A Resource Discovery Protocol for Modern Computing Environments

A thesis submitted for the degree of Doctor of Philosophy at the University of Queensland in March 2005

Ricky Robinson B.InfTech (Hons 1)

School of Information Technology and Electrical Engineering

2005

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original, except as acknowledged in the text, and that the material has not been submitted, either in whole or in part, for a degree at this or any other university.

Ricky Robinson

Brisbane, March 2005

Acknowledgements

Everything I have accomplished is, in part, attributable to others, and this thesis is no exception.

At the end of each of his books, Andrew Tanenbaum states that, to his Ph.D. students, he “resembles a mother hen”. My advisor, Associate Professor Jadwiga Indulska, has been no less to me. In taking me under her wing, she provided me with all the support a Ph.D. candidate could ever need. Without her expert guidance and far-reaching knowledge of pervasive computing, this thesis would not have been possible.

If one person can be singled out for setting me on this course of pervasive computing research, it is Andry Rakotonirainy, my honours supervisor. He supplied inspiration during the initial stages of my Ph.D. candidature, and always challenged me to think outside the square. His enthusiasm for pervasive computing is something that I hope to replicate throughout my career.

The CRC for Distributed Systems Technology (DSTC) furnished me with office space, a workstation and support on numerous other levels. The DSTC has nurtured an environment that is most conducive to research. I would like to thank all the friendly staff and students at the DSTC for creating such a pleasant atmosphere.

I gratefully acknowledge Sun Microsystems Research Labs, and in particular Grzegorz Czajkowski and Laurent Daynès, for giving me the opportunity to work on one of their exciting research projects, and for helping to prepare me for what was to come later in my Ph.D.

This thesis has undergone several revisions. I would therefore like to express my gratitude to those who have checked, revised and edited the chapters of this dissertation, including Jadwiga Indulska, Karen Henricksen, Nigel Robinson, David Robinson, Ryan Wishart and Ted McFadden.

I am eternally grateful to my friends within and without the University of Queensland - including Ryan, Sasi, Qiang and everybody on the UQIT2K mailing list - for their excellent advice and for providing endless banter and timely comic relief. I would especially like to thank Karen for her love and encouragement. This thesis was infinitely easier to complete because of them.

Last, my heartfelt gratitude goes to my family, to whom I am deeply indebted. The unfailing love, support and understanding of my parents, David and Nisha, and my brother and lifelong friend, Nigel, made the many challenges of postgraduate study easier to overcome. This thesis is dedicated to them.

List of Publications

1. Ricky Robinson and Jadwiga Indulska. The Emergence of Order in Random Walk Resource Discovery Protocols. In *The Ninth International Conference on Knowledge-Based Intelligent Information and Engineering Systems (Special Session on Complex Adaptive Systems)*, pages 827–833. Springer-Verlag, volume LNCS 3683, September 14-16, 2005, Melbourne, Australia.

2. Ricky Robinson and Jadwiga Indulska. A Context-Sensitive Service Discovery Protocol for Mobile Computing Environments. In *The Fourth International Conference on Mobile Business*, pages 565–572. IEEE Computer Society, July 11-13, 2005, Sydney, Australia.

3. Ryan Wishart, Ricky Robinson, Jadwiga Indulska, and Audun Jøsang. SuperstringREP: Reputation-enhanced service discovery. In *Computer Science 2005 (CRPIT)* volume 38, Australian Computer Science Communications, 27(1):49–57. Newcastle, Australia, 2005. Australian Computer Society Inc.

4. Ricky Robinson and Jadwiga Indulska. A complex systems approach to service discovery. In *Proceedings of the Database and Expert Systems Applications, 15th International Workshop on (DEXA'04)*, pages 657–661. IEEE Computer Society, 2004. ISBN 0-7695-2195-9.

5. Ricky Robinson and Jadwiga Indulska. Superstring: A scalable service discovery protocol for the wide-area pervasive environment. In *Proceedings of the 11th IEEE International Conference on Networks*, pages 699–704. Sydney, Australia, 2003.

6. Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, and Karen Henricksen. Experiences in Using CC/PP in Context-Aware Systems. In *The*

4th International Conference on Mobile Data Management, volume LNCS 2574, pages 247–261. Springer-Verlag, 2003.

7. Ricky Robinson and Andry Rakotonirainy. Multimedia customisation using an event notification protocol. In *Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS02)*, pages 549–554. IEEE Computer Society, July 2002.

Abstract

Arguably, the world is becoming one large pervasive computing environment. Our planet is growing a digital skin composed of a wide array of sensors, hand-held computers, mobile phones, laptops and web services. Objects like clothes, crates and coffee mugs are also being integrated into the pervasive computing environment via digital identification tags and sensors. These devices, objects and services are deployed in groups, forming dynamic communities of interacting entities. These communities can be linked together to form a wide-area pervasive computing environment, allowing processes in one group to interact with services in another.

This brief characterisation paints a highly heterogeneous picture of the modern computing environment. The entities constituting this digital landscape have vastly different computational capabilities and roles; there is a diverse range of underlying communication protocols and network topologies; and the number of applications that execute in this environment is limited only by human imagination and necessity. Such a broad vision of tomorrow's computing environment gives rise to problems of scalability with respect to the number of entities within it and the differing capabilities of the devices. Heterogeneity and scalability are two very important problems in the domain of pervasive computing research.

Resource discovery protocols, the primary topic of this thesis, are crucial to the operation of any pervasive computing environment because the set of services with which an application may interact constantly changes. This is due to the mobility of users and their devices, the mobility of services, the failure of service instances, network disconnection and changes to the context of an application. Resource discovery protocols are one mechanism by which applications can stay informed about the current state of the environment.

The research contributions afforded by this thesis can be summarised as fol-

lows.

First, it provides a comprehensive critical literature survey of existing resource discovery protocols, which yields a set of elements necessary to construct a resource discovery protocol for pervasive computing environments. A survey of routing protocols, description languages, resource discovery bridges, context-sensitive protocols and preference models is also presented, as these areas can be drawn upon to inspire the design of the required resource discovery components.

Second, it shows that the modern computing environment can be characterised as a complex system. Concepts from complex systems theory are then used to construct a resource discovery protocol that overcomes the problems of scale and heterogeneity. Specifically, using network theory, it describes a mechanism for decreasing the required number of lookups in a structure known as a distributed hash table. The thesis also proposes the use of stigmergy, a naturally occurring method of indirect communication, in service discovery protocols.

Third, it presents context-sensitive extensions to this protocol that can be used to endow applications with context-aware behaviour in dynamic, ad hoc computing landscapes. A context-sensitive protocol will be able to automatically adapt queries to the current situation of the application or user. Context-sensitive preferences, which enable query results to be ranked according to the current context, are also introduced. These context-sensitive features allow applications to act with a greater deal of autonomy in changing environments.

Finally, the thesis discusses a prototype implementation of the core protocol features, and provides a detailed analysis of the protocol.

Contents

List of Figures

CHAPTER

1 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 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 thesis is concerned.

1.1 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 thesis, 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, 2].

A range of resource discovery protocols are in use or have been proposed for use in various computing environments [3–11]. Typically these solutions are focused at specific kinds of environments, such as local dynamic environments [12] or structured wide-area environments [6]. In the increasingly connected world, these narrow application domains are becoming less relevant. This thesis contends that a more holistic approach to resource discovery is required.

1.2 Motivation

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. 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 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 [13]. However, these obstacles must first be hurdled by the resource discovery protocols themselves. It is these challenges which motivate the research documented herein.

1.2.1 Scalability

When a broad view of pervasive computing environments is taken, as is the case in this thesis, scalability becomes a major concern. In the modern computing environment, the issue of scalability takes on several dimensions:

- the number of discoverable resources in the environment is enormous;
- there is a large range of different *types* of resources;
- the number of nodes in the network is extremely large, yet in some scenarios, small groups of devices are isolated; and

- the computational capacity of some devices, such as supercomputers, is high, while other devices, such as mobile phones and sensors are computationally resource poor.

In a typical modern office, one can usually find a computer, which is composed of a processor, storage, a screen and a host of applications software. Sometimes peripheral devices, such as removable storage, web-cams and head-sets, are attached to the computer. Often, the room will be equipped with air-conditioning controls and a kettle. The resident of the office might own a mobile phone. Library books augmented with bar-codes or RFID tags could be strewn over the desk. Each of these items can be considered a pervasive computing resource. This situation is replicated across hundreds of offices within a building, thousands within a city and millions (or billions) globally. Of course, not every resource should be advertised to the world at large. Nevertheless, the number of resources available is so great that no single entity can have full knowledge of the resource set. Furthermore, there is a plethora of resource types that may be discovered, each with its own unique attributes. A resource discovery protocol that hopes to operate in the modern computing environment must scale to large numbers of resources and nodes, and provide facilities to describe diverse resource types. Yet, it must be lightweight enough to scale to resource poor devices with low computational power and little storage. The protocol should distribute work and data fairly among the participating nodes.

1.2.2 Heterogeneity

Heterogeneity is a key characteristic of pervasive computing environments. As with scalability, heterogeneity has several dimensions:

- underlying communications protocols, network topologies and device behaviours give rise to extreme heterogeneity in the network, due to the use of a variety of protocols such as TCP/IP [14, 15] and Bluetooth L2CAP [16], and due to the co-existence of wired and wireless technologies;

- mobile ad hoc networks exhibit a different set of behaviours to fixed networks, such as a greater likelihood of disconnection and frequent changes to the set of nodes surrounding a device;

- applications are diverse in their resource needs in terms of the location of resources and the specificity with which resources must be described; and

- users of the same application often have different requirements.

1.2.3 Mobility

The presence of mobile elements in the environment means that the set of resources available to an application or user constantly changes. Mobility necessitates the ability of devices to interact with other devices opportunistically. Devices within a mobile ad hoc network (MANET) frequently suffer from disconnection due to intermittent network coverage. Typically, these devices are battery powered, implying that the applications executing on them must be power-aware so as not to drain the device of excessive amounts of energy. A resource discovery protocol for the modern computing environment must therefore be power-aware and behave gracefully in the event of disconnection.

1.2.4 Autonomy

Pervasive computing applications are intended to support users in their tasks in a non-intrusive manner. They are expected to operate with little initial configuration

from systems administrators and with minimal user intervention. If applications signalled an alert every time the set of available resources changed, users would quickly become overwhelmed. In mobile computing environments, it is not possible for systems administrators to statically configure devices to use particular resource instances since the set of available resources continually changes. This is the primary reason resource discovery protocols are required in pervasive computing environments. Additionally, applications require some mechanism to be notified of changes to the environment so that they can adapt accordingly. Given that resource discovery protocols are responsible for locating available resources, it follows that applications can use them to achieve a level of autonomy. By placing appropriate facilities within a resource discovery protocol itself, applications can be automatically notified of changes to the environment.

1.3 Thesis Statement

Existing resource discovery protocols do not sufficiently address the challenges arising from the modern computing environment - a composition of wide-area networks, local-area networks, infrastructure-based networks, ad hoc networks and overlays, and a multitude of resources, applications and users. Today's service discovery protocols are built for particular types of computing environment. While bridges or gateways allow some degree of interoperability between resource discovery protocols executing in disparate domains, each bridge maps a single pair of protocols. This approach is hampered by semantic differences in resource discovery protocols and by the large implementation effort required to allow any two protocols to inter-operate. A better approach is clearly needed.

It is hypothesised that it is possible to develop a single resource discovery protocol to meet the above challenges by drawing upon methods from distributed

computing and complex systems theory. This thesis demonstrates how this may be achieved and documents the way in which the above hypothesis is tested.

Specifically, in this thesis, a single resource discovery protocol capable of operation within manifold environments is designed, tested and analysed. Furthermore, as previously argued by others [17], a resource discovery protocol is in a unique position to provide context-sensitivity to applications when resource advertisements are viewed as context information. Thus an additional contribution of this thesis is to augment the service discovery protocol with context-sensitive features that can be utilised within infrastructural and ad hoc environments alike.

1.4 Approach

The research presented in this thesis progressed in several clearly defined stages.

1.4.1 Defining the Modern Computing Environment

In developing a service discovery protocol, the target environment must first be defined and characterised. This characterisation provides the set of challenges that a resource discovery protocol must overcome if it is to be used in the target environment. Taking a broad view of pervasive computing environments necessitates a detailed survey of many of the computing landscapes found in the world today, from grids to ad hoc networks to personal-area networks.

1.4.2 Identifying Resource Discovery Components

There are many resource discovery protocols already in use, but they fall short of the requirements yielded by the characterisation of the modern computing environment. However, a careful study identifies a general set of elements common

to all resource discovery protocols. This study thereby provides a “starting point” for the design and implementation of any new resource discovery protocol.

1.4.3 Searching for Solutions

After characterising the target environment and isolating the core elements of a resource discovery protocol, the process of finding a solution to the core problems identified in Section 1.3 can begin. The classification of the modern computing environment as a complex system suggests the possibility of applying complex systems theory to the problems of scalability, heterogeneity and mobility in pervasive computing.

1.4.4 Solving the Core Problems

A resource discovery protocol is designed using the traditional engineering concepts of abstraction and division in parallel with the ideas formed through the above process. Separate routing layers are developed to cater to static (structured) and dynamic (unstructured) environments. The unstructured layer, in particular, draws upon complex systems theory and biomimetics. This division of routing layers is hidden from applications and users by providing a single application programming interface and resource description language.

1.4.5 Context Augmentation

Having formulated solutions to the core problems presented by the modern computing environment, the resource discovery protocol is extended to include context-sensitive features and preferences. It is shown that with minimal additions to the resource description language, and additional query and advertisement behaviours, powerful, context-aware applications can be developed for stable and ad

hoc environments.

1.4.6 Prototyping and Analysis

Finally, the core aspects of the resource discovery protocol are analysed, again drawing upon complex systems theory and traditional analysis techniques. Mathematical models are developed for the structured and unstructured routing layers, and these are compared with data from experiments performed with a prototype implementation. After verifying the correctness of the mathematical model against the experimental data, the model can be used to predict performance in a range of different network sizes.

1.5 Thesis Structure

This thesis highlights a number of problems in the area of service discovery in pervasive computing environments (for a broad definition of pervasive computing environments), and develops and evaluates solutions to several of these problems.

The remainder of this document is organised as follows. Chapter 2 contains an overview of the current state of resource discovery and a critical literature survey. Chapter 3 provides necessary background information on complex systems theory and then gives an overview of the ways in which complex systems approaches have helped to solve the problems identified in Chapter 2. Following from this, Chapter 4 describes the design of the Superstring resource discovery protocol that provides:

- efficient operation in large networks with large numbers of resources or services;
- operation within highly dynamic networks; and

- scoped queries that dynamically weaken search criteria if no results are found.

Context-sensitive extensions to Superstring are described in Chapter 5. In Chapter 6, details of a prototype implementation of Superstring are discussed, and simplifications and implementation constraints are clarified. Chapter 7 evaluates the prototype, and suggests improvements that might enhance Superstring's performance. Finally, Chapter 8 summarises the contributions of this research, and highlights some areas of future work in the field of resource discovery.

CHAPTER

2 Related Work

This chapter presents a survey and critical analysis of existing resource discovery protocols, highlighting their shortcomings. It also discusses related topics, such as routing protocols, resource description languages and context-awareness, which are relevant to the development of a resource discovery protocol fit for the modern computing environment.

Before engaging in a detailed analysis of existing resource discovery protocols, it is imperative that the range of environments in which they are required 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 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 2.1 describes present-day computing environments in detail.

Section 2.2 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 provide direction for the research reported in the chapters to follow.

Sections 2.3 to 2.6 discuss related work that is important to the construction of a new service discovery protocol. The concepts reviewed include routing protocols, description languages, context-awareness and bridging of resource discovery protocols.

Finally, Section 2.7 reiterates the identified shortcomings in existing protocols, and summarises the set of requirements extracted from the critical analysis.

2.1 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.

2.1.1 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 In-

ternet, stability is replaced by varying degrees of instability as end nodes connect and disconnect, and indeed, entire institutions become connected. 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 (HTTP [18]) and electronic mail (SMTP [19], IMAP [20] and POP [21]). The web has matured rapidly, and is now capable of supporting electronic commerce and other service-oriented applications. The web and electronic mail are both client-server applications. Server processes are long-lived and usually 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, and the web is a typical example of this. In recent times, frameworks such as Microsoft .NET [22] and Sun's J2EE [23] have broadened the client-server paradigm. In these frameworks service objects, whose operations are often invoked via HTTP using SOAP [24], may invoke other web services and objects. So a web service may also act as a client in some scenarios. Among the resources and services that a user or client application may need to discover in this environment are web services and application level proxies. In recent times, peer-to-peer file sharing applications have become popular on the Internet. Section 2.1.4 provides an overview of resource discovery in peer-to-peer environments and applications.

2.1.2 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 predom-

inantly of point-to-point links, while local-area networks generally utilise shared medium protocols such as IEEE 802.3 [25] and IEEE 802.11 [26]. 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 [27]. Thus, these spaces are moving toward pervasive environments (see Section 2.1.6 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.

2.1.3 The Grid

Grid computing environments offer a way to harness unused CPU cycles and storage. Present-day grids, such as NASA's Information Power Grid [28] and Australia's GrangeNet [29], consist of relatively small numbers of very high perfor-

mance 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 [30]. 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 be-

haviour 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.

2.1.4 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 [31, 32]. 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 [33], Gnutella [34], and Freenet [35]. The first two have gained notoriety for the 'swapping' of MP3 [36] 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 [37], 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.

2.1.5 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 [27] and IEEE 802.11 [26]. 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, such as MOTES [38] 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, retrieve the recipient's e-mail address from the address book stored on the handheld computer, send the e-mail via an access point attached to a LAN, and 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 lo-

cal 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 indexes 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.

2.1.6 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. Contrary to the focus of much pervasive computing literature, the pervasive computing environment is *not* limited to the immediate computing environment surrounding a user. Indeed, it has been suggested that the world will become a single, large pervasive computing environment encompassing all manner of applications and computing services [39]. Grid computing environments, for example, may be merged with pervasive computing environments to provide computational resources to lightweight 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 the system scale, dynamic changes in available services, and a rich variety of queries.

2.1.7 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 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.

semi-structured 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 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 2.1 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

Unstructured	Semi-structured	Structured
Pervasive	Pervasive	Pervasive
P2P	P2P	P2P
Ad Hoc	Local-Area	Local-Area
		Internet and Wide-Area
		Grid

Table 2.1: Classification of environments

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 [40], 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. Chord is described in further detail in Section 2.3.

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

2.2 Existing Resource Discovery Protocols

Following is a discussion of the suitability of existing protocols to the three broad environments outlined above. The protocols falling into each category are compared and contrasted.

2.2.1 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) [4] 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 [41]. 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) [5] 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 [16] link layer protocol. SDP is therefore suitable only for very small ad hoc groups of wireless devices.

Service discovery in Salutation [42] 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 [43]. 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 [11], 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 [44]). 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 [45].

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 (which uses a WSDL-like notation [46]) 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. WSDL is described in Section 2.4.

The DEAPspace [47] 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 proactively 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 [48]; 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.

DNS-SD (Domain Name System - Service Discovery) [49] provides service discovery capabilities for Apple's Rendezvous technology [50]. A Rendezvous enabled device first assigns itself an IP address from the link-local range using the ZeroConf draft IETF standard [51]. 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) [52] 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 [53] 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 [33] and Gnutella [34]. JXTA [37] 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.

Chakraborty et al. adopt the use of the Web Ontology Language (OWL) [55] to describe resources in mobile and pervasive computing environments (previously having used DAML+OIL [56], the forerunner of OWL, to describe resources [57]). 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) [58], 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 [59, 60], 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 supertype 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 [60], OWL becomes very powerful. But for the moment the execution of such an en-

gine 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.

2.2.2 Semi-Structured Environments

Jini [8] 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 [61] 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 [62] 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 [63] and a set of keywords. Queries are based on URNs [64]. 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 [18] or NFS [65]), 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 regis-

trations 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) [6] 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 [66] 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 [7] and VIA* [67] 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.

2.2.3 Structured Environments

INS/Twine [3] consists of a core of peer-to-peer resolvers. These resolvers are implemented on top of the Chord distributed hash table protocol (DHT), briefly described in Section 2.3.2. Descriptions and queries utilise the same format as the Intentional Naming System [68]. 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 [69] is a simplified (lighter weight) version of the X.500 [70] 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. Al-

though 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 [71] 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 [72], 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 [73] 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 repre-

sented 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 *businessService*s. 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 [74] queries, though they are marked up using XML. Clients direct queries at a particular node within a registry.

In the approach taken by Schwartz [75], 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 largely on the Small World phenomenon [76],

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 [77, 78]. The Small World phenomenon, its relationship to power-law networks and its consequences in service discovery are further explored in Chapter 3.

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.

2.2.4 Unclassified Protocols

The unclassified protocols can operate in structured, semi-structured and unstructured environments depending upon their configuration. Though they span these multiple environments, they do not solve the problem of operation in the all-encompassing, heterogeneous pervasive environment for several reasons, which are outlined in the following discussion.

The Service Location Protocol (SLP) [79, 80] 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 [81] to specify attributes of interest. It is possible for DAs to contain inconsistent information about available resources. To address this, Mesh Enhanced SLP [82, 83] 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 [84] 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.

2.2.5 Summary

This survey has covered a wide array of resource discovery protocols, and analysed their strengths and weaknesses. It is clear that each of the protocols reviewed is targeted toward particular computing environments. No protocol is suitable for 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. A resource discovery protocol 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. 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.

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).

The first two elements in this list are investigated further in the following sections, since there are other fields of computer science that may provide insights into these areas. Message formats and APIs are not covered further because their design is specific to a given architecture (in this case, resource discovery).

2.3 Routing Protocols

In this section, various routing algorithms are surveyed for their suitability to resource discovery in diverse environments. Address-based and content-based routing protocols are considered.

2.3.1 Routing Algorithms for Unstructured Networks

This section examines various existing routing protocols for ad hoc networks. Guiding the selection of a routing layer for resource discovery in unstructured

networks are several environmental characteristics:

1. high levels of mobility, including node arrivals and departures;

2. target node(s) are unknown - content is more relevant than node addresses; and

3. lightweight devices.

Discussion

Flooding is a search technique often used within unstructured networks. For instance, this technique is used by Gnutella [34] to route queries for files through the overlay network formed by Gnutella peers. A querying node issues a query to each of its neighbours, and the neighbours propagate the query to each of their neighbours and so forth. On each hop, a hop counter in the query message is decremented. When the counter reaches zero, the query message is propagated no further. In general, query hits are routed along the reverse path to that of the query. Retrieval or utilisation of a matching resource usually occurs via a separate mechanism. In Gnutella, for example, when a querier receives a query hit message, it makes a TCP/IP connection directly to the node which contains the matching file. The major problem with flooding is its inability to scale to large networks. In flooding-based query routing, each query generates a message for each neighbour of each node it passes through, thereby often producing redundant messages.

The problems associated with routing in mobile ad hoc networks are well-known. There are many address-based routing protocols designed to overcome these challenges. Three of the better known protocols in this class of routing protocols are Dynamic Source Routing (DSR) [85], Ad Hoc On-Demand Distance Vector routing (AODV) [86], and Greedy Perimeter Stateless routing (GPSR) [87].

DSR uses source routing, whereby the route to be taken by a packet is inserted in the packet header. In ad hoc networks, the route from one node to another can change constantly. Therefore, the idea of route discovery is introduced. When a node wishes to send a packet to a particular destination, it checks its route cache to see if it is already aware of a known route to that destination. If no matching route is found in the cache, the sender performs route discovery, which involves broadcasting a route request packet to each neighbour. This packet is propagated until it reaches a node which knows a route to the destination. Each node through which the route request packet traverses appends its address to the *route record* field of the packet. A route reply packet is generated by any node that knows of a route to the destination. The route reply packet is returned by checking the cache for a route to the initiator. If no route is found, the route record in the route request packet is reversed, and this route is used. If route errors are encountered, a route error packet containing the two nodes involved in the faulty hop is returned to the sender, and the sender must truncate all cached routes involving that hop. DSR is costly in terms of the data that must be cached at each node. It is also costly with respect to packet overhead, since for networks of large diameter, the route inserted into each packet could be very long. In highly dynamic networks, DSR suffers from high protocol overhead (due to route discovery packets) as shown in [87].

AODV shares much in common with DSR. However, while DSR inserts the full route to be taken in each packet header, AODV keeps source and destination pointers at each node. In AODV, routes between source and destination nodes are created dynamically. Routes are established via the use of route request packets (RREQ) and route reply packets (RREP). When a node wishes to establish a route to a destination, it broadcasts a RREQ packet. Each node that receives a RREQ packet stores a backward pointer to the source. The destination node, or nodes

aware of a route to the destination, reply with a unicast RREP packet, which is routed back to the source using the backward pointers previously created. Each node through which the RREP packet traverses stores forward pointers to the destination. The routing protocol is soft-state, so the route remains intact as long as data packets are being sent along the route. If at any time the route is broken (by link failures or node mobility), the source tries to re-establish the route using the above procedure. AODV improves on DSR by reducing packet overhead, though it still requires packets to be broadcast during route discovery.

GPSR forwards packets based on geographic location (whether that location information is gathered from the Global Positioning System in an outdoor setting, from ultra-sonic beacons in an indoor setting, or from some other means entirely). In general, packets are forwarded to the neighbour whose distance to the destination is shortest (making the algorithm *greedy*, since only local information is utilised in the forwarding decision). When a greedy decision is impossible, because the current node's distance to the destination is shorter than any of its neighbours, the packet is routed using an algorithm based on planarised graphs (a graph is planar if no two of its edges cross). This algorithm allows packets to traverse the perimeter of void areas (areas in which no nodes exist) by allowing the packet to momentarily move further away from the destination. The major benefit of this geographically-based algorithm is that there is little protocol overhead. The only non-data packets required are beacon messages, which are broadcast periodically so that each node can keep track of its neighbours, and often, these beacons can be piggybacked on data packets when promiscuous mode is enabled on each node's network interface. One of the shortcomings with GPSR is that it assumes all nodes lie within the same two dimensional plane. Another problem with GPSR is that greedy algorithms do not always result in optimal solutions (in the case of routing, the optimal solution is the shortest path between two nodes).

The problem of routing packets to *particular* nodes in an ad hoc network is an important one. However, it is not the only packet routing problem. Some classes of applications require that packets be routed to particular nodes based on the content contained within them. Data dissemination is the most common form of content-based routing in ad hoc deployments such as sensornets and vehicular networks. In general, the data produced by each node is named or classified. Nodes interested in a particular type of data flood the network with the names or classes of data they are interested in. Nodes that produce matching data then send this data to the requesting nodes along the reverse path. In a variation on this approach, nodes producing a certain class of data forward this data to a third party node that is responsible for this type of data. Consumer nodes can then query this third party node. An overview of a selection of these protocols is given below.

Directed Diffusion [88] is a mechanism for routing data through a sensor network to nodes that have indicated an interest in that data. In the simplest kind of directed diffusion, a node (a *sink*) broadcasts an *interest* to all of its neighbours. The interest is then propagated by the neighbours to their neighbours and so forth, such that the interest is flooded throughout the network of sensors. An interest consists of an event type (such as the detection of an explosive chemical), an event interval, a duration and a region. Upon receipt of an interest, a node will cache the interest or merge it with any identical interests previously cached. Each cached interest is associated with the neighbours from which the interest arrived. Any node within the region specified within an interest (a *source*) begins collecting and emitting data at the specified rate to the neighbours from which it received the interest. The neighbours propagate the data as determined by the cached interests. Thus, to begin with, a sink will receive data from all the neighbours to which it broadcast the interest. To allay the receipt of redundant data, a sink can reinforce a particular neighbour (perhaps the neighbour from which data is first received)

by refreshing its interest to only that neighbour. This neighbour also chooses a particular neighbour to reinforce, and so on. Interests cached at the other nodes will eventually expire due to lack of reinforcement. More advanced versions of directed diffusion are able to aggregate data within the sensor network, thereby further reducing packet redundancy. As with some of its address-based cousins, much of the overhead incurred by directed diffusion is in the initial flooding of packets.

Data-centric storage is another approach for disseminating data in sensor networks. It builds on the idea of distributed hash tables (DHTs), whereby all data hashes to a particular node in the network. Shenker et al. [89] describe an approach for layering a DHT over the above mentioned GPSR routing protocol. Event names or types (as described in the previous section), are used as keys into the hash table. Sensors producing events route their packets toward the node responsible for the event type using the DHT algorithm. Nodes interested in certain kinds of events direct their queries toward the sensors responsible for storing the data related to those events, also by using the DHT algorithm. In some specialised circumstances, data-centric storage based on DHTs can out-perform directed diffusion, since flooding is not utilised. However, reliance on the DHT algorithm limits the application of this protocol to environments in which nodes are stationary and reliable (DHTs do not perform well in highly dynamic networks). Furthermore, since all events of the same type hash to the same node in the network, this dissemination protocol may not scale to cases in which all sensors are tracking the same kind of event.

Summary

The surveyed routing approaches are designed for particular kinds of networks and applications. Resource discovery is inherently data-centric: node addresses are

not important; rather, it is the content of advertisements and queries that guides the route along which a query is propagated. However, the content-based approaches described above are not widely applicable enough to operate in pervasive computing environments. Directed diffusion utilises flooding and reinforcement, which is not scalable to larger networks (especially large networks of the highly dynamic kind), and data-centric storage is certainly not suitable for networks consisting of highly mobile nodes. Dynamic pervasive computing environments, which encompass sensor networks and mobile ad hoc networks, require a resource discovery protocol whose query routing mechanism can adapt to change without burdening the network with protocol overhead, which flooding based service discovery protocols such as RUBI [90] are likely to do. Such a protocol would not strive to find the shortest path to a resource, since the shortest path is short-lived, and the cost of deriving the shortest path is therefore amortised over only a small number of subsequent queries.

2.3.2 Routing Algorithms for Structured Networks

Algorithms for address-based routing in traditional networks are well known [91–93]. They are not covered here because resource discovery is a content-based procedure, as outlined in Section 2.3.1.

This section examines routing protocols used in content-addressable networks and peer-to-peer resource sharing networks.

Discussion

Content-based routing protocols such as Elvin [94] and Gryphon [95] have applications within traditional IP networks. These protocols use the publish/subscribe paradigm to map content to one or more consumers. Elvin is described here to exemplify this class of protocols. *Consumers*, who wish to receive notifications,

subscribe to an Elvin *router* using a highly expressive subscription language. Any notifications the router receives from *producers* that match the consumer's subscription are forwarded to the consumer (and to any other consumers for whom matching subscriptions exist). Elvin routers may be federated into clusters to improve performance and robustness. Clusters or individual routers may be linked with other routers and clusters of routers to form a wide-area messaging network. An Elvin router usually has a well-known address and must be reachable by all consumers and producers.

The use of a central index or directory for lookup is an often employed paradigm for resource discovery. CORBA [72], Jini [8] and Napster [33] are examples of technologies that utilise this technique. In these systems, queries and advertisements are not *routed* in the sense that they traverse many nodes before being resolved. Rather, each client in the network must be aware of the address of the central directory or have some way of discovering its address. Clients then contact the directory directly in order to make a query. Directories process the queries and respond to the clients. Prior to this, nodes offering a resource or service contact the directory in order to register this resource with the directory, which usually entails uploading a resource description. Central directories do not scale to large numbers of resources and clients. In some central directory approaches, the directories can be federated to cater to larger numbers of resources and clients. The intra-directory routing protocols may be based upon one of the other routing algorithms discussed in this section.

Distributed Hash Table (DHT) algorithms have come to prominence in research literature of late. However, they have still to be used in major deployments outside the sphere of research. Routing protocols in this class include CAN [96], Pastry [97], Tapestry [98] and Kademlia [99]. But perhaps the most well known DHT protocol is Chord [40]. Chord maps keys to nodes. Keys of length m bits

are generated by a hash function whose input is the data or the name of the data to be stored. Each Chord node has an ID that is taken from the same keyspace as the keys generated from the data. A key maps to the first node whose ID is greater than the key (the keyspace is circular). Each node maintains a *finger table* of size *m* (the same length as the keys) containing the IDs and addresses of a subset of the other nodes in the DHT. The table is built in such a way that each entry covers a successively larger portion of the keyspace. Specifically, if a node's ID is n, the i^{th} entry in the table contains the first node whose ID succeeds n by at least 2^{i-1}. Upon receiving a *lookup* message for a key *k*, the receiving node consults its finger table to find the node *r* whose ID most closely precedes *k* and refers the querier to this node. When *r* is queried, it carries out the same procedure. This process continues until a node is found whose immediate successor (the next node in the circular keyspace) has an ID greater than *k*. This node returns the address of its successor, and the key (along with its corresponding value) is stored at that node. DHT algorithms scale logarithmically with the number of nodes. However, the performance of these algorithms suffers in the face of network mutability. For this reason, DHT algorithms are best suited to fairly static environments. DHT algorithms can be used directly only by applications with very simple query requirements, since lookup is based on a single key. INS/Twine [3] builds on Chord, such that a number of keys are generated from a single resource description. Further information about Chord and distributed hash tables in general can be found in the excellent survey conducted by Balakrishnan et al. [100].

Summary

Of the algorithms described above, distributed hash tables are most closely matched with the goals of a resource discovery protocol for structured environments. They perform well in fixed networks, and they spread data evenly over the nodes in the

DHT (assuming the use of an appropriate hash function). However, for DHTs to be useful in resource discovery, resource descriptions, which may contain many attributes and values, must be mapped to a key or set of keys. A query, which can likewise contain many attributes and values, as well as expressions, must also be mapped to a key or set of keys. INS/Twine proves that this is achievable, though queries cannot contain expressions.

2.4 Description Languages

The resource discovery protocols examined in Section 2.2 each define their own resource description language in which to formulate advertisements and queries. This section analyses some additional description languages that could potentially be used within a resource discovery protocol.

2.4.1 Discussion

The Resource Description Framework (RDF) [58] is a World Wide Web Consortium (W3C) [101] standard for describing entities that can be retrieved from the Web and entities that can be identified in terms of Uniform Resource Identifiers (URIs) [102] even if those entities cannot be retrieved via the Web. In RDF, statements are made about an entity and its properties. Each statement consists of a subject (the entity being described), a predicate (a property of the entity) and an object (the value of the property). Values may take the form of literals, or URIs which identify another entity. In this way, a web of relationships between entities can be built. RDF forms the basis of languages and models such as the Web Ontology Language (OWL) [55] and Composite Capabilities/Preference Profiles (CC/PP) [103]. While RDF is very flexible in terms of the entities it can describe and the richness with which those entities can be described, it can become

awkward and ungainly to describe anything but the simplest resources. This is acceptable in an environment such as the World Wide Web where most devices are relatively powerful, and adequate facilities exist for creating RDF graphs; however, in pervasive computing environments, where computational resources are scarce, where the means to adequately create complex graphs may be unavailable and where it is difficult or impossible to represent dynamic resources using URLs, RDF is inappropriate.

CC/PP, an RDF vocabulary for describing the capabilities of mobile devices, proxies and user preferences, is already being used by some mobile phones. While these profiles are sometimes stored by the mobile devices themselves, they do not process the profiles. The work of processing is done by end servers (which deliver content) and intermediate proxies (which may add capabilities to a mobile device's profile). In this manner, CC/PP is ordinarily used as a mechanism to negotiate the format of the content retrieved by a mobile device, and these decisions take into account the ability of the device, the user's preferences, and any transcoding abilities of the proxies. These things make CC/PP well-suited to particular application scenarios, but unsuitable for pervasive computing in the general case. The same conclusion was reached by Indulska et al. [104] in attempting to utilise CC/PP to model context information in ubiquitous computing environments.

The Web Services Description Language (WSDL) [46] is yet another XML-based standard from the W3C. The purpose of WSDL is to enable web service providers to describe their services in abstract and concrete forms. At the abstract level, web services are described in terms of the messages they send and receive, without regard to the transmission formats of those messages. The *operation* element describes the sequence of messages to be exchanged by the sender and receiver, and the *interface* element groups these operations together. At the

concrete level, a *binding* describes the specific wire formats used by one or more of the interfaces, and an *endpoint* associates address and port information with a binding. Endpoints that implement a particular interface are grouped together by the *service* element.

WSDL is an excellent means for describing the messages exchanged by services, and for specifying the format of those messages. However, it does not provide elements for describing a service in more general terms. In other words, a client must already know that it wants to use a particular service instance because of the functionality it provides before downloading and parsing the corresponding WSDL document. Compare this with a UDDI description, which provides facilities for describing services in much higher-level terms, thereby making it possible to search for and discover relevant services without needing a priori knowledge about the operations provided by a service. For this reason, WSDL is a good choice for describing the manner in which a service can be invoked, but it is inappropriate for describing *what* a service does. WSDL is therefore a candidate for describing *name records* such as those used by INS/Twine.

Condor ClassAds [105] are utilised by the Condor grid computing system. The ClassAds description model is flat and uses attributes and values to describe resources within a grid environment. Job requests are also described by a ClassAd. Job request ClassAds are matched with grid resource ClassAds by a central matchmaking service. In this manner, tasks can be assigned to grid nodes with the required properties, such as correct CPU type, adequate memory and disk space. The attributes appearing within a ClassAd are specified by the *advertising protocol*, which is particular to each grid environment. Some advertising protocols may define *Constraint* and *Rank* attributes. These attributes indicate the compatibility and utility of a match respectively. The *Constraint* field consists of a Boolean expression over the other attributes in the ClassAd and the attributes in the ClassAd

being matched. Two ClassAds are incompatible unless the *Constraint* field evaluates to true for both of them. The *Rank* attribute consists of a numerical expression which is used to rate the utility of the compatible matches.

The simple nature of ClassAds may prevent it being used in environments that contain complicated resources, better described by a hierarchical model. Furthermore, because the ClassAd model is flat, it is difficult to apply query relaxation in the event of no compatible matches. However, the ClassAd model shows the manner in which utility functions can be used to rank the set of matches. This is an important concept in resource discovery, as it can potentially be used to conserve bandwidth by returning fewer results to a querier.

The final description language considered here is free-form text, which includes self-describing resources such as documents. A document, such as a web page, is a resource whose content provides the best description of it. Search engines such as Google [106] scour the Web, indexing each page they come across. Users can search the index by entering a few key terms into the search engine interface. Documents containing the search term are returned to the user. It is conceivable that a free-form text document could be attached to another object to serve as a description of that object. Search engines could then index the documents associated with objects.

This model works well for application where humans are “in the loop”. Often, keywords will match many documents, and a human is required to sift through the enormous set of results. In environments where applications are required to operate with a high degree of autonomy, such a model is inadequate. A more structured description model is required to enable greater autonomy, and to facilitate techniques such as query relaxation.

2.4.2 Summary

This is by no means an exhaustive survey of description languages. There are many ways to represent knowledge and to describe resources. However, many knowledge representation models can be translated into one or more of the description languages outlined here. For example, it is possible to represent semantic networks and frames [107] (two well-known knowledge representation models) using RDF.

A description language for a scalable resource discovery protocol must be lightweight enough to be used on small devices, yet powerful enough to describe resources in detail and to allow clients to select resources with fine-grained precision so as not to overwhelm them with too many results. Hierarchical and graph models appear to offer the greatest benefits, since they capture the relationships between the attributes comprising the resource description. Furthermore, hierarchical approaches make it possible to easily apply query relaxation.

2.5 Context, Preferences and Result-Ranking

This section discusses some approaches for integrating context, preferences and result-ranking with resource discovery. It also covers preference and result-ranking models from outside the realm of service discovery.

2.5.1 Discussion

Context can be defined as any information pertaining to the environment or circumstance surrounding an entity (such as a person, software component, application or device) [108]. Service discovery protocols can benefit from the use of context information in routing queries and in ranking results. Conversely, some

service discovery protocols can be used to provide context-sensitivity to applications, as in the case below.

Chen and Kotz [17] define a context-sensitive naming framework based on the Intentional Naming System (INS) [68]. Context sources emit low-level advertisements about a particular entity. These take the form of standard INS *name specifications*, which contain a hierarchy of attribute-value pairs. In many cases, a name specification is flat, so that all attributes in the specification are siblings. Applications use *graph specifications* to filter the output of context sources and to define context-sensitive names. A context-sensitive name contains a variable value in place of a literal one. The graph specification designates the way in which the variable is resolved. Normally, the variable is tied to the value of an attribute in another name specification. For example, an application wishes to receive pictures from a camera that is in the same room as Emma within the GP South building. The application creates a graph specification that contains the following context-sensitive name:

```
[sensor='camera', room=$emma–locator:room, building='GP
    South']
```

The graph specification also shows how to resolve the variable *$emma-locator:room*. In this case, it is resolved by taking the value of the *room* attribute from another name specifier which has been emitted by a context source. It might look like this:

```
[user='Emma', room='612', building='GP South']
```

Thus, when the context source that is tracking Emma emits an update about her location, the context-sensitive name specifier is automatically updated, and a new camera from which to receive pictures will be chosen.

In the Cooltown project [109], beacons are placed next to physical objects of interest. These beacons emit URLs (or identifiers that can be translated into URLs) to users' devices that come into close proximity. The URLs are used by

the devices to locate further information about the physical objects. Beacons can also be associated with places, in which case the emitted URL refers to a web page containing information about the other people and things in this place. In other words, the proximity-based discovery mechanism provides an entry point for obtaining further context information.

Another approach for integrating context-sensitivity with service discovery is suggested by Lee and Helal [110]. In their solution, they augmented the Jini [8] lookup service to consider dynamically changing attributes. These dynamically changing attributes are known as context-attributes. A context-attribute that is able to be resolved by the lookup service is called a local context-attribute. A context-attribute which must be resolved by a remote service is known as a remote context-attribute. An example of a local context-attribute is *Ping*, which gives a rough estimate of the network latency to the remote service. An example of a remote context-attribute is *Load*, whereby the lookup service must contact the remote service in order to obtain its current load status. Services define the context attributes, and the lookup service evaluates the attributes when required (that is, when a client performs a lookup). Clients remain unaware of the existence of the context-attributes, as it is argued that services know best which context-attributes are of the most importance. Lee and Helal define a simplistic result ranking mechanism, but this index is defined by the services, and *not* the clients or users, which means that the reasons for the ranking order remain hidden from users. For example, the designers of print services may decide that the most important metric for ranking is the user's proximity to a given printer. A user's query for printers will then return results in order of increasing distance to the user.

The Location Information Server (LIS) [111] supports location-aware applications. It utilises LDAP [69] to access the back-end location directory server. The LIS supports only location context, and is therefore limited in its scope.

Liu et al. [112] describe a framework for discovering resources and then selecting an appropriate candidate from the result set using quality of service (QoS) metrics. An advantage of their approach is that the solution operates in ad hoc mobile networks (that is, with no fixed infrastructure). However, their solution is confined to selecting resources based on only quality of service, and does not consider other potentially more relevant context information. Furthermore, it is unclear how the different kinds of QoS metrics are combined to yield an ordering over the candidates.

Henricksen [113] describes a flexible and powerful approach to modelling preferences in context-aware systems. Henricksen's model can handle ambiguous and incomplete context information, which is advantageous in pervasive computing environments and other distributed systems where gaining complete and accurate knowledge of the system is rarely achievable. Although this context and preference model is highly expressive, extensible and applicable to a wide range of ubiquitous computing scenarios, its reliance on infrastructural support may prohibit it from being used in ad hoc deployments of mobile devices.

2.5.2 Summary

This overview of context-sensitive service discovery protocols, preference models and result-ranking solutions identifies a further aspect of resource discovery that can be improved. Specifically, it should be possible to design a context-sensitive service discovery protocol that does not require users and programmers to learn additional abstractions such as graph abstractions in the case of Chen and Kotz [17]. Instead, the context-sensitive aspects of the protocol should be integrated into the service description language. Furthermore, a lightweight preference and result-ranking model needs to be defined, which can be used by powerful devices and resource-constrained devices alike.

2.6 Bridging Resource Discovery Protocols

While there currently exists a large body of work concerning service discovery protocols, relatively little effort has been invested in their interoperability. Furthermore, where attempts have been made to bridge two service discovery protocols, the approach has generally been to layer one protocol over another, thereby requiring both protocols to be installed on all participating devices and networks. True service discovery protocol bridges or gateways have thus far been products of the academic research domain.

2.6.1 Discussion

Prior efforts at protocol bridging include: Bluetooth SDP and Salutation [114], Salutation and SLP [115], Jini and UPnP [116], Bluetooth SDP and UPnP [117], and Bluetooth SDP and Jini [118]. Only two of these efforts can truly be considered to be bridging solutions. For example in [114], one of the proposed mappings is more correctly considered a transport extension to Salutation, using Bluetooth solely as a transport layer. The other proposed mapping in [114] uses the functional definition of the Salutation API, but passes Bluetooth SDP parameters through the Salutation operations. The Bluetooth Extended Service Discovery Profile (ESDP) [117] specifies two ways in which UPnP can be layered over Bluetooth. The first approach layers UPnP directly over the L2CAP transport. The second approach utilises the IP stack provided by the Bluetooth PAN or LAN Access profiles.

Allard et al. [116] have designed and implemented a bidirectional bridge between Jini and Universal Plug and Play. This bridge not only allows discovery of services between Jini and UPnP, but also enables service invocation via the use of service-specific hand-written proxies. This is a true bridge in that it allows a Jini

client to discover a UPnP service and vice-versa.

Kasper and Bührer [118] employed a semantic translation bridge. This solution was successful in allowing Bluetooth clients to discover Jini services, but was not capable of providing the reverse operation - that is, service advertisement/querying was unidirectional. A bridge is also provided between SLP and Jini, which allows non-Jini clients to access Jini services [119].

Significant problems are encountered when mapping to protocols with limited query capabilities such as SDP, or when bridging two completely decentralised protocols such as INS/Twine [3] and VIA [7], since it is impossible to introduce bridges in a completely transparent fashion, as discussed below.

Mappings between protocols with and without service offer leases (or other soft-state mechanisms) must ensure that the valid service offer set remains consistent between both domains. Therefore none of the existing bridging solutions can claim to successfully bridge the two involved domains, as the mapping of operations and descriptions always incurs some losses due to differences in syntax and semantics, and in underlying type systems used for defining service attributes. The greater the difference between the protocols the greater the losses. The Bluetooth SDP protocol is a good example. Due to its very limited set of service discovery operations, a number of operations from other protocols cannot be mapped to SDP. Moreover, once discovery protocols are bridged, there is the related issue of query scope, as the bridged discovery domains may become unwieldy and present clients with resources that are poorly matched for contextual reasons, such as returning a service offer for a requested printer type that happens to be very remote. The SLP protocol supports the concept of scope to partition query domains, but this partitioning is coarse and one-dimensional.

In bridging two decentralised protocols where queries and advertisements are not sent to a single well-known entity, every resolver in the network is required to

have knowledge of a bridging node, such that unresolved queries can be forwarded to the bridge. Therefore, bridges become an opaque extension to these protocols. There are three distinct possibilities in bridging these protocols:

1. cross-advertise all services from each protocol to the other via the bridge;
2. have all advertisements and unresolved queries from one of the protocols forwarded to the other via the bridge; or
3. forward all unresolved queries from each protocol to the other via the bridge.

The first and last of these choices require both protocols to have knowledge of the bridge. The second option requires only one of the protocols to know about the bridge. The first two options suffer from scalability problems as the number of advertisements increases. The third option suffers from scalability issues only when the number of unresolved queries is large.

2.6.2 Summary

Due to these considerable challenges, bridging is not a long-term and scalable solution to the problem of interoperability. These challenges provide fodder for the argument that a single resource discovery protocol capable of spanning multiple environments is a more viable solution. However, such a protocol should be designed to make it as easy as possible (that is, little or no programming effort required) to forward queries (and possibly advertisements) to domains that utilise other service discovery protocols.

2.7 Conclusions

The focus of resource discovery research is on providing solutions for very specific problem domains. Therefore, numerous service discovery protocols have

been defined, and each is appropriate for a particular kind of computing environment. Due either to their inability to scale or their reliance on homogeneous underlying network layers, these protocols cannot be deployed in environments other than those to which they are targeted. This situation means that if an application is moved from one kind of computing environment to another, it must be re-programmed to utilise the interface of the service discovery protocol offered in that environment. It also prevents queries and advertisements issued in one environment from propagating to other environments.

In addition, existing protocols that integrate context-sensitive features with service discovery are either limited in the range of context-aware services they can provide [110] or introduce additional programming abstractions and interfaces.

An opportunity exists, therefore, to design a resource discovery protocol that does not suffer from these drawbacks. Specifically, in order to maximise the potential of the ubiquitous computing paradigm, a service discovery protocol that exhibits *all* of the following traits must be defined:

- 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

The remainder of this thesis reports research impelled by the above observations. The chapters to follow present a resource discovery protocol which incorporates the features listed above. The design, discussion and analysis of this resource discovery protocol is structured as follows.

Chapter 3 identifies the theories and techniques drawn upon to create a resource discovery protocol that exhibits the above properties. Particularly, ideas from the field of complex systems are expounded, and it is shown how these ideas can form the basis of a highly robust and scalable service discovery protocol.

Chapter 4 details the core elements of the resource discovery protocol, including the query routing protocols required to facilitate a scalable, adaptive service discovery platform, and the description language it utilises. The theories introduced in Chapter 3 influence the design of these routing layers.

Chapter 5 documents the non-core components of the resource discovery protocol. These components include context-sensitivity, preferences and result-ranking. This chapter shows, by way of example, how these additional features allow the development of novel applications and reduce bandwidth consumption.

A prototype implementation is discussed in Chapter 6. This chapter explains the features of the prototype, as well as its limitations and the ways in which it deviates from the design.

Finally, Chapter 7 presents an extensive analysis of the service discovery protocol.

CHAPTER

3 Complex Systems in Resource Discovery

The previous chapter described some existing service discovery protocols and highlighted their strengths and weaknesses. In addition, it summarised the necessary features of a resource discovery protocol suitable for pervasive computing environments. In this chapter, some initial ideas are formed about the way in which a better service discovery protocol can be designed by drawing upon complex systems theory. In particular, this chapter shows how a type of biomimicry and the analysis of complex networks can aid in the development of a resource discovery protocol for modern computing environments.

3.1 Introduction

The modern computing environment is an amalgam of the disparate sub-environments outlined in Section 2.1. These environments are made up of *people, places* and *things* [109] that interact with each other in diverse ways. Furthermore, the *people* can take on different roles, depending upon what time it is or where they are.

The *things* include devices, computationally augmented objects, network protocols and applications. The *places* each have their own unique qualities, which can affect the interactions between other elements.

It is interesting to note the parallels between these characteristics and the traits of complex systems. The Center for the Study of Complex Systems [120] states that a complex system has the following characteristics:

Agent-based The fundamental components of the system are entities that interact with one another by various means.

Heterogeneous The agents and their behaviours are diverse.

Dynamic The individual behaviours of the agents may change over time, as may the composition of the system as a whole. The holistic behaviour of system over time is nonlinear or even chaotic.

Feedback The dynamics of the system is driven by the feedback received by the agents as a result of their own actions.

Organisation The agents are organised into distinct structures, which are either inherent in the system or arise from interactions between the agents (self-organisation).

Emergence System level behaviours arise from the local interactions among agents. A well known emergent phenomenon is self-organisation.

Due to these features, complex systems are immensely scalable, as exemplified by the biological, economic and social systems (among others) that we see and interact with on a daily basis. These systems have evolved unique structures and patterns, and continue to evolve in an ongoing process of adaptation. This ability to adapt to changing conditions, which arise as a result of feedback or external influences, makes these systems highly robust.

If it is presupposed that modern day computing environments *are* complex systems, it seems reasonable to design new applications and network protocols within a complex systems theoretic framework. Specifically, this thesis contends that a resource discovery protocol whose target environment is a complex system may benefit by incorporating ideas from the field of complex systems theory.

The next section gives a brief overview of complex systems. Sections 3.3 and 3.4 cover complex systems phenomena that are central to this thesis. Section 3.5 proposes a query routing algorithm that utilises the concepts of emergence and adaptation. It also describes a method by which a structured peer-to-peer protocol (Chord [40]) can be influenced toward a scale-free routing topology by the use of an appropriate caching strategy, so that key lookups take less time. Finally, Section 3.6 summarises key ideas presented in this chapter, with a view to their use in subsequent chapters.

3.2 Complex Systems: Background

Complex systems is a relatively new field of science concerned with the way in which the behaviour of parts of a system give rise to global behaviours. A key aspect that differentiates complex systems from complicated systems is the nature of the interactions between the components in the system. A complicated system, such as a nuclear submarine, consists of many components that interact in a predictable fashion. In complex systems parlance, such a system is known as *linear* due to the predictable sequence of cause and effect. A complicated system (as opposed to a complex one) may consist of many thousands of components, but each component has a static set of other components with which it interacts. On the other hand, complex systems such as societies, economies and ecosystems, contain components that often interact in a *nonlinear* fashion. An often quoted

example of a nonlinear relationship is the way in which the effectiveness of a particular medication changes as the dosage increases. A medicine may be completely ineffective until a threshold dosage is reached. The medication remains effective for dosage levels above this threshold until the dosage becomes *too* high.

These complex relationships can give rise to emergent behaviours. Emergent behaviours are those that arise at the global or system level and cannot be predicted from observing the behaviour of individual components. Emergence is the formalism behind the common saying "the whole is greater than the sum of the parts". Some examples of emergent behaviour are those of self-organisation such as flocking and swarming in birds and insects, and Adam Smith's "invisible hand" whereby resources in an economy are said to find their way to where they are most efficiently used without the need for a central guiding mechanism [121]. Computer networks such as the Internet organise themselves into scale-free networks. In simplified terms, these are networks in which a small proportion of nodes have a large number of links to other nodes, while the other nodes have fewer links. In more formal terms, these are networks whose degree distributions follow a power-law. Such emergent patterns are not unique to computer networks. Indeed, social networks, the graphs formed by disease propagation and so forth, have all been found to have the scale-free property. In many cases, when emergent patterns and behaviours are observed, a power-law distribution lies at the heart of the interactions. Among other things, this aspect of complexity science is being used to combat the spread of viruses, and also to study social interactions.

Many complex systems, particularly those said to be *chaotic*, are extremely sensitive to initial conditions. A slight variation in the starting conditions may lead to vastly different outcomes. In addition, modifying one part of a dynamical system can have far reaching and non-proportional consequences in other parts of the system. This idea is commonly known as the *butterfly-effect*, a term derived from

the notion that a butterfly flapping its wings in one region of the world might lead to the formation of a cyclone, tornado or other major weather events in another region of the world. In models and simulations of complex systems, initial conditions are usually represented by input parameters. The behaviour displayed by a simulation or predicted by a model is largely dependent on these initial parameters. These initial parameters will play an important role in the analysis presented in Chapter 7. Sometimes a system will settle into a pattern of repeated behaviour. These patterns are called *attractors*, and may take one of several forms. The simplest kind of attractor is the *point* attractor, where the system settles into a single state. An example is rolling a ball-bearing around a bowl. Eventually it comes to a halt at the lowest point of the bowl. *Cyclic* attractors, or *limit cycles*, are another kind of attractor. In this pattern, the behaviour of the system oscillates between a finite set of fixed points, and the pattern repeats continuously. Examples are the seasons in a year and the orbit of the moon about the earth. Other attractors are termed *strange*. These attractors exhibit chaotic behaviour. Chapter 7 shows that the behaviour of the service discovery protocol developed in Chapter 4 is attracted to a point or a limit-cycle depending upon initial parameter values.

These nonlinear dynamical systems are often modelled with differential equations or recurrence equations [122], which are, respectively, the continuous and discrete incarnations of the same mathematical tool [123]. Chapter 7 utilises a recurrence equation to model the nonlinear dynamics of the service discovery protocol presented in the next chapter.

This section has presented a very brief overview of complex systems. It has introduced only those concepts which are of importance to the remainder of this thesis. There exists an extensive body of literature on complex systems; the following is a small selection of publications that provide further information about this burgeoning area of science. Bar-Yam [124] provides an overview of the meth-

ods used to study and model complex systems. Barabási [125] details the recent advances made in the study of complex networks, and Solé et al. [126] explore the differences and similarities between complex bio-systems and the complex networks created by humans (such as the Internet and lexical networks). Finally, Flake [127] presents an in-depth overview of chaos, complex systems and adaptation.

The following two sections provide details of particular complex systems that strongly influenced the design of the service discovery protocol presented in this thesis.

3.3 Complex Systems in Nature

Nature is abundant with examples of complex systems. The entire ecosphere itself is a complex system [128, 129] comprised of an incomprehensible number of agents, such as living things and particles. These agents interact with each other in a multitude of different ways. Even a relatively small ecosystem encapsulates a complex web of interactions, including a food chain and symbiotic relationships. Populations fluctuate or stabilise due to a wide range of interactions [130] between the agents within the ecosystem. It is not Earth's ecosystem as a whole, or even a single ecosystem within it, with which this section is concerned. Rather, it is concerned with the complex phenomenon that arises as a result of simple interactions between individuals within certain kinds of insect colonies - in particular, ant colonies.

Stigmergy is a method of indirect communication used by a large number of insect species [131]. The process of ant foraging is a commonly cited example of stigmergy, and is often used to explain the phenomenon. When one observes a set of foraging ants, it rarely happens that these ants scurry randomly

in every direction. Instead, they usually organise into a single, neat column, with some ants heading in the direction of the food source, and others returning to the nest. How does this occur? Upon locating a source of food, an ant returns to its nest, and in doing so lays a pheromone trail. Other ants in the ant colony follow this pheromone trail back to the food source, and they too lay down pheromone, thereby reinforcing the trail. It may happen that another ant from the colony found the food source by a different route. If this route happens to be shorter than the other, then this route will be reinforced to a greater degree. This occurs because ants using this route will return to the nest more quickly than those using the other route, and therefore, in the same period of time, a greater number of ants will have traversed the shorter route, meaning its pheromone trail is stronger. The shorter trail thus recruits more ants, reinforcing it even further.

Let us examine the processes at work here. The most visible aspect of the stigmergic process is the manner in which it organises the ants into a neat structure. There is no controlling entity that imposes the column structure onto the ants. Rather, the column emerges as a result of the indirect communication between the individual ants. The second aspect of this process is the way that stigmergy can converge on an optimum solution to the food gathering problem. Although it is unlikely that the path chosen by the foraging ants will be a shortest path between the food source and the nest, it is true that the chosen path will usually be the shortest among the available options. In effect, the fittest choice is selected. Again, the shorter path is not selected by a central entity, but is chosen through a distributed process where decisions are made by individuals in the system.

Sumpter and Beekman [132] show that when confronted with food sources of differing energetic value, ants will usually allocate a greater part of the workforce to the source of higher gain. This shows that the stigmergic process differentiates between alternatives, and can select the more profitable option.

These observations are used to develop a simple query routing model in Section 3.5.1.

3.4 Network Analysis

Recently, the analysis of networks has become an extremely popular topic among physicists working in the area of statistical mechanics. Contemporary research in this area has been driven, in part, by the proliferation of human engineered networks such as the World Wide Web and peer-to-peer file-sharing systems.

The relationships between the components within a complex system can be viewed as a network. The networks formed by a complex system are not characterised by any topology in particular; however, studies into real world networks, including the World Wide Web and the Gnutella overlay, have revealed that many of them have scale-free properties [77, 133]. A scale-free network is one in which the link distribution conforms to a power-law. That is, the probability $P(k)$ that a node in the network is connected to k other nodes is proportional to $k^{-\gamma}$, where γ differs slightly depending upon the network in question. For example, in the World Wide Web, the probability that k web pages link to a particular document follows a power-law with $\gamma = 2.1$. Conversely, the probability that a particular web page has k outgoing links follows a power-law with $\gamma = 2.45$. Similar results have been obtained for the link distribution of power grids and the collaboration graph of movie actors [134].

The power-law link distribution means that a relatively small number of nodes are very highly connected, while most of the others are connected only to a few nodes. This gives scale-free networks several interesting properties. First, they are robust against random failure. Since there are only a few highly connected nodes, the probability that one of these *hubs* fails randomly is low. If a less connected

node fails, then it does not usually affect the connectivity of the graph adversely. If a hub were to fail due to a targeted attack or as a result of the small chance of random failure, the connectivity of the network would be impacted severely. This knowledge is now being used to combat the spread of diseases [135]. Second, the presence of hubs decreases the diameter of the network. In other words, the path from one node to another is shorter in scale-free networks compared to networks with normally distributed (random) links. Thus, a scale-free network is also a small world network [76].

Section 3.5.2 shows how these properties can be infused into a distributed hash table - a structure that ordinarily exhibits completely different characteristics.

3.5 Applying Complex Systems Theory to Query Routing and Distributed Hash Tables

Section 2.3.1 concluded that existing query routing protocols for unstructured networks are not scalable and flexible enough to be used in a variety of unstructured networks. That section argued for a service discovery protocol that can adapt to change, without the need for flooding, and that does not expend effort on finding a shortest path from one node to another, since, in a dynamical system, that shortest path is likely to be short-lived. This section proffers the idea of mimicking the biological process of stigmergy, discussed in Section 3.3, to provide the basis for a query routing protocol with the desired characteristics. In addition, using complex network theory, which is outlined in Section 3.4, modifications are made to the Chord distributed hash table algorithm to increase its scalability for use in stable structured environments that contain nodes of varying storage capacity. The work reported in the following two sections can also be found in a previous paper [136].

3.5.1 A Stigmergic Routing Protocol

The Idea

One of the pervading themes in complex systems is the notion of *gaining order for free*. The application of such a concept to software or protocols for resource constrained devices which operate in dynamic environments is highly appealing and appears promising. Protocols for sensor networks, handheld computers, mobile phones and other equally resource constrained devices must be minimalistic due to physical constraints. It is these factors that guide the design of the following query routing protocol.

The environment for which this protocol is designed consists of resource constrained devices distributed randomly throughout an area such as a shopping mall. The devices are mobile with respect to one another, and they form a multi-hop network. Each device is free to leave the network at any time. At a later time it may join another network with similar characteristics.

The query routing protocol is loosely based on ant foraging, as described in Section 3.3 above. However, the protocol necessarily deviates from the ant foraging process for the following reasons.

In ant foraging, there is one nest and potentially several food sources. Pheromone trails direct ants from a single colony to the various food sources. In service discovery, there is not one nest, but several. A nest corresponds to a service discovery client. Clients can cache the results of queries. Thus, in service discovery, the presence of a pheromone trail does not benefit the client who has just issued a query (unless the client does not use caching); rather, it benefits those clients who will issue queries in the future. It is as if different ant colonies shared their pheromone trails. Figure 3.1 depicts this situation.

Furthermore, pheromone trails in service discovery must be differentiated ac-

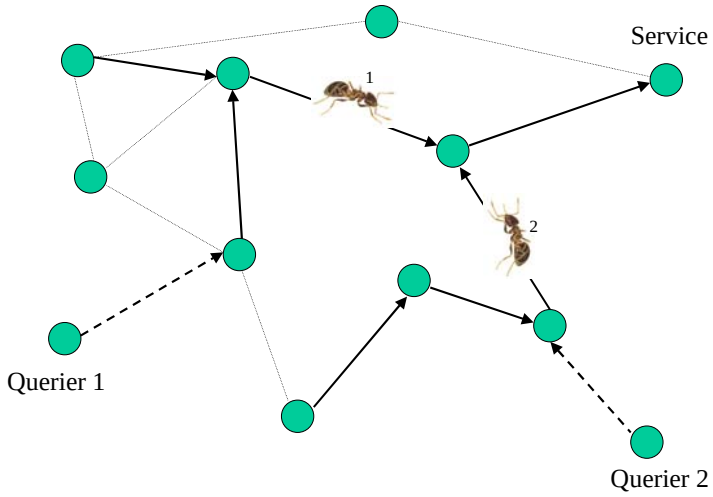


Figure 3.1: “Ants” from different “nests” sharing “pheromone trails”.

cording to the *type* of resource they lead to. A pheromone trail that leads to the description of a printer is of no use to a client searching for a storage device, for example.

The question remains: what constitutes a pheromone trail in the service discovery protocol? Each node along the path from the site of query resolution to the querier could cache the entire response. This means that future queries that match the same resources could be resolved by any of the nodes along the path. However, in the particular environment defined above, this is not a viable solution because the devices are very resource constrained in terms of storage capacity. An alternative is to store a concise summary of the service descriptions and a pointer to the next-hop node. The summary might simply consist of the class or type of service. If a future query matches the concise summary, then it is forwarded to the next-hop node. If the query does not match any stored summaries, it is forwarded randomly. This process continues until the query is resolved or until its hop counter expires (the *TTL*).

In this scenario, we assume there are no service advertisements. Thus, only the service node contains the complete service description. The implication of this is that most queries for this service will be forwarded all the way to the service node to be resolved. Only very simple queries - those that specify type or class constraints and nothing else - can be resolved by the intermediate nodes, by virtue of the pheromone trails.

Such an algorithm is presented below.

```
void receive(message) {
    if(isQuery(message)) {
      if(match(message)) {
        respond(matchingServiceRecords);
      } else if(partialMatch(message)) {
        forwardAlongTrail(message);
      } else if(expired(message)) {
        respond(unresolved);
      } else {
        decrementTTL(message);
        forwardRandom(message);
      }
    } else { // It's a query response
      storePheromone(message);
      forwardNextHop(message);
    }
}
```

The above algorithm is executed at each node. Nodes receive messages. If the message is a query then it is checked against stored query results that have previously traversed through this node on their way back to a querying client. If there is a match, then the query terminates and the matching service record is returned. If

the query matches against a concise summary (partial match), then it is forwarded to the next hop along the relevant pheromone trail. There may be several partial matches, in which case only one of the partial matches is chosen, and the query is forwarded to the corresponding neighbour. The hop count (TTL) is not decremented if the query is following a pheromone trail, since the query will eventually be either fully resolved or the end of the pheromone trail will be reached. If there is no match and the query has reached its hop limit ($TTL = 0$), then the query is unresolved and a message to that effect is sent in response. Otherwise, the query is forwarded to a randomly chosen neighbour. If the message is a positive query response, then the response is stored at this node and then forwarded in the direction of the querying client. Stored records eventually timeout unless they are reinforced by further queries. This simplified version of the algorithm ignores issues such as routing loops and query backtracking. [1]

Discussion

The query routing protocol presented above is exceedingly simple. It is described by a small set of trivial rules that decide where to forward queries and their responses. But from this simple set of rules emerges more complex system behaviour that is not directly encoded in the rule-set.

If a service (or more correctly, queries for a particular class of service) is popular, then the pheromone trails become reinforced. That is, the cached service summaries that constitute the pheromone trail will not time out. If a service is not popular, then the pheromone trail leading to it (which was created as the result of a matching query) will dissipate as a result of cache expiration. This means that an already popular service will have more chance of being discovered than

[1] If a loop or dead end is detected, the query should backtrack to the first node that has other neighbours to which the query can be sent.

a less popular one because there are more pheromone trails leading to it, and thus likely that a query will come across an existing pheromone trail. In complex systems parlance, it can be said that the network *evolves* to favour popular services because queries for popular services will spend less time wandering randomly before stumbling across a pheromone trail.

Another characteristic of complex systems, which is also present in the service discovery protocol, is the ability to adapt to changing conditions. The protocol adapts to node failures by reforming pheromone trails as time goes by, thereby providing some robustness. In the event of many simultaneous node failures, resolution times, even for popular services, may be long, but they regain their former levels as more queries are issued.

These emergent features are not explored further here, and many of the details and complications are glossed over. This simple service discovery algorithm provides the basis for an unstructured routing protocol presented in the next chapter, which is given a more thorough treatment. A discussion and analysis of the emergent features arising from that protocol takes place in Chapter 7.

3.5.2 Making Distributed Hash Tables Scale-Free

The Idea

Distributed hash table (DHT) protocols such as Chord [40] and Pastry [97] offer a way in which to guarantee, in the absence of node failures, the success of key lookups in a distributed environment. This property makes DHTs an interesting base on which to build service discovery protocols for stable, structured environments. In fact, INS/Twine [3] is built upon a DHT to create a reliable service discovery mechanism. However, building on top of a deterministic hash table structure negates any benefits of the scale-free nature of the underlying network

upon which the DHT is built. That is, capable, highly connected nodes are forced to play the same role as less capable and less connected nodes. An obvious question to ask, in light of the above discussion about scale-free networks is, can the DHT be made scale-free? Doing so has the important benefit of reducing the average path length from one node to any other node in the peer-to-peer hash table. In turn, any service discovery protocol overlaying such a scale-free hash table would achieve lower query latency. The remainder of this section is devoted to detailing the mechanism by which the Chord DHT algorithm, presented in Section 2.3.2, can be influenced toward a scale-free topology.

First of all, note that *any* type of caching in Chord will result in a reduced average path length from one Chord node to another. As such, a simple modification to the algorithm and the structures used to support the algorithm can yield a more scalable protocol in terms of the average number of nodes that need to be contacted during lookup. The simplest addition to the Chord algorithm is to cache the addresses of nodes which are discovered during a lookup. The finger table then stores as many identifiers and IP addresses for each interval as it is capable of doing. During lookup, the node can then choose to contact either the node with the closest ID to the target node, or it can measure response times, and choose to contact the node with the fastest response. This elementary improvement, or one very similar to it, is suggested by the authors of Chord. However, they do not discuss the choice of cache replacement policy, which has a direct bearing on the performance of the solution. The remainder of this section discusses a caching algorithm and cache replacement strategy for Chord. It shows why the chosen cache replacement strategy is ideal for the Chord protocol.

The theory of networks, as described in Section 3.4, suggests a way in which this cache can be used to transform the Chord protocol to tend toward a scale-free set of interactions, thereby reducing the average number of nodes contacted

during each lookup further than the above simple caching mechanism could do. In a heterogeneous network, such as one might find in a pervasive computing environment, one will find all manner of devices, some of which are capable of storing large amounts of data and are connected to high bandwidth links. Other devices will be resource poor, and will connect to other devices only through low bandwidth links. If a caching function is introduced to Chord, the size of the cache will vary with the capability of the Chord node. Nodes with large caches will generally have information about nodes closer to the target than would be the case in the original Chord algorithm. This is true because network theory holds that highly connected nodes, or hubs as they are known, can usually reach every other node in the network in a small number of hops. In the context of our hash table protocol, it means that, with high probability, highly connected nodes can *more than halve* the distance to the target on each hop, since it is probable they know of nodes closer to the target than was possible under the original Chord protocol. The new algorithm maintains correctness since each step brings the querying node at least twice as close to the target node as the previous step.

Nodes cache the ID, IP address and cache size of each node contacted during their own lookup operations. This requires no additional messages to be sent. The size of the cache is completely adjustable for each node. When the cache associated with a particular interval in the finger table is full, the node follows a cache replacement strategy that biases the nodes with larger cache sizes. Over time, the cache for each interval will contain the most highly connected nodes in that interval.

Discussion

To prove the viability of such a caching strategy, a simulation model was created. In the simulation, nodes are assigned a random *capability* that represents a space

metric. During testing, the maximum capability assigned to a node was ten percent of the total number of nodes in the simulation. The minimum capability was one.

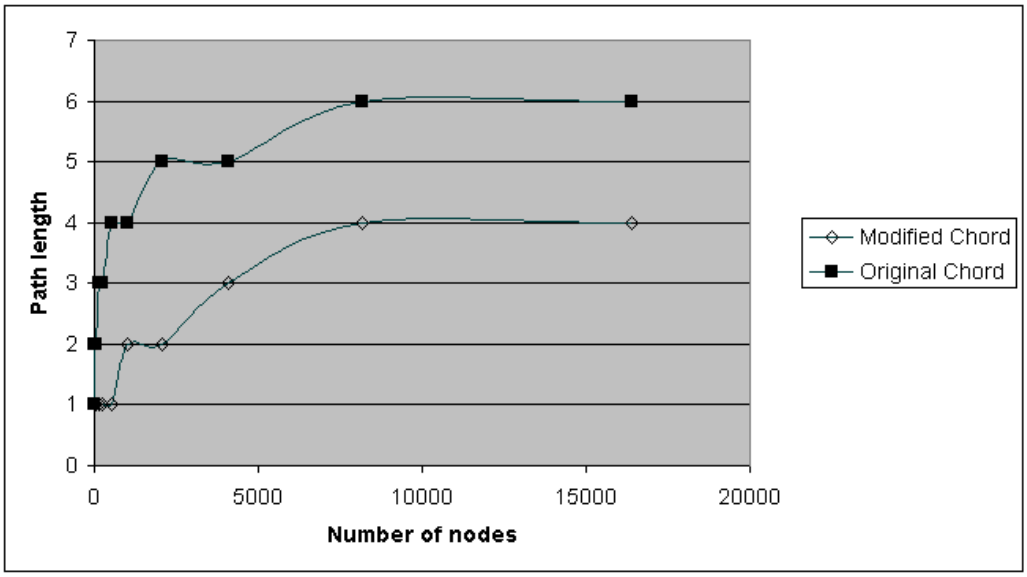


Figure 3.2: Average path lengths of the original and modified Chord algorithms

The results indicate that this modification of the original Chord algorithm results in a significant lowering of the average path length between nodes. In fact, as more and more lookups are performed, the average path length between nodes shortens. Figure 3.2 summarises the results obtained from the simulation of the modified Chord algorithm. For each test, a million lookups were performed, and the average path length calculated. For example, in a network of 512 nodes, the average path length in the original Chord algorithm is approximately four. In the modified algorithm, after a million lookups, the average path length is one. In larger networks, where the simulation shows the hop length has been reduced to two thirds of its original length, the hop length is reduced even further if more than a million queries are issued, since knowledge of highly connected nodes will propagate further through the network.

The choice of caching algorithm does have an effect on the average path

length. To determine this, the caching algorithm was modified to cache the *least connected nodes*. In a network of 512 nodes, the average path length is two; twice that of the scale-free caching algorithm. Therefore, the cache replacement algorithm has a bearing on the average path length in the network.

We do not consider a least-recently used purging policy since node popularity changes dynamically when new nodes join the network and when new keys are added to the DHT. The capability of a node, in terms of its capacity to store pointers to other nodes, is a static property, and therefore a better choice on which to base a cache purging policy.

3.6 Summary and Conclusions

While others have attempted to use complex systems theory and biomimicry in the design of new software and packet routing protocols [137, 138], there is no prior work advocating the use of complex systems theory to allow *resource discovery* protocols to scale to networks of different sizes and to operate in heterogeneous domains.

This chapter presented an overview of relevant aspects of complex systems theory, and showed two ways in which it could be applied to benefit service discovery. The ideas cultivated in this chapter are extended and utilised in the design of a resource discovery protocol for the modern computing environment, detailed in Chapter 4. In particular, the query routing protocol of Section 3.5.1 is expanded and formally described. The scale-free caching strategy applied to DHT protocols is relevant not only to service discovery protocols with an underlying DHT, but to any application that uses DHT protocols.

CHAPTER

4 Superstring

Chapter 2 highlighted a number of research problems that are still to be overcome in the area of service and resource discovery. In this chapter the design of a service discovery protocol, Superstring, is outlined. Superstring provides solutions to many of the identified problems. Its main goal is to operate in a wide range of computing environments, which together comprise a large-scale pervasive computing environment. There are many design elements to be considered in Superstring. Among these are advertisement and query syntax and semantics, and protocol behaviour. Section 4.1 provides an overview of the design goals with respect to the highlighted research problems, and briefly states how the design caters for each of these problems. The remainder of the chapter is dedicated to a detailed description of each of the design elements.

4.1 Goals

Existing protocols are suited to one or another of the computing environments examined in Section 2.1, but none are suitable for multiple environments. This is not unexpected, due to the large discrepancies between the various computing environments. An implication of this state of affairs is that interoperability becomes a major factor if the goal of truly heterogeneous service discovery is to be achieved. Steps can be taken to ensure that discovery can take place across heterogeneous domains. Superstring achieves this by defining two underlying protocols: one tailored to stable structured networks, and another that performs better in dynamic communities of devices. These two protocols, in combination, are adequate to cover a broad range of environments. A common API is provided to hide the differences between the two underlying routing layers.

Section 4.2 provides a high level overview of the Superstring architecture. Its simple API is presented in Section 4.3, and the lightweight description language is documented in Section 4.4. Finally, the two underlying routing protocols are defined in Sections 4.5 and 4.6. These constitute the core elements of Superstring, and address the first five required traits enumerated in Section 2.7. The remaining two traits are addressed in Chapter 5.

4.2 Architecture in Brief

At its core, resource discovery is about issuing queries to locate resources that match a set of constraints, and selecting one or more resources from this matching set. Some resource discovery protocols allow services to *advertise* the resources they offer. If advertisements are used, then it becomes possible for queries to be resolved by a third party node (a node that is neither the querier nor the service node), which enables upward scalability and, in many cases, the balancing

of query processing across the nodes in the network. Discovery thus consists of matching the description contained in a query with a description contained in an advertisement. Superstring offers these two, conceptually simple, operations to applications so that they may offer their own resources and find resources required for their successful execution. While querying and advertising are concepts fundamental to most resource discovery protocols, it is the algorithms underlying these operations which are of primary interest to the research community. Superstring attacks two main problems facing service discovery protocols in pervasive computing environments: scalability and heterogeneity. The approach taken by Superstring to overcome these issues is to provide a single interface to applications, but to utilise one underlying query routing protocol in static, stable networks and another in more dynamic networks. These correspond to structured and unstructured networks, respectively. It is interesting to note that, often, the classification of a network as structured or unstructured correlates directly to its classification as a wide-area or local-area network. Wide-area networks are commonly long-lived networks that link organisations or remote branches of a single organisation, and thus, their structure (or more precisely the infrastructure which supports them, such as computer hardware and network cable) rarely changes. Conversely, the structure of local-area networks, especially those formed by an ad hoc gathering of people and their devices, is very dynamic. Even those local-area networks that can be considered fixed, such as Ethernet LANs, display far more dynamic behaviour than their wide-area cousins, as workstations are turned on and off, and as administrators expand the Ethernet subnet by adding new network segments or installing wireless access points. Network administrators have more flexibility to alter the status quo within the local domain, than they do to change wide-area network links and the hardware that supports them. While it is acknowledged that the term *structured networks* is not equivalent to *wide-area networks*, this thesis

takes license to use the terms interchangeably. Likewise, the term *unstructured networks* is casually equated to the term *local-area networks* for the remainder of this thesis.

Superstring takes advantage of structured networks by utilising a structured query routing algorithm on these networks. On networks that are inherently unstructured, such as those composed of mobile devices or nodes that alternate between an online and offline state (such as the average personal computer used at home), an underlying routing protocol is used that does not rely on the presence of static network components. If a node executing the routing protocol for unstructured networks comes into contact with a resolver on the structured network, it may interact with nodes beyond its immediate neighbourhood by sending queries and advertisements to this resolver.

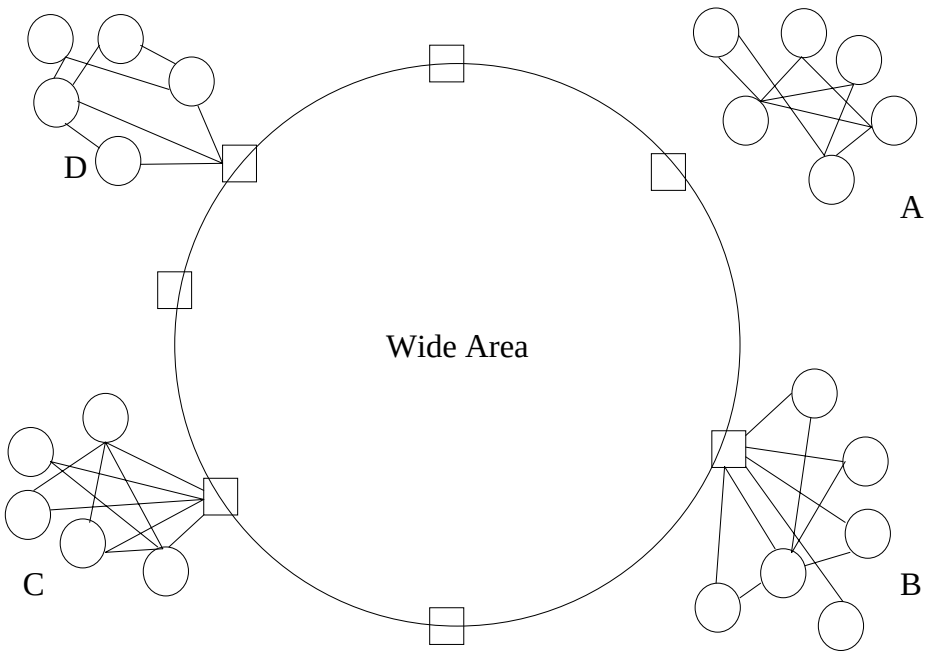


Figure 4.1: High level view of Superstring

Figure 4.1 gives a high-level perspective of the interaction between elements in the wide-area and local-area. The group of nodes labelled *A* is an isolated group. The nodes interact only among themselves. Queries are resolved in that group exclusively by the local-area protocol. Such an isolated community might be formed during an emergency response scenario or by the devices possessed by the members of a scientific expedition. A more commonplace example of a type *A* community is when attendees of a conference meet and exchange their electronic business cards via the Bluetooth interface of their PDAs or mobile phones. Groups *B* and *C* interact among themselves, but each node in those groups chooses to maintain a connection with a resolver that is linked to other resolvers in the wide-area. An office LAN is a typical example of this kind of community. In group *D*, some of the nodes choose to maintain links directly with the resolver. The other nodes do not have a direct connection to the wide-area resolver due to differing underlying protocol stacks. These nodes may interact with the wide-area resolver indirectly via nodes that can communicate directly with the wide-area node. The sections below elucidate the local-area and wide-area routing protocols, and how they combine in the overall Superstring architecture. However, before the routing algorithms are discussed, the description language and API used by Superstring advertisements and queries is explained in detail, as these elements are independent of the underlying routing layers.

4.3 The Superstring API

The Superstring API has been intentionally kept small. As stated in the previous section, resource discovery consists of advertising and querying. The Superstring API exposes these two core processes to applications. The API is described by the following two operations:

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

Query, QueryResponse and Advertisement objects contain Descriptions, which are XML representations of Superstring descriptions. XML was chosen because it is widely used and well-supported. Small XML parsers exist for resource constrained devices. Applications are free to use whatever XML APIs are available to them. The only restriction is that the XML must conform to the description language XML mapping (defined in Section 4.4.4). Advertisements are removed implicitly via a soft-state mechanism (that is, via cache timeouts); therefore, no operation is defined for the explicit removal of advertisements. While this could lead to stale advertisements being returned in query results, in mobile computing environments it is not always possible to ensure that *remove* requests would be propagated to all relevant nodes. Furthermore, short storage lifetimes largely prevent the accumulation of stale advertisements. Finally, the bandwidth overhead incurred in attempting to utilise a stale resource is partially offset by the fact that this protocol incurs no bandwidth overhead for removing advertisements.

The interfaces for Query, QueryResponse and Advertisement are shown below (using Java-like notation):

```
public class Query {
  public static Query
                createQuery(Description desc,
                            int hopsToLive,
                            boolean isWideArea);
  public byte[] getDescription();
  public int getHopsToLive();
  public boolean isWideArea();
}
```

```
public class QueryResponse {
  public static QueryResponse
               createResponse(int type,
                              int numResults,
                              Vector descriptions);
  public int getType();
  public int getNumResults();
  public Vector getDescriptions();
}

public class Advertisement {
  public static Advertisement
               createAdvertisement(Description desc,
                                   NameRecord nr,
                                   int hopsToLive,
                                   boolean isWideArea);
  public byte[] getDescription();
  public NameRecord getNameRecord();
  public int getHopsToLive();
  public boolean isWideArea();
}
```

A *NameRecord* contains a globally unique identifier and contact information for the resource it names. At present, the contact information is limited to a URL, but in future this could be replaced by a resource locator that can cater to mobile environments, as well as static ones.

4.4 Description Language

Superstring defines a hierarchical description language for queries and advertisements. The choice of a hierarchical description format was guided by the fact that it can encapsulate containment relationships in a natural manner, and that it can easily provide the idea of query relaxation. Furthermore, non-hierarchical information can be trivially converted to a hierarchical format. The sections below describe Superstring's description language with reference to the following challenges.

4.4.1 Challenges

As Section 2.4 concludes, a major challenge in designing a description language for a resource discovery protocol that must operate in a wide range of environments is finding a balance between descriptiveness and computational and communication costs. The language should allow resources to be described accurately by resource providers, and it should enable resource consumers to formulate precise, fine-grained queries. On the other hand, query resolution must be computationally feasible on lightweight devices, and the description language itself should not generate network traffic (due to description validation requirements, for example). On the contrary, a good description language will include features to reduce communication overhead. For instance, in attempting to locate a resource, it may happen that an exact match for the query is not found, but the querier will accept a match after weakening the query criteria. Ordinarily, the querier would need to re-issue the query with fewer and fewer constraints until a match is found or until the query can be relaxed no further. A description language can eliminate the need for query relaxation to be an interactive process by defining primitives that will automatically relax a query to a given resolution when no exact match can

be found. Similarly, the description language should provide the option to specify alternative elements for parts of a query, rather than requiring the querier to issue an alternative query when the initial query fails. These challenges are addressed by Superstring's description language, fondly dubbed *AWOL* (Anti-Web Ontology Language), which is described below.

4.4.2 Advertisement

A Superstring advertisement consists of a hierarchy of *components*. Every component, except the root, has a parent component. A component is either an *attribute component*, or a *value component* (simply called a *value* herein). Each attribute component may have an optional value associated with it. All values are leaf components in the hierarchy (that is, no value component has a child). An *attribute name* consists of one or more attribute components, and extends from the root component to an attribute component with an associated value. Attribute names are also called *strands*, and these terms are used interchangeably throughout the rest of this thesis. To denote a particular attribute name within the text of this document, a dot-separated notation of the form $root.c_1.c_2.c_3$ will be used, where *root* is the root component of the strand and c_{i+1} is the child component of c_i. This notation is also used to specify utility functions in the preference language (see Section 5.3.3).

Figure 4.2 shows an example advertisement. Circles denote attribute components, and boxes denote values. It describes a compute server which might be part of a grid computing environment. This particular advertisement describes functional attributes of the compute server such as the number of CPUs (*compute-server.cpu.number*) and their speeds (*compute-server.cpu.MHz*), the amount of RAM (*compute-server.memory.quantity*) and so forth. This example is simplified to omit many functional attributes including information about the operating

system. Non-functional attributes, such as location details, have also been omitted.

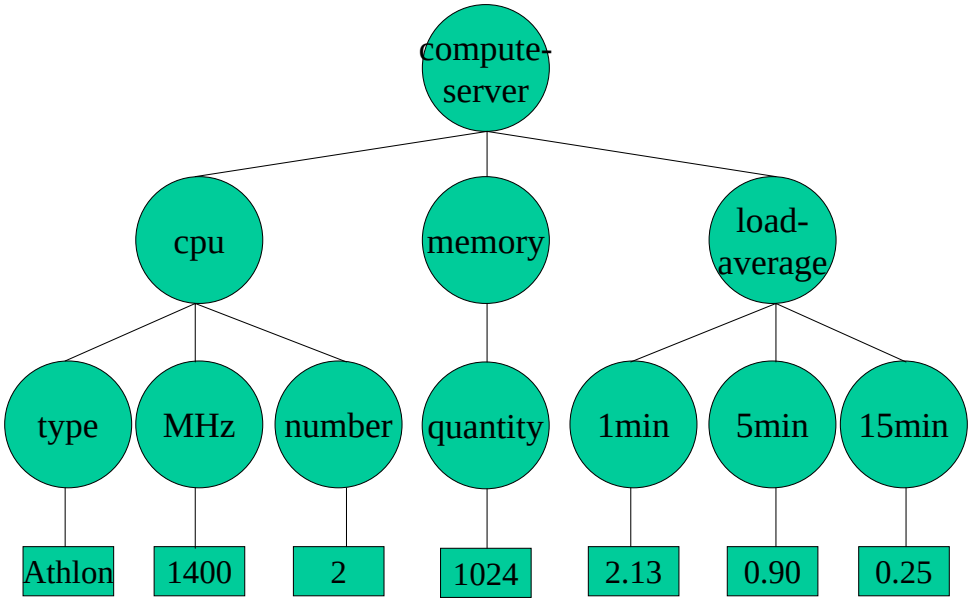


Figure 4.2: The description for a compute-server.

Superstring does not prescribe the semantic meaning of advertisements, nor does it impose any naming convention upon descriptions. This is left to application developers to determine.

4.4.3 Query

Queries have the same structure as advertisements, with the following additions.

Expression Language

Queries may contain expressions in place of exact values. These expressions can use a variety of relational operators, and may also draw upon a small set of built in functions. The expression language is defined by the following EBNF grammar:

```
exp         -> base-exp | '!' base-exp
base-exp    -> and-exp { or-op and-exp }
or-op       -> '||'
and-exp     -> simple-exp { and-op simple-exp }
and-op      -> '&&'
simple-exp -> comp-exp { comp-op comp-exp }
comp-op     -> '==' | '!=' | '<' | '>' | '<=' | '>='
comp-exp    -> term { add-op term }
add-op      -> '+' | '-'
term        -> factor { mul-op factor }
mul-op      -> '*' | '/' | '%'
factor      -> '(' exp ')' | literal | THAT [ call ]
call        -> '.' func '(' arglist ')'
arglist     -> arg { ',' arg } | empty
arg         -> exp | empty
literal     -> integer | float | string
func        -> ID
```

This expression language provides a simple but powerful means to describe required resources in fine detail. Note that although the grammar allows the subtraction, multiplication, division and remainder operators to be used on string types, any attempt to do so will result in the expression evaluating to ‘false’, and so the query will fail. The addition operator may be used on strings, where it acts as a string concatenator. Examples are provided in Appendix A.

The query language also provides three special attribute components: *any*, *not* and *scope*.

Any

The primary use of the *any* attribute component is to allow disjoint query elements. Figure 4.3 displays a query with disjoint query elements. In this example, a query is issued that attempts to match a camera resource whose location might be described geodetically or physically. (Functional attributes are omitted from this example.) A camera that matches any of these location descriptions is acceptable to the querier.

Not

As its name suggests, the *not* attribute component negates the meaning of its child sub-description. Figure 4.4 shows a query that will find all motion sensors in the building except the ones in the control room (because the user issuing the query is currently in the control room, and is not concerned about her own movements).

Scope

The *scope* attribute component is the means by which Superstring achieves query relaxation. Superstring provides the concept of query relaxation to enable queriers to specify that they will settle for a near match in the event that an exact match is unavailable. The *scope* attribute component can deliver this behaviour without the application or user having to modify a failed query and re-issue it with relaxed constraints. Query relaxation is most useful in those attribute names that describe a containment hierarchy. For example, Figure 4.5 depicts a query for a free meeting room. Preferably, the querier would like a room on level 6 of the building. But if there are no free meeting rooms on level 6, the querier will make do with

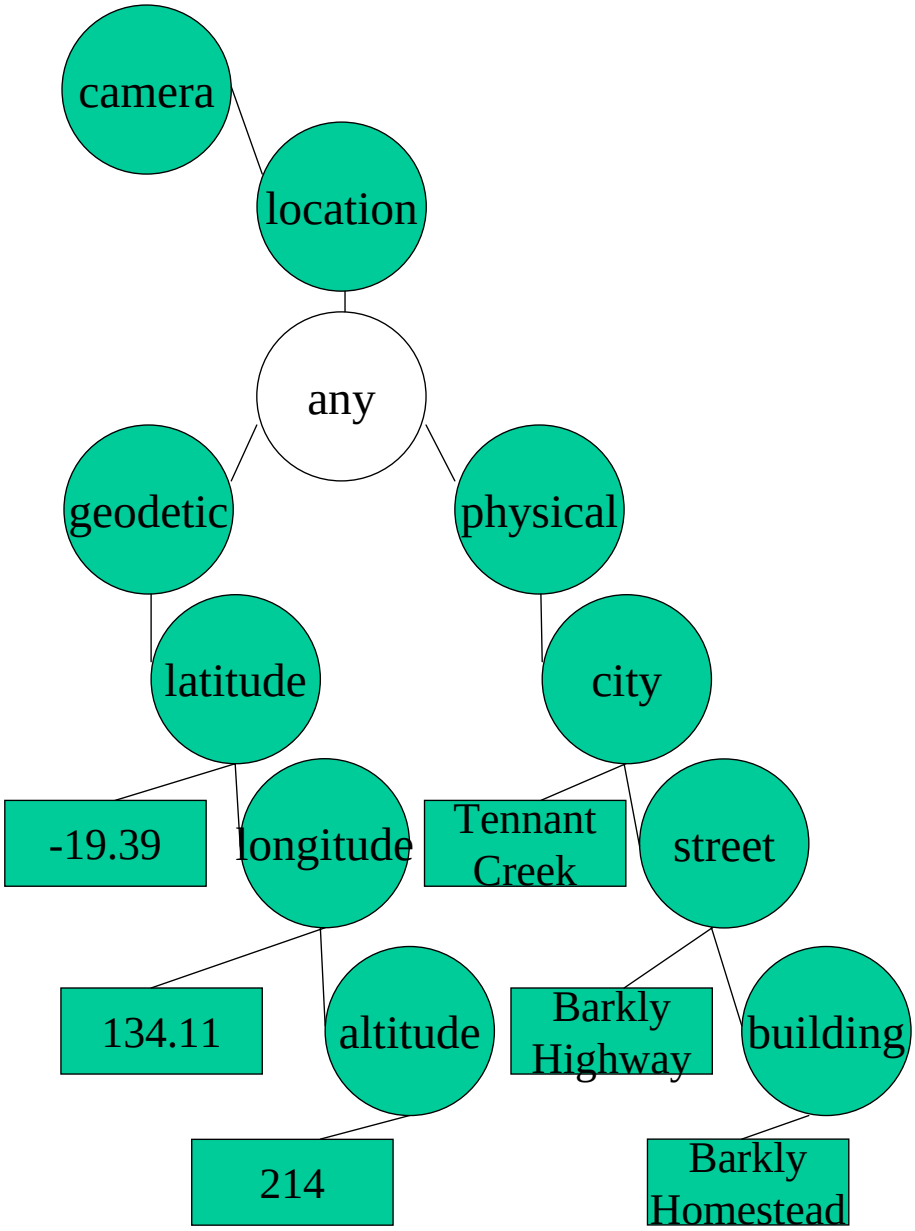


Figure 4.3: A query that will match any of the sub-descriptions.

a free room elsewhere in the building. The query should fail only if there are no free rooms in the building. The *scope* attribute component is inserted as the par-

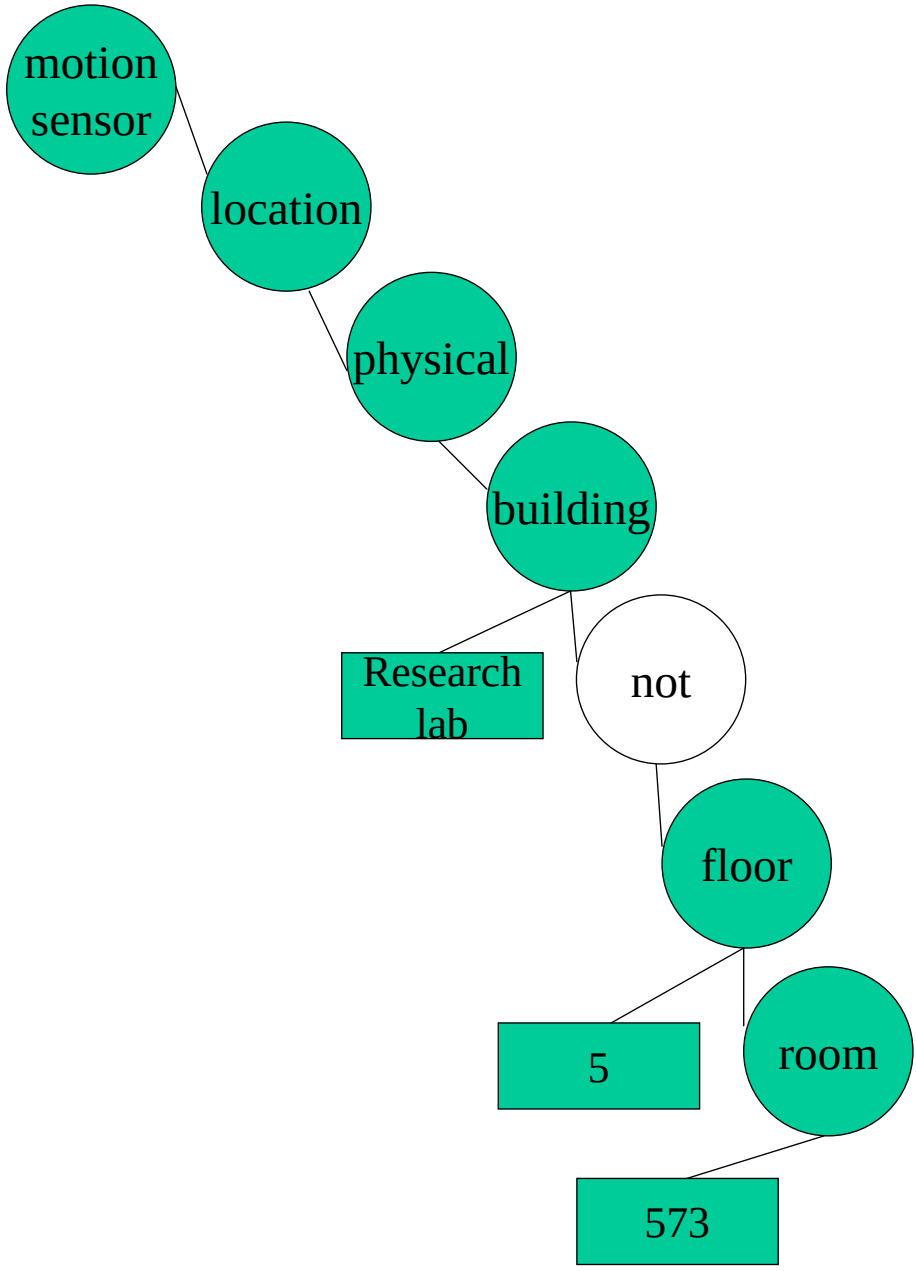


Figure 4.4: A query that excludes resources in particular locations.

ent of the building attribute, to signify that relaxation should apply up to building resolution, but no further.

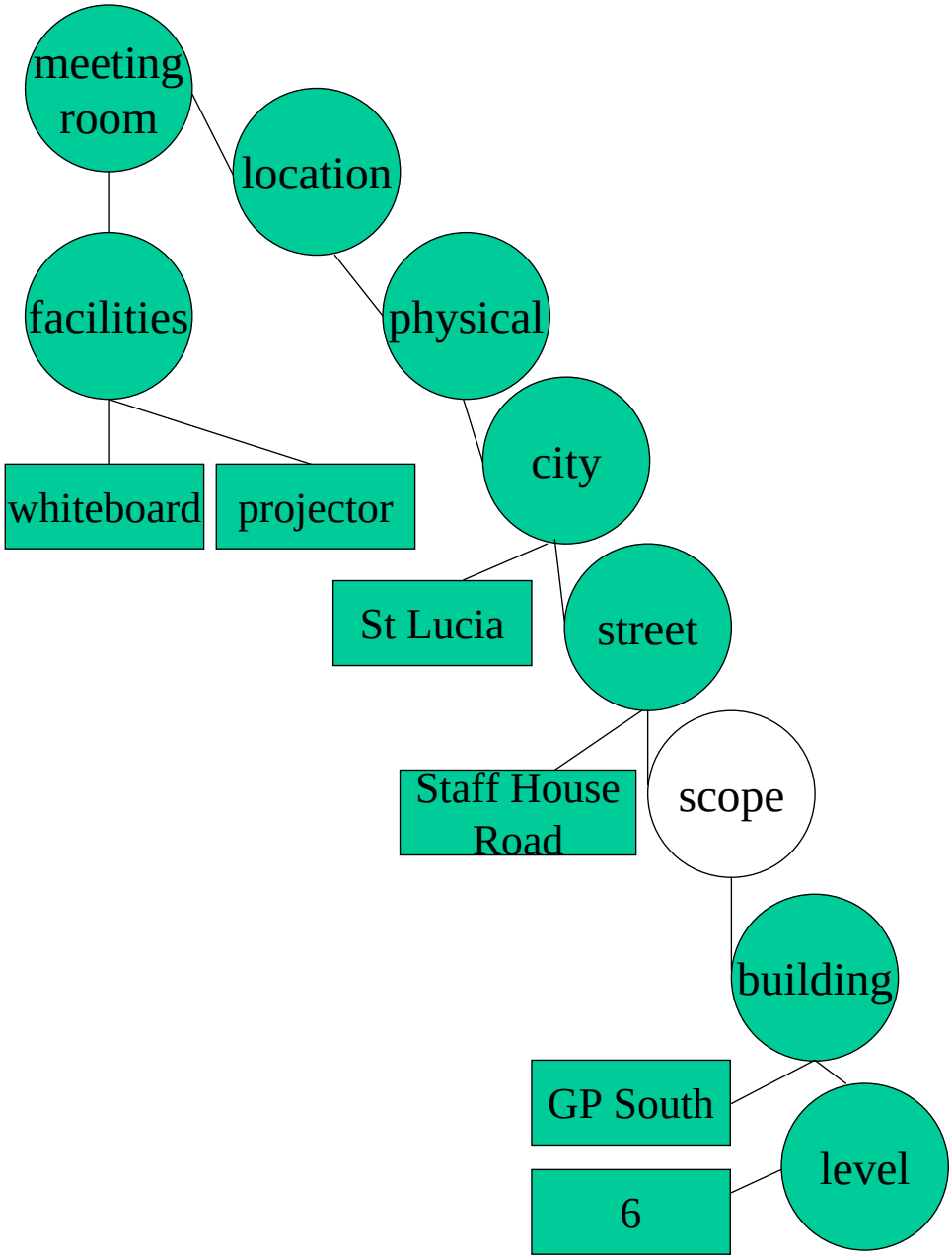


Figure 4.5: An example of the *scope* attribute.

The most common use of query relaxation is expected to be the weakening of location constraints, as is the case in Figure 4.5. However, scoping can also

be used to relax functional constraints. In attempting to discover a video screen, for example, the querier might wish to specify that a screen capable of 1024x768 resolution is preferred, but failing that, any screen will do. The *scope* element provides this behaviour. In conjunction with the preference mechanism described in Chapter 5, query relaxation is a powerful tool. One use of preferences is to control the selection of resources in the event that query relaxation is applied.

4.4.4 XML Mapping

The availability of tools for parsing and processing XML documents, and the wide acceptance of XML as a format for exchanging information, makes it a good choice for serialising Superstring descriptions. Furthermore, there are lightweight tools available for processing XML on small, limited capability devices, such as Scheemaecker's NanoXML [139] and kXML [140].

It is important to note that XML has not been chosen for its ability to be validated by a validating processor. Validation requires that the necessary schema and Document Type Definitions (DTDs) be retrieved, and that XML documents be checked for adherence to these specifications. Such validation comes at the cost of network and computational overhead, unless the set of schemata and DTDs is static, which prevents the creation of new types.

Rather, XML has been chosen because it naturally describes hierarchical relationships, which are key to Superstring's description language, and for the availability of tools to process XML descriptions.

The XML mapping defines five element types: *component*, *scope*, *any*, *not* and *exp*. Each of these element types is described below.

Component

The *component* element describes a component in the description hierarchy as discussed in Section 4.4.2. A *component* element has a *name* attribute and an optional *value* attribute. The *name* attribute defines the name of the component (for example, “camera”, “pixels” or “location”). The optional *value* attribute contains an exact value for the component (for example, “2”, “Australia” or “Bob”). An alternative to the value attribute is to define the value of the component as the content of the *component* element. For queries, if the value is an expression rather than an exact value, the expression appears as the content of an *exp* element, which is described below. Each component may have one or more child components, thus incorporating the hierarchical nature of Superstring descriptions.

Exp

Queries that define expressions must encapsulate expressions in an *exp* element. This is to enable query processors to distinguish quickly and simply between an expression and an exact value. An *exp* element must be a child of a *component*, *any* or *not* element. Its content is encapsulated in a CDATA section in order to prevent an expression being interpreted by the parser as XML markup. The *exp* element has no attributes.

Scope

The *scope* element plays the role of the description language attribute component of the same name. A *scope* element may be inserted as the parent of any *component* in a description except the root. The *scope* element has no attributes.

Any

The *any* element fulfils the role described by its description language namesake. The *any* element may be inserted as the parent of any *component* in a description except the root. It has no attributes.

Not

The *not* element fulfils the role described by the description language attribute component of the same name. The *not* element may may be inserted as the parent of any *component* in a description except the root. It has no attributes.

Examples

Some example descriptions, and a document type definition that formally describes the XML mapping, can be found in Appendix A.

4.5 An Unstructured Routing Layer

Unstructured (local-area) networks exhibit a wide array of behaviours, and encompass a large number of network protocols and device types, from personal computers and workstations to hand held computers, sensors and mobile phones. As stated below, these conditions present a stiff set of challenges to be overcome by a resource discovery protocol required to operate in such an environment. This section describes the design of a resource discovery protocol capable of operating in dynamic, unstructured networks.

4.5.1 Challenges

The local-area service discovery protocol must overcome problems of mobility and network heterogeneity. At the local level of a large-scale pervasive computing environment, devices are constantly joining and leaving the local group owing to a high degree of user mobility. Furthermore, these local groups may contain a wide array of computing devices, as well as disparate network protocols. If two devices *A* and *B* do not share a common network protocol, then it might still be possible for *A* to discover the resources offered by *B* if there is a device *C* which happens to connect to *A* and *B* using two different network interfaces. For example, if *A* uses IP over a wireless LAN interface, *B* uses L2CAP over its Bluetooth interface, and *C* can communicate with *A* using IP and with *B* using L2CAP, then there is a path from *A* to *B* via *C*. This *transitive* routing scheme is a key feature of protocols for unstructured networks, as it facilitates interaction between heterogeneous devices and networks.

A further challenge is to resolve queries quickly at low communication and computational cost. These requirements are particularly important in networks consisting of lightweight devices and where communication is expensive. As pointed out in Section 2.3.1, aiming for shortest path routing in highly mobile networks may incur unjustifiable message overhead, since the shortest path to a particular node may constantly change.

The next section presents a resource discovery protocol for dynamic unstructured networks, which addresses these challenges.

4.5.2 A Biologically-Inspired, Stochastic Resource Discovery Protocol for Dynamic Networks

The survey of routing protocols in Section 2.3.1 leads to the conclusion that there is no existing routing algorithm suitable for use by a service discovery protocol for dynamic networks. The following sections document the design of the local-area service discovery protocol.

Protocol Basis

The proposed query resolution process for unstructured networks has its basis in the ideas introduced in Chapter 3, and borrows from the concept of ant foraging and stigmergy. This idea plays a central role in the protocol, therefore the basic concept is reiterated here. Stigmergy is a method of indirect communication commonly found in nature. For example, ants are able to communicate the presence and direction of a food source by laying a pheromone trail. Pheromone trails are not only a mode of communication, they are also a means of optimisation [132]. Over time, the pheromone trail will approach the shortest path from the ants' nest to the food source by way of positive reinforcement. Several paths to the food source may be formed, but the shorter paths will be reinforced by more pheromone since ants taking this route complete their journey in less time, meaning more trips can be made. Other ants will choose the trail with the strongest pheromone scent, thereby further reinforcing the trail. Superstring's local-area query resolution process makes use of pheromone trails to guide query messages to nodes that are likely to be able to resolve those queries. Although a form of optimisation emerges from this process, it is not an optimisation of the path length to the resolver node. Rather, what happens is the system gains a bias toward popular queries and the resources they match, enabling future queries for those resources

to be resolved more speedily as time goes by. This increase in efficiency results from the fact that descriptions for popular resources are more widely known and queries for these resources therefore have shorter distances to travel (probabilistically) than queries for less popular resources.

The following paragraphs demonstrate the advertisement and query resolution processes of Superstring's local-area protocol. They also illustrate the mechanisms by which a Superstring node can locate other Superstring nodes so that it can participate in the Superstring service discovery protocol.

Overview

The resource discovery protocol for unstructured networks consists of three distinct processes: *joining*, *advertising* and *querying*.

Before a node can issue advertisements or queries, it must first become aware of the other nodes in its vicinity. This process is known as *joining*, and consists of nodes exchanging their node IDs. Superstring does not dictate the manner in which the globally unique identifier of each node is created, as long as it is 128 bits long and its universal uniqueness is guaranteed. Common tools for generating UUIDs utilise random-based UUIDs when there is a secure (non-deterministic) random number generator available on the system, but revert to time- and address-based UUIDs when a high quality random number generator is unavailable. Once the node ID has been generated, the node may issue advertisements for the resources it offers, and it may issue queries to discover the resources it needs. The *joining*, *advertising* and *querying* processes are defined below.

Joining

In general, the join process consists of sending a join packet (whether it be unicast to particular node, multicast to a set of nodes or broadcast to all neighbours within

radio range), and then waiting for a response, called a *peer* message. Once a node has joined, it must continue to listen on all network interfaces for join packets from other nodes. Figure 4.6 shows the general form of the join protocol pictorially.

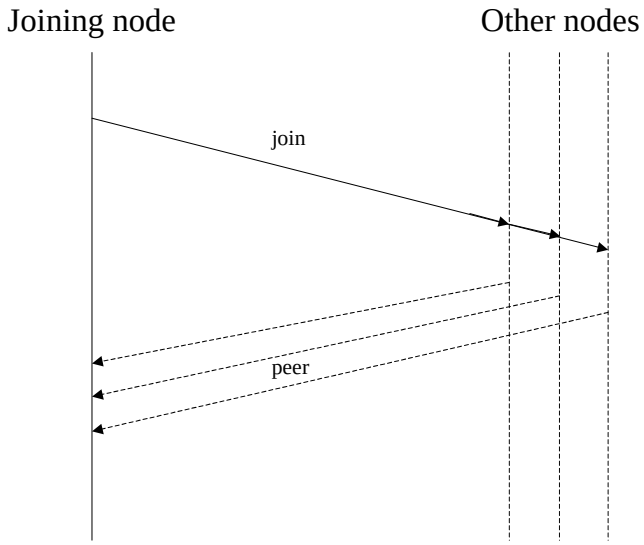


Figure 4.6: The general behaviour of the join protocol. Details differ depending upon the underlying transport.

As stated, pervasive computing environments are heterogeneous in terms of the kinds of devices present within them and the protocols used to communicate between those devices. Thus, the means by which one Superstring node discovers another Superstring node is partly dependent upon the underlying protocols available to it and the nodes surrounding it. So, for example, a device using its Bluetooth L2CAP interface will discover its neighbours in a different way to an IP capable node. To solve this problem, Superstring abstracts away the underlying protocol layers by providing a generic interface for joining (as well as advertising

and querying). The details of the underlying protocol layer are handled by a protocol specific module, which can be loaded when required. The individual transport modules provide a means for sending and receiving messages. They also possess a means of locating neighbouring devices. The exact process by which neighbour discovery occurs is dependent on the transport. It may incorporate several alternative mechanisms. For example, in an Internet Protocol environment such as a LAN, IP multicast discovery using UDP packets may provide one means of neighbour discovery, whilst an alternative means may be a hard-coded list of addresses or an LDAP directory search.

The actual message conveyed in a join packet is defined by Superstring. This packet is shown in Figure 4.7. Any nodes that receive this message respond with their node ID and, if known, the address of the node to which wide-area queries should be sent. This address may correspond directly to a wide-area resolver if the wide-area resolver and the joining node can communicate directly, or it may be the address of a peer node which has a route to the wide-area node.

Neighbour dropping or minimum spanning tree algorithms may be applied to limit the number of neighbours each node must have knowledge of. In many scenarios, the network is either so small, or so dynamic, that the additional overhead incurred by executing these algorithms is not justifiable. Such algorithms are well-known [141, 142], and are beyond the scope of this thesis.

Note that there is no equivalent process for leaving the environment. Nodes must detect failed peers or peers that have left the network. As with the joining process, the exact manner in which failed nodes are detected is partially dependent upon the underlying protocols. When a peer is detected as having failed or left the network, its node ID is removed from the list of neighbours.

JOIN	Local Address	Local Port	Is Wide-Area	Wide-Area Address	Wide-Area Port
String	String	int	boolean	String	int

Figure 4.7: An example join packet format. This is the format used by the UDP transport.

Advertising

After joining, a node can advertise any resources it wishes to share. An application is responsible for determining whether the resources it is offering should be advertised to the world at large, or whether the advertisement should be constrained to the local environment. If the resource is to be propagated to the wide-area, then a flag is set in the advertisement packet. Within the local environment, the advertisement is sent to each neighbouring node and propagated further for some number of hops, as decided by the application. Upon receiving the advertisement, a node may optionally prune the advertisement before storing it and forwarding it to the next hop. This feature allows the protocol to scale to large numbers of resources, even in environments that contain devices with limited storage capabil-

ities. In this fashion, very general queries can be resolved by nodes distant from the node whose advertisement matches the query, whilst more detailed queries will be propagated toward the advertising node.

Each advertisement contains a message ID, a hop counter, a URL, a resource ID and the advertisement itself, as shown in Figure 4.8. Each node that receives the advertisement decrements the hop counter and forwards it. If the hop counter reaches zero, then it is not propagated any further. The message ID allows a node to detect whether it has already seen this advertisement. If it has, it discards the advertisement without propagating it.

ADVERT	Message ID	Hops to Live	Is Wide-Area	URN	Resource ID	Resource Desc.
int	int	int	boolean	String	String	bytes

Figure 4.8: The advertisement packet format.

The cost of advertising is dependent on the per application choice of the hop limit. A larger hop limit means that queries will be resolved faster, but overall storage costs will be higher. The hop limit may be initialised to zero, in which

case the application is indicating that the description should not be advertised beyond the local node. The analysis of the protocol in Chapter 7 suggests an optimal value for the hop limit.

Querying

Once a node has joined, it may issue queries to neighbouring devices. A query initially proceeds as a random walk through the group of local nodes. Each node attempts to resolve the query. If it does resolve the query, then a query response is formulated and returned via the path taken by the query. Each node along the return path caches the resource description returned in the response. In this way, the matched resource becomes more widely known in the local group. Eventually, the response reaches the querying node. Lv et al. [143] call this *path replication*. Unlike peer-to-peer file sharing networks, where the average node has relatively abundant storage space, a typical node in a pervasive computing environment may be extremely lightweight. Storage space is at a premium. For this reason, Superstring couples path replication with a description *pruning* mechanism. In pruning, a hierarchical resource description is *generalised* by removing all those attributes and values deeper than a configurable threshold in hierarchy. Often it is sufficient to prune all those components other than the root. This leads to a situation in which a *pheromone trail* is created, rather than a situation in which entire descriptions are replicated at each node. Two variants of this strategy exist.

Pre-pruning In pre-pruning, a node makes the decision to prune the description as soon as it is received from a neighbouring node. The node then stores and forwards the pruned description. A slight variation on this is to cache a pruned version of the description but to forward the entire description, thereby allowing each node along the route back to the querier make its own decision as to how much of each description it is capable of storing.

Pre-pruning can lead to under-utilisation of the available storage space.

Post-pruning An alternative pruning strategy is possible, whereby complete descriptions are always returned along the entire path to the querying node. Then, when a node becomes overwhelmed with descriptions, it may prune the descriptions to create more storage space. Such an algorithm makes maximum use of the storage available, and reduces query resolution times early in the system lifetime, since queries can be fully resolved at nodes closer to the querier. These two features (maximum utilisation of the available storage space and faster resolution earlier in the piece) make post-pruning a better alternative in most situations.

Subsequent queries proceed as normal (via a random walk) until they encounter a node with a full or partial description. If a node contains a description that fully matches a query, whether it be a full or partial description of resource, then the query is considered resolved, and a response is returned as described above. If the query only partially matches (that is, every component in the description matches a component in the query, but there are still some unresolved components in the query), the node forwards the query to the neighbour from which the partially matching description arrived. In this way, the query is forwarded closer to a node which may have a fully matching description.

In Superstring, random walks are loop-free. This is achieved by adding a Bloom filter [66] to the query message. The Bloom filter keeps a summarised record of the nodes which have previously been traversed. Prior to forwarding the query to a neighbour, a node adds the neighbour's node ID to the Bloom filter. If any bit in the filter changes, then the neighbour has definitely not seen this message before. However, if the filter does not change, then it is *probable* that the neighbour *has* seen this message before. In this case, the query is not forwarded

to this neighbour, and another neighbour is selected. If all neighbours have been exhausted, the query is returned to the neighbour from which it came.

A Bloom filter is a vector *v* of *m* bits. A set of *k* hash functions, each of range 0 to $m-1$, is chosen. An element *e* is added to the filter by hashing it (or something representing it, such as a string identifier) with the first of the *k* hash functions, h_1, and setting the bit at position $h_1(e)$ in *v* to 1. This process is continued for each of the *k* hash functions. To check whether an element *t* is in the filter, the values $h_1(t)..h_k(t)$ are computed. If the bits at each of these *k* positions is set, then it is likely that *t* is known (that is, *t* was added to the filter previously). However, it is not a 100 percent guarantee that *t* is known, since it is possible the bits at these positions were set due to adding other elements to the filter. On the contrary, if any of the bits at the *k* calculated positions is 0, then *t* is certainly unknown.

The use of Bloom filters compounds the chance that a query will return false negative results. However, this chance can be minimised to the point of insubstantiality by an appropriate choice of Bloom filter size. Fan et al. [144] show how Bloom filters can be used by web caches to summarise the list of URLs that have been cached by peer caches, thereby reducing bandwidth consumption and computational costs compared to previous cache protocols. In developing this solution for web caches, the authors suggest configurations for the size of the filter and the number of hash functions to use. These results can be leveraged by Superstring's routing protocol for unstructured networks. Fan et al. suggest that a filter size between 8 and 16 times the average number of documents in the web cache provides good results, if four or more hash functions are used. In other words, the probability that the caching algorithm responds with a cache hit for a neighbour cache, when in fact a cache miss should have resulted, is small when the filter size is adequately large. The magnitude by which the filter size differs from the average number of cached documents is known as the *load factor*. In Superstring,

the size of the filter in a query message is dependent not on the average number of cached descriptions, but on the number of nodes in the network, since it is node addresses that are added to the filter. Based on the results of Fan et al., intuitively an adequate size for the filter would be between 8 and 16 times the *total* number of nodes in the network. Thus, for a network of 128 nodes, the Bloom filter would need to be between 1024 and 2048 bits long! Fortunately, as the analysis in Section 7.2 shows, on average, only a small fraction of the total number of nodes are visited for each query. Therefore, the filter can be much smaller in practice. As the probabilities of false positive results are thoroughly examined by Fan et al. and Bloom discusses the computational costs of his filter at length [66], these topics will not be further expounded here.

Node IDs are used as the basis for the hash function in Superstring. A Superstring node adds a neighbour node's address to the Bloom filter in the following way. Superstring node IDs are 128 bits in length. The number of hash functions, k, is always four (note that this can be made variable as well, at the cost of inserting a field that defines the value of k into the query message), as this number of hash functions is shown by Fan et al. to be a good compromise. The size, m, of the bit vector in the query header is variable, and can be chosen independently by each node. The four hashes are produced by taking the first, second, third and fourth disjoint 32 bit sequences from the node ID. The integer value of each 32 bit sequence modulo m determines which bit in the m bit vector to set.

In addition to the Bloom filter, each node through which a query passes maintains a record of that query, so that a query response may be routed along the return path. This record keeping also caters for the special case in which all bits in the Bloom filter are set to 1, in which case the Bloom filter is ignored, making routing loops possible. In these cases, the loop is detected after it has occurred, and a *loop detected* response is returned to the previous node.

The hop counter is decremented for every forward movement of the query. That is, the counter is not decremented when the query backtracks due to a dead end.

Random walk algorithms have been previously utilised and studied as a means by which to search a network. Lv et al. [143] found that random walks are less expensive than flooding algorithms in terms of the number of messages required to locate the desired resource. However, this saving comes at the cost of higher query latency. Lv et al. also show the benefit of replicating resources in the network to improve query latency and to increase the success rate of queries. Due to the stochastic nature of the query routing algorithm, the performance analysis of the protocol is complex. A rigorous analysis is deferred until Chapter 7, which includes a mathematical model and experimental results.

Figure 4.9 shows the contents of a query packet. Another packet format is used for responding to queries, and this is shown in Figure 4.10.

Several possible values exist for the *Response Type* field. The first possibility is *RESOLVED*, indicating that an exact match for the query was found. Other possibilities include

DEAD_END the node has no further neighbours;

LOOP_DETECTED the node has seen this query previously;

HTL_EXCEEDED the hop counter has expired; and

RELAXED one or more matches were found after the query was relaxed.

If the originating node receives any of the responses in the above list (excluding *RELAXED*), and it has exhausted all its neighbours or the hop counter has expired, then the response type is converted to *UNRESOLVED* and returned to the user. Query relaxation is achieved in the unstructured environment in the following manner. During the processing of a query, if a service description matches a

QUERY	Bloom Filter	Query ID	Hops to Live	Is Wide-Area	Querier	Query Desc.
int	bytes	int	int	boolean	String	bytes

Figure 4.9: The query packet format.

query once the query has been relaxed, then a pointer to the description is stored by the processing node, and the query is forwarded along the pheromone trail (if a trail exists). If no trail exists, the query is forwarded randomly in the usual manner. If a *RELAXED* response is returned, then the contents of the response are merged with the descriptions pointed to by the stored pointers, and the *RELAXED* response is forwarded along the return path. If a negative response is returned, then the node returns a *RELAXED* response containing the descriptions which are pointed to by the stored pointers. This method ensures that exact matches are not pre-empted by relaxed matches. That is, there is opportunity for exact matches to be found *after* a relaxed match has been found.

A potential problem with this protocol is a phenomenon referred to as *resource hiding*. The creation of pheromone trails leading to one resource may, in effect,

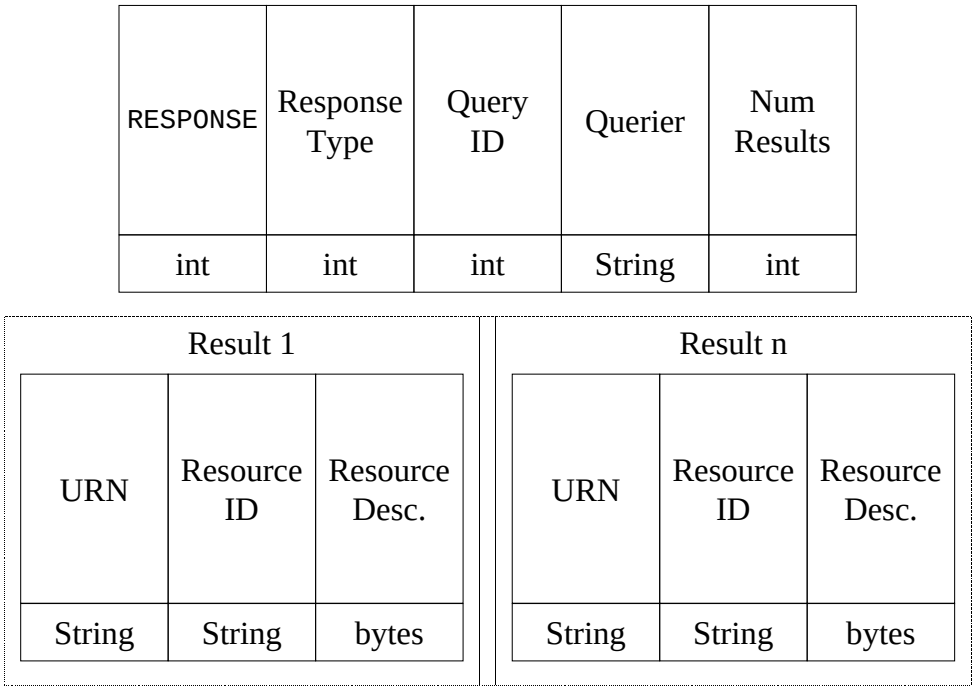


Figure 4.10: The response packet format. The presence and number of result segments are dependent upon the value of the *num results* field.

prohibit the discovery of other resources that would match a query. The reason for this is that a pheromone trail guides a query along a particular path until a match is found (or until it is concluded that the resource at the end of the trail is not a match). This means that resources lying to either side of the pheromone trail will not be discovered, even though they may fulfil the query. Ants themselves provide a possible solution. Deneubourg et al. [145] found that ants sometimes fail to follow the instructions given to them by their fellow ants. Instead of following a pheromone trail, an ant will sometimes wander randomly, thereby facilitating the discovery of new food sources. This behaviour can be adapted to the resource discovery protocol. With small probability, after a query is successfully resolved and a response is returned, the query does not terminate. On the contrary, it continues

on a random walk until it finds another matching resource or until its hop counter expires. A trade-off occurs between the swiftness with which a hidden resource is found, and the extra overhead and load placed on the network. The probability setting acts as a dial to control the trade-off. If the probability of a successfully resolved query carrying on is high, then overhead is increased, but hidden resources are found quickly. If the probability is low, overhead is minimised, but it takes longer for hidden resources to be located.

4.5.3 Summary

The query routing protocol for unstructured networks leverages complex systems theory to provide a lightweight, flexible and adaptable routing layer.

Although this protocol is inherently stochastic and cannot guarantee the discovery of a service (if it exists), in practice a service will always be discovered if the query hop limit is greater than the total number of nodes in the network minus the number of nodes that are covered by the initial advertisement of the service. As the network evolves due to the laying of pheromone trails, the required number of hops becomes even smaller.

Furthermore, note that values for the advertisement and query hop limits can be selected so as to be appropriate for particular applications or environments. For example, if there are n nodes in the network, then by ensuring a description is advertised to at least $n/2 + 1$ nodes and queries are propagated to up to $n/2 + 1$ nodes then the protocol defines a quorum-based query routing layer. Other patterns and behaviours may emerge by adjusting the hop limits, but they are not investigated further.

4.6 A Structured Routing Layer

4.6.1 Challenges

If one of the goals of an unstructured service discovery protocol is to enable dynamic groups of devices to be formed and to share their resources, then analogously, a goal of a wide-area service discovery protocol is to facilitate the interaction of these dynamic pockets of devices. Wide-area networks are generally characterised by a much greater degree of structure and stability than their local-area counterparts. Furthermore, today's wide-area networks share common protocols (TCP/IP), making them less heterogeneous (from the application layer's point of view) than local-area environments. These favourable conditions mean that the focus of service discovery protocols for structured environments is on scaling to large numbers of nodes and resources. How many services and resources might exist in the global pervasive computing environment? In October 2004, there were approximately 56 million web hosts [146] serving a total of around 4.3 billion web pages [106]. Although it is unlikely that there will ever be a need to advertise individual web pages so that they can be found by a service discovery protocol, these figures provide some understanding of the vast number of resources that could potentially exist in a global pervasive computing environment. So if web pages are not the typical example of a pervasive resource, what is? More particularly, what kinds of resources will be advertised to the wide-area? The wide-area pervasive computing environment will consist of a multitude of different kinds of resources, including web services, publicly accessible webcams, non-computational resources (such as restaurants and libraries), groups of remote sensors (such as those that might be deployed in a scientific experiment or in a military operation), grid-computing services (supercomputers, file stores or grid application providers) and items of multimedia. This is just a small selection

of the numerous types of resources that could constitute a wide-area pervasive computing environment of the future. The greatest challenge, then, is to accommodate these vast numbers of resources and the large numbers of nodes that will be required to store the descriptions of these resources. The descriptions should be distributed evenly across the nodes, as should the work involved in resolving a query.

4.6.2 A Deterministic Resource Discovery Protocol for Wide-Area Static Environments

Overview

The resource discovery protocol for structured networks is, at an abstract level, similar to the protocol designed for unstructured networks in Section 4.5. It shares the same three basic processes: *joining*, *advertising* and *querying*. However, the similarities end here. The protocol for structured networks takes advantage of static elements in the network, allowing querying to be deterministic: the wide-area protocol can guarantee no false negative responses.

The resource discovery protocol for structured networks is a hybrid protocol, defining flat, peer-to-peer interactions and hierarchical structures. Peer-to-peer routing is performed using the Chord DHT algorithm. Chapter 3 proposed the use of a scale-free cache to improve the performance of Chord. The design of this service discovery protocol is not predicated on the use of the scale-free version of Chord. Any Chord implementation can be used. Note that the modified version of Chord presented in Chapter 3 is totally compatible with the original Chord protocol. The following sections detail the wide-area protocol.

Joining

A new node in Superstring joins the wide-area network of resolvers using the underlying DHT join algorithm. The Chord join algorithm relies on the new node having prior knowledge of at least one Chord node that is already a member of the resolver network (as discussed in Chapter 6, the Superstring prototype relaxes this requirement in some scenarios by including a simple bootstrapping mechanism that uses IP multicast). The joining node computes its own ID (this can be a hash of its IP address and port number) and asks the existing Chord node to lookup the *m* entries that should constitute its finger table (as described above). Once this is done, the new node contacts those nodes whose finger table must change to reflect the addition of the new node. Initially (for a time equal to the lifetime of a description), all queries routed to the new node are forwarded to the node's successor, since the successor would previously have been responsible for these queries. Once the new node has been a member of the DHT for a period equal to the soft-state timeout, the new node need not forward messages to its successor, since by this time all descriptions should have been refreshed by advertising nodes. This initial span of time is called the *period of uncertainty*.

Advertising

Although it is possible for an application that uses the service discovery protocol to be executing on the same node as a wide-area resolver, the more common case is that a wide-area resolver receives an advertisement from a node that is executing the local-area protocol. As described in Section 4.4, Superstring descriptions are hierarchical. The ingress resolver (the resolver on which the advertising process is executing or the router that receives an advertisement from the local-area) calculates a key using the root component of the description. The resolver responsible for this key, known as the *root resolver* for this description, is located via the

Chord lookup algorithm, then the entire description is sent to this resolver. A core goal of Superstring is to distribute query workload. To achieve this, upon reaching the root resolver, the description in the advertisement is decomposed into the sub-trees constituting the description hierarchy. These sub-trees are distributed to other resolvers in such a manner that a hierarchy of resolvers is formed, which mirrors the description hierarchy.

When the root resolver receives an advertisement, it removes the root component of the description, thereby forming a set of sub-trees. These sub-trees are forwarded to child resolvers. Each child resolver performs the same procedure, removing the root component of the (sub-)description and forwarding the sub-trees to child resolvers. The process continues until the leaves of the description (the value elements) are reached. This process creates a hierarchy of resolvers whose topology matches that of the original description.

Each root node is free to use its own method of locating the nodes which will form the resolver hierarchy for a particular description. One method of forming a resolver hierarchy is to calculate a key for the root of each sub-tree, and then forward these sub-descriptions to the Chord node responsible for these keys. The key and address of the corresponding child resolver are stored by the parent resolver.

This method has the problem that, if the DHT is deployed at a global level, the different parts of the description can find themselves on resolvers all over the world, leading to very high query latency. Alternatively, the resolver hierarchy could be formed from the set of nodes constituting a local Chord network. That is, the root node could be part of the global DHT *and* a local DHT, thereby forming a two-tier wide-area network (the Superstring resolvers in the local DHT, in addition to the nodes executing the local-area protocol that are connected to one of the local wide-area resolvers, are analogous to autonomous systems in the Internet). In this configuration, a description is distributed over a set of local nodes,

improving query latency, but retaining the workload distribution characteristics. A final alternative is not to use Chord during formation of the resolver hierarchy. In fact, if the root node is not overly burdened, it may choose not to use a resolver hierarchy at all; in these cases, it is useful to imagine that the root resolver utilises a *virtual* resolver hierarchy, which is managed internally. As such, a resolver hierarchy may be physical or logical, and the choice is transparent to other nodes.

Superstring is a soft-state protocol, meaning that resources must be periodically refreshed. If an advertisement is not updated or refreshed within the description lifetime period, any components from that advertisement stored in resolvers are purged. Each resolver in the system manages its own timers, and therefore autonomously decides when to purge description components and the name records associated with them.

To advertise a description (or sub-description), a TCP/IP connection is made to the root resolver (or child resolver), and a message adhering to the format shown in Figure 4.11 is sent.

Figure 4.12 depicts the manner in which advertisements (and queries) are routed in the structured routing layer. First the root resolver for the description in question is located via the DHT algorithm. Then the description is propagated down the resolver hierarchy in the manner described above.

Querying

The query process is very similar to advertising. Upon receiving a query, the ingress resolver calculates a key from the root element of the query and sends the entire query to the corresponding root resolver. If there is a value or expression associated with the root component of the description, the value is matched or the expression is evaluated against values stored at the resolver for this component. If there is a match, the root element of the description is removed, yielding a set of

ADVERT	URN	Resource ID	Resource Description
int	String	String	bytes

Figure 4.11: An example join packet using the UDP/IP multicast. In general, join packets for any transport protocol will have a similar format.

sub-descriptions. After calculating a key for a sub-description, the resolver checks to see if there is a mapping for this key to a child resolver. If there is, the sub-description is forwarded to the child resolver. If there is not, then there must not be a matching advertisement, and the query fails. After forwarding sub-descriptions to its child resolvers, the parent resolver waits for a response from each child resolver. Each child resolver performs the same procedure outlined above. The child resolver replies with a set of name records and descriptions for matching resources. The parent resolver performs a set intersection of the name record sets returned by its children, and then returns the yielded set of name records to its parent. The root resolver performs the same process, but returns the resulting set to the ingress resolver. The ingress resolver returns the results to the querier.

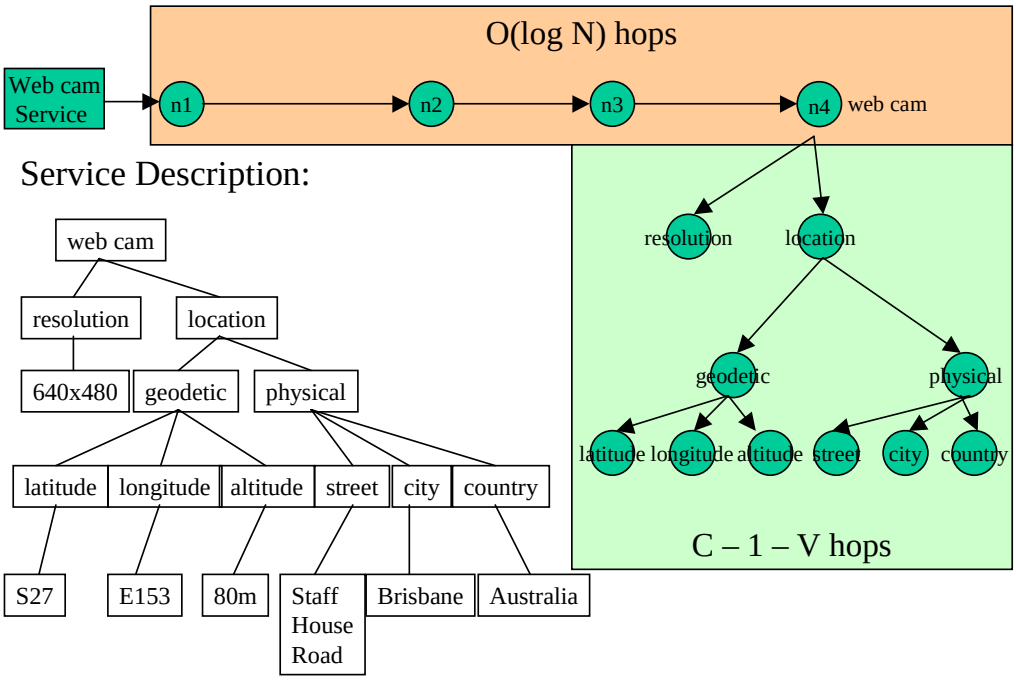


Figure 4.12: Routing in the structured routing layer. N is the number of nodes constituting the DHT, C is the total number of components (including values) in the description and V is the number of values in the description.

A parent node stores a superset of the name records stored by its children. Query relaxation is performed by intersecting the result sets of the child nodes, then calculating the intersection of the resulting set and the set of matching name records stored at the parent. If the resulting set is not empty, it is passed up the hierarchy. If the resulting set *is* empty, then the parent passes its set of matching name records up the hierarchy. Thus, query relaxation is implemented cheaply and easily.

Queries must adhere to the message format depicted in Figure 4.13. Queries to child resolvers conform to the same format, but they contain a different operation

code. Queries to child resolvers are *not* forwarded to their successor nodes during the initial period of uncertainty described in Section 4.6.2. Resulting query responses use the message format shown in Figure 4.14. Since the protocol is connection-oriented, and query responses are returned over the connection opened by the querier, the operation field is omitted from the response format. Instead, the first field is the response status code, which is either *RESOLVED* or *UNRESOLVED*.

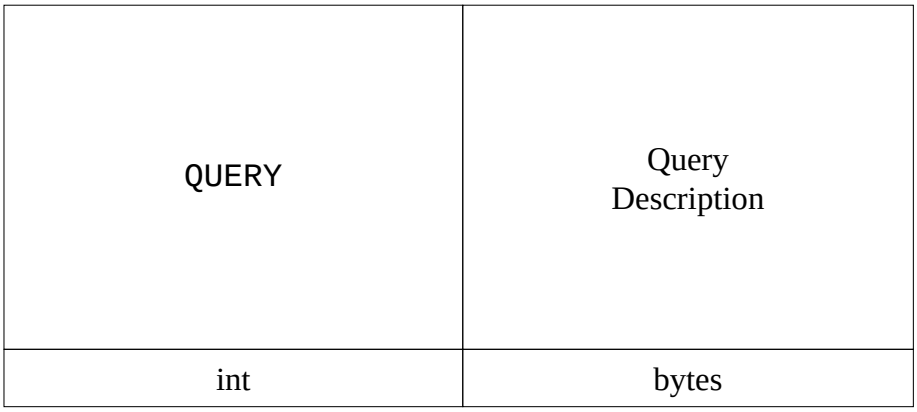


Figure 4.13: The query format for the wide-area.

Reasoning

The reasoning behind this hybrid DHT/hierarchical approach is to provide better scaling of storage and workload costs in situations where advertisements and queries for particular types of resources are more common than others. Studies

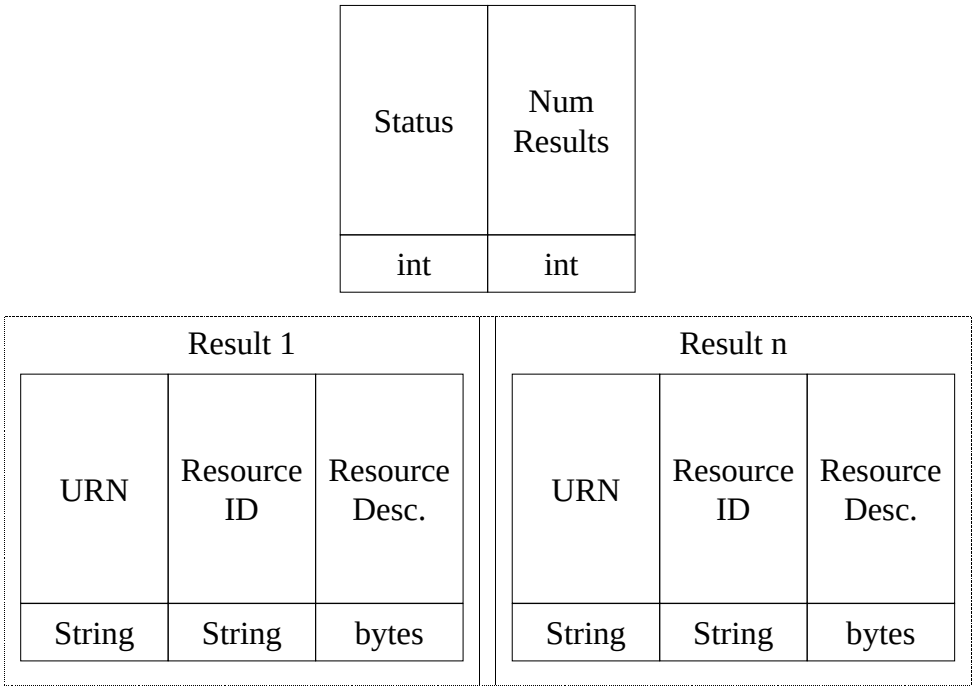


Figure 4.14: The format of a query response in the wide-area. The presence and number of result segments are dependent upon the value of the *num results* field.

have shown that queries to web search engines such as Google [106] and queries in file sharing networks such as Napster [33] and Gnutella [34] follow a power-law or Zipf distribution, rather than a normal distribution [147, 148]. Superstring is designed to acknowledge this query (and advertisement) imbalance, and distribute the workload and storage costs in an equitable fashion. Not only is this important for reasons of scale, but also for reasons pertaining to organisational politics. For instance, if Superstring is deployed within a university, storage and workload costs can be shared equitably between all the departments constituting the university, assuming that resolver hierarchies may be constructed from all Superstring resolvers in the university. Such a scenario provides much fairer sharing of costs even in the face of heavily biased query distributions. Chapter 7 proves

this assertion.

Brief Analysis

In this section, a worst case analysis of the number of nodes that need to be contacted during advertising and querying is provided. A more in-depth analysis of the storage (and by implication, the workload requirements) is deferred to Chapter 7. This analysis considers the case where the description is distributed over a hierarchy of resolvers such that each component of the description is stored on a different resolver in the hierarchy. This scenario represents the worst case in terms of the number of resolvers that need to be contacted and the number of messages that need to be sent. However, it represents the best case in terms of the amount of data stored by each resolver and the amount of work done by each resolver.

The Chord lookup algorithm can locate a node within $logN$ hops, where N is the number of resolvers in the network. Therefore, the cost of locating the root resolver is $logN$. The number of nodes contacted during construction of the resolver hierarchy during advertising or during a query is dependent on the number of components in the description. If C is the total number of components in the description, including the root element, all interior elements and each leaf element (the values or expressions), then $C - 1 - V$ resolvers will be contacted during advertising or querying, where V is the number of value elements in the description. These observations yield the following bound on the number of nodes contacted during advertising or querying (see also Figure 4.12):

$$O((logN) + C - 1 - V) \tag{4.1}$$

The analysis assumes that Chord is not used during construction of the resolver hierarchy. If Chord *is* used during resolver hierarchy construction, then

advertising costs

$$O(logN(C-V)) \tag{4.2}$$

since it costs $logN$ to find each of the $C-V$ internal nodes (including the root node). The cost of querying does not change.

4.6.3 Summary

While the routing layer for unstructured networks draws upon complex systems theory, the structured protocol makes use of a distributed hash table, thereby gaining the property of determinism in the absence of failure. In other words, it can guarantee that queries will locate all matching resources. It allows the formation of hierarchies to distribute storage and computational costs.

The structured protocol can be used in many settings but is especially useful for use in the wide-area. The structured protocol will also be useful in scenarios that utilise a cluster of service discovery resolvers. In this case, the protocol automatically splits workload and storage costs across the cluster based on service type information. In effect, further scalability can be achieved by replacing a single resolver with a tight cluster of resolvers that utilise the structured Superstring routing layer to route advertisements and queries between them. Investigation of these various alternative deployment strategies are beyond the scope of this thesis.

4.7 Routing Layer Interaction

Nodes that execute the local-area routing protocol can interact with the wide-area if there are nodes in the local environment that execute both routing layers. Queries and advertisements from the local-area that are bound for the wide-area are received by these dual routing layer nodes and forwarded to the wide-area. The manner in which this is achieved is implementation dependent. The prototype

documented in Chapter 6 takes advantage of the modular transport architecture of the ant-foraging protocol to define a small pseudo-transport that simply passes messages from the local-area layer to the wide-area layer of a dual-layered node. Figure 4.15 shows a general, high-level view of the configuration of a dual-layered resolver.

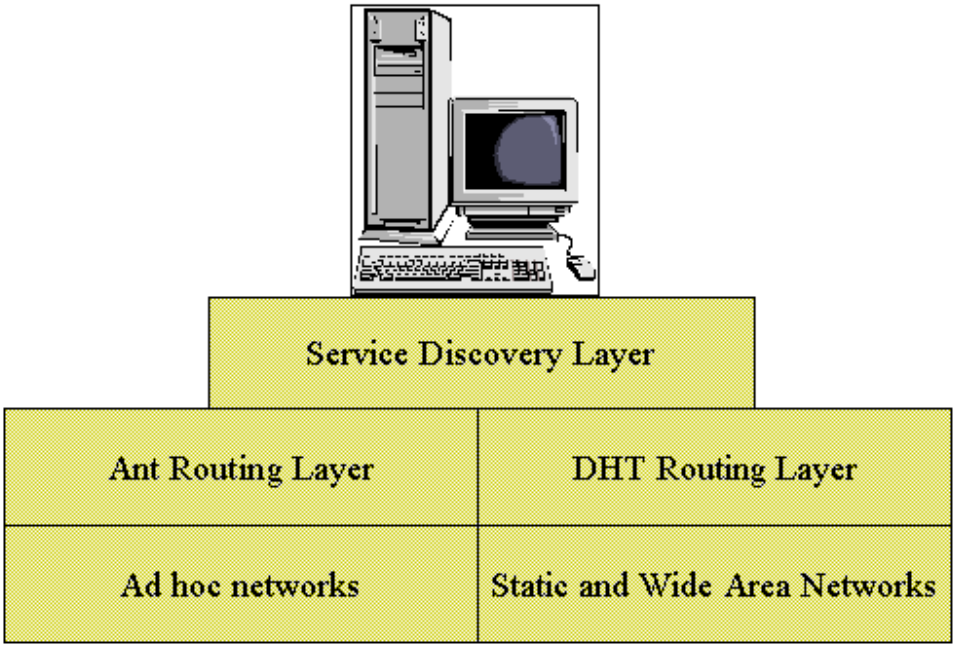


Figure 4.15: A dual-layered node.

Queries and advertisements are marked so that dual-layered resolvers are able to determine whether or not to forward a message to the wide-area. Applications determine whether queries and advertisements are relevant beyond the local-area, and so they are responsible for marking messages correctly. The Superstring API provides a mechanism to create queries and advertisements, and to mark their relevance as *local* (default) or *wide*.

As Figure 4.1 shows, those local-area nodes that cannot communicate directly with a wide-area resolver may do so via one or more intermediary local-area

nodes.

4.8 Summary and Conclusions

The problems identified in Chapter 2 informed the design of Superstring as detailed in this chapter. There are several key design choices in Superstring:

- separation of routing layers from the application interface;
- a structured protocol for static structured networks;
- an unstructured 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 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.

A recent trend in resource discovery is to utilise relatively heavyweight description languages such as RDF [58] and OWL [55], 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 [60], add another layer on top of OWL, introducing further computational costs. Furthermore, existing inferencing engines are known to scale poorly [60].

Superstring *bucks the trend*, by choosing a much lighter-weight description language. 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.

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.

CHAPTER 5

Context-Sensitive Extensions and a Preference Model

The previous chapter defined a scalable resource discovery protocol that is capable of operating within heterogeneous computing environments. With some small additions to the core protocol, Superstring can be made to support context-sensitive queries, advertisements and result-ranking. In very dynamic environments where traditional approaches to context management [109, 149, 150] are not suitable because they rely on centralised or infrastructural components, these context-sensitive extensions to Superstring can be utilised to inform applications of changes in context (the situation within which the application is executing).

5.1 Introduction

The set of services available to a particular mobile application changes over time. Furthermore, the *contexts* in which applications and services execute alter as time passes. The ability to behave sensibly in the face of transmuting contexts presents a major challenge to applications. Applications that adapt to changing contexts

are said to be context-aware.

A change in the computing environment or circumstance in which an application finds itself is known as a context change. The types of context are virtually unlimited, but commonly used context types include location and quality of service attributes.

Thus, in pervasive computing environments, scalability and heterogeneity are not the only issues to be overcome by resource discovery protocols. They must also cope with frequent changes to the situation of the querier or the resources being queried. These environments call for more flexible queries and advertisements whose contents can dynamically adapt to the current situation. For example, an application may wish to locate a resource which is co-located with some other resource, where neither of the resources' locations are known at the time of querying.

In service discovery it often occurs that a query will match many resource descriptions. The name records for all these descriptions will be returned to the querying application. How does the application choose the best match (or the set of best matches) from among these results? Must this selection be left to the querying client, or can it be carried out by the query resolvers? A bandwidth saving is achieved by having only the required number of matches returned to the querier rather than all matches.

It is not only query results that may require ranking. On a device with multiple network interfaces, such as a mobile phone with GPRS and Bluetooth connectivity, applications or users may have reason to favour one interface over another for certain tasks. Applications and users should therefore be able to specify preferences that influence the behaviour of the service discovery protocol.

This chapter describes a context-sensitive framework for service discovery, as well as a preference model that can be used to rank query results. The chap-

ter is organised as follows. Section 5.2 motivates the need for context-sensitive resource discovery and a context-sensitive preference model that can be used to rank query results according to the current situation. Section 5.3 details the design of the context-sensitive extensions and a preference model and language. In Section 5.4, an application that utilises the extended features introduced in this chapter is presented. Section 5.5 compares this context-sensitive framework with the context-sensitive service discovery protocols introduced in Section 2.5. Finally, conclusions are presented in Section 5.6.

5.2 Motivation

5.2.1 Context-Sensitivity

Context is a major thread of research in the fields of pervasive computing and artificial intelligence. As noted in Section 2.5, context can be defined as any information pertaining to the environment or circumstance surrounding an entity (such as a person, software component, application or device) [108]. A context-sensitive or context-aware application is one that adjusts its behaviour when the context changes.

Some context-aware applications gather and manage context information themselves. Cyberguide [151] is an example of this. It uses location context information to adapt its behaviour. In an early prototype, this location information is gathered directly from infra-red beacons, but other Cyberguide prototypes can be built to gather location information from other sources such as the Global Positioning System.

Another approach is to store, manage and evaluate context information in a shared repository. Henricksen and Indulska [152] define one such framework. The repository gathers context information from the environment by way of sensors

and other inputs and manipulates it into a higher level abstraction called *situations* that can be used by applications. Applications describe the situations that are of interest to them, and they can be notified of situation changes. Usually, however, applications interact with situations and context by way of a preference model. Applications define preferences in terms of situations and associated ratings.

Resource discovery protocols are well placed to provide a context gathering service for context-aware applications and frameworks. Knowledge of the availability of resources in a pervasive computing environment is a form of context information. Thus, Superstring and other resource discovery protocols can play an important role in context-awareness.

However, a context-sensitive resource discovery protocol could offer even more. If resource discovery protocols were context-sensitive, then the results of queries could be made dependent upon the context of the querier or some other entity. Furthermore, if queries were persistent - that is, if they remained in the network waiting to be matched against new or updated resource advertisements - then applications could be notified of these changes to the environment. In this fashion, resource discovery protocols can offer context-awareness in environments where traditional approaches are unsuitable. For instance, centralised context frameworks are unsuitable for mobile ad hoc networks.

This completely decentralised approach to context-awareness is useful for creating novel applications in highly mobile and ad hoc environments. Furthermore, this approach utilises only resource discovery resolvers; additional components are not required.

To illustrate the operation of such a protocol, the following example is provided. A streaming media application renders video and audio streams to *devices of opportunity*, shifting the streams from one device to another as required, using a technique called vertical handover [153]. Each device is abstracted as a number

of services, such as screen, audio and controls, and each service instance is associated with a single device. There are three conditions which must be met before a media stream can be handed over to another device:

1. the user must have permission to use the device;
2. the user must be co-located with the device; and
3. the service offered by the device must meet the requirements of the streaming media application.

In the absence of context-aware query completion, the application would be required to issue queries about the devices with which the user is associated, and then issue another query that selects the services which are associated with those devices. A query would be triggered every time the set of devices with which the user is associated changes (that is, when the user's location changes). Since the vertical handover could not be triggered until the query is complete, segments of the media streams would go unheard and unseen by the user. A service discovery protocol that offers persistent queries and context-aware query completion can offer a more satisfactory user experience.

5.2.2 Preferences

Queries should support preferences that can be used to rank query results in the case where multiple resources are matched, and only the best (as defined by some ranking function) are required. The ranking function can further reduce bandwidth utilisation in the mobile network, since the unwanted results will be culled at the query resolver. Preferences can also be used to make more intelligent routing choices in hop-by-hop protocols. For instance, in devices with multiple network interfaces, distance constrained protocols such as Bluetooth L2CAP may be used

in preference to wide-area protocols in some applications. However, this thesis focuses on the use of preferences in ranking query results. Ideally, preferences should also be context-sensitive, such that query result orderings can differ based on the current context of an entity or entities.

5.3 Framework

Recall that Superstring, as defined in Chapter 4, already includes the following features:

- query routing layers for structured and heterogeneous, unstructured networks;
- a rich, but lightweight description language; and
- query relaxation.

In this section, Superstring is extended with a framework for context-sensitive service discovery and preferences. The context-sensitive extensions provide additional *routing protocol behaviour* and *description semantics*, while the preference extensions provide *result-ranking and user preferences*. These are discussed below.

5.3.1 Protocol Behaviour

Persistent Queries

To the existing protocol behaviour of Superstring, the ability to *persist* queries in structured and unstructured networks is added. Persistent queries are akin to subscriptions in content-based routing protocols such as Elvin [94]. Query

persistence has a range of useful applications. Persistent queries, like advertisements, are soft-state. Therefore, querying entities are required to refresh persistent queries on a regular basis. When a persistent query is first injected into the network, any existing advertisements that match the query are returned in a synchronous fashion (as a result of the query operation). Services that match the query which are advertised after the query is first persisted are returned in an asynchronous fashion. The application is notified, by callback, of the newly matching services.

In the unstructured routing layer, the following question arises: to where and how far should a persistent query be propagated? This question does not arise in the structured routing layer since a specific resolver is always responsible for queries of a particular type. In the unstructured routing layer, there is a problem because if the persistent query is not distributed widely enough, it may never match an advertisement. It is important to realise that initially there may be no matching advertisement anywhere in the system, but matching resources may become available at a later time.

There is no convincing solution to this problem. One possible solution is to propagate persistent queries to the extent of the hop limit, thereby increasing the chance of matching future standard and transient advertisements (see below). If the size of the network is known a priori, then the hop limit can be chosen so as to minimise communication and storage overhead in the network as a whole while simultaneously guaranteeing a high probability of resource matching. The protocol analysis in Chapter 7 shows, in detail, how the hop limit can be chosen. Finally, it is possible to dynamically adjust the value of the hop limit of queries and advertisements as time goes by. The hop limit can be gradually increased in the event that resource matches are rare. If matches occur frequently, then the hop limit can be slowly decreased until the point where the frequency of matches

drops. This dynamic adjustment algorithm means that the network size does not have to be known a priori, but initially there will be a cost paid in terms of either excessive communication overhead or missed matching opportunities.

Transient Advertisements

For completeness, transient advertisements are included. Transient advertisements contain information which is meaningful only at the time of advertisement. Transient advertisements are not stored by resolvers in the network. Instead, resolvers attempt to match them against persistent queries, and then discard them immediately. In this respect, they are similar to notifications in asynchronous messaging systems such as Elvin. Transient advertisements are suitable for advertising frequently updated resource information, such as sensor data.

Summary

Thus the protocol now includes ordinary (transient) queries, persistent queries, ordinary (persistent) advertisements and transient advertisements. With this set of protocol primitives, powerful applications can be created, as shown in Section 5.4.

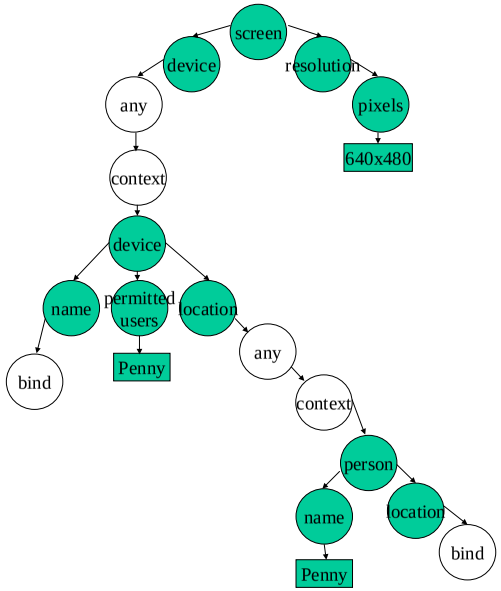
5.3.2 Context-Sensitive Resource Descriptions

The core Superstring query and advertisement features provide a level of expressiveness and flexibility not present in other service discovery protocols. To these core features are added, in a simple and seamless manner, facilities to provide context-sensitive aspects to the protocol. In addition, features that simultaneously conserve bandwidth even further than is possible with the core Superstring description language and that reduce processing burden on lightweight clients are defined.

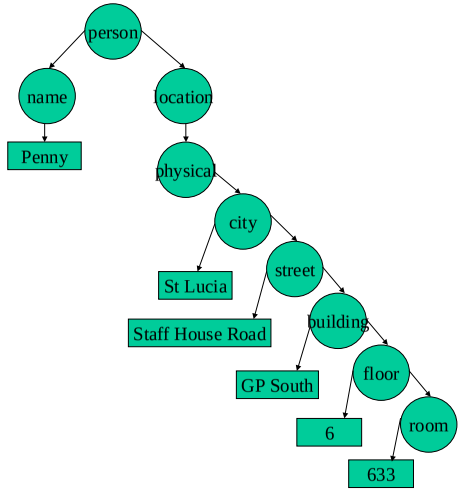
Sub-Queries

Context-sensitive queries and advertisements are realised through the addition of two special attributes to the hierarchical description model. 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 5.1 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. The other *bind* element dictates that the name of any matching device should replace the sub-query.

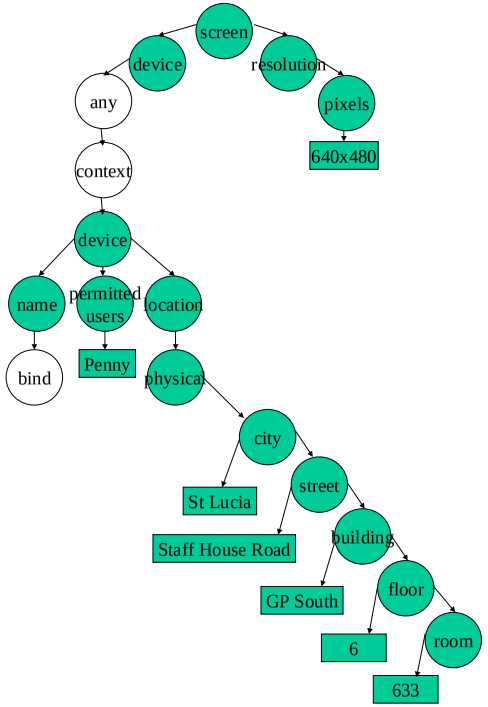
In addition to the *context* and *bind* attributes, the *size* attribute is also introduced. The *size* attribute can be coupled with the *context* attribute such that the number of sub-query results is bound instead of the results themselves. The utility of this attribute is demonstrated in the example application described in Section 5.4.



(a) A context-sensitive query whose sub-query is also context-sensitive.



(b) An advertisement that matches the deepest sub-query in (a).



(c) The context-sensitive query after the deepest sub-query has been resolved and the specified components bound.

Figure 5.1: Context-sensitive query resolution. The advertisement defined in (b) matches the deepest sub-query in (a). The *bind* tag in (a) specifies that the sub-description beneath the *location* component contained in (b) should replace the sub-query in (a), thereby yielding (c). The remaining sub-query in (c) would be similarly resolved.

In the structured environment, the root resolver responsible for the particular query being issued (inferred from the root element of the parent query) receives the context-sensitive query and first resolves any sub-queries by routing those queries to the appropriate resolvers in the usual way. The results are combined with the parent query as dictated by the *bind* element. In unstructured networks, context-sensitive queries are resolved as per ordinary queries. As the query traverses the network, the non-sensitive parts of the parent query are matched against stored descriptions. If a match is found, the sub-queries are issued and the results are combined with the parent query. The entire query can now be matched against stored descriptions. This algorithm ensures that sub-queries are not resolved unless the parent query has a chance of being resolved, thereby conserving bandwidth and power. For persistent queries, the behaviour is modified as indicated in Section 5.3.1.

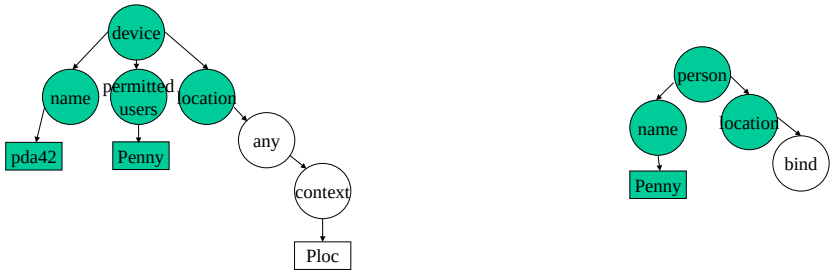
Named Queries

In lieu of specifying sub-queries in place, sub-queries may be pre-defined and associated with an identifier. These are known as *named queries*. Named queries allow context-sensitive queries to be designed in a more modular manner. The *context* attribute may take a single parameter called *name*. If this parameter is present, then its value is the name of a pre-defined query. Figure 5.2 shows a context-sensitive query with a named sub-query.

Furthermore, named queries are utilised by the preference language, described below, in circumstances requiring *context-sensitive preferences*.

5.3.3 Preference Model and Language

This section describes how Superstring is augmented with a mechanism to allow applications and users to specify preferences. Preferences can be used to rank



(a) A context-sensitive query with a named sub-query.

(b) The definition of the *Ploc* named query.

Figure 5.2: A named query. The *Ploc* named query appearing in (a) is a reference to the query defined in (b). It returns Penny's location.

query results, thereby tailoring the results to a user's needs and reducing bandwidth usage. A query may match several advertisements. Often, a querying entity has use for only one result, or some specific number of results. Result-ranking mechanisms allow these results to be ordered according to some qualitative or quantitative metric. For instance, the Google web search engine [106] ranks query hits according to the popularity of a page as defined by the number of pages that link to it [154]. The result-ranking facility draws on decision theory models described by Fishburn [155]. A ranking function is used to assign a utility value to each result, thereby defining a total order over the result set. The utility of a result is calculated by a user-defined mathematical expression that is submitted along with the query. The expression can reference any value contained within an advertisement.

Basic Preferences

Within expressions, attributes forming an advertisement are referred to using a *dot-separated* notation, where a dot separates a parent attribute component from its child. Thus, to create an expression involving the queue length attribute of the printer described in Figure 5.3, one can write:

```
1/printer.queue.length
```

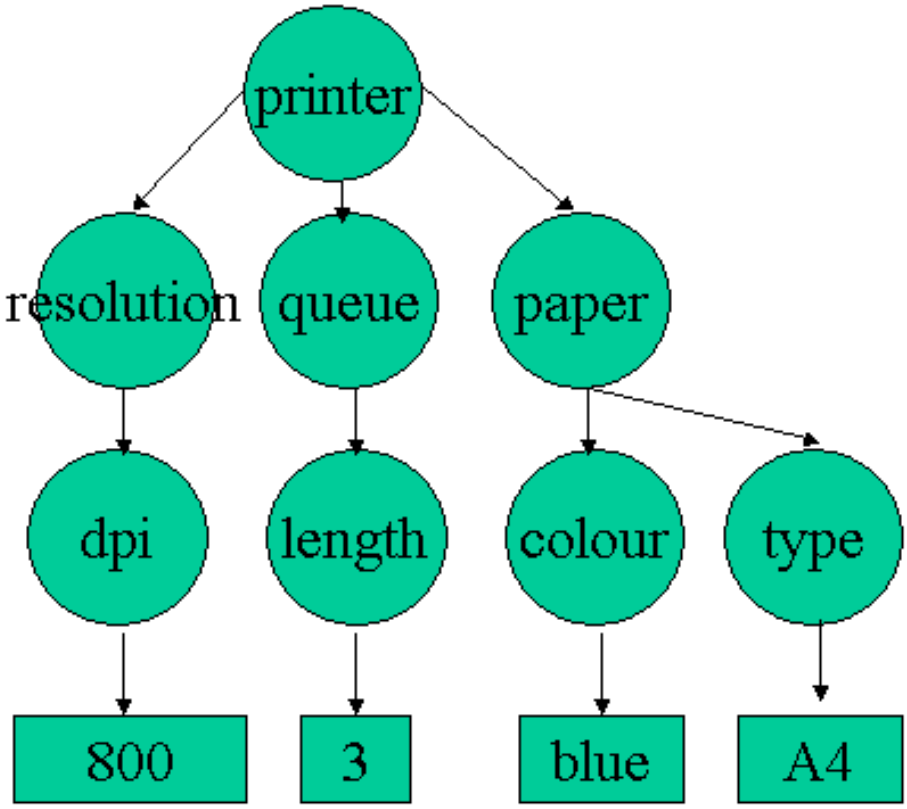


Figure 5.3: Part of a printer description.

The keywords *ASC* (order ascending), *DESC* (order descending), *LIMIT* <*n*> (return only the first *n* results) and *THRESHOLD* <*t*> (return all the results the same or better than the threshold *t*) may optionally appear after an expression. The *ASC*, *DESC* and *LIMIT* keywords play a similar role to their namesakes in SQL [74]. In the absence of either *DESC* or *ASC*, *DESC* is assumed. Specifying ascending order implies that smaller utility values are preferred over larger utility values. Conversely, the specification of descending order implies that larger utility values are preferred over smaller ones. In the absence of *LIMIT* or *THRESHOLD*, all results are returned in the specified order. Note that an expression consisting entirely of the term *LIMIT* <*n*> may be used to return any *n* matching results. The behaviour of *THRESHOLD* is dependent on the ordering of results. In the absence of *DESC* and *ASC*, *DESC* is assumed, and *THRESHOLD* will return any results with a utility of *t* or greater.

The above expression may be equivalently written as:

```
printer.queue.length ASC
```

so that smaller values appear first. Then, to limit the results to exactly one, a *LIMIT* clause is specified:

```
printer.queue.length ASC LIMIT 1
```

The expression grammar can therefore be represented as:

```
[ <expression> [ ASC | DESC ] ]
  [ LIMIT <n> | THRESHOLD <t> ]
```

In addition to the expression itself, a mapping of non-numerical types to numerical types can be declared. This allows non-numerical types, such as strings, to appear in utility expressions. For instance, if there are several printers, each loaded with paper of a different colour, one might specify:

```
mapping (printer.paper.colour) {
  case (‘‘blue’’) : 1;
  case (‘‘red’’)  : 0.5;
  case (‘‘pink’’) : 0;
  default         : 0.1;
}
```

Expressions can then defined as follows:

```
(1/printer.queue.length) *
   printer.paper.colour
```

In the above example, printers which have shorter queues and the preferred kind of paper are assigned higher scores. There are no constraints placed on the range of values that may be returned from an expression.

The mapping grammar is defined as:

```
mapping   -> MAPPING ’(’ var ’)’ ’{’ { map-exp }
                                     [ default ] ’}’
map-exp   -> CASE ’(’ string ’)’ ’:’ number ’;’
default   -> DEFAULT ’:’ number ’;’
```

Context-Sensitive Preferences

Occasionally, the desired order or number of results may be dependent upon the current context of one or more entities, including the context of the querier. For example, queriers may wish results to be ordered differently depending upon their current location, the time of day, the day of the week or even the weather forecast for some particular location. To cater to these situations, the basic preference language above is augmented with conditional rankings.

The ranking order of results can be made dependent upon context through the use of named queries, defined above. Temporally dependent rankings can be achieved by using the built-in *time* function, which returns the number of milliseconds since midnight, January 1, 1970 UTC.

Conditional rankings make use of the traditional *if ... then ... else if ... else* programming construct with the following modification: in place of a Boolean condition is one or more named queries combined with the logical operators && (and), || (or) and ! (not). The presence of a named query *q* in the Boolean condition is interpreted as *the result set of q is not empty*. Thus, the context-sensitive ranking for a query *p*

```
if(q) {
  <preference expression 1>
} else {
  <preference expression 2>
}
```

is read "if the set of results yielded from *q* is not empty, then rank the results yielded from *p* according to preference expression 1, otherwise rank them according to preference expression 2."

The complete grammar for the preference language is given below.

```
pref      -> { mapping } rating
mapping   -> MAPPING '(' var ')' '{' { map-exp }
                                     [ default ] '}'
map-exp   -> CASE '(' string ')' ':' number ';'
default   -> DEFAULT ':' number ';'
rating    -> PREF '{' pexp | pcond '}'
pexp      -> rank-exp | limit-exp
rank-exp  -> exp [ ASC | DESC ] [ limit-exp ]
```

```
limit-exp -> LIMIT integer | THRESHOLD integer
exp       -> func-exp | term { add-op term }
func-exp  -> func '(' exp ')'
func      -> ID
add-op    -> '+' | '-'
term      -> factor { mul-op factor }
mul-op    -> '*' | '/' | '%'
factor    -> '(' exp ')' | number | var
number    -> integer | float
var       -> ID { . ID }
pcond     -> IF '(' ncond ')' { pexp } { ELSE else }
ncond     -> [ not ] cond
cond      -> andcond { or-op andcond }
or-op     -> '||'
andcond   -> scond { and-op scond }
and-op    -> '&&'
scond     -> name | ncond
name      -> ID
not       -> '!'
else      -> pcond | { pexp } | { pcond }
```

An example utilising the preference language and several context-sensitive queries is given in Section 5.4.

5.3.4 Complete Superstring API

With the addition of persistent queries, transient advertisements and preferences, the complete API exposed by context-sensitive Superstring is (in a Java-like syntax):

```
public class Superstring {
  public QueryResponse query(Query q);
  void advertise(Advertisement a);
}

public class Query {
  public static Query
               createQuery(Description desc,
                           Map namedQueries,
                           Preferences prefs,
                           int hopsToLive,
                           boolean isWideArea);
  public static Query
               createPersistentQuery(Description desc,
                                     Map namedQueries,
                                     Preferences prefs,
                                     int hopsToLive,
                                     boolean isWideArea,
                                     QueryCallback qc);
  public byte[] getDescription();
  public Preferences getPreferences();
  public int getHopsToLive();
  public boolean isWideArea();
  public boolean isPersistent();
}

public class QueryResponse {
  public static QueryResponse
               createResponse(int type,
```

```
                                int numResults,
                                Vector descriptions);
  public int getType();
  public int getNumResults();
  public Vector getDescriptions();
}

public class Advertisement {
  public static Advertisement
               createAdvertisement(Description desc,
                                   NameRecord nr,
                                   Map namedQueries,
                                   Preferences prefs,
                                   int hopsToLive,
                                   boolean isWideArea);
  public static Advertisement
         createTransientAdvertisement(Description desc,
                                      Map namedQueries,
                                      Preferences prefs,
                                      int hopsToLive,
                                      boolean isWideArea);
  public byte[] getDescription();
  public NameRecord getNameRecord();
  public Preferences getPreferences();
  public int getHopsToLive();
  public boolean isWideArea();
  public boolean isTransient();
}
```

```
public interface QueryCallback {
  public void persistentQueryMatch(QueryResponse qr);
}

public class Preferences {
  public static Preferences
               createPreferences(String prefExpression,
                                 Map namedQueries);

  public String getPrefExpression();
  public Map getNamedQueries();
}
```

The appropriate members have been added to the Query and Advertisement classes, and an additional interface has been added, which allows applications that post persistent queries to be notified asynchronously of matches against those queries. A Map of named queries stores a mapping between identifiers and Queries. The Preferences class stores a result-ranking expression which may refer to one or more named queries. The Preferences argument in each of the *create** operations of Query and Advertisement may be *null*. The message formats shown in Figures 4.8, 4.9, 4.11 and 4.13 are extended to carry information about query persistence or advertisement transience, in addition to preference information. The same query response formats shown in Figures 4.10 and 4.14 are used for synchronous and asynchronous responses.

As is the case for advertisements (Section 4.3), persistent queries cannot be removed explicitly. Rather, they are allowed to expire.

5.3.5 Scalability

A context-sensitive query pays for the parent query and for every sub-query appearing within it. Thus, while it is possible that strategies such as query result caching may improve performance, in general the context-sensitive framework scales linearly with the number of queries comprising context-sensitive queries, advertisements and preferences. It is difficult to imagine any non-centralised framework that can better this situation.

5.4 Application: Locating Free Parking Spaces With *iCarpark*

This section describes a hypothetical application called *iCarpark*. iCarpark is an application that enables drivers to find a free parking space within a multi-level car park. The application is intended to illustrate the use of the context-sensitive resource discovery protocol; the viability of the application design is untested. iCarpark can be implemented by using a small set of context-sensitive Superstring queries and preferences. The iCarpark application demonstrates all of the context-sensitive extensions described above, including context-sensitive queries and advertisements, context-sensitive preferences and the mapping of non-numerical values to numerical ones for use within ranking expressions. iCarpark was designed specifically for the Indooroopilly Shopping Centre car park; however, it could be reused for any multi-level car park and trivially modified to be suitable for a single-level sectioned car park.

A more orderly car parking system might reduce the incidence of the phenomenon known as *car park rage*, *parking lot rage* or simply *parking rage* [156]. By identifying free parking spaces early, a driver can navigate to an area where

there is a high chance of acquiring a parking space. Although there is a chance that one or more cars converge on the same free parking space, car park congestion ought to be reduced because drivers already know where the free spaces are. Therefore, they spend less time driving around trying to locate free spaces.

5.4.1 Requirements

The Indooroopilly Shopping Centre has a multistory car park of five *levels*, each assigned a *colour* (except the uppermost level, which is called the *rooftop* level). Each level is split into a *southern zone* and a *northern zone*, and these are further divided into *sections* or aisles (the rooftop is less uniform and is divided into areas instead).

The shopping centre management and consumer groups wish to devise a solution to the free parking space problem: finding a free parking space within user specified constraints, such as preferred level, wheelchair spaces and "parents with prams" spaces. Users should be free to utilise any other preferences and context information they see fit. The shopping centre management sees this as a consumer driven project, though they acknowledge that, since the car park is owned and managed by the shopping centre, it must provide some of the necessary infrastructure. A minimal infrastructure means the solution will be more easily applied to other car parks. Thus, the consumer group sees a large benefit in having a minimal infrastructure solution. The consumer group imposes the following constraints:

- the solution must not come at a high cost to consumers;
- modifications made to vehicles, if any, must be optional, so that participation in the scheme is not mandatory for all shoppers;

- sufficient parking spaces must be made available on each level to those customers who do participate in the scheme.

Both parties recognise that some shoppers will be excluded from participation in the scheme due to the minimum technological requirements imposed by the solution. The solution should be designed with tomorrow's vehicles in mind, rather than today's, so that it can be deployed incrementally as more shoppers acquire modern vehicles. A technological requirement that is obvious even before a solution has been proposed is that vehicles participating in the scheme must be equipped with a graphical console so that drivers can view information about free parking spaces. Any car with a trip computer, such as the Holden Astra CDX [157], already possesses the required interface.

With these requirements in place, a solution can be formulated. The solution, in the form of iCarpark, is described below.

5.4.2 Solution

The solution can be broken into several components. Each of these components is described below.

Identifying and Monitoring Parking Spaces

There are several possible alternatives for identifying parking spaces, and each depends on the mechanism used to monitor the state of parking spaces.

Gibbons et al. [158] utilise a set of cameras to monitor parking spaces. This solution is problematic for a number of reasons. First, cameras are expensive. Second, the computer vision algorithms required to delineate each parking space are computationally expensive. A central server must be installed to perform this task. A final complication is in determining suitable locations for each camera

such that all parking spaces are covered by the set of cameras. Cameras must be placed so that all parking spaces can be established as being free or occupied. If too few cameras are deployed, then free parking spaces will be obscured by occupied parking spaces. In low ceiling car parks, the number of required cameras may be prohibitively expensive, since each camera can capture only a small area. This solution also places a heavy burden on the shopping centre management.

Read-only, chip-less RFID tags are, perhaps, the best option in terms of robustness, and potentially even in cost. These tags sell for between 0.25 USD and 0.50 USD today, and the price is expected to fall to 0.05 USD in the near future, as mass production of tags begins. At 0.25 USD each, a 4000 space car park could be completely fitted out at a cost of 1000 USD. But this only covers the expense incurred by the managers of the parking lot. Each vehicle participating in the scheme must have an RFID reader installed. Toppan Printing Co. recently announced that they will start producing RFID readers that cost 20 USD [159]. In terms of the total cost of a vehicle, 20 USD is negligible (in Australia even a small car like the above mentioned Holden Astra retails for approximately 25500 AUD).

An RFID tag stores a 64 bit identifier, and sends this identifier to an RFID reader when queried (using the power afforded by electromagnetic waves emitted by the reader). In this way, each parking space is uniquely identified.

The state of a parking space is adjusted by the car which enters or leaves the parking space. A car entering a parking space reads the ID of the embedded tag with its RFID reader, and sends out an advertisement update that changes the state of the advertisement for that parking space. When it leaves, it sends another advertisement that changes the state of the parking space to “free”.

Queries and Advertisements

This application requires several context-sensitive queries and advertisements. Furthermore, individual shoppers may tailor query results to their preferences and any contexts they specify. The required queries and advertisements and an example result-ranking preference are detailed below.

A device, such as a mid-range workstation, owned by the shopping centre management emits context-sensitive advertisements for each parking space in the complex. These advertisements are stored by vehicles that are participating in the scheme. An example of such an advertisement is shown in Figure 5.4.

The sub-query enables the status of the space to be updated dynamically by parking and leaving cars.

In addition, it issues another context-sensitive advertisement for each level of the car park. The purpose of these advertisements is to aggregate the free parking spaces for each level. Thus, an advertisement exists for each level of the car park, and it includes the name (or number) of the level and the number of free spaces in the level. Figure 5.5 shows the advertisement for Indooroopilly Shopping Centre's pink level car park.

The sub-query in the above advertisement matches advertisements sent by parking and leaving cars. When a car enters a free space, it reads the ID of the space from the embedded RFID tag. It then issues a transient advertisement that updates the status of the space. This advertisement is matched by the sub-query in the advertisements shown in Figure 5.4. This update is shown in Figure 5.6.

These inter-related advertisements are the core of the iCarpark application. With these in place, shoppers can issue queries to find the levels where there are free spaces. These queries can be adjusted to the needs of individual shoppers. Furthermore, the results can be ranked according to user preference.

Upon approaching the shopping centre car park, the user (or the iCarpark ap-

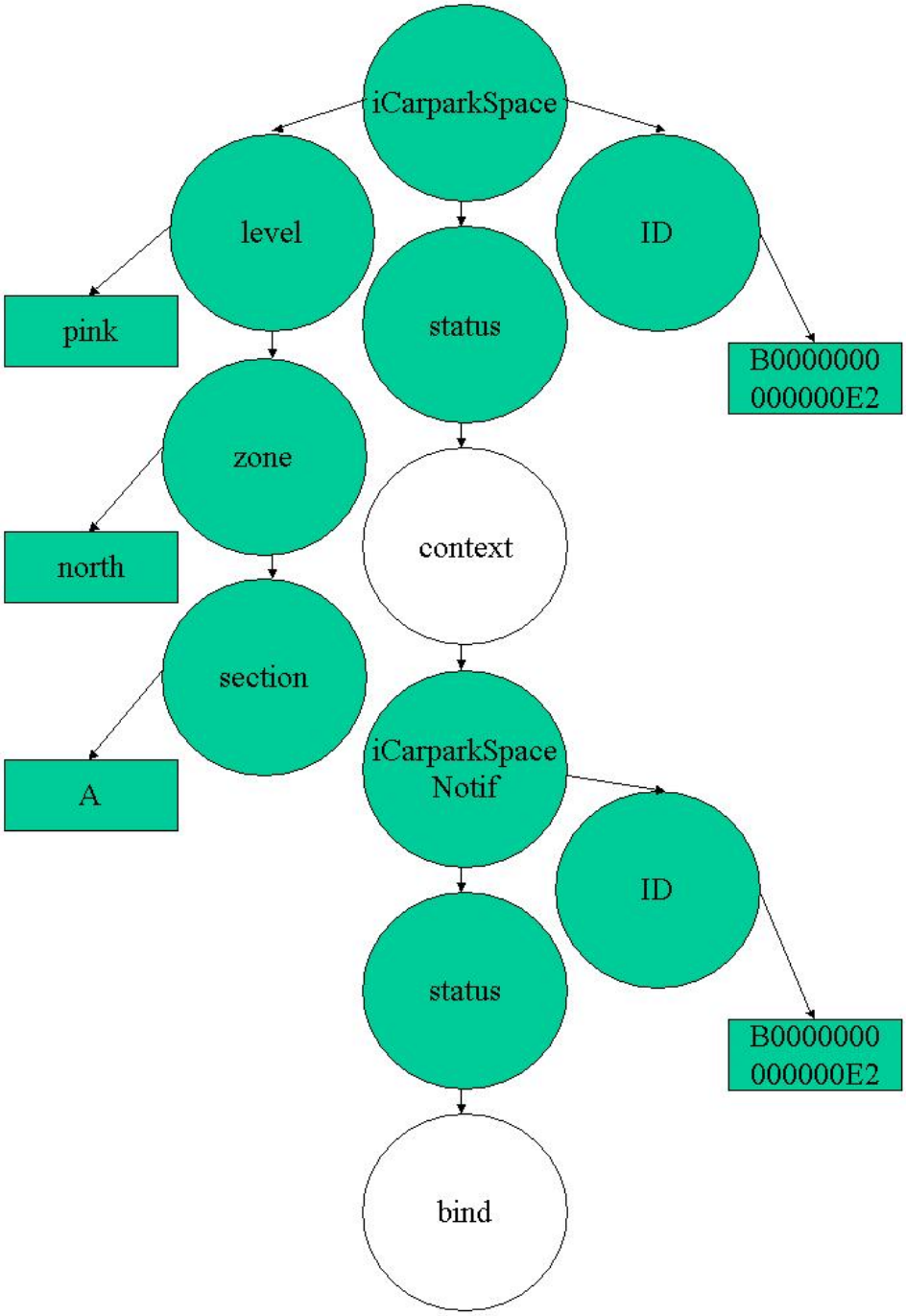


Figure 5.4: The iCarparkSpace advertisement.

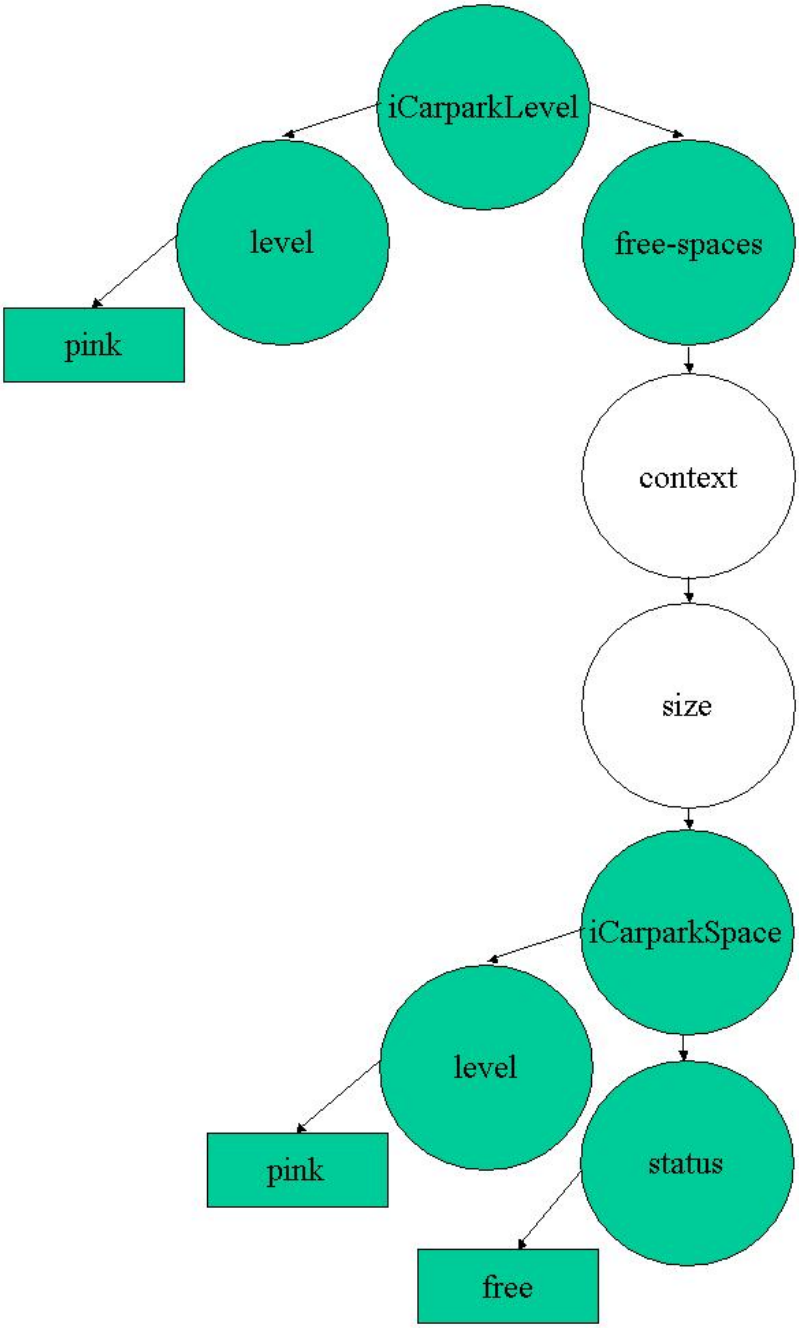


Figure 5.5: The iCarparkLevel advertisement.

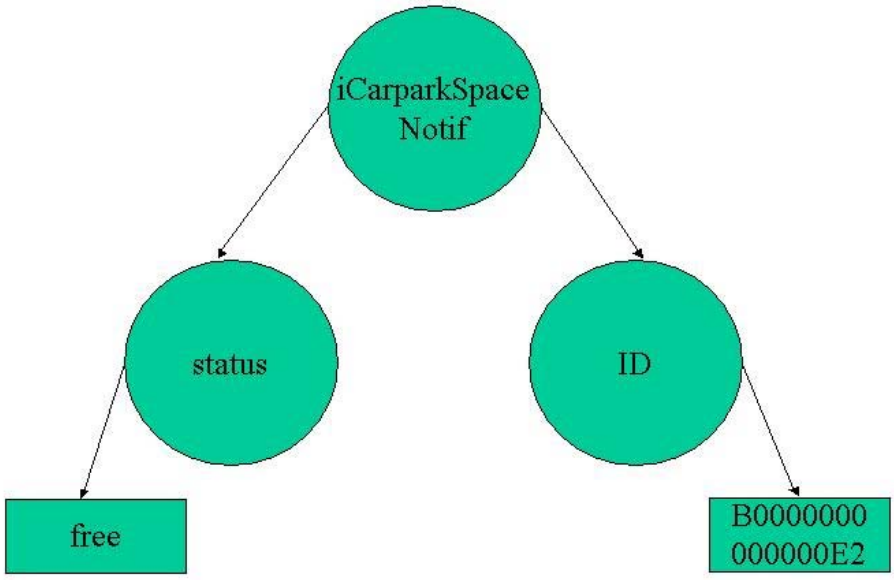


Figure 5.6: The iCarparkSpaceNotif advertisement.

plication running on the car's computer) issues the query shown in Figure 5.7, where the free spaces threshold can be adjusted to the user's taste.

Accompanying the query is a preference that specifies the order in which results should be returned. The preference presented below specifies an ordering dependent on the weather forecast.

```
MAPPING (iCarparkLevel.level) {
  case(''yellow''): 1;
  case(''blue''): 2;
  case(''green''): 3;
  case(''red''): 4;
  case(''pink''): 5;
  case(''rooftop''): 6;
```

```
}
PREF {
  if(good-weather) {
    abs(iCarparkLevel.level - 5) ASC;
  } else {
    iCarparkLevel.level % 6 DESC THRESHOLD 1;
  }
}
```

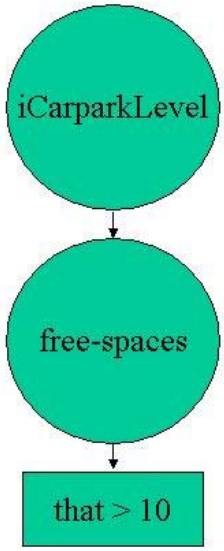


Figure 5.7: The iCarparkLevel query.

If it is forecast to rain later in the day, at which time the shopper emerges from the shopping centre with bags full of groceries, clothing and other items, then rooftop parking is ill-advised. But ordinarily, the rooftop is a good choice because it is fairly close to the food court and the cinemas. However, the pink level (level

5) is always the most preferred level because it is undercover *and* it is close to the food court and cinemas. The specified preference takes these things into account. Thus, if it is a fine day, the pink level is our preferred destination, while the levels above (rooftop) and below (blue) are next best, and so on. If the prediction is for rainy weather later on, then the pink level is still preferred, but any level except the roof (level 6) is acceptable if pink is full. Levels just below pink are preferred, therefore the results are returned in descending order (as per the mapping in the preference above).

The exact specification of the *good-weather* query is not shown. However, that query details weather conditions that are acceptable to the shopper. If the query is matched (the result set is not empty), then the weather will be fine. Otherwise, the outlook is not good, and the shopper is prevented from using the rooftop. Note that the weather is only one kind of context. Other kinds of context might also be relevant for different shoppers, such as the day of week or the time of day.

Once the vehicle arrives at the chosen level, the query can be refined and reissued so that a specific zone and section can be selected. This process can be entirely automated and combined with a car park mapping service to provide a complete car park navigation service.

Communication Between Vehicles

Communication among vehicles and between vehicles and the shopping centre infrastructure occurs via IEEE 802.11. In this fashion, a large, mobile, multi-hop ad hoc network is formed between the vehicles participating in the iCarpark application. The query and advertisement packets can be encapsulated directly by 802.11 frames, where the protocol retransmits after a timeout due to dropped or corrupted packets. This scheme nullifies the requirement for separate network and transport layers to be used. Since the service discovery protocol for unstructured

environments does not make use of end-to-end routing (its utility is derived from its hop-by-hop nature), a network routing layer only adds overhead and complexities. For example, each mobile node would need to dynamically establish an IP address using a mechanism such as Zeroconf [51].

Onboard Computer

As stated in Section 5.4.1, each participating vehicle requires a small onboard computer. Modern vehicles already have many computerised components, which control the fuel injection mechanism, the spark plug timings, keyless entry, car radio and CD stacker. Recently, cars have been built with more sophisticated computers that provide an interface for the driver. These computers offer simple services to the driver, such as the distance the vehicle can go on the remaining fuel, but increasingly they are offering applications one might expect to find in a PDA. For instance, some onboard computer systems include a calendar application which can be programmed to remind the driver of a scheduled car servicing or an upcoming birthday. The iCarpark software, including the Superstring protocol, will be installed in these onboard computers.

Modern vehicles utilise controller area networks (CANs) [160] to facilitate communication between computerised components and to provide auto-mechanics with access to diagnostic information. CANs are capable of data rates up to 1Mbit/s [161], but only half that rate is used in practice. iCarpark harnesses the CAN to provide communication between the RFID reader installed underneath the car and the onboard computer.

An alternative is to use a laptop or handheld computer instead of a built-in onboard computer. However, this would require the RFID reader to have a wireless network interface, such as Bluetooth, so that it can communicate with the laptop. Such devices exist [162], but cost considerably more. Nevertheless, this does offer

a means by which customers with older cars can participate in the scheme.

In either case, a simple graphical interface is provided to the driver in order to set threshold values for the queries. In the case of the onboard trip computer, complicated preferences must be uploaded to the car's computer prior to the use of the iCarpark application.

5.4.3 Summary

The iCarpark application provides one example of the way in which a context-sensitive application can be built by utilising the features provided by Superstring. Importantly, it shows that context-sensitive applications can be built in ad hoc network environments where there is no centralised component. Furthermore, it highlights the power of being able to combine totally orthogonal context elements and personal preferences (for example, location and the weather) to create highly individualised applications. The ability to easily integrate context types not envisaged at application design time is a critical factor in the reuse of the application in other scenarios.

5.5 A Comparison to Existing Approaches

In Section 2.5, a number of existing attempts to integrate context and service discovery were surveyed. In this section, those works directly comparable to Superstring are contrasted to Superstring's context-sensitive framework.

The work of Chen and Kotz [17] is most similar to the context-sensitive service discovery protocol outlined in this chapter; however, 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 chapter 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.

Another difference is that Chen and Kotz require the programmer to learn a different abstraction in order to utilise context-sensitive queries. In their work, context-sensitive queries are specified using a graphing abstraction that combines INS [68] names with user-defined functions and variable names. Thus users are expected to program functions that filter, transform or aggregate advertisements (or events). In contrast, the solution outlined in this chapter utilises the same language for queries and advertisements whether they are context-sensitive or not. The preference language *is* different to the description language; however, this is acceptable given that preferences solve an orthogonal problem.

Lee and Helal [110] augmented the Jini [8] lookup service to consider dynamically changing local and remote context attributes. Services define relevant context attributes and these are evaluated by the lookup service on demand. Clients remain oblivious to the existence of context attributes, meaning that clients cannot define the types of context that are relevant to them. Result-ranking expressions are defined by the services, meaning that clients and users are completely unaware of the relevance of the ordering. Although hiding context attributes from the clients implies that existing Jini clients can operate with the new lookup ser-

vice and Jini services that specify context attributes, it means that only a small portion of the power of context-awareness can be exploited by their solution. The assumption that services know best does not always hold. For instance, implicit in their use of a *Domain* context attribute to estimate the distance between the user and the service is the heuristic that "closer is better". This is only sometimes true. There are occasions when one would prefer to select a particular service (such as a printer) because it is conveniently located on one's path from location A to location B, while not necessarily being the closest printer at the current time. A further problem with hiding context attributes and result-ranking functions from users is that the user has no way of knowing *why* the results were returned in the order that they were returned. Result-ranking is a futile exercise if the semantics governing that ranking are hidden from the user. Superstring does not suffer from any of these drawbacks. Instead, it enables users to integrate arbitrary kinds of context information into the discovery and result-ranking processes. It also allows service advertisements to change dynamically as the context changes.

The Location Information System (LIS) [111] could be deployed in parallel with our context-sensitive service discovery protocol to provide advertisements about changes in location of the entities within the mobile computing environment.

None of the above solutions contain specific support for very lightweight mobile devices that operate within a heterogeneous network. Each of them assumes that a mobile node can reach the infrastructural components of the respective solutions (called *Planets* in the work of Chen and Kotz, *lookup services* in Jini and the work of Lee and Helal, and *LISes* in the work of Maaß [111]) via the underlying network protocols. None of the above solutions provide any support for context-aware operation in the event that the infrastructure is unreachable or non-existent, as in the case of iCarpark.

5.6 Conclusions

This chapter described a set of context-sensitive extensions to the core Superstring protocol introduced in the previous chapter. These extensions provide a lightweight, flexible mechanism to endow applications with context-awareness. This solution requires neither a central context repository nor a monolithic design approach whereby context management is built into the application. Rather, context information remains in the network where it is matched against resource discovery queries, making it highly suitable for mobile ad hoc networks.

Superstring enables users to formulate advertisements and queries dependent upon arbitrary contexts, and it provides facilities for context-sensitive result-ranking. When these features are combined with the core components of Superstring, such as the rich description language and query relaxation, powerful context-aware applications can be realised.

CHAPTER

6 Prototype

This chapter describes the prototype implementation of the core Superstring protocol described in Chapter 4. The chapter is divided into four main sections. Section 6.1 outlines the purpose of the prototype. Section 6.2 details an implementation of the unstructured routing layer. Section 6.3 documents an implementation of the structured Superstring protocol and the Chord routing layer. In Section 6.4, the details of various test configurations are given.

6.1 Prototype Scope and Purpose

The purpose of the prototype was to enable a formal analysis of the scalability of Superstring. The analysis is documented in Chapter 7. In addition, a set of tests were conducted with the prototype to demonstrate the routing layers operating separately and together. Both of these layers share the same programming interface which has already been documented in Chapter 4. In practice, this interface is implemented as an abstract class (*Superstring*) which is then extended by

the two specialised layers. The prototype was implemented using J2SE [163] and J2ME [164]. Both layers use NanoXML [139] for parsing XML descriptions. A lightweight XDR [165] implementation was written for marshaling messages in both layers.

A further goal of the prototype of the unstructured layer was to demonstrate the ease with which additional transport layers can be plugged into the unstructured routing protocol. Two UDP-based transports were implemented for the prototype.

6.2 Unstructured Layer

Section 4.5 described a query routing protocol for unstructured environments. This section describes the implementation of that protocol. This implementation has some limitations: it does not prune query responses, and it ignores *scope* attributes. These features are not important to the analysis contained in Chapter 7 as the analysis is primarily concerned with scalability and the manner in which descriptions are propagated throughout the network, whether or not they are pruned.

The prototype implementation was constructed in a manner that highlights Superstring's ability to operate in heterogeneous environments. Figure 6.1 shows a component diagram for the unstructured protocol layer.

The *Unstructured* class extends the base *Superstring* class. It provides implementations of the *query* and *advertise* operations. The *Unstructured* class contains the core logic for deciding where to forward queries and advertisements based on information given to it by the *Resolver* class. The *Resolver* class is responsible for storing advertisements and cached query responses (pheromone), and matching queries against these stored descriptions. The *DescriptionParser* aids in the matching process by parsing the descriptions contained in advertisements and queries. Upon being given a query to resolve, the *Resolver* returns a set of match-

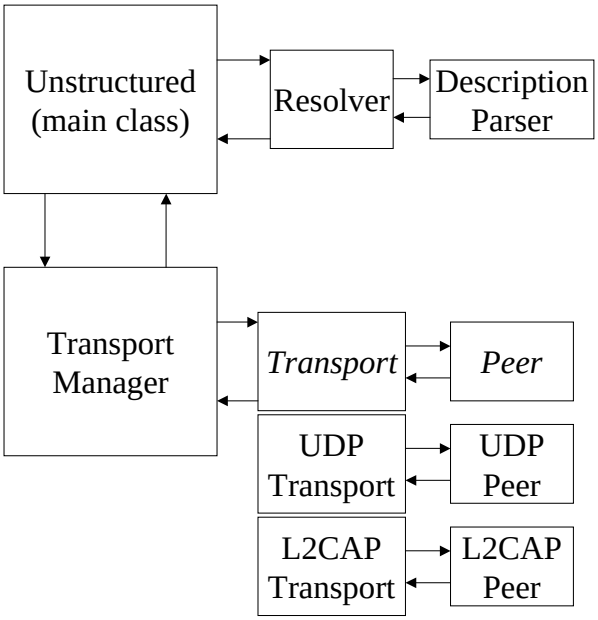


Figure 6.1: The unstructured protocol implementation.

ing resources. If the returned set is empty, then the *Unstructured* class forwards the query to a random neighbour. Otherwise a response message is formulated and forwarded along the return path. A background thread monitors the descriptions stored by the *Resolver* and purges descriptions that have timed out.

To provide a level of abstraction from underlying protocol layers, a *TransportManager* is used. The *TransportManager* contains one or more *Transport*s. The prototype includes a *UDPTransport* for sending messages over UDP/IP. The *TransportManager* initialises each *Transport* at start-up, which involves executing the join protocol described in Section 4.5.2. Each neighbour that responds to the *join* message is represented by an instance of the *Peer* class. For example, a neighbour that uses UDP/IP for communication is represented by a *UDPPeer* object. All subsequent communication with that neighbour occurs through that

instance of *UDPPeer*. The *TransportManager* contains operations to broadcast messages to all neighbours (used during advertising) and for directed communication. Invocations of these operations are delegated to the *Transport* and *Peer* instances, which handle the details of transport specific communication.

The prototype was written so that it may be executed on either a standard Java platform (J2SE) or a micro Java platform (J2ME). Only where the implementation details differ (such as the network communication) were separate classes required. For instance, there are separate classes for *UDPTransport* on the two platforms (*J2SEUDPTransport* and *J2MEUDPTransport*) and separate classes for the corresponding *UDPPeer*s (*J2SEUDPPeer* and *J2MEUDPPeer*).

6.3 Structured Layer

This section details the implementation of the structured routing layer designed in Section 4.6. It also outlines the underlying Chord implementation.

Figure 6.2 shows the class diagram of the structured routing layer.

The *Structured* class extends the *Superstring* base class. The *EntryTable* is a structure that stores component names of the form *printer.paper.size* and their corresponding values. The *Structured* class uses a *DescriptionParser* to convert XML descriptions to the dot-notation form. A *StoreThread* is used to process each child of the attribute currently being processed. Similarly, during queries, a *QueryThread* is used to process each child of the attribute currently being processed. Thus, sibling sub-descriptions can be processed in parallel for queries and advertisements. These threads are given the sub-description to be processed and the address of the child node that is responsible for the sub-description, as determined using the algorithm specified in Section 4.6. A switch is provided to toggle the use of node hierarchies. If the switch is off, then the entire description is stored

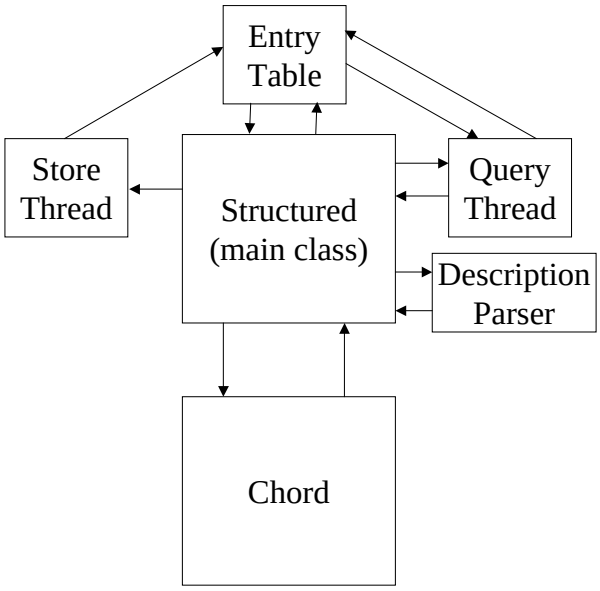


Figure 6.2: The structured protocol implementation.

at the root resolver for a particular description. No hierarchy will be used. In this case, each *StoreThread* stores all components at the current node. Communication between Superstring nodes that use the structured protocol occurs via TCP/IP.

The Chord implementation provides a *lookup* operation. It does not include a background process that ensures integrity in the case of node failures and simultaneous joins. This simplification is justified since the target environments for this layer are stable and structured; it will not affect the analysis contained in Chapter 7. The background process can be easily added to the implementation at a later stage. Furthermore, a node discovery protocol is added to Chord so that a new Chord node can find an existing Chord node. First, a new Chord node attempts to locate existing Chord nodes on the same subnet using UDP/IP multicast. If this process fails to locate an existing node, then a list of known Chord nodes is read

from a file at a known URL (usually on the local file system). The Chord node uses TCP/IP during the lookup process.

6.4 Testing

Before an experimental analysis of the protocol was carried out (see Chapter 7), the prototype was tested in a number of configurations to demonstrate the feasibility of the Superstring resource discovery protocol.

First of all, each layer was tested separately in a configuration mirroring the intended experimentation environment. The structured protocol was tested on a network of fifty resolvers using the J2SE platform, and the unstructured protocol was tested on a network of ten nodes. The unstructured protocol was tested under J2SE and J2ME.

Second, the two layers were tested together using several dual-layered nodes. In this configuration, nodes executing the unstructured routing layer issued advertisements and queries that were propagated to the structured environment. The gateway between the unstructured and structured environments was implemented as a *virtual* transport, by inheriting from the *Transport* class documented above. The role of this transport is to pass messages between the structured and unstructured layers on a dual-layered node. Thus, owing to the prototype design, implementation of the gateway was trivial.

In a real deployment situation, it is intended that the structured protocol be installed on stable infrastructure to support wide-area operation. Applications that are traditionally deployed to stable infrastructure, such as web services and grid computing applications may find themselves executing on a node configured to use the structured protocol, in which case they may utilise the Superstring protocol via the structured protocol layer. All other nodes will execute the unstructured

protocol layer. Applications deployed on these nodes can discover resources in the structured environment via a dual-layered node. Nevertheless, much of the time, resource discovery will be constrained to the local environment, due either to application requirements or network isolation.

The tests, conducted in the environment described above, were performed to ensure the correct operation of the prototype. They enabled debugging to be performed prior to the experimental analysis.

6.5 Summary

This prototype, available from http://rickyrobinson.id.au/superstring.tgz, is just one possible implementation of Superstring. This implementation was designed so that new transports can be added easily to the unstructured layer, making it deployable to heterogeneous networks. The structured layer prototype includes a lightweight implementation of Chord, which is suitable for use in stable networks, and which meets the experiment requirements. However, other prototypes can be built as long as they meet the specifications outlined in Chapter 4. For example, a lighter-weight version of the unstructured protocol layer could be implemented, which does not have the overhead of a transport manager. Such an implementation would be suitable for devices with only a single network interface.

CHAPTER

7 Performance Analysis

This chapter evaluates the performance of Superstring in terms of the distribution of descriptions over nodes in the wide- and local-area. For the wide-area, description distribution is measured directly, by counting the number of description components stored by each resolver. For the local-area, the task is not so simple, as the description distribution evolves over time. The analysis of the local-area query routing protocol therefore warrants a more detailed discussion. Since the number of descriptions in the local-area is relatively small, the fact that descriptions are distributed evenly over the set of nodes is less important than the manner in which the distribution evolves to reduce query times. In other words, the distribution scheme in the wide-area is optimised to share storage and computation costs as equitably as possible, while the distribution scheme in the local-area becomes optimised to reduce query times, especially for popular resources.

Mathematical models of the local- and wide-area data distributions are formulated and compared to the experimental results. The mathematical models have several purposes. First, if the output generated by the mathematical model

matches the results obtained from the experiments, then this is a validation of implementation correctness, especially in the case of the wide-area protocol, which is completely deterministic. Large discrepancies between the mathematical model and the experimental results would therefore uncover problems with the protocol implementation that may have gone undetected using standard testing approaches. Such a validation assumes that the possibility of the implementation *and* the model being incorrect in exactly the same way is negligible. The second purpose of the mathematical models is to allow prediction and simulation of data distribution in much larger networks than would be achievable via direct experimentation. Of course, this second purpose is predicated on the first. That is, the mathematical model must first be found to concur with the experimental results on smaller networks. Finally, the mathematical model may reveal interesting behaviour that suggests ways in which the protocol could be improved.

An in-depth analysis of query latency is not provided. There are multiple reasons for this. First, query latency is contingent upon the route length of a query, and an expression for route length is given in the design of Superstring (see Chapter 4). Furthermore, absolute latency data are dependent upon network data rates, processing speeds and the amount of memory at each node. For this reason, the route length expression provides the best indicator of query latency. Finally, and most importantly, there are no data against which to compare the latency results (for example, the latency of INS/Twine is unknown).

7.1 Wide-Area Data Distribution

In this section, the distribution of descriptions in the wide-area is analysed quantitatively. First of all, a theoretical model of the data distribution is introduced. Then the results of an experiment are compared to this theoretical model. The

theoretical model and the experiment utilise the same set of 980 resource descriptions. These resource descriptions were produced from a set of seventy different resource types, with fourteen descriptions generated for each resource type, so that there would be approximately one thousand resource descriptions overall. INS/Twine is used as a benchmark against which the results are compared.

7.1.1 A Theoretical Model of Data Distribution

Model Derivation

In Superstring, the unit of granularity is the *strand*. A *strand* is the concatenation of attribute components from the root of the description to any internal or leaf node. An analysis of the sample set of 980 resource descriptions found there to be an average of 19.4 strands per description, giving a total of 19012 strands. On a network of fifty resolvers, it is expected that $19012/50 = 380.24$ strands will be stored on each resolver. However, an analysis showed that only approximately 16 percent of strands were unique in the data set. Superstring does not store duplicate strands. It stores identical strands once, and then keeps pointers to the name record for each resource containing the strand. Therefore, this figure can be lowered to $380.24 * 0.16 = 60.8384$ strands. This model shows that it is expected that an average of 61 strands will be stored at each resolver, a number representing less than one third of one percent of all strands, or just two percent when only the unique strands are considered.

This analysis leads to a simple expression that can be used to calculate the expected number of strands at a resolver.

$$S = \frac{D \times P \times U}{R} \tag{7.1}$$

where D is the total number of descriptions, P is the average number of strands

per description, U is the fraction of unique strands, and R is the number of resolvers on the network.

Comparison

In comparison, INS/Twine pays a much larger storage cost, since descriptions are sent to multiple resolvers. In fact, each description is sent to as many resolvers as there are strands in the description. Using the same set of descriptions, INS/Twine must store a total of $19012 * 19.4 = 285180$ strands. Thus, in a network of fifty resolvers, each resolver would be expected to store 5703.6 strands. When only the unique strands are considered (since INS/Twine does not store duplicate strands either), $5703.6 * 0.16 = 912.576$ strands will be stored at each resolver. This represents just under five percent of all strands, or thirty percent of unique strands. Therefore, Superstring provides an improvement over INS/Twine of about 93 percent in terms of storage costs.

Equation 7.2 shows an expression for calculating the storage requirements for INS/Twine, derived from the discussion in the preceding paragraph.

$$S = \frac{D \times P^2 \times U}{R} \tag{7.2}$$

The P^2 term represents the fact that each description must be sent to as many resolvers as there strands in the description.

Figure 7.1 shows a graphical comparison of the storage requirements of Superstring and INS/Twine, for $D = 1000$, $P = 19.4$, $U = 0.16$ and varying R. As can be seen, Superstring outperforms INS/Twine by an order of magnitude.

In the data set used for experimentation (which consisted of descriptions of all manner of resources, including but not limited to printers, radiation meters, naming services, thermometers, screens and file stores), the average strand was 68 bytes in size. Therefore, in one gigabyte of memory, it is possible to store

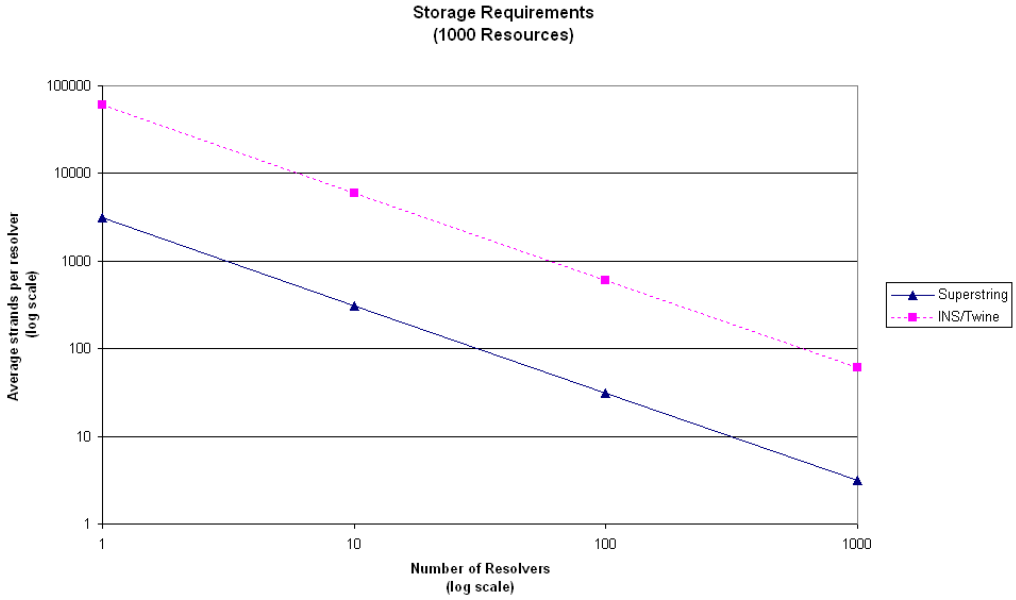


Figure 7.1: A comparison of the storage requirements of Superstring and INS/-Twine.

approximately 15.8 million strands. If the parameters of the equations are set to the same values as the experiment ($R = 50$, $P = 19.4$ and $U = 0.16$), 15.8 million strands equates to 254.5 million resources under Superstring as compared to 13.1 million resources under INS/Twine. Essentially, this means that in this scenario, Superstring can afford to store twenty times as many descriptions as INS/Twine (or twenty replicas of each description, which can be achieved using keyed hashing). In general, Superstring can store P times as many descriptions as INS/Twine for the same cost.

Workload

As Iamnitchi et al. [166] point out, queries in grid computing environments follow a power-law distribution, meaning that, in simple terms, searches for certain types of resources are more common than others. In fact, this is true for all manner of en-

vironments, including the world-wide web [147] and peer-to-peer networks [148]. If the assumption is made that the query distribution in pervasive environments is similarly non-uniform (no studies have been performed to date), then it is clear that Superstring distributes query processing workload more fairly than do INS/-Twine and other existing resource discovery protocols. The reason for this is that Superstring can distribute resource descriptions, and thus query processing, over a number of resolvers even when some resource types are far more common than others. An in depth study of the distribution of workload is outside the scope of this analysis.

7.1.2 Experimental Results

The theoretical model shows that Superstring should outperform INS/Twine by 93 percent in terms of storage costs. The experiments, which were conducted on a network of fifty resolvers within a single subnet, revealed that each Superstring resolver stored an average of 61.48 strands. This figure resonates strongly with the expected figure yielded by the theoretical model. It represents two percent of the unique strands. The greatest number of strands stored by any resolver was 279, which is approximately nine percent of unique strands. Figure 7.2 shows a graphical representation of this strand distribution.

Since the experimental results agree with the mathematical model, meaning the model can be used to accurately predict storage requirements (and therefore workload requirements), and because an asymptotic analysis of routing length was conducted in Section 4.6 as part of the design, no further analysis of the wide-area protocol is given here.

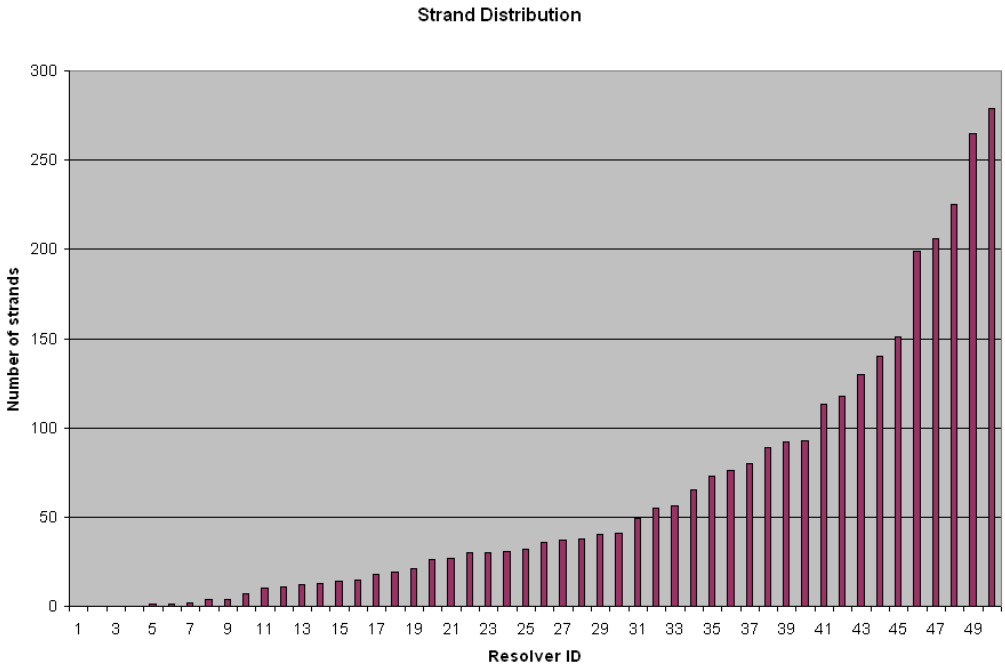


Figure 7.2: The strand distribution obtained in an experiment utilising 50 resolvers and 980 descriptions. The resolver IDs were assigned especially for this figure and ordered by increasing number of strands.

7.2 Local-Area Data Distribution

Lv et al. [143] have studied random walk algorithms and replication in the domain of peer-to-peer file sharing networks. The study contained in this section presents an alternative mathematical model of random search, and documents the results of experiments performed with the prototype implementation of the local-area protocol. The expected results provided by the theoretical model are compared to the results from the experiments.

7.2.1 A Theoretical Model of Data Distribution

Recall that the local-area protocol is predicated on the process of stigmergy, a method of communication used by social insects such as ants. Stigmergy is not a deterministic mechanism. Rather, it is a probabilistic and stochastic process.

Simplifications and Assumptions

An important simplification is made to the protocol described in Section 4.5. Pruned (generalised) descriptions (see Section 4.5.2) are not utilised in the process of forming ant-trails. Instead, entire descriptions are propagated in responses to successful queries. In the world of ants, this is analogous to replicating the entire food source along the path back to the ant nest.

In addition, some assumptions are made. This model assumes that from any node, it is equally probable to hop to any other node in the network. Such a network is formed in many situations, such as a Wireless LAN where each node is within radio range of every other node, or an Ethernet where each node shares the same medium. Thus, this assumption is not an invalid one, and in fact matches the testing environment precisely. Further, the model assumes that the number of nodes in the network does not exceed the hop limit for a query. It also assumes that, for each query, a matching resource exists somewhere in the network. The advertisement radius is set to zero, so that, initially, only the originating node is aware of the service it provides. Specifically, the mathematical model assumes the existence of a solitary resource, and all queries issued match this single resource. Therefore, the model describes the way in which knowledge of the existence of this single resource spreads through the network. The final assumption made is that queries are issued at a constant interval, and the interval is greater than the time taken for any query to be resolved and returned.

With these simplifications and assumptions in mind, the protocol behaviour

can be represented by a mathematical expression. Specifically, the problem is that of calculating a dynamically changing expected value, where the expected value represents the number of nodes in the network that are aware of a particular resource. The nodes that are aware of a resource are said to be *covered* by that resource.

Coverage Equation

In general, an expected value, $E(X)$, is calculated as

$$E(X) = \sum_{x} xP(x) \tag{7.3}$$

where P is the probability function of X.

However, the number of covered nodes changes dynamically as queries are issued. In particular, the number of covered nodes after query $q + 1$ is dependent upon the number of covered nodes after query q. Therefore, the specification of P is complex.

Let χ represent the expected number of covered nodes. Initially, $\chi = 1$, indicating that only one node has knowledge of the resource to begin with. We distinguish between subsequent values of χ by introducing a subscript, q, so we can rewrite the above as $\chi_0 = 1$.

The coverage increases as successful queries create pheromone trails, replicating the resource description along the path between the querier and the query resolver. After the first query, more nodes will be aware of the existence of the resource, thereby increasing the probability that the resource will be located in fewer hops for the next query. Therefore, the probability that a particular number of nodes must be visited before the resource is found is dependent upon the current cover.

Recall from Section 3.2 that differential equations or their discrete analogues, recurrence equations, are often used to model dynamic systems. The recurrence

equation shown below provides a way to calculate the expected cover of the network. The formulation of this equation is explained below.

$$\chi_{q+1} = \lceil \frac{\chi_q}{n} + \sum_{i=2}^{n+1-\chi_q} i \times (\prod_{k=0}^{i-2} \frac{n-\chi_q-k}{n-k}) \times \frac{\chi_q}{n-i+1} + \chi_q - 1 \rceil \quad (7.4)$$

Model Derivation

Although Equation 7.4 appears markedly dissimilar to the standard expected value calculation depicted in Equation 7.3, it is, in fact, the same equation with the appropriate probability function and possibilities substituted. Its derivation is explained using an example. Consider the network of four nodes, one of which has been seeded with the resource to be discovered. Then, according to Equation 7.3, the expected value (the number of nodes that must be visited before the resource is found) is

$$E(X) = \sum_{x=1}^{4} xP(x) = 1 \times \frac{1}{4} + 2 \times \frac{3}{4} \times \frac{1}{3} + 3 \times \frac{3}{4} \times \frac{2}{3} \times \frac{1}{2} + 4 \times \frac{3}{4} \times \frac{2}{3} \times \frac{1}{2} \times 1 \quad (7.5)$$

In plain language, each possibility (that one, two, three or four nodes are visited) is associated with a probability. The possibility that three nodes are visited is used to illustrate this concept. In a network of four nodes, where one node contains the resource, in order to visit three nodes before finding that resource, two nodes that are not aware of the existence of the resource must be visited before the node containing the resource is visited. To begin with, the chance of the query originating from an uncovered node is three in four (since three out of four nodes are not covered). The chance of then visiting another uncovered node diminishes to two in three (since there are only three nodes left to choose from, and two out of those three nodes are uncovered). Finally, on the last hop, only two nodes are left to choose from, and the chance of choosing the covered node is one in two.

Now imagine that two out of the four nodes in the network are covered. Again, consider the possibility of visiting three nodes. The chance of the query originating from an uncovered node is two in four, or one half. For the next hop the chance of selecting an uncovered node is one in three. On the final hop, the chance of picking a covered node is 100 percent.

In general, the probability function is constituted by the chance of picking an uncovered node for all except the last hop, in which a covered node is chosen.

Thus, the number of terms and the value of each term in the probability function varies with the number of covered nodes and the value of the possibility in question, thereby introducing a degree of complexity which cannot be adequately captured by the standard expected value calculation.

The discussion above suggests the need to find a general expression for the probability function. If i is the value of the possibility under consideration, then a general expression for the probability function is

$$(\prod_{k=0}^{i-1} \frac{n-\chi_q-k}{n-k}) \times \frac{\chi_q}{n-i+1} \tag{7.6}$$

This probability function works for all possibilities except $i = 1$, since this possibility is simply the probability of the query originating from a covered node ($\frac{\chi_q}{n}$). The first possibility is therefore treated specially, and the probability function is adjusted to hold for values of i greater than one. Pulling this possibility out of the probability function, subtracting one and adding the number of nodes covered already (χ_q) yields Equation 7.4. The reason the result must be decremented by one is because the expected value calculates the number of nodes visited *including the final node*, which is already aware of the resource.

Decay

To complete the model, a decay factor, δ, is introduced. The decay factor simulates dissipation of the pheromone trail, or in other words, the purging of replicated

descriptions from nodes along the query route.

$$\chi_{q+1} = \lceil \frac{\chi_q}{n} + \sum_{i=2}^{n+1-\chi_q} i \times (\prod_{k=0}^{i-2} \frac{n-\chi_q-k}{n-k}) \times \frac{\chi_q}{n-i+1} + \chi_q - 1 - \chi_q\delta \rceil \quad (7.7)$$

A decay factor of 0.75 means that during the space of time it takes to issue and resolve four queries, three cache timeouts should occur. Thus, if one query is issued every minute, $\delta = 0.75$ means that the lifetime of a description is eighty seconds. Similarly, a decay factor of 0.25 means that one timeout should occur for every four queries issued, and so on.

Model Results

This model was implemented as a Java program, which was responsible for generating the following results. The implementation of this model can be viewed as a simulation of the way resources are propagated and replicated in the actual protocol.

Figures 7.3-7.6 show the way in which coverage evolves over time, according to the mathematical model, for varying network sizes and varying rates of decay.

Order From Chaos

Each network size and decay pair is associated with a particular cover *attractor*. An attractor is a point or pattern to which a system evolves. There are several kinds of attractors, including point attractors, periodic attractors and strange attractors. Figures 7.3-7.6 show the attractors for various combinations of network size and decay. For example, in the fifty node network (Figure 7.5), with $\delta = 0.25$, the system is attracted to a single cover value (13), and is therefore classified as a point attractor. On the other hand, when the decay rate is 1.0 or 0.75, the system settles into a simple cycle between two values, and is therefore properly classified

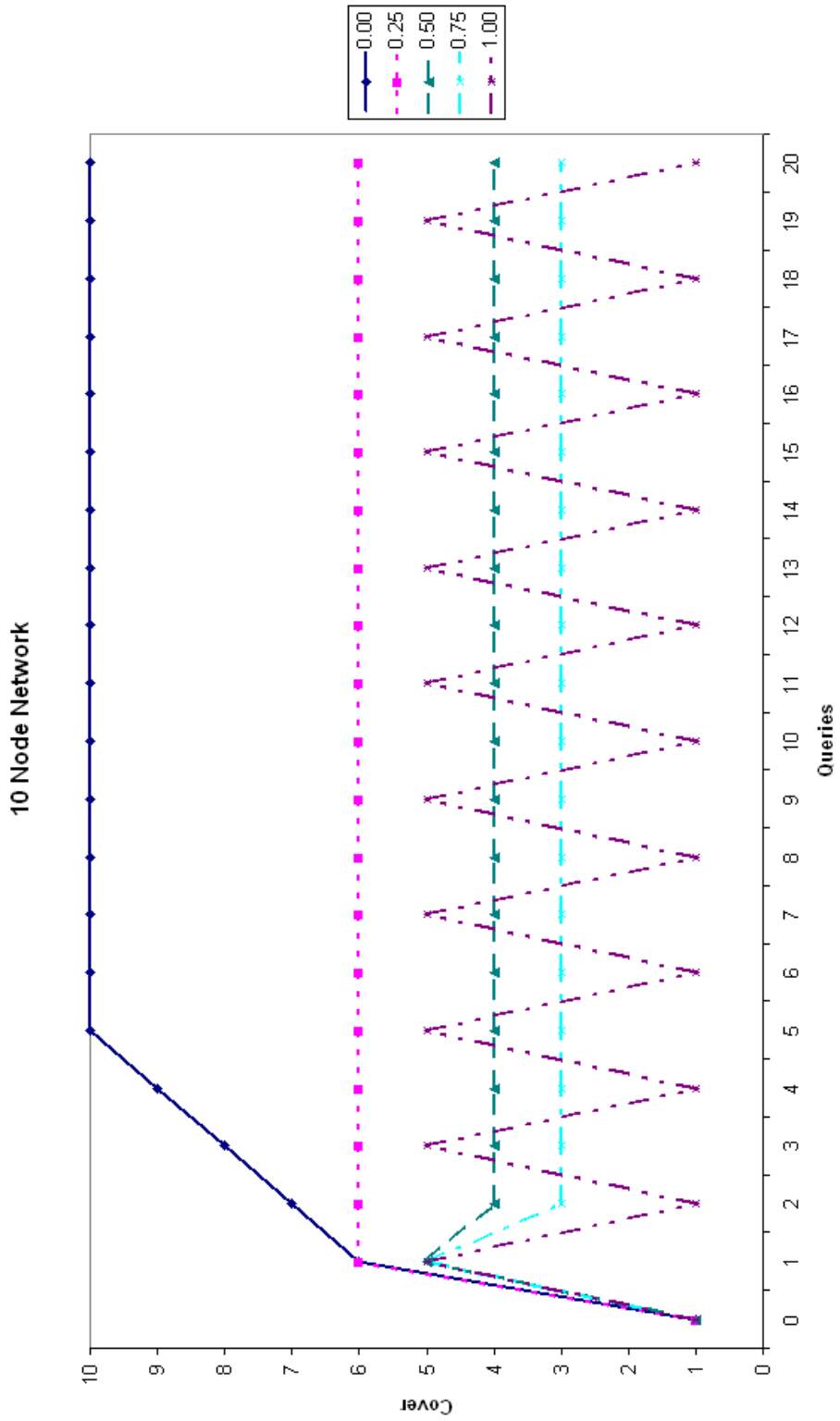


Figure 7.3: Change in coverage over time for a 10 node network and varying rates of decay.

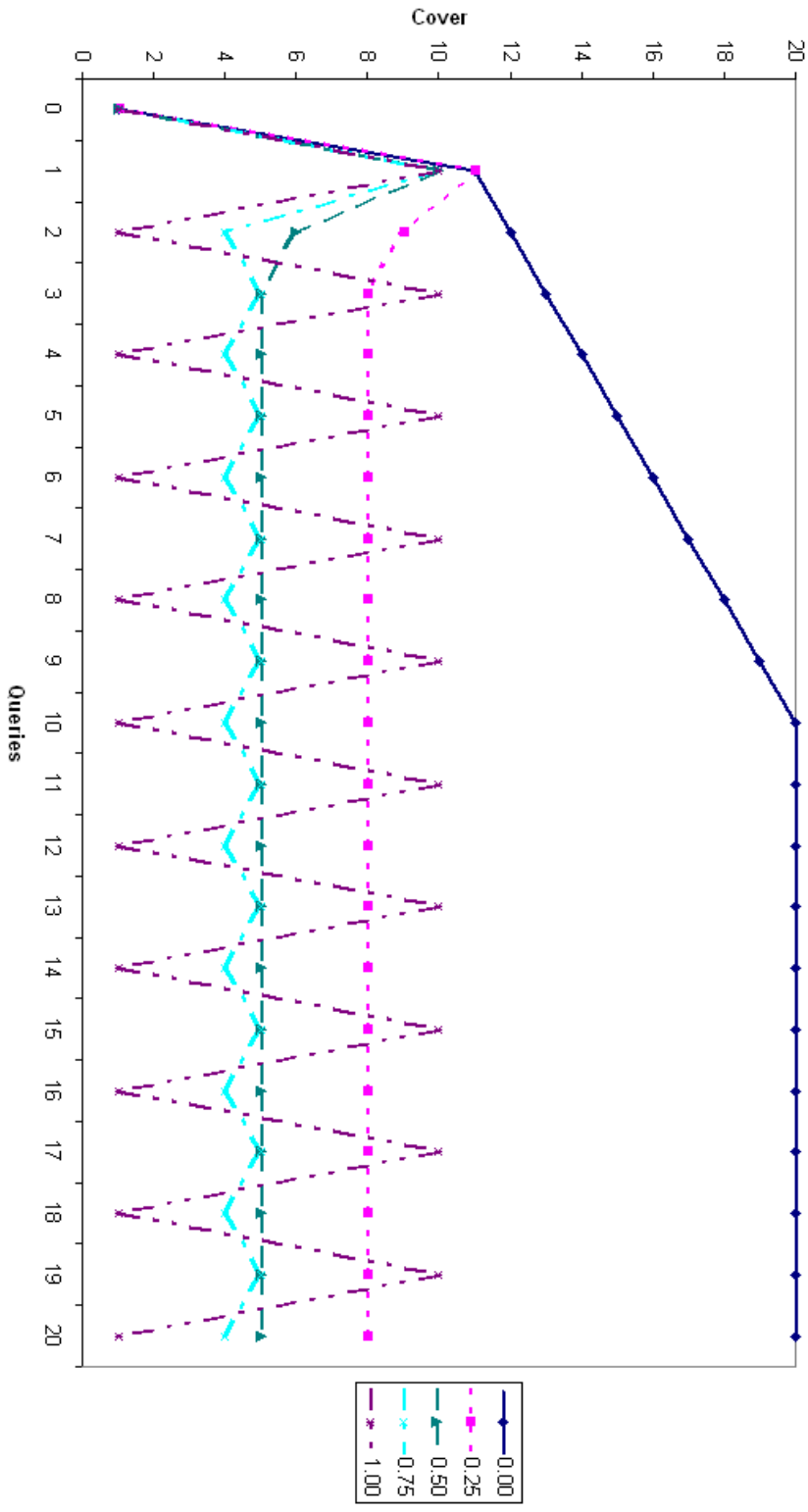


Figure 7.4: Change in coverage over time for a 20 node network and varying rates of decay.

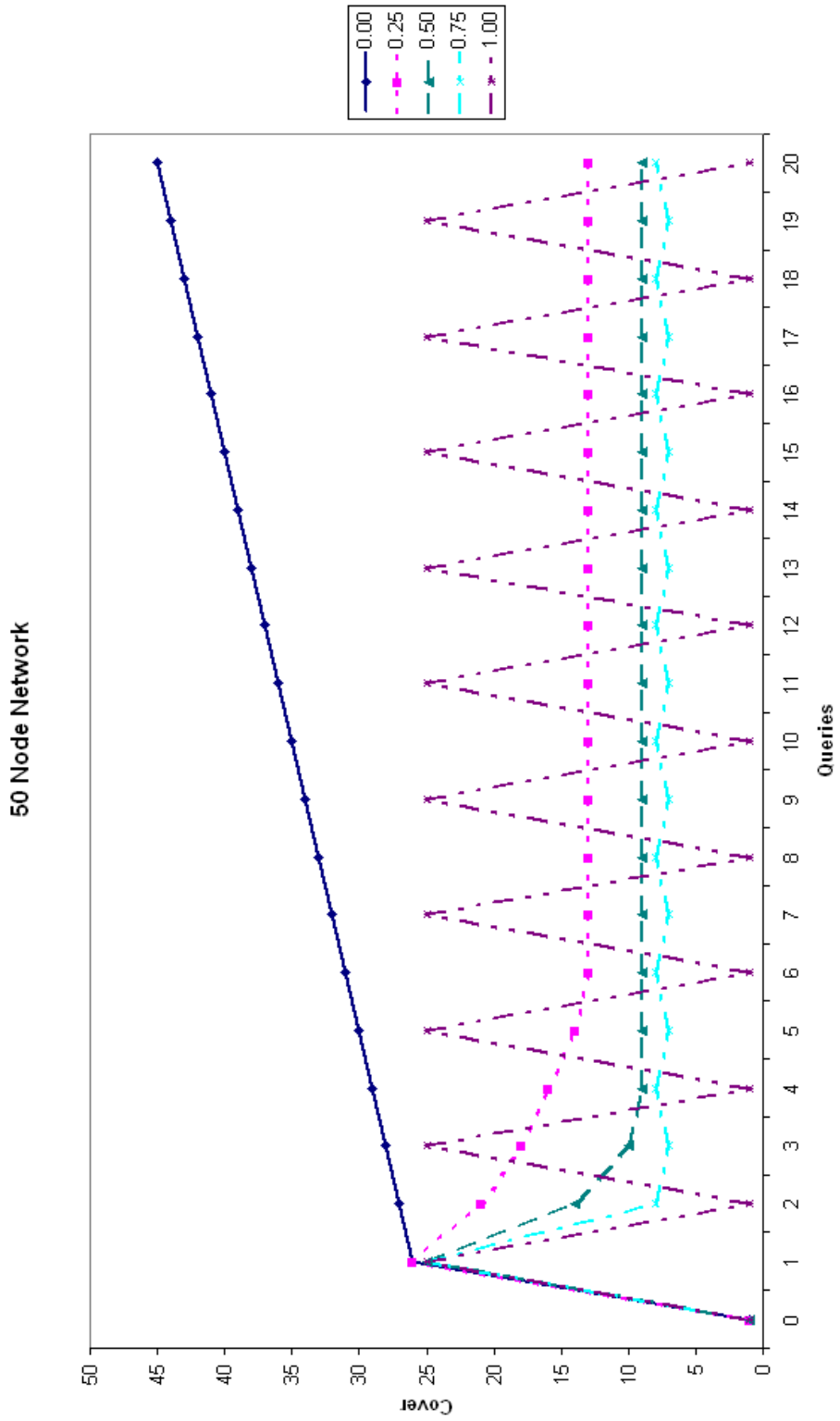


Figure 7.5: Change in coverage over time for a 50 node network and varying rates of decay.

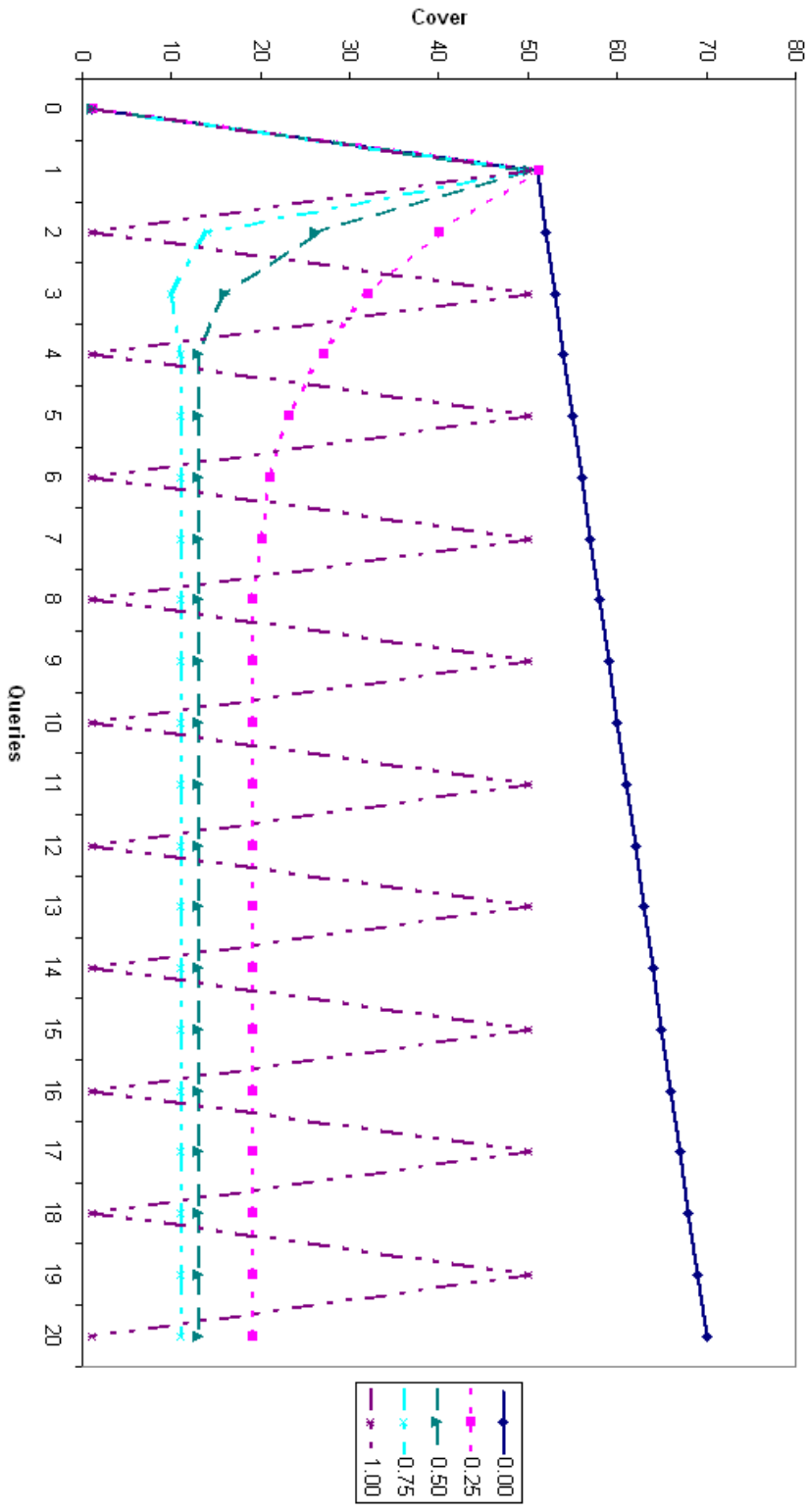


Figure 7.6: Change in coverage over time for a 100 node network and varying rates of decay.

as a periodic point attractor. This shows that order can be achieved in a resource discovery protocol whose foundation lies in a stochastic process.

In each figure, the extreme value of decay, $\delta = 0.00$ (no decay), shows the cover approach and reach the maximum number of nodes. Note that in the graphs for network sizes of fifty nodes and above, it takes more than twenty queries before maximum coverage is reached, and therefore the diamond line does not appear to plateau. The other extreme value of decay, $\delta = 1.00$ (maximum decay), shows the coverage fluctuating between one node and fifty percent of the number of nodes. The intermediate values of decay are of most interest.

In general, when the decay rate is high, the overall storage requirements of the protocol are relatively low, but the expected number of hops before a query is resolved is relatively high. On the other hand, when the decay rate is low, the overall storage requirements of the protocol are relatively high, but the expected number of hops before a query is resolved is relatively low.

When $\delta = 0.75$, it is noticeable that for some network sizes (twenty and fifty nodes), the cover settles into a simple cyclic attractor of two fixed points. For some rates of decay greater than 0.75 (not shown), the fluctuations become even more pronounced, until the most extreme fluctuation case arises at a decay rate of 1.00.

Fixed point coverage is consistently achieved with decay rates of 0.25 and 0.5. For example, on the 100 node network, when $\delta = 0.25$, the cover evolves to a fixed point of 19 nodes. When $\delta = 0.5$, cover stabilises at 13 nodes.

Route Lengths

Expected cover translates into an expected route length, where route length is defined to be the number of nodes traversed en route to a node covered by the resource. Clearly, there is a trade-off between coverage and route length: a high

coverage leads to shorter route length, while a low coverage leads to longer route length. Ideally, coverage should be as high as possible, without overwhelming the nodes (maintaining a high coverage is expensive in terms of storage costs when there are many advertised resources) and without settling into a periodic point attractor whose points are widely dispersed (as is the case for $\delta = 1.0$). Figure 7.7 graphs the attractors for cover and route length with varying rates of decay on a 100 node network. Note that where the decay rate leads to a periodic point attractor, the average of the fixed points is shown. Most notably, this occurs for decay rates above 0.85. The graph shows that as the rate of decay increases, the expected path length increases much more slowly than the coverage decreases. In other words, the expected path length stays relatively low, even as the coverage is lowered. This suggests that route lengths stay fairly short even for large values of δ.

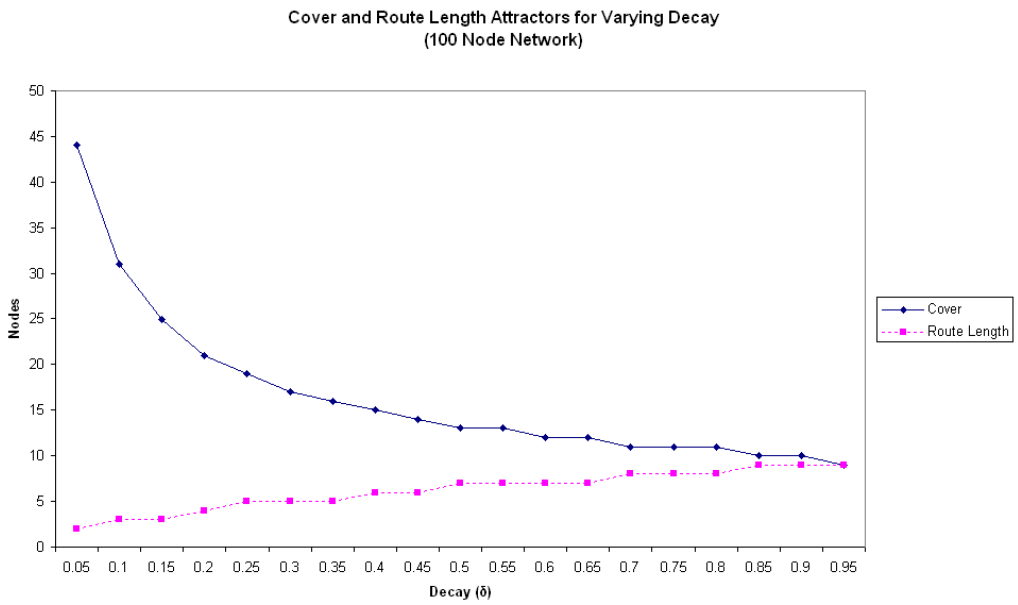


Figure 7.7: Cover and route length attractors for increasing decay.

These results are important in terms of identifying an appropriate size for the

Bloom filter in each query message. Recall from Section 4.5.2 that the size of the Bloom filter should be dependent upon the expected number of nodes that a query will traverse. For example, as Figure 7.7 shows, in a network of 100 nodes and a decay rate of 0.75, we expect to traverse no more than 10 nodes. Thus, the Bloom filter in the query message should be set between 80 and 160 bits in size. In the prototype, the Bloom filter is set to 128 bits in size. The size of the filter can be adjusted to match the deployment size. In fact, the choice of filter size need not be agreed upon by all nodes in the network. Each node is free to choose its own filter size.

Another way of evaluating the cost of a query is by examining the number of nodes that process a query. For this protocol, the number of nodes that process the query is equal to the route length. Therefore, the expected number of nodes that process any given query is a small proportion of the total number of nodes on the network. In comparison, a broadcast based protocol requires every node on the network to process every query (assuming the diameter of the network is less than the maximum hop limit, which, if the assumptions applied to the rest of this analysis are carried over, is certainly the case), and in fact, most nodes will receive each query multiple times. The benefits of the ant foraging approach are clearly evident in light of this.

This analysis shows that, even in networks on the order of hundreds of devices and where no pro-active advertising is performed (an advertisement radius of zero), the number of hops required by a query to discover a matching service is not large, meaning that queries are processed in a timely manner. Unlike DEAPspace [47], excessive network bandwidth is not consumed when no devices are emitting queries. Moreover, the protocol analysed here is capable of operation in heterogeneous and multi-hop networks - a claim that DEAPspace cannot make. Superstring's protocol for unstructured networks is also free of overhead

incurred by address-based routing protocols, such as AODV [86] and DSR [85], and multicast routing protocols for mobile ad hoc networks. Other service discovery protocols for ad hoc networks, such as Konark [11] and SSDP [4], rely on these routing schemes, and thus must incur the communication overhead associated with them. In short, this analysis shows that the paradigm used by Konark and DEAPspace, whereby all nodes have a full view of the network, is by no means the only choice for service discovery in MANETs. In fact, the analysis shows that in the same environments addressed by Konark and DEAPspace, namely those wireless networks in which all nodes are within broadcast range of one another, the Superstring protocol for unstructured networks outperforms these protocols in terms of communication overhead. It also demonstrates that, in many cases, it is unnecessary to design protocols which explicitly guarantee that services will be located if they exist, since the evolution of resource coverage in conjunction with the initial advertisement radius and maximum query hop settings mean that resources are discovered with high probability. This probability reaches 100 percent when the maximum number of query hops exceeds the number of uncovered nodes in the network (for a particular resource).

Suggested Improvements

The model also suggests some improvements that could be made to the protocol. At present, advertising is performed by broadcasting a service description within an h hop radius, where h can be 0, in which case no advertising is performed (recall that $h = 0$ is an assumption of this analysis). The above analysis shows that the first query usually incurs the greatest cost in terms of the number of nodes the query must visit before finding the required resource. However, if the description were to be advertised to a set of nodes whose cardinality is equal to the stable cover for a given network size and rate of decay, the first query would incur no

greater expense than the succeeding queries. Instead, the cost would be borne by the advertisement. But this improvement relies on knowledge of the network size, which is not a value that can be precisely calculated by each node individually, unless the nodes are fully interconnected.

The theoretical model argues for aggressive cache purging (trail dissipation). The implementation, as it stands, utilises a time-based configurable parameter for controlling the purging of resource descriptions cached by nodes and, for testing, this parameter was set to five minutes. In hindsight, and with the benefit of the mathematical model, five minutes is a very conservative lifetime for descriptions when queries are issued at an interval much less than five minutes. As the preceding graphs show, the risk of a query having to visit a large proportion of the total number of nodes is exceedingly small, even for high rates of decay. In fact, consistently long routes (routes where fifty percent or more of the nodes must be visited) are likely only when cached resources expire more frequently than queries are issued. These results suggest that a shorter lifetime for cached resources will not have a large impact on the performance of the protocol. Furthermore, a shorter lifetime is preferable in environments with highly mobile nodes, to ensure that the risk of returning a stale description (a description pertaining to a resource hosted by a node that has left the network) is kept as low as possible. The model also suggests a different way in which to manage description lifetimes. Instead of associating a description in the cache with a purge time and then checking at each time step to see whether that purge time has been reached, the node can discard the description with a given probability at each time step, thereby mirroring the theoretical model. One benefit of this approach is that a timer (or time value) need not be associated with each resource.

7.2.2 Experimental Results

The experiments for the local-area protocol were conducted on a stable network of ten nodes. In each experiment, twenty queries were issued at a constant interval of fifteen seconds. Thus, each experiment lasted for approximately five minutes. Two variations of the prototype were used in these experiments. In the first experiment, rates of decay were mapped to cache lifetimes. Thus, since the query interval was fifteen seconds, a decay of 0.25 mapped to a cache lifetime of sixty seconds, a decay of 0.5 mapped to a cache lifetime of thirty seconds, and a decay of 0.75 mapped to a cache lifetime of twenty seconds. For the second experiment, the prototype was modified so that purging of descriptions at each time step was probabilistic, as suggested by the mathematical model derived in the previous section. To achieve this, at each time step (which was set equal to the query interval), a resource is purged with a probability equal to the rate of decay.

Each of these experiments was run twenty times (that is, the experiment was run twenty times for each decay rate and prototype combination) so that average figures could be calculated. The experiments were automated by a distributed test framework. In each experiment, one node was selected randomly to store a resource description initially. For each query, the test framework selected one node at random to issue the query.

Figure 7.8 shows the evolution of coverage for the experiment in which decay rates were mapped to cache lifetimes. The figure shows that the results of this experiment strongly resonate with the output from the mathematical model (Figure 7.3). The attractor values for each of the decay rates in the experiment are practically identical to those produced by the theoretical model. The only real point of divergence is at the beginning of each experiment (most notably for a decay rate of 0.25), but this is because initially there are no cache timeouts for a period equal to the cache lifetime. Thus, for a decay rate of 0.25, a whole minute passes

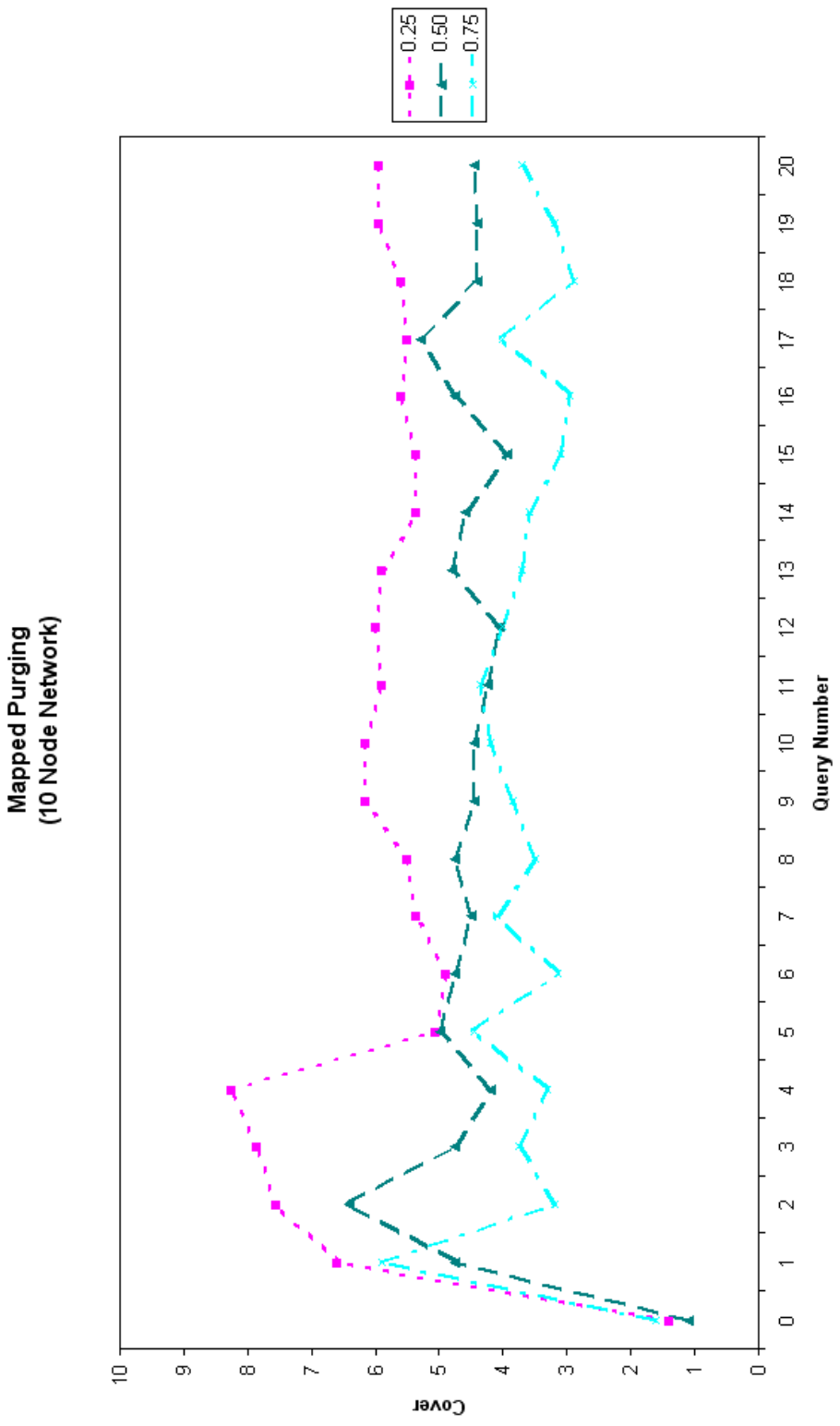


Figure 7.8: Prototype variant 1. Decay rates (0.25, 0.50 and 0.75) were mapped to cache lifetimes (60, 30 and 20 seconds).

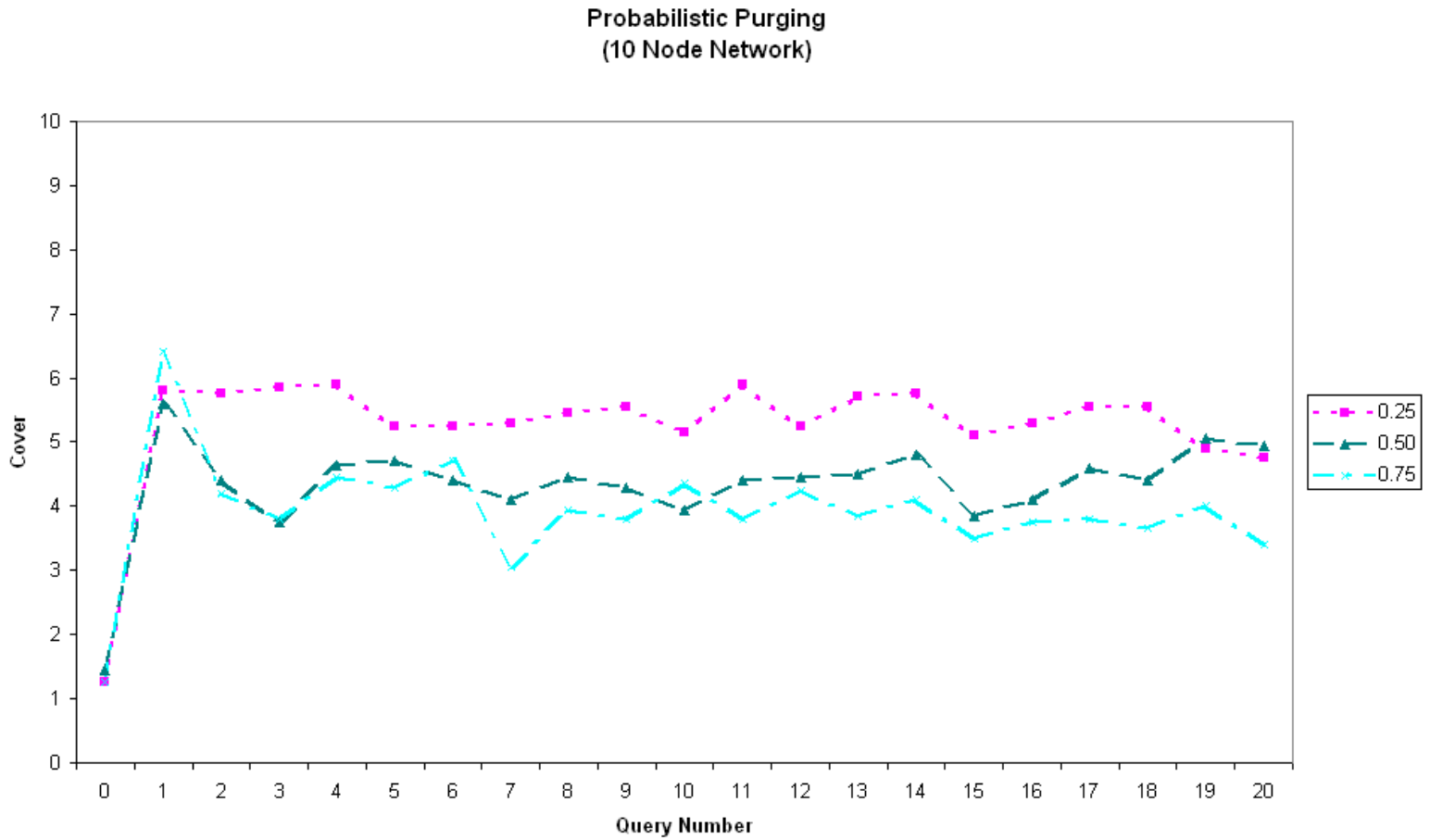


Figure 7.9: Prototype variant 2. Resource descriptions are purged with a probability equal to the decay rate in each experiment.

(in which four queries are issued) before a single cache timeout occurs. After this period, the experiment concurs precisely with the model.

Figure 7.9 depicts the evolution of coverage for the experiment in which decay rates are used directly by the prototype, where they define the probability with which a description will be purged from the cache. This version of the prototype bears a more direct resemblance to the mathematical model, which explains why there is no divergence from the mathematical model in the initial part of each experiment.

The experiments show that there is no appreciable difference between the two implementations once the system has reached equilibrium. Furthermore, the closeness with which the experiments agree with the mathematical model indicates that the model can be used to predict performance of the protocol in networks of various sizes and under different decay rates.

Finally, although node disconnection has not been explicitly modelled or incorporated into the experiments, the idea of node disconnection can, to some extent, be factored into the decay rate. A high rate of disconnection is comparable to a high rate of decay (or short cache lifetimes). In fact, in environments where the rate of disconnection is similar to the rate of node connection (that is where the churn rate is high, but the network size stays fairly constant), the decay rate encapsulates the churn rate.

7.3 Summary

The results presented in this chapter show that the core goals of this research have been reached; that is, to build a service discovery protocol that is scalable to small dynamic networks, and to large wide-area networks. In addition, the results vindicate the decision to treat the wide-area and local-area routing protocols separately.

This allowed each protocol to evolve independently of the other, and to solve the problems specific to each environment. In the wide-area, where nodes can be counted upon to utilise standardised, homogeneous protocols such as TCP/IP, efforts can be focused on achieving greater scalability. In Superstring, this scalability is achieved through the use of a protocol that distributes resource descriptions equitably over the set of resolvers, thereby enforcing a balanced query processing workload. In local-area pervasive environments, where mobility and heterogeneity are key characteristics, the nodes form a looser community, and organisation *emerges* from the bottom up. Such a strategy enables mobility to be handled in an elegant manner, and without the expenses incurred by broadcasting or multicasting solutions (indeed, because of the inherent network heterogeneity of these pervasive environments, such solutions are often not even possible).

This analysis also justifies the use of the complex systems theoretic approaches inherent in many aspects of this research (see Chapter 3). First, the local-area protocol utilises ideas from complex systems and biology to achieve a high level of performance even in the absence of more rigid guidelines (rigid in the sense that, for example, the wide-area protocol defines a routing structure specifically to achieve a high degree of scalability). Second, the analysis itself makes use of methods from complex systems (such as investigation of complex networks topologies and analysis of dynamic systems), which have highlighted improvements that could be made to the protocol. Order can usually be found in even the most complex, chaotic and nonlinear systems, and in analyses of those systems, order often manifests itself as an attractor. This is precisely what was found in the analysis of the local-area protocol, which is based upon a purely stochastic process.

CHAPTER

8 Conclusion

This chapter summarises the contributions this thesis has made to the field of resource discovery, and outlines areas of future work.

8.1 Contributions

The core goal of this thesis was to develop a resource discovery protocol capable of operating in the modern computing environment. Chapter 2 summarised the characteristics of the sub-environments that constitute the modern computing environment, thereby deriving a set of challenges which resource discovery protocols must overcome, and identifying the areas in which current resource discovery protocols fall short of the requirements.

The key challenges for resource discovery protocols arising from the survey were found to be

- scalability in terms of the numbers of resources and nodes, and the capability of devices;

- heterogeneity with respect to underlying network protocols, topologies and behaviours, as well as device differences and application requirements; and
- providing context-sensitive features that improve the utility of resource discovery protocols in pervasive computing environments.

A rich, yet lightweight resource description language and scalable and adaptable routing layers for queries were identified as the chief components in meeting these challenges.

Current resource discovery protocols fail to meet these challenges, primarily because they are targeted toward a specific subset of the modern computing environment. Therefore, if they have been designed to scale to large numbers of nodes in the wide-area they are unsuitable for use in smaller, ad hoc networks. Similarly, those protocols designed for mobile ad hoc networks are not usually adaptable to fixed networks because they do not scale to the wide-area.

Chapter 3 showed that the modern computing environment is an excellent example of a complex system. Having established this, the chapter proposed initial ideas that demonstrated the way complex systems theory could be utilised to overcome the key challenges present in designing resource discovery protocols for the modern computing environment.

These ideas were elaborated in Chapter 4, and then combined with the traditional engineering concepts of abstraction and separation of concerns to develop Superstring, a resource discovery protocol equipped to meet the challenges of contemporary computing environments. Two routing layers were developed, which together cover the immense diversity of computing environments found in the modern world. These were 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.

In addition to solving the key problems associated with resource discovery in

modern computing environments, the thesis presented a set of extensions to the core protocol that can endow applications with context-awareness. These extensions, documented in Chapter 5, introduced query persistence and advertisement transience to enable applications to be notified of changes to resources and the appearance of new resources. Furthermore, context-sensitive queries and advertisements were added to the description language so that queries and advertisements for resources could be made dependent upon the state of other resources. Finally, a context-sensitive result-ranking mechanism was introduced in order that matching resources could be ordered according to user preference or the state of one or more entities. These features were demonstrated in a proposed, context-sensitive application called *iCarpark*, which assists drivers to find free parking spaces when they approach a multilevel parking complex.

Chapter 6 documented a prototype of Superstring. This prototype incorporated the core features of the Superstring resource discovery protocol, and was designed to show the ability of the unstructured routing layer to operate in different network environments. It was tested on J2SE and J2ME platforms with a UDP transport. The structured layer included a lightweight implementation of Chord. The prototype embodied only the core features of Superstring, so that these could be tested during protocol analysis.

Finally, an analysis of the protocol was presented in Chapter 7. The analysis highlighted the ability of the structured layer to scale to extremely large numbers of resources by distributing data and workload among resolvers. The analysis for the unstructured layer was concerned with the manner in which knowledge of resources spreads across the network and the efficiency with which resources can be found by queriers. The analysis found that the unstructured protocol *evolves* the network so that a small proportion of nodes become aware of a particular resource, while the number of hops required to find a resource also remains small.

Mathematical models were created for both routing layers, enabling predictions to be made about the scalability of each layer.

8.2 Future Work

The contributions outlined above, while identifying solutions to many of the problems associated with resource discovery in modern computing environments, do not exhaust the domain of resource discovery research. The following sections propose areas for further research within the realm of resource discovery in general, and Superstring in particular.

8.2.1 A Superstring Gateway

To date, every bridging solution is either restricted to mapping two protocols [114, 118] or constrained to a narrow domain [12]. Section 2.6 also concluded that mappings will often be incomplete since some protocols support features that are not supported by others. Nevertheless, no single resource discovery protocol will ever gain complete dominance over the others. For that reason, new protocols must be designed with interoperability in mind, so that the task of bridging the new protocol with existing ones is simplified. A preliminary effort to define a generic bridge for Superstring has been made. This bridge simplifies the design requirements for bridging other protocols with Superstring, and removes a large portion of implementation work from developers. The approach is outlined here.

Executable and non-executable specifications have been utilised in many fields of computer science and engineering. For example, they have been used in the development of human-computer interfaces [167], rapid prototyping [168], software engineering simplification [169], protocol implementation [170] and translation of machine language syntax and semantics [171–173]. The last of these allows a

language to be fully described so that it can be dynamically translated to another language. However, these specifications do not describe the *behaviour* of the programs that are written in the language being translated. That is not their purpose. Therefore, if translation specifications were to be used in a bridging solution, an extra component would be needed so that the behaviour of one protocol could be bridged with that of another.

The proposed bridging solution is comprised of a generic bridge, *B*, and a small protocol specific component, *T*, for each protocol that Superstring interfaces with. *T* is responsible for shielding *B* from protocol *behaviour*. *B* is given specifications that describe the syntax and semantics of the description languages for each of the protocols it interfaces with. *B* and *T* communicate via a thin communication layer. Thus, *T* handles communication with a specific resource discovery protocol, while *B* is responsible for interpreting resource descriptions and queries. This enables *T* to be an extremely small component, thereby releasing developers from a large programming burden. The bulk of the work is done by *B*. Developers need only write a syntax and semantics specification and load that into *B*. Figure 8.1 shows the architecture of the bridge from a high level perspective. The bridge *B* mediates between Superstring resolvers (*R*) and entities executing a foreign protocol (*C*) via another intermediary component *T*. As with any bridging solution, in some cases it is impossible to map every element of Superstring's description language to third party description languages and vice-versa.

Research in this area is required to yield candidate specification languages, or to develop a new one drawing on research conducted in related areas [173]. This bridging solution must be tested in terms of its ability to translate between Superstring and other resource discovery protocols, the ease with which specifications can be written and *T* components can be implemented, the simplicity with which the solution can be deployed in the real world, and performance.

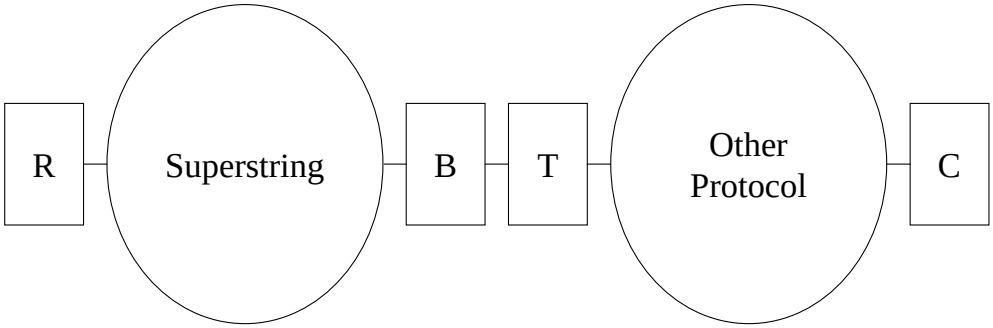


Figure 8.1: High level view of bridge architecture

8.2.2 Trust and Reputation Management

Preliminary research has been conducted on integrating reputation management with Superstring [174]. This has several significant benefits. First, the risky assumption that all services are trustworthy can be discarded. Second, clients can include reputation constraints in their queries, just as any other query constraints would be specified. Thus, only those services that meet the required reputation level are returned to the client. Initial work has yielded a set of API extensions and an algorithm for calculating service reputations.

One problem yet to be solved by the framework is the lack of incentive for clients to provide reputation ratings after they have used a service. Furthermore, while the algorithm takes precautions against a single entity providing multiple

reputation ratings for a particular service, there is no defence against entities that collaborate to artificially raise or lower the reputation of a service. This problem could be overcome (at least in part) by enabling *services to rate clients*. This solution could also be used to filter out extreme ratings provided by clients that have a history of doing so. However, no research has been conducted on defining appropriate algorithms to solve these problems and integrating them into Superstring.

8.2.3 Super-Lightweight Superstring

Superstring has been tested on a range of platforms, from high-end servers to mobile phones. However, it has not been tested on extremely resource poor platforms such as sensors [38]. While the unstructured protocol was designed specifically with energy-awareness in mind, the description language may not be suitable for such devices because it uses XML serialisation. The process of parsing XML is likely to be too intensive for small sensors. This assertion must be tested and an appropriate replacement found if Superstring is to scale down to the very smallest of devices in the modern computing environment.

8.2.4 Information Fusion

Information Fusion is the process of gathering data from various, often heterogeneous, sources and aggregating, combining and correlating that data according to a set of rules, usually for the purposes of making decisions about a given situation. Since Superstring is designed for a range of heterogeneous environments, including mobile computing environments, it is well placed to facilitate a dynamic kind of information fusion known as Opportunistic Information Fusion, or O'Fusion [175], in which data sources are discovered and integrated dynamically. Furthermore, it is possible that Superstring could be augmented with rules

that specify how information gathered from disparate sources should be fused together. In this manner, Superstring itself may be capable of providing O'Fusion natively with only minimal additions to the protocol.

8.2.5 Pervasive Computing Environments as Complex Systems

Chapter 3 showed that the modern computing environment is a complex system, and the work reported in this thesis was heavily influenced by complex systems theory. However, resource discovery is just one element of the modern computing environment. Complex systems theory can be applied to many other areas of pervasive computing research, just as it is being applied to diverse fields of study, including sociology, biotechnology, immunology and economics.

Previous work on context indicates that the relationships between types of context, such as location, user preferences and device capability are not simple or linear [104]. Rather, they are affected by each other and can be affected by user actions. Therefore, the interactions between context elements define a complex system. Treating context as a complex system may provide new insights into creating a context management system. Scalability is a major concern in context management. The range of context types and the frequency of context updates means that a context management system must deal with copious amounts of data, and it must decide how to filter and aggregate this data so it is usable by applications. The poor scalability of traditional approaches to context management has resulted in a tendency to focus on a limited subset of context information (usually location). This problem must be overcome if context-aware systems are to be widely deployed in modern computing environments. Natural complex systems provide a rich source of ideas and solutions to the problems of scale in context management (and indeed, in many domains of engineering), because they have evolved to overcome similar problems.

APPENDIX

A XML Mapping Examples

A.1 Example 1: Advertisement

```
<?xml version="1.0" ?>
<component name="xradiation-meter">
  <component name="name-record">
    <component name="resourceID">
      uuid
    </component>
    <component name="url">
      http://xray1.dar.csrio.au/
    </component>
  </component>
  <component name="location">
    <component name="physical">
      <component name="country" value="Australia">
```

```
        <component name="state" value="Northern
            Territory">
          <component name="suburb" value="Tennant
              Creek">
            <component name="street" value ="Barkly
                Highway">
              <component name="building" value="Barkly
                  Homestead"/>
            </component>
          </component>
        </component>
      </component>
      <component name="geodetic">
        <component name="latitude">
          −19.39
        </component>
        <component name="longitude">
          134.11
        </component>
        <component name="altitude">
          214
        </component>
      </component>
    </component>
    <component name="purpose">
      Measuring the intensity of x−radiation reaching the
```

```
        earth.
    </component>
</component>
```

A.2 Example 2: Query With Expression

```
<?xml version="1.0" ?>
<component name="xradiation-meter">
  <component name="location">
    <component name="geodetic">
      <component name="latitude">
        <exp>
          <![CDATA[(that < -19.00) && (that > -20.00)
              ]]>
        </exp>
      </component>
      <component name="longitude">
        <exp>
          <![CDATA[(that > 134.00) && (that < 135.00)
              ]]>
        </exp>
      </component>
    </component>
  </component>
</component>
```

A.3 Example 3: Query With Scoping

```
<?xml version="1.0" ?>
<component name="xradiation-meter">
  <component name="location">
    <component name="physical">
      <component name="country" value="Australia">
        <component name="state" value="Northern
          Territory">
          <scope>
            <component name="suburb" value="Tennant
              Creek">
              <component name="street" value ="Barkly
                Highway">
                <component name="building" value="
                  Barkly Homestead"/>
              </component>
            </component>
          </scope>
        </component>
      </component>
    </component>
  </component>
</component>
```

A.4 AWOL Document Type Definition

```
<!DOCTYPE AWOL [

<!ELEMENT component (component*|scope*|any?|not?|exp?|
    context?|bind?)>
<!ATTLIST component name CDATA #REQUIRED>
<!ATTLIST component value CDATA #IMPLIED>

<!ELEMENT scope (component+)>

<!ELEMENT any (component+|not?|context?)>

<!ELEMENT not (component+|any?|context?)>

<!ELEMENT exp (#PCDATA)>

<!ELEMENT context (component+)>
<!ELEMENT name CDATA #IMPLIED>

<!ELEMENT bind EMPTY>

]>
```

Bibliography

[1] Ricky Robinson and Andry Rakotonirainy. Multimedia customisation using an event notification protocol. In *International Workshop on Distributed Event-Based Systems (DEBS02)*, pages 549–554. IEEE Computer Society, July 2002.

[2] Louise Barkhuus and Anind Dey. Is context-aware computing taking control away from the user? Three levels of interactivity examined. In *The Fifth International Conference on Ubiquitous Computing*, pages 149–156, October 2003.

[3] 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.

[4] Yaron Y. Goland, Ting Cai, Ye Gu, and Shivaun Albright. Simple Service Discovery Protocol/1.0. IETF draft specification, October 1999.

[5] Bluetooth SIG. Bluetooth Specification version 1.1, February 2001.

[6] 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.

[7] 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.

[8] Sun Microsystems, Inc. Jini Technology Core Platform Specification v1.2. Technical report, Sun Microsystems, Inc, 2001.

[9] James Beck, Alain Gefflaut, and Nayeem Islam. Moca: a service framework for mobile computing devices. In *MobiDe '99: 1st ACM international workshop on Data engineering for wireless and mobile access*, pages 62–68, Seattle, Washington, United States, August 1999. ACM Press. ISBN 1-58113-175-5.

[10] HAVi Inc. The HAVi Specification. Technical report, HAVi Inc, 2001.

[11] Sumi Helal, Nitin Desai, Varum 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.

[12] Adrian Friday, Nigel Davies, Nat Wallbank, Elaine Catterall, and Stephen Pink. Supporting service discovery, querying and interaction in ubiquitous

computing environments. *Wireless Networks*, 10(6):631–641, November 2004. ISSN 1022-0038.

[13] Mike Esler, Jeffrey Hightower, Tom Anderson, and Gaetano Borriello. Next century challenges: data-centric networking for invisible computing: the portolano project at the university of washington. In *MobiCom '99: 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 256–262, Seattle, Washington, United States, August 1999. ACM Press. ISBN 1-58113-142-9.

[14] J. Postel. IETF RFC 793: Transmission Control Protocol, September 1981. URL `http://www.ietf.org/rfc/rfc793.txt`.

[15] J. Postel. IETF RFC 791: Internet Protocol, September 1981. URL `http://www.ietf.org/rfc/rfc760.txt`.

[16] Bluetooth SIG. Bluetooth Specification version 1.1, February 2001.

[17] 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.

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. IETF RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, June 1999. URL `http://www.ietf.org/rfc/rfc2616.txt`.

[19] Internet Society. IETF RFC 2821: Simple Mail Transfer Protocol, April 2001. URL `http://www.ietf.org/rfc/rfc2821.txt`.

[20] M. Crispin. IETF RFC 3501: Internet Message Access Protocol, Version 4rev1, March 2003. URL `http://www.ietf.org/rfc/rfc3501.txt`.

[21] J. Myers and M. Rose. IETF RFC 1939: Post Office Protocol - Version 3, May 1996. URL `http://www.ietf.org/rfc/rfc1939.txt`.

[22] Microsoft .NET homepage, December 2004. URL `http://www.microsoft.com/Net/`.

[23] Java 2 platform, enterprise edition (j2ee), December 2004. URL `http://java.sun.com/j2ee/`.

[24] W3C Recommendation: SOAP Version 1.2 Part 0: Primer, June 2003. URL `http://www.w3.org/TR/2003/REC-soap12-part0-20030624/`.

[25] IEEE. IEEE Std 802.3-2002, Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, 2002.

[26] IEEE. IEEE Std 802.11-1999, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.

[27] Bluetooth SIG. Bluetooth Specification version 1.1, February 2001.

[28] NASA. Information Power Grid, 2004. URL `http://www.ipg.nasa.gov/`.

[29] GrangeNet - GRid And Next GEneration Network, 2004. URL `http://www.grangenet.net/`.

[30] 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.

[31] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Laboratories, 2002. URL `http://citeseer.nj.nec.com/milojicic02peertopeer.html`.

[32] Krishna Kant, Ravi Iyer, and Vijay Tewari. A framework for classifying peer-to-peer technologies. In *CCGRID '02: 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 368. IEEE Computer Society, May 2002. ISBN 0-7695-1582-7.

[33] Napster. The napster homepage, 2003. http://www.napster.com/.

[34] Clip2. The gnutella protocol specification v0.4, 2003. URL `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`.

[35] 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.

[36] ISO/IEC. ISO/IEC 13818-3:1998 – generic coding of moving pictures and associated audio information – part 3: Audio, 1998.

[37] Sun Microsystems, Inc. JXTA v2.0 Protocols Specification, June 2004. URL `http://www.ietf.org/internet-drafts/draft-duigou-jxta-protocols-05.txt`.

[38] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Emerging Challenges: Mobile Networking for 'Smart Dust'. *Journal of Communication and Networks*, 2 (3):188–196, September 2000.

[39] 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.

[40] 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.

[41] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen (eds). W3C Recommendation: Extensible markup language XML 1.0, February 10 1998.

[42] The Salutation Consortium. Salutation architecture specification (part 1) v2.0c, June 1999.

[43] 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.

[44] 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.

[45] 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.

[46] W3C Working Draft: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, December 2004. URL `http://www.w3.org/TR/2004/WD-wsdl20-primer-20041221/`.

[47] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. DEAPspace: transient ad hoc networking of prevasive devices. *Comput. Networks*, 35(4):411–428, December 2001. ISSN 1389-1286.

[48] 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, September 1993. URL `http://www.ietf.org/rfc/rfc1521.txt`.

[49] Stuart Cheshire and Marc Krochmal. DNS-based service discovery. IETF Draft Specification, February 2004. URL `http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt`.

[50] Apple Computer, Inc. Rendezvous technology brief, October 2003. URL `http://images.apple.com/macosx/pdf/Panther_Rendezvous_TB_10232003.pdf`.

[51] Stuart Cheshire, Bernard Aboba, and Erik Guttman. Dynamic configuration of IPv4 link-local addresses. IETF Draft Specification, July 2004. URL `http://files.zeroconf.org/draft-ietf-zeroconf-ipv4-linklocal.txt`.

[52] Stuart Cheshire and Marc Krochmal. Multicast DNS. IETF Draft Specification, February 2004. URL `http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt`.

[53] P. V. Mockapetris. RFC 1035: Domain names — implementation and specification, November 1987. URL `http://www.ietf.org/rfc/rfc1035.txt`.

[54] 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`.

[55] W3C Recommendation: OWL Web Ontology Language Overview, February 2004. URL `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[56] W3C Note: Daml+oil (march 2001) reference description, December 2001. URL `http://www.w3.org/TR/daml+oil-reference`.

[57] Dipanjan Chakraborty, Anupam Joshi, Tim Finin, and Yelena Yesha. GSD: A novel group-based service discovery protocol for MANETs. In *4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN)*, pages 140–144, Stockholm. Sweden, September 2002. IEEE.

[58] W3C Recommendation: Resource Description Framework Primer, February 2004. URL `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

[59] Nenad Stojanovic, Rudi Studer, and Ljiljana Stojanovic. An approach for the ranking of query results in the semantic web. In *2nd International Semantic Web Conference*, pages 500–516. Springer-Verlag, October 2003.

[60] Youyong Zou, Tim Finin, and Harry Chen. F-OWL: an inference engine for the semantic web. In Michael Hinchey, James L. Rash, Walter F.

Truszkowski, and Christopher A. Rouff, editors, *Formal Approaches to Agent-Based Systems*, number 3228 in LNCS. Springer-verlag, November 2004.

[61] 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.

[62] R. Droms. IETF RFC 2131: Dynamic host configuration protocol, March 1997. URL `http://www.ietf.org/rfc/rfc1541.txt`.

[63] T. Berners-Lee, L. Masinter, and M. McCahill. IETF RFC 1738: Uniform resource locators (URL), December 1994. URL `http://www.ietf.org/rfc/rfc1738.txt`.

[64] K. Sollins. IETF RFC 2276: Architectural principles of Uniform Resource Name resolution, January 1998. URL `http://www.ietf.org/rfc/rfc2276.txt`.

[65] Sun Microsystems, Inc. IETF RFC 1094: NFS: Network File System Protocol specification, March 1989. URL `http://www.ietf.org/rfc/rfc1094.txt`.

[66] 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.

[67] Paul Castro and Richard Muntz. An adaptive approach to indexing pervasive data. In *MobiDe '01: 2nd ACM international workshop on Data engineering for wireless and mobile access*, pages 14–19, Santa Barbara, California, United States, May 2001. ACM Press. ISBN 1-58113-412-6.

[68] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *SOSP '99: seventeenth ACM symposium on Operating systems principles*, pages 186–201, Charleston, South Carolina, United States, December 1999. ACM Press. ISBN 1-58113-140-2.

[69] 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`.

[70] International Telecommunication Union (ITU). X.500 - Open Systems Interconnection - The Directory: Overview of concepts, models and services, April 2002.

[71] ISO/IEC. Open Distributed Processing - Trading function: Specification. ISO/IEC 13235-1:1998, February 2002.

[72] OMG. Trading Object Service version 1.0, May 2000. URL `http://www.omg.org/technology/documents/formal/trading_object_service.h%tm`.

[73] OASIS UDDI Specification TC. UDDI Version 3.0.1, October 2003. URL `http://uddi.org/pubs/uddi_v3.htm`.

[74] ISO/IEC. Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation). INCITS/ISO/IEC 9075-2-2003, December 2003.

[75] 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.

[76] Stanley Milgram. The small world problem. *Psychology Today*, pages 60–67, May 1967.

[77] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernado A. Huberman. Search in power-law networks. *Physical Review E*, 64(046135), October 2001.

[78] Nima Sarshar and Vwani Roychowdhury. Scale-free and stable structures in complex ad hoc networks. *Physical Review E*, 69(026101), February 2004.

[79] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. IETF RFC 2165: Service location protocol, June 1997. URL `http://www.ietf.org/rfc/rfc2165.txt`.

[80] E. Guttman, C. Perkins, J. Veizades, and M. Day. IETF RFC 2608: Service location protocol, version 2.

[81] T. Howes. RFC 2254: The string representation of LDAP search filters, December 1997. URL `http://www.ietf.org/rfc/rfc2254.txt`.

[82] 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.

[83] W. Zhao, H. Schulzrinne, and E. Guttman. IETF RFC 3528: Mesh-enhanced service location protocol, April 2003. URL `http://www.ietf.org/rfc/rfc3528.txt`.

[84] 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.

[85] David B Johnson and David A Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353, pages 153–181. Kluwer Academic Publishers, January 1996.

[86] C. E. Perkins and E. M. Royer. Ad Hoc On-Demand Distance Vector Routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, USA, February 1999.

[87] Brad Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *ACM/IEEE MobiCom: Mobile Computing and Networking*, pages 243–254. ACM Press, August 2000.

[88] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: 6th annual international conference on Mobile computing and networking*, pages 56–67, Boston, USA, August 2000.

[89] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. Data-centric storage in sensornets. *SIGCOMM Computer Communication Review*, 33(1):137–142, January 2003. ISSN 0146-4833.

[90] Rae Harbird, Stephen Hailes, and Cecilia Mascolo. Adaptive resource discovery for ubiquitous computing. In *2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 155–160, Toronto, Ontario, Canada, October 2004. ACM Press. ISBN 1-58113-951-9.

[91] Y. Rekhter and T. Li. IETF RFC 1771 - A Border Gateway Protocol 4 (BGP-4), March 1995. URL `http://www.faqs.org/rfcs/rfc1771.html`.

[92] J. Moy. IETF RFC 2328: OSPF Version 2, April 1998. URL `http://www.ietf.org/rfc/rfc2328.txt`.

[93] G. Malkin. RFC 2453: RIP Version 2, November 1998. URL `http://www.ietf.org/rfc/rfc2453.txt`.

[94] B. Segall and D. Arnold. Elvin has left the building: a publish/subscribe notification service with quenching. In *AUUG97: Australian UNIX and Open Systems Users Group Conference*, Brisbane, Australia, September 1997.

[95] G. Banavar, T.D Chandrea, B.Mukherjee, , J. Nagarajarao, R.E Strom, and D.C Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*, pages 262–272, Austin, USA, May 1999.

[96] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, San Diego, USA, August 2001. ACM Press. ISBN 1-58113-411-8.

[97] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, number 2218 in LNCS, pages 329–350, Heidelberg, Germany, November 2001. Springer-Verlag.

[98] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale over-

lay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

[99] Petar Maymounkov and David Maziéres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer-Verlag, March 2002. ISBN 3-540-44179-4.

[100] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, February 2003. ISSN 0001-0782.

[101] World Wide Web Consortium, 2004. URL `http://www.w3.org/`.

[102] T. Berners-Lee, R. Fielding, and L. Masinter. IETF RFC 2396: Uniform Resource Identifiers (URI): Generic syntax, August 1998. URL `http://www.ietf.org/rfc/rfc2396.txt`.

[103] Franklin Reynolds, Chris Woodrow, Hidetaka Ohto, and Graham Klyne. W3C Working Draft: Composite capability/preference profiles (CC/PP): Structure and vocabularies, January 2001.

[104] Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, and Karen Henricksen. Experiences in Using CC/PP in Context-Aware Systems. In *The 4th International Conference on Mobile Data Management*, number 2574 in LNCS, pages 247–261. Springer-Verlag, January 2003.

[105] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Seventh IEEE International Symposium on High Performance Distributed Computing*, page 140. IEEE Computer Society, July 1998. ISBN 0-8186-8579-4.

[106] Google search, October 2004. URL `http://www.google.com`.

[107] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[108] Anind K. Dey, Daniel Salber, Gregory D. Abowd, and Masayasu Futakawa. The conference assistant: Combining context-awareness with wearable computing. In *3rd International Symposium on Wearable Computers*, pages 21–28, Atlanta, USA, October 1999.

[109] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, places, things: web presence for the real world. *Mob. Netw. Appl.*, 7(5):365–376, October 2002. ISSN 1383-469X.

[110] Choonhwa Lee and Sumi Helal. Context Attributes: An Approach to Enable Context-awareness for Service Discovery. In *2003 International Symposium on Applications and the Internet (SAINT-2003)*, pages 22–30. IEEE Computer Society Press, January 2003.

[111] Henning Maaß. Location-aware mobile applications based on directory services. In *3rd annual ACM/IEEE international conference on Mobile computing and networking*, pages 23–33, Budapest, Hungary, August 1997. ACM Press. ISBN 0-89791-988-2.

[112] Jiangchuan Liu, Qian Zhang, Bo Li, Wenwu Zhu, and Jun Zhang. A Unified Framework for Resource Discovery and QoS-Aware Provider Selection in Ad Hoc Networks. *ACM Mobile Computing and Communications Review*, 6(1):13–21, January 2002.

[113] Karen Henricksen. *A framework for context-aware pervasive computing applications*. PhD thesis, School of Information Technology and Electrical Engineering, The University of Queensland, 2003.

[114] Brent Miller. Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer. white paper, Bluetooth Special Interest Group, July 1999. URL `http://www.salutation.org/whitepaper/BtoothMapping.PDF`.

[115] Pete St. Pierre and Tohru Mori. Salutation and SLP, 2002. URL `http://www.salutation.org/techtalk/slp.htm`.

[116] J. Allard, V. Chinta, S. Gundala, and G. G. Richard III. Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability. In *2003 Symposium on Applications and the Internet*, pages 268–275, Orlando, USA, January 2003. IEEE.

[117] Arun Ayyagari. Bluetooth ESDP for UPnP, 2001. URL `https://www.bluetooth.org/foundry/adopters/document/ESDP_for_UPnP_V0.95%/en/1/ESDP_for_UPnP_V0.95.pdf`.

[118] S. Kasper and L. Bührer. Jini Discovers Bluetooth. Semester thesis, Technische Informatik und Kommunikationsnetze, 2002. URL `http://www.csg.ethz.ch/people/lenders/JiniBluetooth.pdf`.

[119] E. Guttman and J. Kempf. Automatic Discovery of Thin Servers: SLP, Jini and the SLP-Jini Bridge. In *25th Annual Conference IEEE Industrial Electronics Society*, pages 722–727, San Jose, USA, November 1999. IEEE Press.

[120] The center for the study of complex systems, 2004. URL `http://www.cscs.umich.edu/complexity.html`.

[121] Adam Smith. *An Inquiry into the Nature and Causes of the Wealth of Nations*. Methuen and Co., Ltd, 1904. URL `http://www.econlib.org/library/Smith/smWN1.html`.

[122] Steven H. Strogatz. Exploring complex networks. *Nature*, 410:268–276, March 8 2001.

[123] Recurrence equation, 2004. URL `http://mathworld.wolfram.com/RecurrenceEquation.html`.

[124] Yaneer Bar-Yam. *Dynamics of complex systems*. Perseus Books, 1997. ISBN 0-201-55748-7.

[125] Albert-László Barabási. *Linked*. Perseus Publishing, 2002. ISBN 0738206679.

[126] Ricard V. Solé, Ramon Ferrer-Cancho, Jose M. Montoya, and Sergi Valverde. Selection, Tinkering, and Emergence in Complex Networks. *Complexity*, 8(1):20–33, September 2003.

[127] Gary William Flake. *The computational beauty of nature*. MIT Press, 1998. ISBN 0-262-06200-3.

[128] K.-H. Robèrt, B. Schmidt-Bleek, G. Basile, J. L. Jansen, R. Kuehr, P. Price Thomas, M. Suzuki, P. Hawken, and M. Wackernagel. Strategic sustainable development - selection, design and synergies of applied tools. *Journal of Cleaner Production*, 10(2):192–214, 2002.

[129] Philip A. Lawn. Ecological tax reform: Many know why but few know how. *Environment, Development and Sustainability*, 2(2):143–164, 2000.

[130] Robert M. May. *Stability and Complexity in Model Ecosystems*. Princeton University Press, 2001. ISBN 0691088616.

[131] Guy Theraulaz and Eric Bonabeau. A brief history of stigmergy. *Artificial Life*, 5(2):97–116, 1999. ISSN 1064-5462.

[132] David J. T. Sumpter and Madeleine Beekman. From nonlinearity to optimality: pheromone trail foraging by ants. *Animal Behaviour*, 66:273–280, 2003.

[133] Albert-László Barabási, Reka Albert, and Hawoong Jeong. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A*, 281:69–77, 2000.

[134] Albert-László Barabási, R. Albert, and H. Jeong. Mean-field theory for scale-free random networks. *Physica A*, 272:173–187, 1999.

[135] Zoltán Dezsö and Albert-László Barabási. Halting viruses in scale-free networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 65(5):055103, 2002. URL `http://link.aps.org/abstract/PRE/v65/e055103`.

[136] 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.

[137] Özalp Babaoglu, Hein Meling, and Alberto Montresor. Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems. In *22nd International Conference on Distributed Computing Systems*, pages 15–22, Vienna, Austria, July 2002. IEEE Computer Society.

[138] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research (JAIR)*, 9:317–365, 1998.

[139] Marc De Scheemaecker. NanoXML, October 2004. URL `http://nanoxml.cyberelf.be/`.

[140] kXML, October 2004. URL `http://kxml.org/`.

[141] Andrew S. Tanenbaum. *Computer Networks, 4th Edition*. Prentice Hall, 2002.

[142] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw-Hill, Inc., 1990. ISBN 0070131430.

[143] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *16th ACM International Conference on Supercomputing(ICS'02)*, pages 84–95, New York, USA, June 2002. ACM Press.

[144] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, June 2000.

[145] J. L. Deneubourg, J. M. Pasteels, and J. C. Verhaeghe. Probabilistic behaviour in ants: A strategy of errors? *Journal of Theoretical Biology*, 105: 259–271, 1983.

[146] Netcraft. Netcraft Web Server Survey. web, October 2004. URL `http://www.netcraft.com/`.

[147] Virgílio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing Reference Locality in the WWW. In *IEEE Conference on Parallel and Distributed Information Systems (PDIS)*, pages 92–103, Miami Beach, USA, December 1996. IEEE Computer Society.

[148] Kunwadee Sripanidkulchai. The popularity of Gnutella queries and its implications on scalability. Technical report, Carnegie Mellon University, February 2001. URL `http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html`.

[149] Bill N. Schilit. *A Context-Aware System Architecture for Mobile Distributed Computing*. Ph.D. thesis, Columbia University, May 1995.

[150] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. In *MobiCom '99: 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 59–68, Seattle, Washington, United States, 1999. ACM Press. ISBN 1-58113-142-9.

[151] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: a mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, October 1997. ISSN 1022-0038.

[152] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *2nd IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 77–86. IEEE Computer Society, March 2004.

[153] Sasitharan Balasubramaniam and Jadwiga Indulska. Vertical Handover Supporting Pervasive Computing. *Future Wireless Networks. Computer Communication Journal, Special Issue on 4G/Future Wireless Networks*, 27(8):708–719, 2004.

[154] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *seventh international conference on World Wide Web*

7, pages 107–117, Brisbane, Australia, 1998. Elsevier Science Publishers B. V.

[155] Peter Fishburn. Preference structures and their numerical representations. *Theoretical Computer Science*, 217(2):359–383, 1999. ISSN 0304-3975.

[156] Parking rage, December 2004. URL `http://www.drdriving.org/rages/parking.htm`.

[157] New Astra CDX Hatch Overview, December 2004. URL `http://www.holden.com.au/www-holden/action/modeloverview?modelid=26002`.

[158] Phillip B. Gibbons, Brad Karp, Suman Nath Yan Ke, and Srinivasan Seshan. IrisNet: An architecture for a worldwide sensor web. *IEEE Pervasive Computing*, 2(4):22–33, 2003.

[159] Toppan to produce $20 rfid reader, January 2003. URL `http://www.rfidjournal.com/article/articleview/279/1/1/`.

[160] ISO. ISO 11898-1:2003 – road vehicles – controller area network (CAN) – part 1: Data link layer and physical signalling, 2003.

[161] ISO. ISO 11898-2:2003 – road vehicles – controller area network (CAN) – part 2: High-speed medium access unit, 2003.

[162] EuroTag. New Bluetooth RFID Reader from Baracoda and Cathexis, September 2004. URL `http://www.eurotag.org/news/archives/Sep_2004/New_Bluetooth_RFID_Reader%.html`.

[163] Java 2 Platform, Standard Edition (J2SE), December 2004. URL `http://java.sun.com/j2se/`.

[164] Java 2 Platform, Micro Edition (J2ME), December 2004. URL `http://java.sun.com/j2me/`.

[165] R. Srinivasan. IETF RFC 1832: XDR: External data representation standard, August 1995. URL `http://www.ietf.org/rfc/rfc1832.txt`. Status: DRAFT STANDARD.

[166] Adriana Iamnitchi, Ian Foster, and Daniel C. Nurmi. A Peer-to-Peer Approach to Resource Discovery in Grid Environments. Technical Report TR-2002-06, Department of Computer Science, University of Chicago and Mathematics and Computer Science Division, Argonne National Laboratory, March 2002.

[167] Robert J. K. Jacob. Executable specifications for a human-computer interface. In *SIGCHI conference on Human Factors in Computing Systems*, pages 28–34, Boston, USA, December 1983. ACM Press. ISBN 0-89791-121-0.

[168] H. Klaeren and P. Thiemann. Merging formal methods with rapid prototyping. In *ICSE-17 Workshop on Formal Methods Application in Software Engineering Practice*, pages 39–46, 1995.

[169] A. Gerber, M. J. Lawley, K. Raymond, J. Steel, and A. Wood. Transformation: The Missing Link of MDA. In *1st International Conference on Graph Transformation (ICGT 2002)*, pages 90–105. Springer-Verlag, October 2002.

[170] Yen-Min Huang and Chinya V. Ravishankar. Linguistic support for controlling protocol execution. In *14th International Conference on Distributed Computing Systems*, pages 581–588, Pozman, Poland, June 1994.

[171] Norman Ramsey and Mary F. Fernández. Specifying Representations of Machine Instructions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(3):492–524, May 1997.

[172] Cristina Cifuentes and Shane Sendall. Specifying the Semantics of Machine Instructions. Technical Report 422, The University of Queensland, December 1997.

[173] C. Cifuentes and M. Van Emmerik. UQBT: Adaptable Binary Translation at Low Cost. *Computer*, 33(3):60–66, March 2000.

[174] Ryan Wishart, Ricky Robinson, Jadwiga Indulska, and Audun Jøsang. SuperstringREP: Reputation-enhanced service discovery. In *Computer Science 2005 (CRPIT)*, pages 49–57, Newcastle, Australia, February 2005. Australian Computer Society Inc.

[175] S. Challa, T. Gulrez, Z. Chaczko, T. N. Pranesha, and J. Indulska. Opportunistic information fusion: A new paradigm for next generation networked sensing systems. In *The Eighth International Conference on Information Fusion (Fusion 2005)*, July 2005.