

# Resource Discovery in Modern Computing Environments

Ricky Robinson

*National ICT Australia*

*ricky.robinson@nicta.com.au*

Jadwiga Indulska

*School of Information Technology and Electrical Engineering*

*The University of Queensland*

*jaga@itee.uq.edu.au*

Ted McFadden

*CRC for Enterprise Distributed Systems Technology*

*mcfadden@dstc.edu.au*

**Abstract**—Resource discovery is an integral part of many kinds of computing environments, including emerging heterogeneous environments for pervasive computing. This paper characterises a number of current and future computing environments and summarises their resource discovery requirements. It then analyses, with respect to the requirements of each environment, several established service discovery protocols and some newer protocols that are still in the research domain. In addition, the key features of a new resource discovery protocol that has been developed to operate with heterogeneous computing environments are described.

**Index Terms**—pervasive computing, resource discovery, service discovery

## I. INTRODUCTION

This paper presents a comprehensive survey of existing resource discovery protocols. It classifies resource discovery protocols using a novel taxonomy that enables these protocols to be compared. The secondary purpose of this paper is to summarise the chief benefits of our own resource discovery protocol, Superstring [1, 2], which overcomes many of the problems highlighted by the literature survey.

The emergence of pervasive computing and its predicted interfusing with grid computing [3] will see an increased requirement for service discovery protocols that are capable of operating in very large, heterogeneous environments. This next generation of service discovery protocols needs to scale to much larger networks and support much richer kinds of service descriptions and queries. Expressive queries are required in the

wide-area for two main reasons. Firstly, they aid the preservation of expensive long-haul bandwidth: the more expressive the query, the less chance there is of matching irrelevant services, and the smaller the query response. Secondly, expressive queries allow fine grained selection of services by clients.

Today's resource discovery protocols are specialised for particular kinds of computing environments because different computing environments have dramatically different characteristics. A protocol intended to operate in a conventional office environment cannot be expected to operate with the same degree of efficiency in a mobile ad hoc network. Bridging provides a partial solution; however, due to the difficulty of bridging resource discovery protocols, new ways of dealing with environment heterogeneity are necessary. One way is to provide a single service discovery interface to applications and then to provide alternative query routing mechanisms that are suited to a particular environment. This allows applications (both clients and services) to migrate from one environment to another without requiring separate service discovery interfaces for each environment. Furthermore, it allows advertisements and queries to be forwarded from one environment to another with minimum effort. This is the approach explored in this paper.

The paper is organised as follows. Section II presents a comprehensive and thorough survey of existing resource discovery protocols. It defines the various computing environments that can be found in the modern world, and then proceeds to classify resource discovery protocols according to their suitability to these environments. In Section III, further detail is given about Superstring, a protocol that is well suited to heterogeneous computing environments. Section IV concludes the paper, and suggests some areas of future work.

## II. A SURVEY OF RESOURCE DISCOVERY PROTOCOLS

Before engaging in a detailed analysis of existing resource discovery protocols, it is imperative that the range of envi-

---

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.

The work reported in this paper has been funded in part by the Co-operative Research Centre Program through the Department of Industry, Science and Tourism of the Commonwealth Government of Australia.

vironments in which they are used is fully understood. Only by surveying the kinds of environments, and the unique characteristics each of them exhibits, is it possible to engage in a meaningful discussion of existing protocols, and to determine their strengths and weaknesses. This approach is all the more appropriate when one considers that existing protocols are targeted to only one, or a small selection, of the variety of contemporary computing environments. Section II-A describes present-day computing environments in detail.

Section II-B discusses past and present resource discovery protocols, and matches them to the kinds of environments they support. Each protocol is also analysed in depth, thereby yielding a collection of shortcomings which were used to inform the design of Superstring.

#### A. Environment Characterisation and Requirements

Service discovery is a necessary component of many kinds of computing environments. In this section, these environments are characterised according to node mobility and stability, the type of underlying network, network size and the typical number of clients and resources. The following categories are not disjoint. For example, peer-to-peer networks overlay the Internet, but these can still be considered as disparate computing environments.

1) *The Internet and Wide-Area Networks:* There are numerous examples of wide-area networks ranging from telecommunications networks to proprietary data networks, each of which contains a variety of services and resources. But perhaps the best known wide-area network is the Internet. The Internet is a global scale “network of networks” consisting of heterogeneous computing devices that share common network and transport layer protocols (TCP/IP). Generally speaking, the core of the Internet is comprised of a stable set of routers. As one moves from the core toward the edges of the Internet, stability is replaced by varying degrees of instability as end nodes connect and disconnect. The Internet Protocol and transport layer protocols (TCP and UDP) provide a basis on which application layer protocols are built. Two of the most popular applications used on the Internet today are the World-Wide Web and electronic mail. The web and electronic mail are both client-server applications. On the web server processes are usually long-lived and execute on stable infrastructure, while the client processes that access them are generally much shorter lived. Application level interactions are often mediated by a proxy for purposes of providing increased security and efficiency. The web has rapidly moved from being a set of static documents to a dynamic, service-oriented platform. Standards such as SOAP [4] have emerged to enable the interoperation of web services, and allow clients to invoke operations on web services in standard ways. These web services, as well as documents and other kinds of objects, are typical resources that can be found on the Internet. In recent times, peer-to-peer file sharing applications have become popular on the Internet. Section II-A.4 provides an overview of resource discovery in peer-to-peer environments and applications.

2) *Smaller Area Networks:* Often, local-area networks (LANs) are connected to wider area networks such as the

Internet. However, when viewed in isolation, these networks exhibit characteristics distinct from those outlined in the previous section. LANs are often built from the same network and transport layer protocols as the Internet. However, the lower layer protocols are very different. The wide-area is comprised predominantly of point-to-point links, while local-area networks generally utilise shared medium protocols such as IEEE 802.3 and IEEE 802.11. Also, LANs are usually contained within a single administrative domain, making it feasible to deploy applications that use multicast communications. While the users in LAN environments are ultimately the same users who utilise web services and other services available in the wide-area, they have needs much closer to home as well. These users need to print and scan documents, and their web browsers must send requests to nearby proxies. Each of these scenarios necessitates service discovery, or at the very least, can be made more autonomous by using service discovery.

In recent times, home and office environments are being augmented with networked appliances such as entertainment and security systems in an effort to make these spaces *smarter*. These appliances generally utilise lower bandwidth protocols such as Bluetooth [5]. Thus, these spaces are moving toward pervasive environments (see Section II-A.6 below). Other small area networks are coming into use with the advent of controller-area networks (CANs), which connect the computerised components within modern motor vehicles, some planes and other machinery. Personal-area networks (PANs) have also been developed to provide communication between the devices carried by a person. For instance, a personal-area network could be used to connect an audio headset, head-mounted display and mobile computer to provide navigation assistance to a soldier in the field. Each device within a small network must establish physical connections with its neighbours and then discover the features or services they provide.

3) *The Grid:* Grid computing environments offer a way to harness unused CPU cycles and storage. Present-day grids 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 [6]. The typical use of a grid is to allow users to submit tasks for execution to the grid infrastructure. This involves matching the computational requirements of a task to nodes in the grid that can cater to those requirements. Such requirements include machine architecture, operating system, execution environment (for example, Java 2 or Python), number of CPUs and available memory. Grids also support the migration of tasks from one node to another and the execution of a single task across multiple nodes if the task is amenable to such a distributed and parallel execution. As well as offering these computational resources, grids may also offer value-added services such as specialised software components. Clients may invoke applications leased by Application Software Providers (ASPs), and these ASPs in turn may require additional computational resources from

the grid. Grids may one day support pervasive environments by offering computational resources to resource-poor mobile devices.

The grid offers both low-level, primitive computational resources as well as collections of computational resources (such as distributed storage) and software services. Each primitive resource must enable discovery of its state, structure, quality of service and other capabilities. Above this layer are aggregate resources. These should be discoverable by clients in the same way as primitive resources. To enable this, the grid infrastructure provides a coordination layer, which manages the aggregation of these compound resources. For example, if a client needs to store a large amount of data, and no single storage resource can satisfy the client's requirements, then several storage resources can be combined to satisfy the request. Either the resource discovery mechanism needs to support this behaviour directly (which implies the resource discovery protocol itself should support transactions so that consistency is preserved during a query involving multiple resources), or it needs to provide rich enough description and query capabilities such that the application or a mediation layer can construct a set of queries which collectively satisfy the client's needs. Relational comparison operators (less-than, greater-than and so on) are a definite requirement in these aggregation scenarios.

To support a large array of resource types and instances, grid computing environments offer discovery protocols which allow a detailed specification of requirements, including operating system, memory size and scope (to constrain the results to a particular administrative domain). Often, they also support the specification of a benefit or ranking function, which allows some or all of the attributes in the list of requirements to be weighted.

In grid environments, resource discovery protocols often operate alongside resource management and reservation protocols. Discovery protocols are utilised to gain information about specific resources, while management protocols arbitrate access to those resources and enforce any policies that are in use. Other roles of the resource management protocol are to ensure fair access to resources, and to enable accounting and payment. The resource manager may play a role in updating the description of a resource, and might also find it necessary to issue queries for resources.

4) *Peer-to-Peer*: 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 other network environments. 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.

The most widely known P2P applications are content-sharing systems such as Napster [7], Gnutella [8], and Freenet [9]. The first two have gained notoriety for the 'swapping' of MP3 music files (although, Napster was shut down

and subsequently reinvented itself as a legal music subscription service) and the last for its anonymity features, such as masking the publishers and consumers of content. In both Napster and Gnutella, P2P routing is used only for resource discovery, not to retrieve content. Retrieval is done using the underlying network directly, whereas Freenet routes retrieved content through its P2P overlay. In JXTA [10], an application-neutral P2P infrastructure, both queries and content are routed through the P2P virtual network. This allows JXTA to operate in situations where underlying network nodes do not have direct connectivity or are of heterogeneous types. For example, JXTA uses relay peers to route messages (XML documents) between peers sitting on either side of a firewall.

Gnutella and Napster allow resources (files) to be discovered by title or keywords, although the underlying discovery mechanisms are very different, with Napster relying on a single, centralised service (which meant that it was easy to close down) and Gnutella on multicast flooding. Freenet does not support discovery, requiring the client to obtain a content identifier key, by an out-of-band means, which is used directly to retrieve the content.

These types of P2P applications are built in homogeneous peer environments where inter-peer communication is simple and query constructs to discover resources are adequately represented by a title, ID, or keywords. More complex P2P applications requiring heterogeneous resources and richer query constraints are described in the grid and ad hoc networks sections.

5) *Ad Hoc Networks*: 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 [5] and IEEE 802.11. Bluetooth is limited in scope: its primary purpose is to replace the cables on devices such as printers, mice and headsets. However, it is also capable of providing communication within an isolated set of devices, and allows the establishment of ad hoc communication with fixed infrastructure where an access point is available. Other, even more ambitious kinds of ad hoc networks include sensor deployments and pervasive computing environments built from a variety of wired and wireless networks providing communication between a variety of computing devices.

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. A common application of ad hoc networks is to enable a user's laptop computer, handheld computer, printer and scanner (among other devices), to work seamlessly together, so that the user may accomplish their task (which, in this case, might be to prepare an e-mail that includes the scanned image of a document, to retrieve the recipient's e-mail address from the address book stored on the handheld computer, to send the e-mail via an access point attached to a LAN, and to print the recipient's response). To cater to a scenario such as this, the resource discovery protocol must initiate device discovery (if low-level network communication

with neighbouring devices has not yet been established) and locate the devices that offer the services necessary to complete the user's task. In this scenario, queries for resources should not be propagated beyond the local environment. Since some of the devices taking part in this scenario are very lightweight, the resource description language should be simple enough and small enough to be managed by the smaller devices.

On the other hand, if an ad hoc network supports a team of people working together on a particular task (for example, emergency response) a more sophisticated resource discovery protocol may be required. This is the case if a variety of computing services, both within the ad hoc network and in the fixed networking infrastructure to which the ad hoc community is connected, need to be discovered. The queries must be able to describe services, their computational requirements, and Quality of Service (QoS) provided by the services. The QoS description can be complex as it may need to describe a variety of indices including communication QoS (bandwidth, latency, delay, jitter), display quality, security levels (integrity, confidentiality, non-repudiation), reliability, and so on. Some of the query parameters may be flexible, specifying a range of acceptable values. This requires service discovery protocols which are able to utilise procedures for parameter evaluation. The queries also need to capture user preferences in order to build benefit functions used to decide which resources provide the best match for user requirements. Scoping of the queries to a particular region is also essential, as ad hoc networks may be linked to wide-area networks.

6) *Pervasive Computing Systems*: 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 systems will comprise a mixture of dynamic and stable, structured and unstructured, wired and wireless, networks. This type of network organisation is termed a *hybrid wireless mesh network* in the literature, and is expected to be an extremely common network architecture in the future [11]. Contrary to the focus of much pervasive computing literature, the pervasive computing environment is *not* limited to the immediate computing environment surrounding a user. Grid computing environments, for example, may be merged with pervasive computing environments to provide computational resources to remote mobile devices.

Service discovery protocols will be an integral part of the pervasive computing environment for the following (non-exhaustive) set of reasons:

- 1) to enable autonomy and to facilitate adaptation in a changing environment, applications must have awareness of the services and resources around them;
- 2) the creation of novel applications, many of which will rely on knowledge of the changing environment (dynamic context information), will be feasible with the aid of resource discovery;

- 3) some applications will require resources from beyond the local-area, and these need to be discovered, just as a local resource would be; and
- 4) often, an application in the pervasive computing environment is not a single monolithic process; rather it is constituted of several distributed components, each of which performs a specialised task.

It can be argued that the pervasive computing environment subsumes the environments previously described. Service discovery protocols for pervasive computing environments must therefore deal with issues of system scale, dynamic changes in available services, and a rich variety of queries.

7) *Summary*: Although computing environments are many and varied, it is possible to loosely classify them in terms of the level of structure they exhibit. The spectrum of structure level may be continuous; however, the discrete classes *unstructured*, *semi-structured* and *structured* suffice to categorise each of the environments described above. These classes are defined as follows.

- **Unstructured** environments usually contain dynamic elements and often hostile conditions. In these environments, disconnections occur frequently, due either to mobility or intentional termination by users. Sometimes, as in the case of many peer-to-peer protocols, an unstructured environment may be layered on top of a stable infrastructure.
- In **semi-structured** environments, the core components of the environment may exhibit structure while the edges exhibit a higher degree of disconnection and uncertainty.
- **Structured** environments rely on stable infrastructure. Disconnection is rare in these environments, and redundant components are often used to cover for failed components. Thus, from the user's perspective, failure rarely, if ever, occurs.

Note that an environment may span several or all of the classes: there is no requirement that an environment must be assigned to one and only one category. Table I shows the classification.

In light of the descriptions of each environment above, this categorisation is largely intuitive. However, the classification of P2P environments into all three classes bears further discussion.

It would be easy to incorrectly classify P2P protocols as being only unstructured or semi-structured, since the highly visible P2P protocols in use today fall into these categories. But the imparting of structure onto a protocol does not preclude it from being classified as a P2P protocol. Perhaps the primary instance of such a protocol is Chord [12], which constructs a distributed hash table in a flat routing domain. In essence, each Chord node is created equal and can be considered to be a "bucket" in the hash table.

The next section examines a range of resource discovery protocols, and places each protocol within one or more of the above categories.

## B. Existing Resource Discovery Protocols

Following is a discussion of the suitability of existing protocols to the three broad environments outlined above.

| Unstructured               | Semi-structured                | Structured   |
|----------------------------|--------------------------------|--|
| Pervasive<br>P2P<br>Ad Hoc | Pervasive<br>P2P<br>Local-Area | Pervasive<br>P2P<br>Local-Area<br>Internet and Wide-Area<br>Grid |

TABLE I  
CLASSIFICATION OF ENVIRONMENTS

The protocols falling into each category are compared and contrasted.

Although each of the protocols surveyed is unique, most of them share common components. This general set of components informs the design of any resource discovery protocol. These components are listed here:

- 1) resource description language;
- 2) routing protocols;
- 3) message formats; and
- 4) application programming interface (although some protocols, such as Bluetooth SDP, are defined purely in terms of their message formats).

However, the ever-widening scope of computer-enhanced environments brings the following challenges, which follow from the survey of modern computing environments in the previous section.

- 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

We find that, of the protocols examined below, only Superstring meets these challenges. Of particular importance is the ability of resource discovery protocols to operate in heterogeneous environments. This is critical due to the major challenges associated with bridging resource discovery protocols. The vast differences in descriptive capabilities, semantics, routing and protocol behaviour makes bridging an unattractive solution to the problem of routing queries between multiple types of environments. Typical bridging solutions are designed on a case-by-case basis and they generally do not constitute a complete bidirectional mapping between the two protocols in question.

1) *Unstructured Environments*: A large number of commercial service discovery protocols are targeted at small, unstructured communities of devices. This section surveys a comprehensive selection of such protocols.

The Simple Service Discovery Protocol (SSDP) [13] is utilised by Universal Plug and Play 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. 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.

The Service Discovery Protocol (SDP) [14] 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 link layer protocol. SDP is therefore suitable only for very small ad hoc groups of wireless devices.

Service discovery in Salutation [15] 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 [16]. 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.

Helal et al. have designed a protocol named Konark [17], 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 [18]). 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 [19].

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 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 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.

The DEAPspace [20] 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 overhead 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; 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). Like most of the other protocols surveyed here, 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.

DNS-SD (Domain Name System - Service Discovery) [21] provides service discovery capabilities for Apple's Rendezvous technology [22]. A Rendezvous enabled device first assigns itself an IP address from the link-local range. 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 to register their services with other devices on the network. Clients can also use multicast DNS-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 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.

The rise of peer-to-peer computing can largely be attributed to applications such as Napster [7] and Gnutella [8]. JXTA [10] 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 message 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.

Chakraborty et al. adopt the use of the Web Ontology Language (OWL) [24] to describe resources in mobile and pervasive computing environments. 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) [25], 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, 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 notebook 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 [26], 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.

2) *Semi-Structured Environments*: Jini [27] 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.

Perkins and Harjono [28] 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 the Dynamic Host Configuration Protocol (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 *n2l*, 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), 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.

The Secure Service Discovery Service (SSDS) [29] from Berkeley is an attempt to provide service discovery to larger scale networks. 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 [30] 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.

VIA [31] forms 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.

3) *Structured Environments*: INS/Twine [32] consists of a core of peer-to-peer resolvers. These resolvers are implemented on top of the Chord distributed hash table protocol (DHT). 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.

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.

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 Trader is a centralised component, making it unsuitable for dynamic networks such as MANETs. It does not provide query relaxation, though services can be selected with some degree of granularity.

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 queries, though they are marked up using XML. Clients direct queries at a particular node within a registry.

In the approach taken by Schwartz [38], 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 maintain 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 primarily on the Small World phenomenon [39], 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 [40, 41].

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 (described above) bears a striking resemblance to Schwartz's solution.

4) *Unclassified Protocols*: The unclassified protocols can operate in structured, semi-structured and unstructured environments depending upon their configuration. Though each protocol can span these multiple environments, only one of them can solve the problem of operation in the all-encompassing, heterogeneous pervasive environment. The reasons for this are outlined in the following discussion.

The Service Location Protocol (SLP) [42] 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 to specify attributes of interest. It is possible for DAs to contain inconsistent information about available resources. To address this, Mesh Enhanced SLP [43] 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.

NEVRLATE [44] 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 the diversity 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.

The final protocol examined in this class is known as Superstring [1, 2]. Superstring defines a single resource discovery API and two underlying routing protocols to allow it to operate efficiently in disparate environments. The following points summarise the key design choices made in Superstring:

- separation of routing layers from the application interface;
- a protocol for static structured networks;
- a protocol for dynamic unstructured networks; and
- a lightweight, but expressive description language.

These key elements allow Superstring to scale to large numbers of nodes in a variety of environments, and to scale 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 [45]. It thus becomes possible to quickly impart simple context-awareness to applications by issuing Superstring queries and advertisements.

The routing layers, description language and single API combine to define a powerful resource discovery protocol capable of operating in a multitude of heterogeneous environments.

5) *Summary*: This survey has covered a wide array of resource discovery protocols, and analysed their strengths and weaknesses. It is clear that most of the protocols reviewed are targeted toward particular computing environments. Superstring is arguably the only protocol suited to 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, such as Superstring, that can overcome the problems of heterogeneity and scale posed by the set of contemporary computing environments facilitate the development of ubiquitous computing applications that can operate in a wide range of situations. Furthermore, such a protocol will enable the creation of a new breed of distributed applications, which can draw upon the resources offered in disparate environments.

### III. SUPERSTRING

In the previous section, the traits of various computing environments were identified and existing resource discovery protocols were classified according to the extent to which they meet the operational requirements of each computing environment. It was concluded that of the reviewed protocols, Superstring offers the best support for heterogeneous computing environments. This section presents the key features that allow Superstring to operate in the disparate computing environments that constitute the modern computing landscape.

#### A. Small API

Powerful computers and tiny pocket computers alike access the features of Superstring through a common, small programming interface. The interface defines two core operations:

```
QueryResponse query(Query q);
void advertise(Advertisement a);
```

When utilised with the powerful but lightweight description language, these two operations suffice to provide Superstring with the core resource discovery functionality as well as the context-sensitive features. In the interests of conserving space, the programming interfaces for Queries and Advertisements are not shown here; they define functions to create transient and persistent queries and advertisements respectively. Transient queries are regular queries that attempt to match advertisements at the time of issue. Persistent queries are akin to subscriptions in publish/subscribe protocols. Transient advertisements are analogous to notifications in publish/subscribe

protocols. They have meaning only at the time of publishing. Persistent advertisements are regular advertisements that are stored within the network for an extended period (until a soft-state timer expires).

#### B. Powerful, Lightweight Description Language

A recent trend in resource discovery is to utilise relatively heavyweight description languages such as RDF [25] and OWL [24], as they impart *semantics* to information that is exchanged by distributed entities. This, in turn, allows applications to perform limited reasoning about the information they receive. But this comes at the cost of computational and communication overhead. While these costs may be borne by a structured network containing high bandwidth links and high-end computers (as is the case for the majority of nodes constituting the World-Wide Web), the cost may become prohibitive in unstructured networks. Document validation incurs communication *and* computation costs. Inferencing engines, such as F-OWL [26], add another layer on top of OWL, introducing further computational costs. Furthermore, existing inferencing engines are known to scale poorly [26].

Superstring *bucks the trend*, by choosing a much lighter-weight description model. The model is mapped to XML in practice. XML was chosen as the serialisation syntax due to the wide availability of XML parsing tools, and specifically because of the existence of XML parsing libraries for lightweight devices. This allows Superstring to scale upward, to high-end devices, and downward, to small, mobile devices. The description language offers powerful features such as expressions, which are lacking in other description languages for small devices such as Bluetooth SDP, and query relaxation, which is missing from description languages for high-end and low-end devices alike.

Context-sensitivity is introduced to the description model via the *context* and *bind* elements. A *context* attribute appearing in a query or advertisement signals to the protocol that what follows is a sub-query that should be resolved first, the result of which should be bound to the super-query in place of the sub-query. The precise information that should be bound in place of the sub-query is controlled with a *bind* attribute. This model does not prohibit multi-level sub-queries, meaning that context-sensitive queries can be arbitrarily complex. The hierarchical description abstraction means that the author of queries need not be aware of the detailed structure of sub-descriptions. For example, if a resource defines its location using several sub-components (logical, geodetic and physical location), the author of the context-sensitive query can merely specify that location information is of interest, without concern for the different types of location sub-components. As with ordinary Superstring queries, parts of a context-sensitive query may be wrapped in a *scope* element to signify that query relaxation should apply in the event of no exact matches. Figure 1 demonstrates the use of the *context* and *bind* attributes. This example specifies a context-sensitive query in which the sub-query is also context-sensitive. The deepest *bind* element specifies that the sub-query should be replaced by the location components that appear in the result returned by the sub-query.

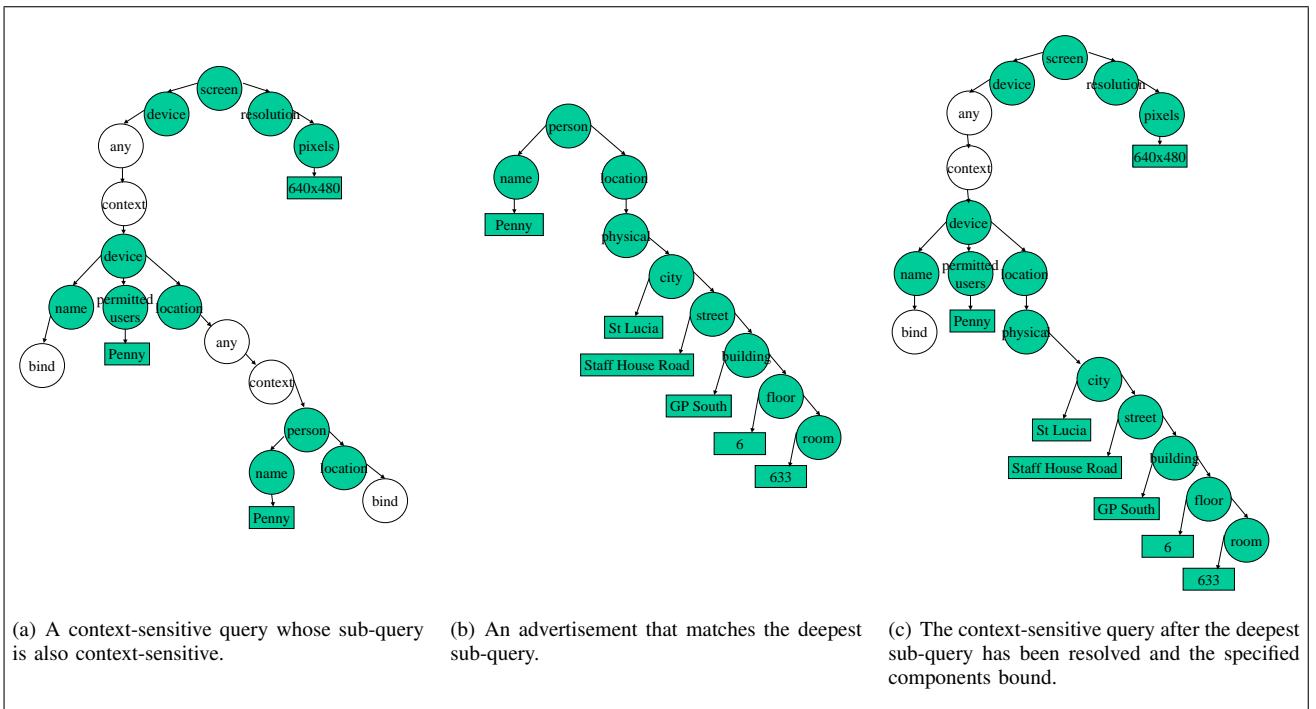


Fig. 1. Context-sensitive query resolution.

The other *bind* element dictates that the name of any matching device should replace the sub-query.

While the work of Chen and Kotz [46] is similar to the context-sensitive service discovery protocol outlined above, there are several important differences that will now be highlighted. The first major divergence is that, although Chen and Kotz define a context-sensitive naming mechanism, the design is such that the author of the context-sensitive query must always have knowledge of the structure of those names. The hierarchical naming approach employed by the protocol described in this paper places no such restriction on the author. For example, a mobile, location-sensitive application need not know the detailed structure (physical, logical, geodetic or something else entirely) of the location information utilised by the current environment. Instead, the location component can be specified in very abstract terms. This feature enables much greater flexibility than the solution of Chen and Kotz. Furthermore, Chen and Kotz do not define a mechanism for query relaxation. Query relaxation is a powerful tool in its own right, but is especially useful when combined with context-sensitive query completion and preferences.

Augmenting the description language is a companion preference language that enables query results to be culled at the resolver and ranked before they are returned to the querier, resulting in reduced bandwidth consumption. Preferences are also useful for fine-tuning the results that are returned in the event that query relaxation is applied. For instance, if a query is issued to find a resource in a particular room on a particular floor in a particular building, and there are no matching resources in the room or on the specific floor but there are matching resources on all the floors below, a preference can be used to favour the floors nearest to the current floor when query relaxation is applied (that is, when the query is relaxed firstly

from the room to floor resolution, then from floor resolution to building resolution). Furthermore, preferences can be context dependent. An example use of context-sensitive preferences is to rank results one way if it is raining and another way if it is sunny. Superstring’s preference language draws on decision theory models described by Fishburn [47].

### C. Two Distinct Routing Layers

There are two complimentary routing layers in Superstring, and the way in which these layers interact, from a high-level perspective, is shown in Figure 2.

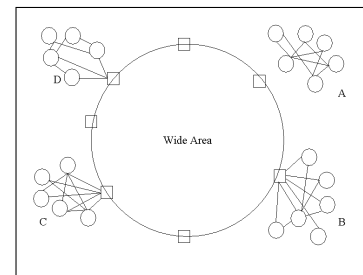


Fig. 2. The wide- and local-area components of Superstring. Local network A is an independent, completely ad hoc network, unsupported by the wide-area infrastructure. Local networks B and C show scenarios where every mobile node is connected to the wide-area node. Finally, local network D shows a network in which only some nodes are directly connected to the wide-area node. In each case, the mobile nodes utilise a transport independent hop-by-hop query routing protocol.

One, based on distributed hash tables, is suitable for the wide-area, as it scales to large numbers of nodes and resources. This routing layer guarantees no false negative query results. Briefly, the routing layer deterministically selects the nodes on which a description should be stored. Queries are then routed

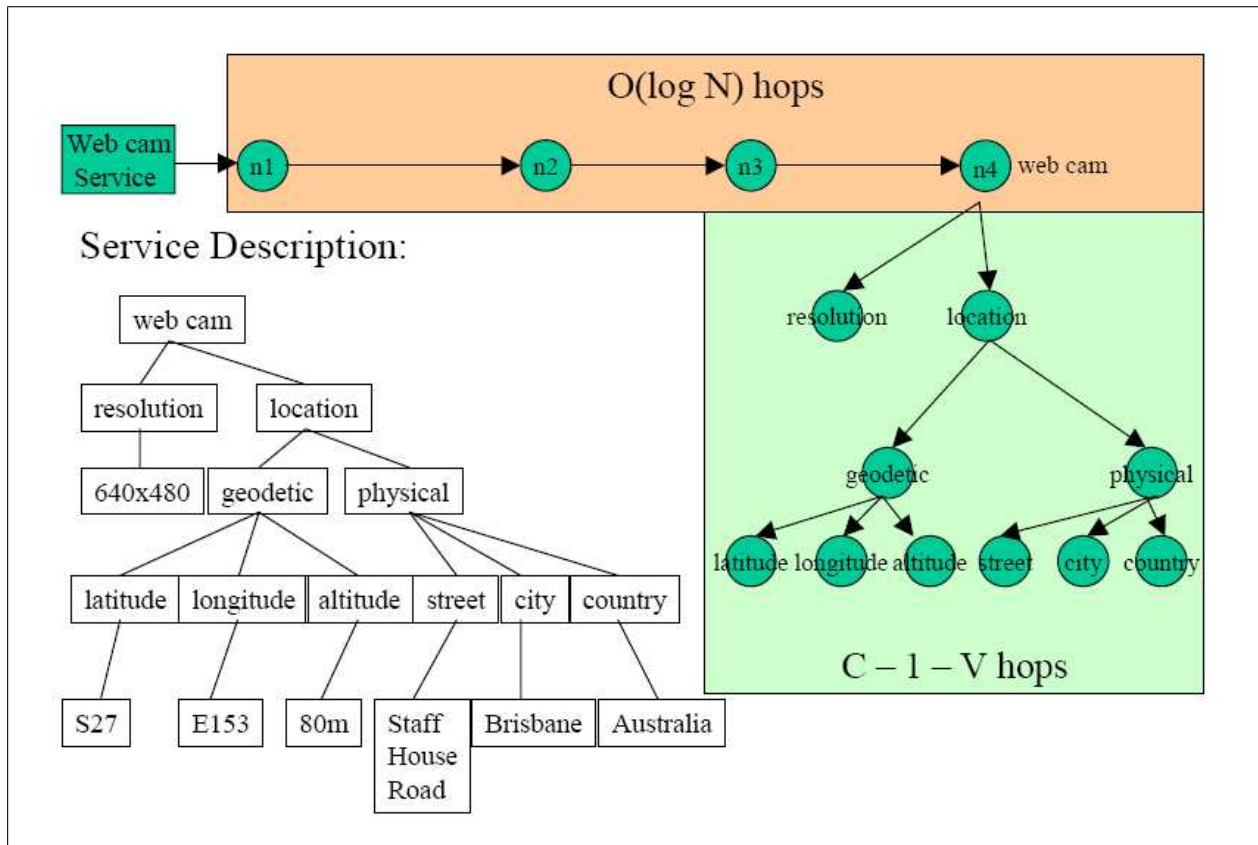


Fig. 3. The routing mechanism in the structured protocol layer. The root element of the description is used to locate the root of the appropriate resolver hierarchy.

to the appropriate nodes for resolution. On the contrary, the other routing layer is based upon the stochastic process of ant foraging and stigmergy. It involves establishing a *pheromone trail* by caching entire or partial descriptions along the route from a querying node to the node that resolves the query. This layer is appropriate for smaller, dynamic networks, as it is designed to adapt to changing network conditions. The ant foraging protocol degrades to a best-effort search in the case where the number of nodes on the network exceeds the maximum hop limit (more accurately, it degrades to a best-effort search in the case where the number of nodes *that do not have any knowledge of a matching resource* exceeds the maximum hop limit). This routing layer is designed to operate over a range of underlying transports. Some nodes in the network will execute both routing layers, thereby facilitating interaction between the wide-area and dynamic ad hoc networks. In both cases, the routing layers are concerned with locating nodes on which to store resource descriptions, and locating the appropriate nodes to perform query resolution.

The structured routing protocol, whose operation is shown in Figure 3, has been shown to have a worst case cost of  $O((\log N) + C - 1 - V)$  where  $N$  is the number of resolvers in the structured environment,  $C$  is the total number of components in the hierarchical description and  $V$  is the number of attribute values (leaf nodes) in the description [1].

The unstructured routing protocol has a stochastic element, and its cost is therefore much harder to quantify. However,

experiments indicate that in simple networks where nodes can propagate a query to any other node in the network with equal probability, the number of nodes that are aware of any particular description at a given point in time (the expected *coverage*) can be described via the following equation:

$$\chi_{q+1} = \lceil \frac{\chi_q}{n} + \sum_{i=2}^{n+1-\chi_q} i \times \left( \prod_{k=0}^{i-2} \frac{n - \chi_q - k}{n - k} \right) \times \frac{\chi_q}{n - i + 1} + \chi_q - 1 - \chi_q \delta \rceil \quad (1)$$

where  $\chi$  represents the number of covered nodes,  $n$  is the total number of nodes in the network and  $\delta$  is a decay factor (the rate at which cached descriptions expire) [48]. The route length can be determined from the expected coverage. Figure 4 shows the relationship between route length and coverage in a network of 100 nodes for increasing rates of decay. The graph indicates that the average route length increases very slowly as the cache purging policy becomes more aggressive, while the overall storage costs (that is, the number of nodes that are aware of a particular resource) falls sharply as  $\delta$  increases.

#### D. Summary

The above overview of Superstring shows that it is a resource discovery protocol equipped to meet the challenges of contemporary computing environments. Its two routing protocols together cover an immense diversity of computing

**Cover and Route Length Attractors for Varying Decay  
(100 Node Network)**

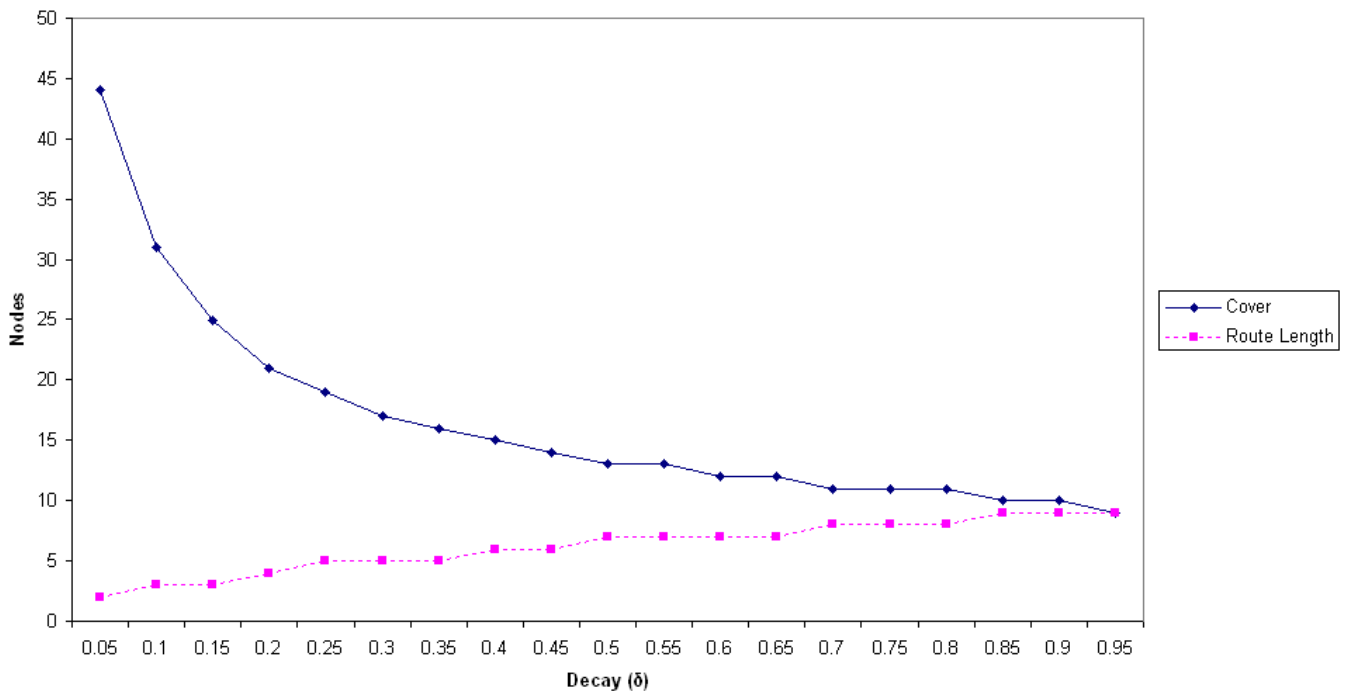


Fig. 4. Cover and route length attractors for increasing decay.

environments that can be found in the modern world. These routing layers are coupled with a single applications programming interface and resource description language to yield a resource discovery protocol that can be used across heterogeneous domains, and that scales upward and downward. The context-sensitive features of Superstring, which are missing from other resource discovery protocols, add enormous power to the protocol.

#### IV. CONCLUSION

This paper discussed the issue of service discovery in current and future computing environments. There is a number of different computing environments and therefore there also exists a large number of service discovery protocols which were designed for particular environments. This paper described the requirements of different computing environments, including grid computing and pervasive computing, and showed the extent to which current service discovery protocols, both well established and research-oriented protocols, meet these requirements. As pervasive computing becomes more widespread, service discovery protocols will increasingly need to work in large and heterogeneous domains.

The paper presented a solution for future service discovery in heterogeneous domains: Superstring is suitable for a wide range of domains. It behaves differently in a large, static environment than in a small, dynamic environment. Therefore it can meet differing requirements of particular environments while providing one unified interface for all applications

issuing service discovery queries. Ongoing research on this protocol will further increase its suitability for heterogeneous domains and increase its usability. Specifically, although a preference language has been defined to allow control over the number and ranking of items in a result set, it is often difficult to find a mathematical expression that produces the desired effect. However, Henricksen [49] describes a more flexible and powerful approach to modelling preferences in context-aware systems, which does not exhibit these difficulties. It remains to be seen whether resource discovery protocols such as Superstring can benefit from a similar preference modelling approach.

This paper provided an extensive survey of resource discovery protocols, which we hope the pervasive computing community can draw upon to meet the resource discovery requirements inherent in the new and exciting applications being developed for the increasingly dynamic world in which we live.

#### REFERENCES

- [1] Ricky Robinson and Jadwiga Indulska. Superstring: A scalable service discovery protocol for the wide-area pervasive environment. In *11th IEEE International Conference on Networks*, pages 699–704, Sydney, Australia, September 2003.
- [2] Ricky Robinson and Jadwiga Indulska. A complex systems approach to service discovery. In *Database and*

- Expert Systems Applications, 15th International Workshop on (DEXA'04)*, pages 657–661, Zaragoza, Spain, August 2004. IEEE Computer Society. ISBN 0-7695-2195-9. doi: <http://dx.doi.org/10.1109/DEXA.2004.3>.
- [3] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.
- [4] W3C Recommendation: SOAP Version 1.2 Part 0: Primer, June 2003. URL <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>.
- [5] Bluetooth SIG. Bluetooth Specification version 1.1, February 2001.
- [6] Ian Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd International Workshop on Peer-to-Peer Systems*, pages 118–128, February 2003.
- [7] Napster. The napster homepage, 2003. <http://www.napster.com/>.
- [8] Clip2. The gnutella protocol specification v0.4, 2003. URL [http://www9.limewire.com/developer/gnutella\\_protocol\\_0.4.pdf](http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf).
- [9] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, January 2002. ISSN 1089-7801. doi: <http://dx.doi.org/10.1109/4236.978368>.
- [10] Sun Microsystems, Inc. JXTA v2.0 Protocols Specification, June 2004. URL <http://www.ietf.org/internet-drafts/draft-duigou-jxta-protocols-05.txt>.
- [11] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Comput. Netw. ISDN Syst.*, 47(4):445–487, 2005. ISSN 0169-7552. doi: <http://dx.doi.org/10.1016/j.comnet.2004.12.001>.
- [12] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, February 2003. ISSN 1063-6692. doi: <http://dx.doi.org/10.1109/TNET.2002.808407>.
- [13] Yaron Y. Goland, Ting Cai, Ye Gu, and Shivaun Albright. Simple Service Discovery Protocol/1.0. IETF draft specification, October 1999.
- [14] Bluetooth SIG. Bluetooth Specification version 1.1, February 2001.
- [15] The Salutation Consortium. Salutation architecture specification (part 1) v2.0c, June 1999.
- [16] 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, December 2002.
- [17] Sumi Helal, Nitin Desai, Varun Verma, and Choonhwa Lee. Konark - a service discovery and delivery protocol for ad-hoc networks. In *Third IEEE Conference on Wireless Communication Networks (WCNC)*, pages 2107–2113, New Orleans, USA, March 2003.
- [18] Choonhwa Lee, Abdelsalam Helal, Nitin Desai, Varun Verma, and Bekir Arslan. Konark: A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 33(6):682–696, November 2003.
- [19] Sung Ju Lee, William Su, and Mario Gerla. On-demand multicast routing protocol in multihop wireless mobile networks. *Mobile Networks and Applications*, 7(6):441–453, December 2002. ISSN 1383-469X. doi: <http://dx.doi.org/10.1023/A:1020756600187>.
- [20] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. DEAPspace: transient ad hoc networking of pervasive devices. *Comput. Networks*, 35(4):411–428, December 2001. ISSN 1389-1286. doi: [http://dx.doi.org/10.1016/S1389-1286\(00\)00184-5](http://dx.doi.org/10.1016/S1389-1286(00)00184-5).
- [21] Stuart Cheshire and Marc Krochmal. DNS-based service discovery. IETF Draft Specification, February 2004. URL <http://files.dns-sd.org/draft-cheshire-dnsextdns-sd.txt>.
- [22] Apple Computer, Inc. Rendezvous technology brief, October 2003. URL <http://images.apple.com/macosx/pdf/PantherRendezvous.TB-10232003.pdf>.
- [23] Dipanjan Chakraborty, Anupam Joshi, Tim Finin, and Yelena Yesha. Towards distributed service discovery in pervasive computing environments. Technical report, University of Maryland, Baltimore County, July 2004. URL <http://ebiquity.umbc.edu/v2.1/get/a/publication/114.pdf>.
- [24] W3C Recommendation: OWL Web Ontology Language Overview, February 2004. URL <http://www.w3.org/TR/2004/REC-owl-features-20040210/>.
- [25] W3C Recommendation: Resource Description Framework Primer, February 2004. URL <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.
- [26] Youyong Zou, Tim Finin, and Harry Chen. F-OWL: an inference engine for the semantic web. In Michael Hinchey, James L. Rash, Walter F. Truskowski, and Christopher A. Rouff, editors, *Formal Approaches to Agent-Based Systems*, number 3228 in LNCS. Springer-Verlag, November 2004.
- [27] Sun Microsystems, Inc. Jini Technology Core Platform Specification v1.2. Technical report, Sun Microsystems, Inc, 2001.
- [28] Charles E. Perkins and Harry Harjono. Resource discovery protocol for mobile computing. *Mobile Networks and Applications*, 1(4):447–455, September 1996. ISSN 1383-469X.
- [29] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *MobiCom '99: 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 24–35, Seattle, Washington, United States, August 1999. ACM Press. ISBN 1-58113-142-9. doi: <http://doi.acm.org/10.1145/313451.313462>.

- [30] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13 (7):422–426, July 1970. ISSN 0001-0782. doi: <http://doi.acm.org/10.1145/362686.362692>.
- [31] Paul Castro, Benjamin Greenstein, Richard Muntz, Parviz Kermani, Chatschik Bisdikian, and Maria Papadopouli. Locating application data across service discovery domains. In *MobiCom '01: 7th annual international conference on Mobile computing and networking*, pages 28–42, Rome, Italy, July 2001. ACM Press. ISBN 1-58113-422-3. doi: <http://doi.acm.org/10.1145/381677.381681>.
- [32] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Pervasive 2002 - International Conference on Pervasive Computing*, number 2414 in LNCS, pages 195–210. Springer-Verlag, August 2002.
- [33] M. Wahl, T. Howes, and S. Kille. IETF RFC 2251: Lightweight Directory Access Protocol (v3), December 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, April 2002.
- [35] ISO/IEC. Open Distributed Processing - Trading function: Specification. ISO/IEC 13235-1:1998, February 2002.
- [36] OMG. Trading Object Service version 1.0, May 2000. URL [http://www.omg.org/technology/documents/formal/trading\\_object\\_service.htm](http://www.omg.org/technology/documents/formal/trading_object_service.htm).
- [37] OASIS UDDI Specification TC. UDDI Version 3.0.1, October 2003. URL [http://uddi.org/pubs/uddi\\_v3.htm](http://uddi.org/pubs/uddi_v3.htm).
- [38] Michael Schwartz. The networked resource discovery project. *IEEE Computer Society Technical Committee Newsletter on Operating Systems and Application Environments*, 5(2):10–11, June 1991.
- [39] Stanley Milgram. The small world problem. *Psychology Today*, pages 60–67, May 1967.
- [40] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physical Review E*, 64(046135), October 2001.
- [41] Nima Sarshar and Vwani Roychowdhury. Scale-free and stable structures in complex ad hoc networks. *Physical Review E*, 69(026101), February 2004.
- [42] E. Guttman, C. Perkins, J. Veizades, and M. Day. IETF RFC 2608: Service location protocol, version 2.
- [43] Weibin Zhao, Henning Schulzrinne, and Erik Guttman. mSLP - Mesh Enhanced Service Location Protocol. In *IEEE International Conference on Computer Communications and Networks (ICCCN'00)*, pages 504–509, Las Vegas, USA, October 2000.
- [44] Ajay Chander, Steven Dawson, Patrick Lincoln, and David Stringer-Calvert. NEVRLATE: Scalable Resource Discovery. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 352–358, Berlin, Germany, May 2002. IEEE Computer Society.
- [45] Ricky Robinson and Jadwiga Indulska. A context-sensitive service discovery protocol for mobile computing environments. In *The Fourth International Conference on Mobile Business*, Sydney, Australia, July 2005. IEEE Computer Society.
- [46] Guanling Chen and David Kotz. Context-sensitive resource discovery. In *First IEEE International Conference on Pervasive Computing and Communications*, pages 243–252. IEEE Computer Society Press, March 2003.
- [47] Peter Fishburn. Preference structures and their numerical representations. *Theoretical Computer Science*, 217(2): 359–383, 1999. ISSN 0304-3975. doi: [http://dx.doi.org/10.1016/S0304-3975\(98\)00277-1](http://dx.doi.org/10.1016/S0304-3975(98)00277-1).
- [48] Ricky Robinson and Jadwiga Indulska. The emergence of order in random walk resource discovery protocols. In *Ninth International Conference on Knowledge-Based Intelligent Information and Engineering Systems (Special Session on Complex Adaptive Systems)*, LNCS/LNAI, Melbourne, Australia, September 2005. Springer-Verlag.
- [49] Karen Henriksen. *A framework for context-aware pervasive computing applications*. PhD thesis, School of Information Technology and Electrical Engineering, The University of Queensland, 2003.