advertisement. Resources include peers, peer-groups, pipes and modules, as well as anything else that can be described by a JXTA advertisement. JXTA uses PDP to discover advertisements within a peer-group. Specifically, PDP discovers advertisements in the *world* peer-group. Custom discovery protocols can be implemented for use within non-world peer-groups, and these protocols may leverage PDP for bootstrapping purposes. If a custom protocol does not exist for a specific peer-group, PDP may be utilised directly. Provision for custom discovery protocols is made because the authors believe that detailed discovery information can only be known by higher-level discovery protocols. Discovery queries and advertisements are based on XML. JXTA makes few assumptions about the underlying network. It provides a messaging layer that binds the six protocols, including PDP, onto the underlying transport, whatever it might be.

By leaving the definition of higher-level service discovery protocols to particular peer-groups, interaction between peer-groups may be minimised, which inhibits upward scalability. The assumption that higher-level discovery protocols are better suited to cater to specific applications allowed PDP to be defined for minimality. Advertisements and queries contain attributes and values. Values in queries may specify exact or wild card matches. A wild card may appear at the beginning or end of a value, or both. A wild card may not appear in the middle of a value. Wild card matching is an optional feature, which vendors may choose not to implement.

Queries and query responses are routed through a JXTA peer-group with the Peer Resolver Protocol (PRP). The PRP directs each method to a particular named handler. The named handler defines the semantics of the message but is not associated with a particular peer within the peer-group. The message may be sent to one peer or multiple peers. The Rendezvous Protocol is responsible for the actual sending and receiving of messages. Some nodes in JXTA may become rendezvous peers. Other nodes subscribe to the rendezvous peers to receive particular kinds of messages. The rendezvous peers propagate messages to the peers that have subscribed to receive those messages. The rendezvous protocol uses flooding to locate resource advertisements, which further limits its scalability.

F. Jini

Jini [7] is a technology for the spontaneous creation of communities of networked devices. It is ultimately a Java-centric solution, though it is possible for non-Java

enabled devices to participate in a Jini community. Jini utilises registries, or lookup services, that accept advertisements from services and resolve queries from clients.

Unlike SSDP, described above, Jini cannot operate without a lookup service. Furthermore, there is no ad hoc mechanism for electing a Jini enabled node to provide the lookup service. However, the lookup service can be located in an ad hoc fashion by means of IP multicast. For these reasons, it is classified as a service discovery protocol for semi-structured environments such as the home or office, though it could potentially operate within structured environments such as grids.

A service registers itself with the lookup service when it comes online by sending a proxy object and a set of attributes. The proxy object is used by clients to access the service, and the attributes are Java objects that can be matched by a client query.

Jini's centralised lookup mechanism limits the number of resources a Jini community can scale to. Although lookup services may be federated, this is not achieved in a transparent fashion. The leasing mechanism provides an elegant way in which to deal with disappearing services.

G. RDP

Perkins and Harjono [32] designed a protocol targeted at mobile nodes that move from one fixed network to another, such as when a laptop moves from a work LAN to a home LAN. The protocol uses DHCP as a bootstrap mechanism for locating the central resource database that resides in the local network. In this respect, their solution is similar to Jini. However, resource descriptions take the form of a URL and a set of keywords. Queries are based on URNs. The first part of the URN defines the type of service, and can be either *n21*, specifying that only one matching resource should be returned, or *n2c*, requesting that all the matches should be returned. Following this is an optional resolution path (used to override the address of the resource database provided by DHCP), and an optional naming authority (often the name of the institution in which the mobile device currently finds itself). The naming authority defines how to interpret the following scheme field, which identifies the protocol used to retrieve the resource (such as HTTP or NFS), and the following keywords, which describe the resource. Only exact matching of keywords is supported.

Perkins and Harjono's solution utilises UDP for delivering description registrations to the resource database (advertisements) and for queries. The solution is firmly tied to environments that use IP, and local-area networks in particular.

H. LDAP

LDAP [33] is a simplified (lighter weight) version of the X.500 [34] standard. Like other directory services, LDAP

is not technically a resource discovery protocol. Nonetheless, it exhibits many features required of a discovery protocol, and it would be a simple matter of programming to build a discovery service utilising LDAP at its core. Moreover, LDAP provides features necessary in a distributed environment, and therefore necessary to a resource discovery protocol. Such features include replication and referrals, as well as security features. Although LDAP is primarily a directory containing simple data types, objects may be bound to directory nodes. Almost any kind of object may be bound to a directory entry. For example, Java objects may be stored in the directory for later retrieval. The only requirement is that a schema exists for describing how specific kinds of objects are stored in the directory.

LDAP may be used as a directory-oriented resource discovery protocol, whereby services register themselves with the LDAP directory, and clients search the LDAP directory for the relevant resources.

I. CORBA Trader

The Open Distributed Processing (ODP) Trading function [35] provides a means of offering a service and the means to discover services that have been offered. These capabilities are known as exporting and importing, respectively. The ODP trading function is a model; it has not been implemented. However, there *are* implementations of the CORBA trading service [36], which has its basis in the ODP trading function. The CORBA Trader has five key interfaces: Lookup, Register, Link, Admin and Proxy.

The Lookup interface is used by clients to find services. The Register interface is the means by which exporters advertise services in the Trader. Inter-operation between Traders is performed via the Link interface. Trader policies are set via the Admin interface, and the Proxy interface is used to hide or wrap legacy services.

The CORBA Trader model is unsuitable for dynamic networks such as MANETs, since the Trader should be deployed on a static, well known node in the network. It does not provide query relaxation, though services can be selected with some degree of granularity.

J. UDDI

Universal Description, Discovery and Integration [37] is a specification for enabling businesses to find one another and the services they offer. Once a business finds a suitable service offered by another business, it can integrate its applications with the discovered service (in some non-specified manner).

Information in UDDI is described by four entity types, each of which is represented in XML. The *businessEntity* is the top-level structure. It contains information such as

the name of the business (or other entity, such as a department within an institution), its address and the type of service the business provides. Each *businessEntity* contains one or more *businessServices*. This entity type logically groups a set of related web services provided by the business. The *businessService* structure is purely descriptive. It does not contain technical information about how the web services should be invoked. A *businessService* has one or more *bindingTemplate* structures. These contain technical information which describes how an application can interact with a particular web service. Finally, the *tModel* is a structure that exists outside of the hierarchy described above. It defines reusable components that can be utilised within any one of the above structures. For example, *tModels* can be used to describe protocols, the format of postal addresses and so forth.

UDDI provides a logically centralised but physically distributed view of its service registry database. Each UDDI entity is associated with a key which is unique within a registry and across all interacting registries. UDDI entities submitted to a node within a single logical registry are replicated among the other nodes within that registry. They may also be replicated between registries, but there is no direct channel of communication between registries. Instead, an *importer* retrieves UDDI entities from one registry and publishes them to a different registry. Before interacting with any particular UDDI registry, clients, importers and other UDDI registries must be aware of the registry's key generation policies, so as to prevent problems in the future. For instance, if two registries A and B wish to interact in the future, they must ensure that their keyspaces do not overlap (or equally, that they each take keys from the same keyspace). A root registry can help with this problem, but it still means that both interacting registries must be affiliated with the same root prior to sharing data.

Within a registry, whenever an entity is added, removed or updated from a particular node, that node issues a notification to all other nodes comprising the registry. Those nodes may then pull the new information from the notifying node.

UDDI allows clients to search using constructs similar to SQL [38] queries, though they are marked up using XML. Clients direct queries at a particular node within a registry.

K. Schwartz et al.

In the approach taken by Schwartz [39], resource information repositories (RIRs) are encapsulated by *brokers*, which hide the heterogeneous aspects of individual RIRs, including the unique access control policies governing them. Brokers are designed specifically for each RIR, since each RIR may have its own internal protocols and interfaces. Brokers announce a set of keywords characterising the information contained within their associated RIRs. The keywords are received by *agents*, which main-

tain links between the RIRs. Agents use a form of multicasting to inform each other about the set of keywords they know about. *Clients* initiate searches by sending a query to a nearby agent. In the event that the agent cannot resolve the query, it is forwarded to other agents based on the set of cached keywords. The use of these keywords means that several overlapping graphs may be formed. Some graphs are organised according to services offered, while other graphs might be organised according to geographic location, and so on.

While the links formed between these components may be dynamically added and removed, these relationships are generally long-lived and the components themselves are static in nature. That is, agents and brokers do not appear and disappear. Furthermore, this resource discovery architecture is targeted at large-scale networks such as the Internet.

Oddly, Schwartz's work has gone largely unnoticed in the last decade. The scalability of Schwartz's solution relies largely on the Small World phenomenon [40], in which, counter to intuition, the number of steps or hops to traverse from one node to another node within a large network is often quite small. The recent interest shown by physicists in the statistical dynamics of real-world networks such as the World-Wide Web and Gnutella has resulted in suggestions as to the way the Small World phenomenon can be incorporated into new and existing protocols [41, 42].

While Schwartz was arguably the first to suggest these links between small world networks and resource discovery, several problems were left unaddressed. First, the granularity of search is very coarse. Brokers advertise category descriptions on behalf of RIRs. Thus, the volume of responses to queries may be overwhelming in large networks. Second, there is no way to order or limit the number of responses. Finally, although it is suggested that descriptions would consist of a list of keywords, this is not a requirement. Instead, descriptions may consist of more formal structures. The lack of a consistent approach to resource description will inhibit the interoperability of agents, clients and brokers in a large-scale system consisting of multiple administrative domains. This approach may be better suited to smaller environments, and in fact, Chakraborty et al.'s framework [43] (described below) bears a striking resemblance to Schwartz's solution.

L. Service Location Protocol

The Service Location Protocol (SLP) [44, 45] provides a discovery mechanism that scales from small ad hoc groups to large enterprise networks controlled by a single administrative authority. SLP supports scoping to provide logical resource grouping. A user agent (UA) may discover services advertised by a service agent (SA) in two ways: by a multicast query to which SAs respond directly if there are no lookup or directory agents (DA)

present, or by a unicast query to an available DA. It is expected that DAs will be present in larger networks with centralised authority. If DAs are present, SAs register with them. DA/SA responses to queries are unicast. Service advertisements contain a service type and a set of service attributes and need to be periodically refreshed with DAs. Service queries support the use of LDAP compatible search filters [46] to specify attributes of interest. It is possible for DAs to contain inconsistent information about available resources. To address this, Mesh Enhanced SLP [47, 48] introduces a peer communication protocol between DAs. This also allows for simpler SA/DA interaction.

SLP is suitable for an entire organisation. Full implementations of SLP provide expressive query capabilities, and it is therefore viable to use SLP in a simple grid computing environment.

While it might be possible to deploy SLP in a mobile ad hoc network, its reliance on the Internet Protocol means that it cannot operate in heterogeneous MANETs, where IP might not be supported by all devices.

M. Summary

It is clear that each of the protocols reviewed in this section is targeted toward particular computing environments. None of the protocols is capable of supporting ad hoc interactions within the vast range of computing environments that can be found in the modern world. As more applications are found for computing technology and new devices are created to perform novel tasks, the range of computing environments will expand and become more diverse. Resource discovery protocols that can overcome the problems of heterogeneity and scale posed by the set of contemporary computing environments would facilitate the development of ubiquitous computing applications that can operate in a wide range of situations.

IV. EMERGING APPROACHES TO RESOURCE DISCOVERY

The previous section examined a number of currently available and historical resource discovery protocols. Those protocols are largely aimed at specific computing environments, and therefore fail to support ad hoc interactions across heterogeneous environments. This section examines a range of emerging protocols, some of which are designed to operate in heterogeneous computing environments, and are therefore better equipped to deal with the realities of modern computing environments.

A. Konark

Helal et al. have designed a protocol named Konark [10], which is targeted to the discovery and delivery of m-commerce oriented software services (though it is not clear what makes the protocol more suitable for m-commerce in particular, nor do the authors explain why resource discovery is required in m-commerce). The basic Konark architecture uses IP multicast for service queries and advertisements, leading to high message overhead (as conceded by the authors in their subsequent paper [49]). An extension to the base protocol, known as the *Service Gossip Protocol*, reduces message overhead by having each node only broadcast the *differences* between the services it learns about from its neighbours' broadcasts and its own service description cache. Konark relies on IP multicast, which means it cannot be used in some environments. Furthermore, the use of IP in mobile ad hoc networks incurs overhead which is not considered by the authors [50].

Each Konark node maintains a Service Registry, where service descriptions are stored in a class hierarchy structure called a service tree. Concrete service instances are stored at the leaves of the tree. A parent node encapsulates all its child nodes, such that a query that specifies a non-leaf node will match all the service instances directly or indirectly under that node. Konark allows for the specification of keywords in queries and advertisements. Similar to SSDP in UPnP, Konark specifies that initial discovery of a service uses only the service type and keywords. The second discovery phase requires the client to download the entire description (which uses a WSDL-like notation [51]) from a URL. The full description contains components that indicate how a service should be invoked, and also includes a human-readable description of the service.

B. DEAPspace

The DEAPspace [52] protocol from IBM focuses on very small networks of a scale similar to Bluetooth. In particular, DEAPspace relies on all nodes being within broadcast range of one another. Unlike Konark, it does not *rely* on the Internet Protocol, though it can operate over TCP/IP (indeed, IBM's prototype operates over TCP/IP and an underlying IEEE 802.11 link). DEAPspace uses a randomised slotted broadcast scheme, whereby advertisements are pushed pro-actively to all nodes within the network. Therefore, each node has full knowledge of all the resources in the network. (Konark's Service Gossip Protocol, described above, borrows heavily from the DEAPspace algorithm.) The authors argue that such a scheme delivers service descriptions in a timely fashion. However, a large communication overhead is incurred when queries for services are infrequent. It is questionable whether the smaller discovery delay is an acceptable payoff for the large communication over-

head in most of the environments at which this solution is targeted.

In DEAPspace, services are defined by their input or output formats. These format descriptions are hierarchical and based upon MIME [53]; however, any element in the hierarchy may be qualified with attributes. For instance, in the description *Application* \rightarrow *PostScript* \rightarrow *version2*, the *PostScript* element can be qualified with the attributes *colour = yes* and *ppm = 20* (where *ppm* stands for pages per minute). DEAPspace supports neither query relaxation (automatic relaxation of query constraints so as to make it more likely services will be matched) nor expressions over the attributes.

C. OWL-based Service Discovery

Chakraborty et al. [43] adopt the use of the Web Ontology Language (OWL) [54] to describe resources in mobile and pervasive computing environments (previously having used DAML+OIL [55], the forerunner of OWL, to describe resources [56]). The choice of OWL means that, in theory, new resource types can be added to the system without needing to alter the query resolution mechanisms. This feature is derived from the inheritance model implicit within OWL and the Resource Description Framework (RDF) [57], which OWL is built upon. Semantic matching of the description contained within a query can thus take place against service instances of the same type as that contained within the query, and against sub-types. While such a scheme is appealing in theory, and indeed has been shown to be viable in the web environment through a number of applications [58, 59], some doubt must be cast upon its employment within mobile ad hoc networks (MANETs) for a number of reasons.

As Chakraborty et al. state, and as outlined above, one of the advantages of using OWL as a description language is the relative ease with which new service types can be introduced to an environment. However, this extensibility is predicated on the ability to validate previously unseen service types against schemata, such that its place within the inheritance hierarchy can be determined. This introduces several problems. First, the relevant schema for the new service type must be downloaded by *every* node that does not have previous knowledge of the service type. In a mobile environment, where should the schema be downloaded from? Presumably, the schema can be treated as *just another resource*, in which case a query can be initiated for the schema. Regardless of the solution, a considerable amount of communication overhead is generated during schema retrieval. Also note that it may be necessary to fetch more than one schema, since the super-type of the service might also be unknown, or the schema defining the service type could be dependent upon several other schemata. Second, assuming that the relevant schema has been found and downloaded, validation can be a computationally expensive exercise. If the nodes constituting the MANET are laptop or note-

book computers, then the cost of computation can be borne without trouble. However, for smaller, lightweight devices, the cost of validation must be considered. It is certainly not clear that validation costs will comprise only a small portion of the entirety of the work carried out by a device. Related to this problem is the size of the in-memory footprint of any RDF parser. When the OWL layer is added to this, the challenge to implement the system described by Chakraborty et al. becomes formidable. At the time of writing, to the author's knowledge there is no RDF parser, let alone OWL tool, that is small enough to execute on handheld devices such as the iPAQ or Palm Pilot. By using an inference engine such as F-OWL [59], OWL becomes very powerful. But for the moment the execution of such an engine on a lightweight device is out of the question. If schema validation is turned off, and inferencing is not used, then OWL offers little over much lighter-weight alternatives.

The real novelty of Chakraborty et al.'s solution is the selective forwarding of queries, which is made possible by the exchange of abstract service information, known as *service groups*. A service group is a set of service instances that share the same service type. If a node receives a query which it cannot satisfy, it checks its list of cached service groups that it has built from previous service advertisements to see if there is a match. If there is, the query is forwarded to the neighbours which informed this node about the service group in question. In the event that there is no matching service group, the query is broadcast to all neighbours. The authors do not investigate alternatives to broadcast routing when selective routing fails. Note that the use of OWL is neither necessary nor sufficient for the operation of this selective routing protocol. That is, OWL is not necessary because the same query routing protocol can be used with other, lighter-weight description languages to the same effect; and it is not sufficient because it provides nothing without the group-based routing protocol.

D. SSDS

The Secure Service Discovery Service (SSDS) [5] from Berkeley is an attempt to provide service discovery to larger scale networks. Although SSDS is at least as old as some of the protocols in Section III, we classify it as an emerging protocol because its features are yet to be incorporated into any commercially available protocols. It consists of a hierarchy of service discovery service (SDS) servers (resolvers), each one responsible for the services and clients in its immediate vicinity. SDS servers solicit service information from services by announcing themselves on a well-known multicast channel. Services respond by sending XML descriptions of themselves to the SDS server. Clients send XML queries to the SDS server, which then attempts to match queries to registered services. Bloom filters [60] are used to limit the amount of service data which is propagated between SDS servers in

the hierarchy. A child SDS server sends a Bloom filter, which is a compact summary of the service information contained at a server, to its parent. The parent merges the summaries from all its children, and then merges this with its own service summary before forwarding the resultant summary to its parent. To save processing time and bandwidth, queries are checked against the Bloom summaries before being locally resolved or forwarded. Using such a mechanism guarantees that there are no false negative query resolutions, but there may be false positives. This just means that the occasional query is propagated further up or down the hierarchy than is optimal.

SSDS will scale to a large institution like a university campus. The bottleneck formed by the root node of a hierarchy of SDS servers prohibits SSDS from scaling beyond a network on the order of thousands of nodes; updates in SSDS (and service discovery protocols in general) are necessarily more frequent than in other hierarchical systems such as DNS. If queries are prevented from traversing the root node, results become dependent on the location of the querier. SSDS does include an expressive description and query language, making it a possible candidate for grid computing environments.

E. VIA

VIA [6] and VIA* [61] form cluster-based hierarchies of resolvers, where each level of the hierarchy does less work than the one above it. The hierarchy is such that each cluster at the same level of the hierarchy filters on the same attribute as its sibling clusters, but on a different value for the attribute. The nodes at the top level listen on a global multicast channel, and thus they receive all queries. Queries are only propagated to child clusters if the query matches at the top level of the hierarchical description. A node joins a cluster only if it would benefit in terms of the amount of work it does. Otherwise it stays on the multicast channel and processes all queries. There is a non-negligible cost associated with joining a cluster and maintaining membership of a cluster. Therefore it is not an obvious or automatic choice to join a cluster. A VIA node may become a child of another node only if the set of service descriptions at the child is a subset of the descriptions at the parent. If this is not the case, then queries that may have been matched by the child are filtered out by the parent, resulting in false negative responses.

VIA is aimed at the same environments as SSDS: a large campus or enterprise. It may be suitable for grid computing environments, as it allows fairly expressive queries. It handles node failure gracefully, though it is expensive if nodes higher in the hierarchy fail or disconnect. VIA also relies on IP multicast, making it unsuitable for many kinds of environments.

F. Splendor

The Splendor service discovery protocol [62] is somewhat similar in concept to other registry-based service discovery protocols, in that it incorporates clients, services and directories. The difference is that Splendor incorporates an advanced security scheme to accomplish authentication, non-repudiation, privacy and integrity. It also introduces proxies, which release services that execute on resource poor mobile devices from participating in the computationally expensive security protocol. The system utilises a public key infrastructure scheme to enable authentication and secure communication between two entities who are previously unknown to each other. Splendor therefore relies upon some existing infrastructure.

G. INS/Twine

INS/Twine [2] consists of a core of peer-to-peer resolvers. These resolvers are implemented on top of the Chord distributed hash table protocol (DHT). Descriptions and queries utilise the same format as the Intentional Naming System [63]. A resource or service description is a hierarchy of attribute-value pairs. Dependency of one A-V pair on another is signified by a parent-child relationship in the hierarchy. So, the pair [Room=633] is a child of [Level=6], which in turn is a child of [Building=GPSouth]. During advertising, the hierarchical description undergoes a strand extraction process, whereby all paths from the root of the description to each leaf and each sub-path from the root to each internal node are extracted and subjected to a hashing function. These hashed strands act as keys into the distributed hash table. The complete resource description is then stored at each resolver in the network that is responsible for each of the generated keys. For queries, the longest strand from the query is extracted and hashed. The query is then forwarded to the resolver responsible for the yielded key. If the query is successful, a *name record* containing contact information for the matching service or services is returned to the client. INS/Twine utilises a soft state protocol, so services must periodically refresh their advertisements.

INS/Twine will scale to environments the size of a large city, such as New York. However, because INS/Twine operates on a flat peer-to-peer network, scoping queries to the local area is a problem. Although INS/Twine allows for rich descriptions, the query language may not be expressive enough for many situations. Specifically, INS/Twine does not support relational operators such as “less-than” and “greater-than”. Rather, it relies on exact matching of a subset of the service description.

H. NEVRLATE

NEVRLATE [64] organises its nodes into a roughly square grid where advertisements are sent in one dimension and queries in the other. This way, advertisements and queries cost $O(\sqrt{N})$, where N is the number of nodes in the network. NEVRLATE can optimise lookup by enforcing an ordering on the resource descriptions so that a binary search can be used to find the node that stores the resource. A further optimisation can be made by making each server responsible for a section of the ordering, thereby providing lookup in constant time. It is not clear that ordering can be performed on many types of resource descriptions.

While NEVRLATE can dynamically adjust to node arrivals and departures, it cannot be used in networks consisting of heterogeneous protocols, since any imbalance in the number of underlying protocols leads either to a skewed grid or a very inefficient routing structure, whereby an advertisement or query may traverse a single node several times (if that node acts as a bridge between two or more diverse link-layers). This problem is not unique to NEVRLATE; any solution that uses address-based routing (such as the Internet Protocol) over heterogeneous link-layers may face a similar problem. The process of joining a NEVRLATE network can be expensive, since it may result in *set-splitting*, meaning that the number of rows in the grid must be increased. Likewise, when a node leaves, the number of rows in the network may shrink. This is called *set-absorption*. The protocol becomes very expensive when the protocol is caught in a cycle of set-splitting and set-absorption, which will occur if the number of nodes in the network is poised just below the set-splitting threshold, and if a node join is followed by a node departure.

I. Superstring

The final protocol examined in this class is known as Superstring [65–67]. Superstring defines a single resource discovery API and two underlying routing protocols to allow it to operate efficiently in structured (static) and unstructured (dynamic) environments.

Superstring scales to large numbers of nodes in a variety of environments, and scales from powerful computers to lightweight devices. Each routing layer addresses the concerns particular to its intended deployment environment. The protocol for unstructured networks evolves the network over time to reduce query times for popular resource types and adapts to changes in the network. The routing protocol for structured networks, on the other hand, focuses on scaling to large numbers of resources, since it is intended to be deployed in the wide-area, which may contain millions of resources. The less dynamic nature of structured networks meant that upward scalability, rather than adaptation, became the focus of the design effort for the protocol for structured

environments.

Superstring has a concise, yet powerful hierarchical description model that includes a lightweight expression language defined over attribute values for use in situations where exact attribute matches do not suffice. The description model contains a small number of reserved elements, including an element that allows query relaxation (the ability to dynamically and automatically weaken the constraints of a query in the event that there are no exact description matches) and an element that allows the specification of disjoint query components.

In addition to the core resource discovery elements identified above, Superstring defines a small set of primitives in the description language that allow queries and advertisements to become context-sensitive [68]. It thus becomes possible to quickly impart simple context-awareness to applications by issuing Superstring queries and advertisements.

J. Summary

The protocols surveyed in this section seek to overcome some of the shortfalls of those protocols in the previous section. Although none of the protocols, perhaps with the exception of Superstring, is designed to surmount all the problems inherent in pervasive computing environments, they each go some way toward solving particular problems. For instance, INS/Twine is highly scalable. OWL-Based Service Discovery provides rich resource descriptions. DEAPspace is focused on facilitating service discovery in dynamic networks. Superstring attempts to address many of these problems, and introduces features to enable context-sensitive resource discovery. It is these characteristics that make the protocols surveyed in this section a better match to modern computing environments than those surveyed in the previous section.

V. CONCLUSION

This chapter showed why modern computing environments require the use of resource discovery protocols to facilitate ad hoc interactions. User and device mobility means that applications can no longer rely on the presence of a pre-configured, static set of services and other resources to be available. Furthermore, changing user preferences, organisational policies and increased choice of service providers mean that applications require a mechanism that allows resources to be located

dynamically in real-time. While a range of current commercial protocols facilitate resource discovery in traditional computing environments, most of them fall short of supporting truly ad hoc interactions in modern computing environments because they fail to address the problems of scalability and heterogeneity. Resource discovery protocols that overcome many of the problems presented by modern computing environments are beginning to emerge.

However, there are still many problems to be solved. For instance, the bridging of resource discovery protocols is a major problem. There will always exist multiple resource discovery protocols. Thus, to enable interoperability between devices that support different protocols, some kind of bridge needs to be devised. Solutions to this problem to date are somewhat ad hoc in nature. They each define mappings, either unidirectional or bidirectional, between two protocols. A more general approach is required, but it is a difficult problem to solve. Resource discovery protocols define different description languages and type systems, and they support disparate modes of operation whereby some utilise lookup servers, others work in a peer-to-peer manner using a combination of queries and advertisements and still others utilise only queries or advertisements. In addition, it is still unclear how to build trust between mobile users and resources. While it is possible to locate a device that purports to provide a particular resource, it is an entirely different matter as to whether that resource should be trusted. Does a service really do what it is supposed to do? Another problem is scaling resource discovery protocols down to the very smallest of devices, such as sensors. The resource discovery protocols presented here require too many computational resources to operate on very small devices. To solve this problem they either need to be adapted to operate on small devices, or bridges need to be implemented between sensor networks and other networks so that sensor network resources may be discovered and utilised by applications in other networks.

The development of new resource discovery protocols is driven by breakthroughs in computer hardware and low-level network protocols that facilitate greater user mobility, which makes it possible to create novel kinds of applications. In addition, application designers are beginning to appreciate the flexibility and power of dynamically composing services from various resources in the computing environment. These are the forces that will drive the development and refinement of resource discovery protocols into the future.

-
- [1] L. Barkhuus and A. Dey, in *The Fifth International Conference on Ubiquitous Computing* (2003), pp. 149–156.
 [2] M. Balazinska, H. Balakrishnan, and D. Karger, in *Pervasive 2002 - International Conference on Pervasive*

- Computing* (Springer-Verlag, 2002), no. 2414 in LNCS, pp. 195–210.
 [3] Y. Y. Goland, T. Cai, Y. Gu, and S. Albright, *Simple Service Discovery Protocol/1.0*, IETF draft specification

- (1999).
- [4] Bluetooth SIG, *Bluetooth Specification version 1.1* (2001).
 - [5] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, in *MobiCom '99: 5th annual ACM/IEEE international conference on Mobile computing and networking* (ACM Press, Seattle, Washington, United States, 1999), pp. 24–35, ISBN 1-58113-142-9.
 - [6] P. Castro, B. Greenstein, R. Muntz, P. Kermani, C. Bisdikian, and M. Papadopouli, in *MobiCom '01: 7th annual international conference on Mobile computing and networking* (ACM Press, Rome, Italy, 2001), pp. 28–42, ISBN 1-58113-422-3.
 - [7] Sun Microsystems, Inc, Tech. Rep., Sun Microsystems, Inc (2001).
 - [8] J. Beck, A. Gefflaut, and N. Islam, in *MobiDe '99: 1st ACM international workshop on Data engineering for wireless and mobile access* (ACM Press, Seattle, Washington, United States, 1999), pp. 62–68, ISBN 1-58113-175-5.
 - [9] HAVi Inc, Tech. Rep., HAVi Inc (2001).
 - [10] S. Helal, N. Desai, V. Verma, and C. Lee, in *Third IEEE Conference on Wireless Communication Networks (WCNC)* (New Orleans, USA, 2003), pp. 2107–2113.
 - [11] A. Friday, N. Davies, N. Wallbank, E. Catterall, and S. Pink, *Wireless Networks* **10**, 631 (2004), ISSN 1022-0038.
 - [12] M. Esler, J. Hightower, T. Anderson, and G. Borriello, in *MobiCom '99: 5th annual ACM/IEEE international conference on Mobile computing and networking* (ACM Press, Seattle, Washington, United States, 1999), pp. 256–262, ISBN 1-58113-142-9.
 - [13] NASA, *Information Power Grid* (2004), URL <http://www.ipg.nasa.gov/>.
 - [14] *GrangeNet - GRid And Next GEneration Network* (2004), URL <http://www.grangenet.net/>.
 - [15] I. Foster and A. Iamnitchi, in *2nd International Workshop on Peer-to-Peer Systems* (2003), pp. 118–128.
 - [16] D. S. Milojevic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, Tech. Rep. HPL-2002-57, HP Laboratories (2002), URL <http://citeseer.nj.nec.com/milojevic02peertopeer.html>.
 - [17] K. Kant, R. Iyer, and V. Tewari, in *CCGRID '02: 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid* (IEEE Computer Society, 2002), p. 368, ISBN 0-7695-1582-7.
 - [18] Bluetooth SIG, *Bluetooth Specification version 1.1* (2001).
 - [19] IEEE, *IEEE Std 802.11-1999, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* (1999).
 - [20] T. Bray, J. Paoli, and C. M. S.-M. (eds), *W3C Recommendation: Extensible markup language XML 1.0* (1998).
 - [21] Bluetooth SIG, *Bluetooth Specification version 1.1* (2001).
 - [22] The Salutation Consortium, *Salutation architecture specification (part 1) v2.0c* (1999).
 - [23] ITU-T/ISO/IEC, *Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation*, ITU-T Recommendation: X.680 (2002) — ISO/IEC 8824-1:2002 (2002).
 - [24] S. Cheshire and M. Krochmal, *DNS-based service discovery*, IETF Draft Specification (2004), URL <http://files.dns-sd.org/draft-cheshire-dnsextdns-sd.txt>.
 - [25] Apple Computer, Inc., *Rendezvous technology brief* (2003), URL http://images.apple.com/macosx/pdf/Panther_Rendezvous_TB_10232003.pdf.
 - [26] S. Cheshire, B. Aboba, and E. Guttman, *Dynamic configuration of IPv4 link-local addresses*, IETF Draft Specification (2004), URL <http://files.zeroconf.org/draft-ietf-zeroconf-ipv4-linklocal.txt>.
 - [27] S. Cheshire and M. Krochmal, *Multicast DNS*, IETF Draft Specification (2004), URL <http://files.multicastdns.org/draft-cheshire-dnsextdnsmulticastdns.txt>.
 - [28] P. V. Mockapetris, *RFC 1035: Domain names — implementation and specification* (1987), URL <http://www.ietf.org/rfc/rfc1035.txt>.
 - [29] Napster, *The napster homepage* (2003), <http://www.napster.com/>.
 - [30] Clip2, *The gnutella protocol specification v0.4* (2003), URL <http://www9.limewire.com/developer/gnutella.protocol.0.4.pdf>.
 - [31] Sun Microsystems, Inc., *JXTA v2.0 Protocols Specification* (2004), URL <http://www.ietf.org/internet-drafts/draft-duigou-jxta-protocols-05.txt>.
 - [32] C. E. Perkins and H. Harjono, *Mobile Networks and Applications* **1**, 447 (1996), ISSN 1383-469X.
 - [33] M. Wahl, T. Howes, and S. Kille, *IETF RFC 2251: Lightweight Directory Access Protocol (v3)* (1997), URL <http://www.ietf.org/rfc/rfc2251.txt>.
 - [34] International Telecommunication Union (ITU), *X.500 - Open Systems Interconnection - The Directory: Overview of concepts, models and services* (2002).
 - [35] ISO/IEC, *Open Distributed Processing - Trading function: Specification*, ISO/IEC 13235-1:1998 (2002).
 - [36] OMG, *Trading Object Service version 1.0* (2000), URL <http://www.omg.org/technology/documents/formal/trading-object-service.htm>.
 - [37] OASIS UDDI Specification TC, *UDDI Version 3.0.1* (2003), URL <http://uddi.org/pubs/uddi.v3.htm>.
 - [38] ISO/IEC, *Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation)*, INCITS/ISO/IEC 9075-2-2003 (2003).
 - [39] M. Schwartz, IEEE Computer Society Technical Committee Newsletter on Operating Systems and Application Environments **5**, 10 (1991).
 - [40] S. Milgram, *Psychology Today* pp. 60–67 (1967).
 - [41] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, *Physical Review E* **64** (2001).
 - [42] N. Sarshar and V. Roychowdhury, *Physical Review E* **69** (2004).
 - [43] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin, *IEEE Transactions on Mobile Computing* **5**, 97 (2006), ISSN 1536-1233.
 - [44] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, *IETF RFC 2165: Service location protocol* (1997), URL <http://www.ietf.org/rfc/rfc2165.txt>.
 - [45] E. Guttman, C. Perkins, J. Veizades, and M. Day, *IETF RFC 2608: Service location protocol, version 2*.
 - [46] T. Howes, *RFC 2254: The string representation of LDAP search filters* (1997), URL <http://www.ietf.org/rfc/rfc2254.txt>.
 - [47] W. Zhao, H. Schulzrinne, and E. Guttman, in *IEEE International Conference on Computer Communications*

- and Networks (ICCCN'00) (Las Vegas, USA, 2000), pp. 504–509.
- [48] W. Zhao, H. Schulzrinne, and E. Guttman, *IETF RFC 3528: Mesh-enhanced service location protocol* (2003), URL <http://www.ietf.org/rfc/rfc3528.txt>.
- [49] C. Lee, A. Helal, N. Desai, V. Verma, and B. Arslan, *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* **33**, 682 (2003).
- [50] S. J. Lee, W. Su, and M. Gerla, *Mobile Networks and Applications* **7**, 441 (2002), ISSN 1383-469X.
- [51] *W3C Working Draft: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language* (2004), URL <http://www.w3.org/TR/2004/WD-wsdl20-primer-20041221/>.
- [52] R. Hermann, D. Husemann, M. Moser, M. Nidd, C. Rohner, and A. Schade, *Comput. Networks* **35**, 411 (2001), ISSN 1389-1286.
- [53] N. Borenstein and N. Freed, *RFC 1521: MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of Internet message bodies* (1993), URL <http://www.ietf.org/rfc/rfc1521.txt>.
- [54] *W3C Recommendation: OWL Web Ontology Language Overview* (2004), URL <http://www.w3.org/TR/2004/REC-owl-features-20040210/>.
- [55] *W3C Note: Daml+oil (march 2001) reference description* (2001), URL <http://www.w3.org/TR/daml+oil-reference>.
- [56] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, in *4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN)* (IEEE, Stockholm, Sweden, 2002), pp. 140–144.
- [57] *W3C Recommendation: Resource Description Framework Primer* (2004), URL <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.
- [58] N. Stojanovic, R. Studer, and L. Stojanovic, in *2nd International Semantic Web Conference* (Springer-Verlag, 2003), pp. 500–516.
- [59] Y. Zou, T. Finin, and H. Chen, in *Formal Approaches to Agent-Based Systems*, edited by M. Hinchey, J. L. Rash, W. F. Truszkowski, and C. A. Rouff (Springer-verlag, 2004), no. 3228 in LNCS.
- [60] B. H. Bloom, *Communications of the ACM* **13**, 422 (1970), ISSN 0001-0782.
- [61] P. Castro and R. Muntz, in *MobiDe '01: 2nd ACM international workshop on Data engineering for wireless and mobile access* (ACM Press, Santa Barbara, California, United States, 2001), pp. 14–19, ISBN 1-58113-412-6.
- [62] F. Zhu, M. Mutka, and L. Ni, in *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications* (IEEE Computer Society, Washington, DC, USA, 2003), p. 235, ISBN 0-7695-1893-1.
- [63] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, in *SOSP '99: seventeenth ACM symposium on Operating systems principles* (ACM Press, Charleston, South Carolina, United States, 1999), pp. 186–201, ISBN 1-58113-140-2.
- [64] A. Chander, S. Dawson, P. Lincoln, and D. Stringer-Calvert, in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid* (IEEE Computer Society, Berlin, Germany, 2002), pp. 352–358.
- [65] R. Robinson, Ph.D. thesis, School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia (2005), URL <http://www.rickyrobinson.id.au/publications/RobinsonPhDThesis.pdf>.
- [66] R. Robinson and J. Indulska, in *11th IEEE International Conference on Networks* (Sydney, Australia, 2003), pp. 699–704.
- [67] R. Robinson and J. Indulska, in *Database and Expert Systems Applications, 15th International Workshop on (DEXA'04)* (IEEE Computer Society, Zaragoza, Spain, 2004), pp. 657–661, ISBN 0-7695-2195-9.
- [68] R. Robinson and J. Indulska, in *The Fourth International Conference on Mobile Business* (IEEE Computer Society, Sydney, Australia, 2005).