APPROVAL SHEET

Title of Thesis: Service Discovery and Composition in Pervasive Environments

Name of Candidate: Dipanjan Chakraborty
Doctor of Philosophy, 2004

Thesis and Abstract Approved: _____
Dr. Anupam Joshi
Associate Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

CURRICULUM VITAE

Name: Dipanjan Chakraborty.

Permanent Address: 19A P.G.M Shah Road. Flat 502. Kolkata: 700033. India

Degree and date to be conferred: Doctor of Philosophy, 2004.

Date of Birth: November 1, 1976.

Place of Birth: Kolkata. India

Collegiate institutions attended:

- Jadavpur University, India.

  Bachelor of Computer Science and Engineering, 1999.

- University of Maryland, Baltimore County,

  Master of Science, Computer Science, 2002.

- University of Maryland, Baltimore County,

  Doctor of Philosophy, Computer Science, 2004.

Major: Computer Science.

Professional publications:

- Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, Timothy Finin, “Service Composition for Mobile Environments”, Journal on Mobile Networking and Applications, Special Issue on Mobile Services (MONET), 2004. Accpeted for publication.

- Olga Ratsimor, Dipanjan Chakraborty, Anupam Joshi, Timothy Finin, “Service Discovery in Agent-based Pervasive Computing Environments” Journal on Mobile Networking and Applications (MONET), Special Issue on Mobile and Pervasive Commerce, 2003. Accepted for publication.

- Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, “Integrating Service Discovery with Routing and Session Management for Ad Hoc Networks”, Ad Hoc Networks Journal by Elsevier Science, 2004. Accepted for publication.

- Dipanjan Chakraborty, Hui Lei, “Pervasive Enablement of Business Processes”, 2nd International Conference on Pervasive Computing and Communications (PerCom), Orlando, Florida, March 2004. In Proc.

- Harry Chen, Filip Perich, Dipanjan Chakraborty, Tim Finin, Anupam Joshi, “Intelligent Agents meet Semantic Web in a Smart Meeting Room”, 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systsms (AAMAS), New York, July 2004. In Proc.

- Dipanjan Chakraborty, Avinash Shenoi, Yacov Yesha, Yelena Yesha, “A Queuing Theoretic Model for Service Discovery in Ad Hoc Networks”, Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS), San Diego, California, January 2004. In Proc.

- Dipanjan Chakraborty, Yelena Yesha, Anupam Joshi, “A Distributed Service Composition Protocol for Pervasive Environments”, IEEE Wireless Communications and Networking Conference (WCNC), Atlanta, Georgia, March 2004. In Proc.

- Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Timothy Finin, Yelena Yesha, “A Reactive Service Composition Architecture for Pervasive Computing Environments”, 7th Personal Wireless Communications Conference (PWC), Singapore, October 2002.

- Dipanjan Chakraborty, Anupam Joshi, “GSD: A Novel Group-based Service Discovery Protocol for MANETS”, 4th IEEE Conference on Mobile and Wireless Communication Networks (MWCN), Stockholm, Sweden, September 2002. In Proc.

- Filip Perich, Sasikanth Avancha, Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, “Profile Driven Data Management for Pervasive Environments”, 13th International Conference on Database and Expert Systems Applications (DEXA), Aix-en-Provence, France, September 2002. In Proc.

- Olga Ratsimor, Dipanjan Chakraborty, Anupam Joshi, Timothy Finin, “Allia: Policy-bsed Alliance Formation for Agents in Ad-hoc Environments”, 2nd ACM Mobile Commerce Workshop held in Conjunction with MobiCom, Atlanta, Georgia, September 2002. In Proc.

- Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Timothy Finin, Yelena Yesha, “Middleware for Mobile Information Access”, 5th International Workshop on Mobility in Databases and Distributed Systems (MDDS), in conjunction with DEXA, Aix-en-Provence, France, September 2002. In Proc.
- Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, Anupam Joshi, “DReggie: A Smart Service Discovery Technique for E-Commerce Applications”, Workshop in conjunction with 20th Symposium on Reliable Distributed Systems (SRDS), New Orleans, October 2001. In Proc.
- Sasikanth Avancha, Dipanjan Chakraborty, Dhiral Gada, Tapan Kamdar, Anupam Joshi, “Fast and Effective Wireless Handoff Scheme using Forwarding Pointers and Hierarchical Foreign Agents”, Invited paper to Conference on Modeling and Design of Wireless Networks, Denver, Colorado, August 2001. In Proc.
- Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, Anupam Joshi, “An Agent Discovery Architecture using Ronin and DReggie”, First GSFC/JPL Workshop on Radical Agent Concepts (WRAC), NASA Goddard Space Flight Center, Maryland, September 2001. In Proc.
- Harry Chen, Dipanjan Chakraborty, Liang Xu, Anupam Joshi, Tim Finin, “Service Discovery in the Future Electronic Market”, Working notes of the Seventeenth National Conference on Artificial Intelligence, Eleventh Innovative Applications for AI Conference, Austin, Texas, July 2000.
- Chaitanya Pullela, Liang Xu, Dipanjan Chakraborty, Anupam Joshi, “Component-based Architecture for Mobile Information Access”, Workshop in conjunction with International Conference on Parallel Processing (ICPP), August 2000. In Proc.
- Samiran Chattopadhyay, Dipanjan Chakraborty, Aditya Chakraborty, “Design and Performance of Modeling of a Distributed Computing Environment based on CORBA”, International Conference on Modeling and Simulation, Jaipur, India, November 1999. In Proc.
- Dipanjan Chakraborty, Hui Lei, “Extending the Reach of Business Processes”, IEEE Computer, 2004, To appear.
- Dipanjan Chakraborty, Anupam Joshi, “Anamika: Distributed Service Composition Architecture for pervasive Computing Environments”, ACM SIGMOBILE Mobile Computing and Communications Review (MC2R), Volume 8, Issue 1, January 2003.

- Sasikanth Avancha, Dipanjan Chakraborty, Filip Perich, Anupam Joshi, “Data and Services for Mobile Computing”, Handbook of Internet Computing, CRC Press, 2003.
- Dipanjan Chakraborty, Harry Chen, “Service Discovery in the Future for Mobile Commerce”, Association for Computing Machinery (ACM) Crossroads, Winter 2000 (7.2).
- Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, Timothy Finin, “Towards Distributed Service Discovery in Pervasive Computing Environments”, Submitted to Journal.
- Dipanjan Chakraborty, “Service Composition in Ad-hoc Environments”, Ph.D. Dissertation Proposal, Technical Report TR-CS-01-21, Department of Computer Science and Engineering, University of Maryland, Baltimore County, 2001.
- Dipanjan Chakraborty, Filip Perich, Anupam Joshi, “An Agent-based Approach to Ubiquitous Information Access”, Technical Report TR-CS-01-21, Department of Computer Science and Engineering, University of Maryland, Baltimore County, 2001.
- Dipanjan Chakraborty, Anupam Joshi, “Dynamic Service Composition: State-of-the-Art and Research Directions”, Technical Report TR-CS-01-19, Department of Computer Science and Engineering, University of Maryland, Baltimore County, 2001.

Professional Positions held:

- Research Staff Member (Present)
  IBM Research Labs, India.
- Instructor (January 2004 - June 2004)
  Department of Computer Science and Engineering, University of Maryland, Baltimore County.
- Summer Intern (May 2003 - August 2003)
  IBM Thomas J. Watson Research Center, Yorktown Heights, New York
- Contractor (May 2002 - August 2002)
  IBM Software Laboratory, Toronto, Canada
- Contractor (May 2001 - August 2001)
  IBM Software Laboratory, Toronto, Canada

- Summer Intern (May 2000 - July 2000)

  Aether Systems, Owings Mills, Maryland

- Graduate Research Assistant (January 2000 - December 2003)

  Department of Computer Science and Engineering, University of Maryland, Baltimore County

- Graduate Teaching Assistant (August 1999 - December 1999)

  Department of Computer Science and Engineering, University of Maryland, Baltimore County

ABSTRACT

Title of Dissertation: Service Discovery and Composition in Pervasive Environments

Dipanjan Chakraborty, Doctor of Philosophy, 2004

Dissertation directed by: Dr. Anupam Joshi
Associate Professor
Department of Computer Science and
Electrical Engineering

Service discovery and composition has been an active area of research in the domain of wired web-based environments as well as mobile and pervasive environments. The advent of mobile computing elements (handhelds) along with the concomitant development of wireless and ad-hoc networking technologies such as Bluetooth, 802.11, Infrared etc have recently led to the growth of *infrastructure-less pervasive environments* - environments that do not assume the existence of the umbilical cord: a stable network backbone. Examples of such environments range from war front activities, fire fighting, futuristic malls, and smart traffic scenarios to deep space exploration research. Services - resources, data as well as computation components are resident on heterogeneous devices and connected through heterogeneous networking technologies in such environments. These type of environments pose several interesting and challenging issues in the development of end applications that intend to utilize the resource-rich vicinity. Service discovery and composition are key technologies to enable end applications in such environments.

Research in service discovery and composition has mostly focused on wired/web-based environments and infrastructure-based mobile and pervasive environments - environments where devices can connect to the stable network backbone and access services in the wired infrastructure. Understandably, architectures proposed for service discovery and composition are mostly centralized or semi-centralized and registry-based. However, such architectures are unsuitable for infrastructure-less pervasive environments. Current solutions

for service discovery and composition in infrastructure-less environments are primarily broadcast-based and are inefficient in the utilization of networking and computing resources. Solutions focus on discovering services and composing them in a constrained network following a peer-to-peer model and do not address the issues of scalability and network-wide reachability. Moreover, issues of mobility and adaptability of protocols with respect to varying environments is also not addressed.

This dissertation identifies key issues related to service discovery and composition in infrastructure-less pervasive environments. It proposes a conceptual architecture to enable integrated service discovery, service-centric routing and service composition in infrastructure-less pervasive environments. We develop, implement and extensively evaluate distributed protocols at various layers of the conceptual architecture. The design of the architectures and the protocols use several cross-layer optimizations that improve the overall performance of the system.

The primary contributions of this dissertation are the architecture and the novel and innovative protocols that enable distributed and integrated service discovery, service-centric routing and service composition in pervasive environments. Additionally, we also propose and evaluate a queuing theoretic formalism to model the service cache - the basic unit that affects discovery and composition protocols. We design an ontology grounded in Web Ontology Language (OWL), to describe services in pervasive environments in terms of its inputs and outputs, platform constraints and device capabilities. This information is used in various aspects of distributed discovery and composition in pervasive environments.

Service Discovery and Composition in Pervasive Environments

by
Dipanjan Chakraborty

Dissertation submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2004

To my parents and my lovely sister Piku

Firstly, I want to acknowledge my father and my mother. Their encouragement, support and blessings inspired me to achieve this goal in my life. Piku: my sister has been there beside me through various phases of my life in the last 5 years. I also need to acknowledge many other people who contributed to this work. I like to thank Anupam Joshi, my advisor and mentor whose guidance was invaluable. To my committee members: Tim Finin, Yelena Yesha, Arya Gangopadhyay, Hillol Kargupta and Mukesh Singhal - for your support and contributions. To Filip Perich and Sasikanth Avancha amongst many others in the lab - A wonderful collaborative environment in the lab would not have been possible without you. To all my friends, who made this phase of my life, a memory worth remembering.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I

INTRODUCTION

The paradigm of distributed computing underwent a significant change with the advent of mobile computing and wireless networks. Mobile and pervasive computing - as it is called - introduces the concept of computing without the support of the umbilical cord - a stable network backbone. With the growth of numerous handheld gadgets and the birth of new network paradigms like home networks, office networks, and ad-hoc and wireless networks, the importance of mobile and pervasive computing as the next enabling computing technology has increased tremendously. A very important concept of pervasive computing is the incorporation of human beings as an important and significant component in the computation cycle. Mobile devices, often times represent their "masters" - the human beings themselves - and contain information as well as services belonging to the person. This has led to the consequent advancement of various user-centric computing paradigms such as graphic display technologies (everywhere displays), context-aware computing, profile-driven computing, embedded computing, wearable computing and many others.

Pervasive computing promises to bring the world to the user's finger tips *anytime and anywhere*. In a shopping mall, handheld devices promise to collect discount coupons for materials that one is interested in. On a road, devices around us promise to provide our accurate GPS location and provide directions to our destination. In smart home environments, the coffee maker promises to turn on and start brewing coffee the moment the garage detects that the owner has pulled in after a long day. The Bluetooth headset promises to automatically disconnect with the MP3 player in the car and connect to the music player in the room. Pervasive computing promises rich enhancements in various other areas apart from helping us in our daily lives. Wireless sensor networks promise to track down tornadoes and warn us ahead of time. Applications of

pervasive technologies range from fire fighting, war-front activities, smart monitoring of patients in hospitals to deep space exploration research.

We categorize pervasive environments into *infrastructure-based* and *infrastructure-less* or *ad-hoc* environments. Infrastructure-based mobile environments have access to computing elements in the wired infrastructure through gateways, proxies and base stations. Examples of such environment are wireless office networks based on WiFi. These environments are able to leverage from the high bandwidth resource-rich wired environments. Infrastructure-less pervasive environments refer to environments where computing elements (mobile devices) do not have any access to the wired infrastructure and network connections with peer devices are created and broken down on-the-fly and on an as-needed basis. Examples of such environments are warfront activities, ad-hoc sensor networks, walkways in cities, futuristic malls etc.

Pervasive environments facilitate sharing and using of the resources, data and services on peer devices to enable maximum benefit of the computation-rich vicinity. A very important question in this area is: How do we model resources, data, computation components so that we can identify them unambiguously and utilize them? Over the last decade, various concepts have been researched starting from basic uniform programmatic function calls to object-level encapsulation and modeling of systems. Recently, service-centric modeling [53, 58] of devices/resources/computation entities has been introduced that enables much higher flexibility in the modeling of components in this environment. Consequently, service-oriented modeling and computing has come up over the past few years. Service discovery and composition are very important to enable development of end applications in pervasive environments.

I.A Definitions

Before we proceed, we define the terms service, service discovery and service composition as it applies to our research.

- **Service:** At its lowest level of granularity, the term *service* can be defined as any hardware or software entity that resides on a mobile device or platform and has distinct functional attributes that characterizes it and is accessible by other services. For example, we can consider the GPS on a mobile device to be a service that provides location coordinates of that device.

- **Service Discovery:** Service Discovery refers to the act of (1) discovering hardware components or software entities (resources, data or computation components) on peer devices; (2) determining how to

invoke or utilize the services. Some protocols (e.g. Bluetooth [14]) constrain itself to only discovering services though. This dissertation looks at obtaining service invocation information as an integral part of service discovery.

- **Service Composition:** Service composition refers to the technique of creating *composite services* with the help of smaller, simpler and easily executable services or components.

Service discovery and composition has been an extremely active area of research over the last five years. However, most of the research has been done in the context of wired or infrastructure-based pervasive and mobile environments [96, 20, 28, 35, 58, 66, 75]. Solutions have firstly focused on developing languages and descriptive models (DAML, OWL etc) to represent services, device resources, user profiles and queries. Architectures proposed for service discovery and composition have focused primarily on wired web-based environments and hence solutions are mostly centralized or semi-centralized and registry-based. Centralized or semi-centralized solutions are unsuitable for infrastructure-less ad-hoc environments.

There is very little work in service discovery and composition in the domain of infrastructure-less pervasive environments. Current solutions for service discovery and composition in ad-hoc environments come from the domain of peer-to-peer technologies. However, such solutions are primarily broadcast-based and are inefficient in the utilization of networking and computing resources. Solutions primarily focus on discovering services and composing them in a constrained network and do not address the issues of scalability and network-wide reachability. Moreover, issues of mobility and adaptability of protocols with respect to varying environments is also not addressed. This dissertation addresses the issues of service discovery and composition in a general purpose infrastructure-less pervasive environments. Secondly, it develops an extremely flexible, adaptable and scalable distributed architecture and associated protocols for discovery and composition in such environments. Thirdly, it utilizes semantic web technologies as needed in various aspects of developing the architecture.

We focus our work on infrastructure-less pervasive environments since we believe that infrastructure-based pervasive environments are just a special case of the infrastructure-less environment where some devices are fixed, resource-rich and have infrastructure support. Moreover, solutions for this environment can be as easily adapted to infrastructure-based mobile environments. For the remaining of the dissertation, we shall use the term *mobile environments (ME)* to mean *infrastructure less pervasive environments*.

I.B Problem Domain and Description

Centralized or semi-centralized and registry-based solutions for service discovery are unsuitable as a scalable and efficient solution for service discovery in mobile environments [22, 26, 23]. Fixed broker-based or fixed central-entity based service composition techniques conventionally applied to infrastructure-based environments cannot be applied as a solution to compose services on the fly especially when the services themselves are mobile and are interconnected through an ad-hoc network. This is due to the fact that such approaches presuppose the existence of a persistent central entity or coordinator in the client's neighborhood, which is reliable and constantly available on request. The questions addressed in this dissertation are:

- How can we develop a distributed architecture to enable service discovery, data routing and composition in an integrated manner for pervasive environments?
- How can we make the architecture scalable to enable efficient usage of the underlying network bandwidth while facilitating network-wide reachability?
- Can we assess the feasibility as well as sustainability of these approaches in pervasive environments?

I.C Research Contributions

The dissertation makes the following research contributions to the field of mobile and pervasive computing.

- The first contribution is the architecture to enable integrated service discovery, service-centric routing and service composition in infrastructure-less pervasive environments (also referred to as mobile environments in this dissertation). The design of the architectures and the protocols therein offers several cross-layer optimizations that improve the overall performance of the architecture.
- Design, implementation and evaluation of a novel group-based service discovery protocol (dubbed GSD) that is registry-less, decentralized and based on the concepts of peer-to-peer caching of service advertisements and group-based selective forwarding of service discovery requests. The protocol also displays the coupling of the traditionally de-coupled fields of *service matching* with the *discovery architecture*. The protocol also addresses the issues of scalability and network-wide reachability of discovery protocols in ad-hoc networks.

- Design, implementation and evaluation of an unique service-centric group-based routing and session management protocol (dubbed GSR) for ad-hoc networks as an integral part of the service discovery infrastructure. Traditional approaches place routing at a layer below service discovery. While this distinction is appropriate for wired networked services, we found that in ad hoc networks this layering is not as meaningful and show that integrating routing with discovery infrastructure increases system efficiency.

- Design, implementation and evaluation of two novel distributed reactive service composition protocols that are immune to central point of failure and are adaptable, efficient and take into consideration mobility, dynamic changing service topology and device resources. The composition protocols are based on distributed brokerage mechanisms and utilize the GSD and GSR protocols for service discovery and invocation.

Additionally, the dissertation also presents a bottom-up formalism to model the concept of proactive composition in mobile environments. We present an M/G/c/c queuing theoretic model to formalize a service cache: the basic unit that determines the efficiency and efficacy of composition in mobile environments. We also design and implement an ontology grounded in Web Ontology Language (OWL), to describe services in mobile environments in terms of its inputs and outputs, platform constraints and device capabilities. This information is often used in various aspects of our above-mentioned distributed discovery and composition in mobile environments.

I.D Thesis Outline

This dissertation proposes a novel architecture for service discovery and composition in pervasive environments. Additionally, it proposes and implements new and innovative distributed protocols to enable efficient, distributed and highly scalable service discovery, service-centric routing and service composition for mobile environments. The following is a chapter by chapter overview of the dissertation.

Chapter I provides an overview of the dissertation and presents the problem being addressed in the dissertation. It also provides a brief outline of the research area and presents a summary of the various chapters in the dissertation.

Chapter II describes the work being done in the field of service discovery and composition for wired as well as infrastructure-less pervasive environments. It brings out the drawbacks and limitations of currently

existing solutions and also emphasizes areas that this dissertation has leveraged from. It also provides an overview and contrasts related areas that are analogically similar to the problem being addressed in this dissertation.

Chapter III unveils the integrated architecture that enables service discovery and composition for mobile environments and encompasses various distributed protocols developed as a part of this dissertation. It provides an overview of the broad functionalities of the various layers and briefly summarizes the protocols developed.

Chapter IV describes the peer-to-peer caching based and selective forwarding of request-based distributed service discovery protocol dubbed GSD. GSD utilizes underlying ad-hoc routing protocols and is designed to minimize the network bandwidth and achieve network-wide reachability. This chapter describes the protocol and provides experimental results to validate the claims of the protocol.

Chapter V describes our work in developing a routing protocol for service data. Service discovery is followed by service invocation. Service invocation requires data to be routed from the source to the destination service. We found that integrating service discovery with routing protocols rather than using a standalone routing protocol underneath the discovery layer increases system efficiency. This chapter describes our Group-based Service Routing (GSR) protocol that is service-centric (apart from being node-centric) and has end-to-end session maintenance. We also demonstrate our results of comparing our protocol with AODV (Ad-hoc On-demand Distance Vector protocol [85].

Chapter VI demonstrates the design of our reactive distributed service composition protocols. Our protocols are based on the principle of distributing the load of discovering, integrating and executing services for a composite request to the most appropriate node(s) in the environment. We carry out simulation experiments to measure various parameters such as composition efficiency, broker arbitration efficiency, discovery efficiency etc.

Chapter VII describes the Anamika framework: our proof-of-concept implementation of parts of our integrated architecture. Our implementation utilizes Bluetooth and ThinkPads to form an ad-hoc network and demonstrates discovery and composition of services in a piconet. We also share implementation experiences in this chapter.

Chapter VIII presents a queuing theoretic model for a service cache in mobile nodes in our attempt to formalize the concept of proactive composition in mobile environments. Proactive composition refers to precomputation of composite services even before the user requests such a service. We model the environment

in a bottom-up manner and present our M/G/c/c based model for formalizing a service cache: the most elementary unit that determines the efficiency and efficacy of composition in mobile environments.

Chapter IX summarizes the contributions of the dissertation and enumerates some future directions of research in this domain. It also concludes the dissertation.

I.E Topics excluded from the Thesis

This dissertation primarily focuses on developing a distributed architecture to enable efficient service discovery and composition in mobile environments. In doing so, it limits itself to developing distributed protocols that are network-efficient and primarily focus on the theme of being able to utilize resources/services and the environment around a device. Understandably, there are several other interesting issues that it leaves open.

The dissertation leaves open concerns of security and privacy of mobile users and services in this environment. Other colleagues in our group are building distributed trust and belief based systems for security and privacy in pervasive environments [4, 69, 107]. This dissertation however, complements the work by opening up several other related ways of accessing services in peer devices.

Along with the option of discovering and accessing of services comes the question of how to pay for these services. There has been some research in micropayments and minipayments [48] in the domain of eCommerce that forms an interesting starting point for payment-related research in the space of mobile services. This dissertation does not concern itself with payments.

Service composition is also defined in the related field of planning as the generic problem of generating a complex task by conglomerating smaller sub-tasks and vice versa. Splitting a task into generic or optimal sub-tasks is very complex and goes into the domain of planning [43] in AI. This branch of research also includes developing complex planning engines [43, 79] that utilize service descriptions to generate declarative specification of workflows that compose different services. The composition protocols in this dissertation assumes the pre-existence of such a workflow specification of a composite service and addresses the problem of actually discovering, integrating and executing atomic services in a distributed mobile environment.

Chapter II

BACKGROUND AND RELATED WORK

In this chapter, we review the work done in service discovery, ad-hoc routing and service composition in the general domain of wired as well as wireless environments. We also review and critically examine the work in the domain of infrastructure-less environments. We present an overview of ad-hoc routing protocols and contrast them with respect to their effectiveness as a standalone layer underneath a discovery infrastructure. We also present other related areas where similar concepts has been modeled and researched.

II.A Roots of Service Discovery and Composition

The original concept behind service discovery and composition is not new to the field of computer science. In software engineering, we have seen effective use of previously written software to write new applications. This is called software reuse [98]. There has been work in upgrading these software components dynamically at run-time [46] to enhance the functionality of the whole system. Here, the software code was usually modified to suit the functionality of the new application. Following the trend, we saw object-oriented and component-based application [2, 54, 11] development. The modularity and componentization of software was considered primarily important here. The software design followed this pattern to ensure minimal interdependence between software components. The various subcomponents were coupled together to ensure integrity and proper functioning of the whole software. In other words, the various subsystems carried out their own specific functions to make a whole system work. The functions of the subsystems were clearly defined and there was a static way in which information flow used to take place in these systems. Service discovery and composition in pervasive environments is one way of re-using the components that are avail-

able in vicinity of a device. However, in pervasive environments, the rigid structural framework of software re-use fails when the multiple components are loosely defined and is in a state of continuous change.

II.B Service Discovery and Composition in Infrastructure-based Environments

Service discovery and composition is an important and active area of research [19, 42, 105, 66, 68, 89, 21] and has been studied widely in the context of web services. Research in the field of service discovery has forked along two branches, namely *service description and matching* and *service discovery architectures*.

Service description languages like Web Services Development Language (WSDL) [111], Web Services Flow Language (WSFL) [112] and DARPA Agent Markup Language for services(DAML-S) [40] have been developed to describe web services in a flexible manner. The Web Services Description Language (WSDL) by W3C [111] is an XML format for describing network services as a set of endpoints operating on document-oriented or procedure-oriented messages. The DAML project by DARPA and the W3C focuses on standardizing OWL as the language to use to describe information available on any data source, in order that the information may be understood and used by any class of computers, without human intervention. We have used a OWL-based ontology to describe our services and our logic behind using OWL is explained in chapter III.

Service Discovery Architectures like Jini [3], Salutation and Salutation-lite [96], UPnP [62], Service Location Protocol [53], have been developed over the past few years to efficiently discover wired infrastructure based services from wired as well as wireless platforms. However, most of these service discovery infrastructures have a central lookup server type architecture for service registration and discovery. Central lookup server/registry-based mechanism for doing service discovery is inappropriate in ad-hoc/pervasive environments due to the dependence of the whole infrastructure on a central point/node, which might as well be mobile and unreliable.

Research in service composition has predominantly followed two directions. One direction defines languages to formally specify services and composite services [40, 112, 50] in terms of service input/output, service pre-condition and post-conditions, fault handling, joint exception handling and service invocation mechanisms. This research also includes developing complex planning engines [43, 79] that utilize these service descriptions to generate declarative specification of workflows that compose different services. The

other direction of research aims to develop architectures [105, 66, 68, 89, 21] that enable service composition. These architectures assume a declarative specification of a composite service[1] and performs the task of discovering, integrating and executing the actual services. We focus on service composition architectures since we believe that it is critically important in a mobile environment and requires a different approach from currently available wired-infrastructure based composition architectures.

Current service composition architectures [20, 66, 75] have been designed with the inherent assumption that the services are resident in the wired infrastructure. Thus, services are assumed to be on stable nodes/devices and connected to each other over high bandwidth reliable communication channels. Composition architectures are centralized and consist of a preconfigured *composition manager* or *broker* residing on high-end machines. The *composition manager* performs *service coordination and management* that involves managing issues like service path creation [104], delegation of service discovery to the proper discovery manager, appropriate combination of different services and management of the information flow between service components.

Such architectures do not work well in mobile environments, especially ad-hoc mobile environments. Some of the notable limitations are: (1) Central Point of Failure: Centralized design of service composition architectures for wired-infrastructure based environments is prone to single point of failure in a mobile environment. (2) Mobility: Mobility changes the topology of a mobile environment. This changes the *service topology* (distribution of services on various mobile nodes). Current composition architectures lack support for mobility. (3) Fault Management: Mobile nodes are prone to various kinds of faults. Faults range from network level disconnection, service discovery failures, and service execution failures to node failures. Composition architectures for mobile environments need to be adaptive and resilient to these failures.

There has been work on content-centric networking and content-based message routing architectures [109, 18] that use publish-subscribe based architectures to route data based on its content. However, such architectures do not perform well in a distributed ad-hoc environment due to their centralized/semi-centralized architecture.

Recently, some work has been done by Garcia-Molina et. al [34] by addressing resource discovery using routing indices. They use routing indices to measure the "goodness" of neighbors to be able to answer a query or provide a resource. However, the solution places index values on different paths in the peer-to-peer system and hence requires huge amount of updating in the event that the paths change dynamically (as they

[1] we shall use the term *composite service* and *composite request* interchangeably throughout the dissertation. This is because composite request in this work is nothing but a declarative specifi cation of a composite service.

do in pervasive environments). Our group-based service discovery protocol described in chapter IV does not place any weightage on paths; rather it adapts itself depending on the movement of the devices in the vicinity.

Advertisements in pervasive environments is coming up as a new area and our protocol can easily benefit by using intelligent schemes for adaptive advertising of services. For example, Anand et al. [90] talks about serendipitous advertising and Ratsimor et al. [91] talks about policy-based advertising that can easily perform better than periodic advertising of services. Our architecture is extensible and can easily be enhanced to accommodate these protocols.

II.C Service Discovery and Composition in Infrastructure-less Environments

Research in the area of service discovery for ad-hoc networks is relatively new. Solutions [55, 103] primarily utilize the broadcast-driven nature of the underlying ad-hoc network to carry out service discovery. We have shown in chapter IV, that broadcast-driven protocols do not work well in terms of scalability and efficiency of discovery for large-scale pervasive environments. There has been work in the field of wired networks to develop serverless peer-to-peer architectures like [92, 97, 73]. However, some key limitations of such approaches with respect to pervasive environments are: (1) Traditional P2P networks derive basic boot-strap support from some *trusted hosts* that are robust and available. We cannot assume such support in an ad-hoc environment (2) Underlying protocols to discover resources are essentially broadcast-driven thus potentially generating significant network load (3) The virtual network topology of these P2P networks do not use the underlying physical Internet topology effectively, thus affecting their scalability and efficiency. Service discovery architectures in pervasive environments not only has to utilize the underlying dynamically changing topology, but also has to be independent of any boot-strap servers.

The Bluetooth Service Discovery protocol [99] is a peer-to-peer service discovery protocol that can be used over ad-hoc environments. However, apart from the fact that it supports very rudimentary unique-identifier based matching, the discovery is also driven by broadcast in a piconet. Our distributed protocol for service discovery is targeted towards generalized ad-hoc networks that are better representative of pervasive environments. Avancha et al. [4] enhances the Bluetooth service discovery protocol to include service description-based reasoning using Prolog. However, it only enhances the service matching part of Bluetooth and does not address discovery architecture.

There has been very limited work in the development of distributed protocols for service composition targeted towards mobile environments (ME). In prior work [87], we developed a middleware-based architecture to handle composite requests from mobile devices with the help of infrastructure-based services. Our middleware platform sits on the edge of the wired network and serves as a proxy to integrate various web services and present them to mobile users. Limitation of such a system is that it depends only on the wired infrastructure for services. Moreover, middleware-oriented protocols are essentially semi-centralized and hence unsuitable for ME.

Basu et al. [10] have described a hierarchical task-graph based approach to enable service composition in ad-hoc networks. In this work, a composite service is represented as a task-graph with leaf nodes representing atomic services. Different sub trees of the graph are computed in a distributed manner in a MANET (Mobile Ad-hoc Network). Service composition is coordinated by the source of the request. Their approach selects the source of the request as the composition manager for itself. Our architecture considers the source of the request and the composition manager as logically separate entities. We believe that the proper selection of the composition manager for a request increases the efficiency of composition and decreases the network load. Moreover, their approach employs network intensive global broadcasting techniques whereas, our architecture is based on the strategy of localized search thus reducing the protocol overhead.

II.D Survey of Discovery and Composition Protocols

We present a survey of important service discovery and composition protocols in this section.

Service Location Protocol (SLP): The Service Location Protocol (SLP) [53] is a language-independent protocol for automatic resource discovery on IP networks. The base of the SLP discovery mechanism lies on predefined service attributes, which can be applied to universally describe both software and hardware services. The architecture consists of three types of agents: User Agent, Service Agent and Discovery Agent. The User Agents are responsible for the discovery of available Directory Agents, and acquiring service handles on behalf of end-user applications that request for services. The Service Agents are responsible for advertising the service handles to Directory Agents. Lastly, the Directory Agents are responsible for collecting service handles and maintaining the directory of advertised services. The Service Location Protocol also works without the presence of the Directory Agents. In the later case, the User Agent and the Service Agent follow a multicast message-passing mechanism to exchange service related information.

Jini: Jini [62]is a distributed service discovery architecture developed by Sun Microsystems, whose services can be realized to represent hardware devices, software programs or their combination utilizing the Java language. Its overall goal is to transform the traditional network into a flexible, easily administered tool on which human and computational clients can find services in a robust manner. A key component of Jini is the Jini Lookup Service (JLS), which maintains the dynamic information about the available services in the Jini federation. Each service is responsible for discovering one or more JLS by using either a known location or by using a Jini multicast discovery mechanism. After discovering a JLS, the service also is responsible for uploading its service proxy and to periodically refresh its registration at the JLS. Similarly, each client is responsible for discovering a JLS before it can download the matching service and invoke various methods by contacting the original service. However, in the Jini architecture the client is solely responsible for knowing the precise name of the Java class representing the service.

Universal Plug and Play (UPnP): UPnP [62] extends the original Microsoft Plug and Play peripheral model to support service discovery provided by network devices from numerous vendors. UPnP works and defines standards primarily at the lower-layer network protocol suites. Thus devices can natively, i.e. language and platform indecently, implement these standards. UPnP uses the Simple Service Discovery Protocol (SSDP) for discovery of services over IP networks, which can operate with or without a lookup service in the network. In addition, the SSDP operates on the top of the existing open standard protocols utilizing HTTP over both unicast (HTTPU) and multicast UDP (HTTPMU). When a new service wants to join the network, it transmits an announcement to indicate its presence. If a lookup service is present, it can record such advertisement to be subsequently used to satisfy client's service discovery requests. Additionally, each service on the network may also observe these advertisements. Lastly, when a client wants to discover a service, it can either contact the service directly through the URL that is stored within the service advertisement, or it can send out a multicast query message, which can be answered by either the directory service or directly by the service.

Salutation and Salutation-lite: Salutation is an open standard, communication, operating system, and platform-independent service discovery and session management protocol. The goal of Salutation is to solve the problem of service discovery and utilization among a broad set of appliances and equipments in a wide-area or mobile environment. The architecture provides applications, services and defines a standard method for describing and advertising their capabilities, as well as locating and determining other services and their capabilities. In addition, the Salutation architecture defines an entity called the Salutation Lookup Manager

(SLM) that functions as a service broker for services in the network. The SLM can classify the services based on their meaningful functionalities, called Functional Units (FU), and the SLM can be discovered by both a unicast and a broadcast method. The services are discovered by the SLM based on a comparison of the required service types with the service types stored in the SLM directory. Finally, the service discovery process can be performed across multiple Salutation managers, where one SLM is a client to another SLM.

The Salutation-lite standard prototypes the required architectural changes required by the Salutation technology to support multiple operating systems and small form-factor devices. It provides a means to determine the operating system, processor type, device class, amount of free memory, display capabilities and other characteristics of a hand-held device. The message exchange protocols are tailored to reduce the quantity of data exchanged. By limiting the functions of the Service Discovery, the footprint of the implementation has been reduced significantly.

Ninja Secure Service Discovery System: The Ninja Secure Service Discovery System (SDS) [58] is a research level service discovery engine developed at University of California, Berkeley. The architecture consists of clients, the services and the SDS servers. The SDS server architecture is a scalable, fault-tolerant, secure and available service discovery repository. The architecture offers support for local area as well as wide area services. The system has client-repository-server type architecture. However, SDS servers are hierarchically arranged for scalability and availability across wide area networks. The architecture allows push or pull-based access to the servers. Service descriptions and queries are specified using XML, leveraging the flexibility of semantic-rich content. The SDS uses encryption to ensure privacy of the transactions and uses capability-based access control on the clients to discover permissible services. Ninja services and clients uses well-known global SDS multicast channels to communicate with the service discovery servers. The architecture also has a certificate authority and capability managers to handle the access of authorized clients to services.

Bluetooth Service Discovery Protocol: The Bluetooth [99] has a specification for a very simple service discovery mechanism for peer-to-peer type networks. The architecture does not have a centralized directory for storing information about services. The information about a service is stored in the local Service Discovery Server in each mobile/static device. The services are advertised using unique 128 bit identifiers. The attributes of a service also have unique identifiers to distinguish themselves from attributes of another service. The client who wants to find out the existence of a particular service has to know the unique identifier corresponding to that service class. The Bluetooth Service Discovery server does not specify a means

to access the service. The server only replies whether a service is available on the queried platform. Kagal et. al describes a higher level service discovery and management system in the centaurus project [65]. The service discovery in Centaurus is carried based on XML description of the services and is independent of the underlying communication medium. Moreover, the architecture is also secure and clients are authenticated using a Kerberos-type [77] authentication. The architecture is based on a client-registry-service type model.

eFlow: Service Composition Engine in HP Labs: eFlow [20] is an e-commerce service composition system developed at HP Laboratories, Palo Alto. The system provides a platform for integrating various heterogeneous e-services and using the individual functionalities to compose various complex e-commerce transactions. These services, which are a concoction of different individual 'small' services, are termed 'composite services' in eFlow. The eFlow system supports the specification, enactment and management of composite e-services. The users of the system define the specifications of composite e-services. The eFlow architecture consists of the following entities

1. The eFlow composition engine
2. Service Discovery system/Broker
3. Elementary e-services

The eFlow engine maintains state information of every running composite process instance. A composite service is modeled as a graph defining the order of execution of the different processes. The graph that models a composite service may consist of service nodes, event nodes or decision nodes. The graph also may contain 'transactional regions' that identifies portion of the process graph that should be executed in an atomic fashion. Service nodes are instances of simple or composite service. Event nodes specify the rules that control the execution flow. Event nodes enable services and the process to receive various kinds of event notifications. Event notifications are used to communicate completion of a service, suspension of a process in the event of dynamic process instance modification and other errors to the eFlow engine. The service discovery broker is primarily responsible for finding out different e-services that comply with the required specifications of different service nodes in the eFlow engine. The service discovery broker has the independence of contacting other vendor specific service brokers to discover different e-services. The main function of the engine is to maintain state information of every running process instance, coordinate the different events with the corresponding service nodes and schedule the corresponding service for execution.

Ninja Service Composition Architecture: Researchers at UC, Berkeley have developed a platform to

enable dynamic service composition in the Ninja Project [104, 66]. The main objective of the platform is to enable diverse clients with varying resources and network accessibility to access the same unchanged network service. Changing a network service to suit the needs of various clients with respect to bandwidth, resource capability is not a very good design choice. It duplicates functionality and sometimes makes the service more complex. This is because the service now has to perform other actions like transcoding a reply to suit a particular browser type and similar other jobs. Their service composition platforms address this problem by allowing the service to remain as it is and dynamically plugging in required services to cater to the needs of the various clients. Their research goal is to be able to compose arbitrarily complex services from simpler services over a wide area network. However, the research still depends on high-end powerful infrastructure-base support for composition.

Descriptive Language Frameworks: There is a plethora of work in trying to define languages to represent services, invocation mechanisms and composite services. This class of work in service composition, nonetheless important is not an issue of this dissertation. For the purpose of this dissertation, we assume that the composite service can be represented in any of these languages. Examples of such work include DAML-S[40], OWL [41], WSFL[112], XLANG[49] and BPEL[50]. In particular, we have used DAML-S to represent composite services. This is primarily because DAML-S derives semantically rich reasoning capabilities with the service description from DARPA Agent Markup Language (DAML) [70]. It also offers a rich set of declarative constructs to appropriately represent composite services.

II.E Ad-hoc Routing Protocols

An ad-hoc network is a temporary network, operating without the aid of any established infrastructure or centralized administration. Such a network can be envisioned as a collection of routers, equipped with wireless transceivers, which are free to move about arbitrarily. The basic assumption in an ad-hoc network is that, two nodes willing to communicate may be outside the wireless transmission range of each other but may be able to communicate in multiple hops, if other nodes in the network are willing to forward packets for them. The mobile nodes would be arbitrarily located and would be moving in a dynamic manner. Typical applications of ad-hoc networks are outdoor special events such as conferences, communications in regions with no wired infrastructure support, in emergencies and natural disasters and in military operations.

Routing of information is very important to maintain seamless connectivity between different mobile

devices. This is because all the mobile devices might not be in the radio contact of each other, but they might be reachable by using the radio links of intermediate nodes. The primary goal of routing protocols in an ad-hoc environment is to ensure correctness and efficiency in route establishment. The route construction should also utilize minimum possible resources and bandwidth. There is a direct correlation of the ability of the different ad-hoc routing protocols in determining the extent to which service composition might be possible in an ad-hoc environment. This is due to the fact that service composition will require nodes, which are multiple hops away to communicate with each other in an efficient manner. Existing ad-hoc routing protocols deals with typical limitations of these type of networks, such as high power consumption, low bandwidth and high error rates. Routing protocols [94] in this environment can be categorized as:

- Table-driven
- Source-initiated (demand-driven)

II.E.1 Table-Driven Ad-hoc Routing Protocols

Table-driven protocols as the name suggests, attempt to maintain consistent, up-to-date routing information from each node to every other node in the *reachable* vicinity. They respond to network topology changes by detecting it and propagating the new information to other nodes.

Destination-Sequenced Distance Vector Routing (DSDV): DSDV [84] is a table-driven algorithm based on the Bellman-ford routing mechanism [64]. In this routing mechanism, every node in the network maintains a routing table in which possible destinations in the network and the number of hops to each destination is recorded. Each entry is marked with a sequence number assigned by the destination node. The algorithm is free from routing loops. Routing table updates are periodically transmitted throughout the network in order to maintain consistency. Bandwidth is conserved by sending partial routing information called *partial dump* most of the times and *full dump* occasionally. Duplicate routes with the same sequence number are rejected and only the shortest of the lot is accepted. Delaying route updates for the length of the 'settling time' optimizes network traffic.

Clusterhead Gateway Switch Routing (CGSR): CSGR[31] is a variant of DSDV where the network is hierarchically organized. The nodes are grouped into clusters and a cluster head controls groups of ad-hoc nodes. A distributed cluster-head selection algorithm is used to elect a cluster head. However, frequent cluster head changes may degrade the performance of such an algorithm. Cluster heads control the routing of traffic to different clusters. Clusters communicate with each other using 'gateway nodes'. Gateway nodes are those

nodes that are in the range of two or more cluster heads. Each node maintains a cluster member table and a routing table. Routing takes place through a coordination of the cluster member table and the routing table. The Wireless Routing Protocol (WRP) [76] is another example of a table-based ad-hoc routing protocol.

II.E.2 Source-Initiated Ad-hoc Routing Protocols

The source-initiated on-demand routing creates a route only when the source node requires it. When a node requires a route to a destination, it initiates a route discovery process within the network. If a route can be discovered, it is maintained using a route maintenance procedure unless every path from the source is no longer available.

Ad-hoc On-Demand Distance Vector Routing (AODV): AODV [85] is an on-demand type of routing protocol which is a variant of the DSDV protocol. It minimizes the number of required broadcasts by creating routes on an on-demand basis. A source node that tries to send data to a node for which a route does not exist initiates a ”path discovery” process to locate the other node. The request is broadcasted until it reaches a node that has got fresh information about the route to the destination node. This intermediate node then replies to the source node and all the intermediate nodes also cache that route entry. If an intermediate node moves away, a ‘link failure notification’ is transmitted upstream to the source node. The source node might choose to initiate a route discovery again.

Dynamic Source Routing (DSR): DSR [63] is an on-demand routing protocol that is based on the concept of source routing. The protocol consists of route learning and maintenance policy accompanied by a route discovery process. When a mobile node wants to send a data packet, it first checks whether it has an ‘unexpired’ route to the destination. It initiates a ‘route discovery’ if it does not have one. A ‘route reply’ is generated when the request reaches the destination or an intermediate node that knows the route to the destination. The reply is then routed back through the same path through which it had come. Route mantainance is accomplished through the use of route error packets and acknowledgements. Temporally Ordered Routing Algorithm (TORA) [80], Associativity-Based Routing (ABR) [106], Signal Stability Routing (SBR) [44] are other examples of similar on-demand ad-hoc routing protocols. Adaptability and self-configuration are important parameters of any ad-hoc routing protocol. The efficiency and performance of ad-hoc routing protocols will affect the efficiency of service execution and composition at the higher layers.

Bluetooth: As far as the development of commercial ad-hoc networking systems are concerned, Bluetooth [14] is the most important product in the market. Bluetooth is a short-range radio frequency based

networking technology, which facilitates peer-to-peer networking in an ad-hoc fashion among various different kinds of heterogeneous Bluetooth-enabled mobile devices. Bluetooth uses frequency-hopping technique to transmit data and has a radio range of 10 to 100 meters. A number of Bluetooth-enabled mobile devices can form a piconet. Multiple piconets can join together to form a scatternet. Bluetooth also has a simple service discovery protocol as part of its specification. Interesting issues come up when a Bluetooth piconet/scatternet is visualized as an ad-hoc environment. Current specification states that a piconet can contain a maximum of 8 nodes with one node being the controller or ”master”. A scatternet can be formed by a maximum of 10 piconets. The more the number of piconets in the vicinity, the more is the chance of frequency interference leading to data loss. Routing in Bluetooth scatternets has not reached maturity and is in a stage of research and development.

II.E.3 Integration of Routing with Discovery

Protocols like DSR [63], AODV [85], TORA [80], DSDV [84] are only a few of the routing protocols that have come up over the last 10 years. However, all these protocols are node-centric and the source has to know the destination address before ensuing an interaction. Standard node-centric routing protocols cannot be used for service discovery. This is primarily due to couple of reasons: (1) Discovery protocols do not assume that they know the node addresses of the devices in the vicinity to be able to use standard routing protocols to reach the devices and check whether the required service exists on those devices (2) Unique Address assumption: Routing protocols assume device addresses are unique. However, services are not unique and there could be multiple instances of the same service in a network. However, routing protocols have been traditionally used to route service invocation data once the service has been discovered.

The idea of integrating routing with service discovery has been discussed earlier in [6, 109]. There has been work [6] in looking at the issues and benefits involved in cross-layer optimizations in Bluetooth scatternets. The principal idea is to integrate the link, routing and service discovery layer so that efficient handling of power is possible. This body of work calls for a unified network stack instead of the traditional protocol design. Our work also follows along these lines except that our integrated protocol is general for any ad hoc network. It also develops on the concept of a “bottom-up” integration where the routing layer is integrated into the discovery infrastructure above it. We also show that careful design of the service discovery protocol can provide better routing support.

In [109], Balakrishnan et. al, present an integrated message routing and service discovery architecture.

However, this solution assumes an underlying wired network infrastructure support and solutions like DNS for bootstrap. Moreover, message routing is done by piggybacking service data along with discovery request. Thus, the discovery is in some sense tied to the routing layer. GSR does not rely on any wired network infrastructure and also supports use of the discovery layer as a separate protocol along with other traditional routing protocols if needed.

Content-based Networking [18] mentions about message routing based on message content rather than node-address. They employ a publish-subscribe-based middleware solution for data routing and in some sense is geared towards wired networks. In essence, this is similar to service-driven routing. However, publish-subscribe or middleware solutions do not perform well in a distributed ad-hoc environment due to their centralized/semi-centralized architecture.

Ad-hoc On-Demand Distance Vector Protocol (AODV) [85, 86] has defined an extension that is based on the idea of enhancing the routing protocol by adding a service discovery extension to it. It uses the AODV_RREQ message as a service discovery request when its 'S' flag is set. The IP address field contains the service address and port number. Even though this is one way of performing service discovery and enables service-based routing, the key limitations/differences from GSR are: (1) It assumes only one service per node whereas GSR does not have any such assumption. (2) The discovery protocol is dependent on the broadcast-based architecture of AODV. Our group-based service routing protocol described in chapter V on the other hand, uses selective forwarding based on the service group information and is more efficient than simple broadcasting used by AODV. This reduces network overhead in the integrated protocol.

II.F Related Research Areas

Even though there has been very little work done in the field of service discovery and composition is particular, the problem is not unique in itself. We can find resemblance of the concept in various other fields in computer science. In the Agent World, there has been research on agent cooperation and heterogeneous agent coordination and teamwork theories. Pynadath, Tambe and Chauvat [88] address the problem of heterogeneous agent coordination in team-oriented programs. They describe a framework that enables developers to construct large-scale agent organizations. The agents in these organizations have individual roles to play as part of a team. Their system addresses the problem of locating different agents by constructing different agent lookup mechanisms like matchmakers, local white-pages directories and other registry services. However,

the system is targeted to solve problems that have been pre-defined and individual goals are fed as inputs to the agents taking part in it. In ad-hoc service composition, there is no strict coordination and role assignment to the different services. Infact, the role or purpose of a service is only known to itself. Moreover, unlike teamwork type environments where each agent is pre-assigned certain duties during the creation of the agents, here the goal is set by a single client/agent only when he needs to satisfy that goal.

Jennings and Wooldridge [61] have described an agent-oriented way to look at software engineering. The components that make up a component-based software can be best designed using agents since it ensures least coupling and autonomy. The various agents in a software system interact with each other to fulfill both their individual and collective objectives. This view of looking at component-based software development is similar to the concept of service composition. Service composition however, is more dynamic and has to take into account the targeted environment also.

Helal et. al [17] have addressed the problem associated with the collaboration of mobile devices in a wireless or ad-hoc environment. Collaboration in mobile environments have to address issues like bandwidth, latency, processing power, battery usage, restricted GUI and limited input methods. Any system developed in this environment has to be fault-tolerant and de-centralized. The system focuses on collaboration in a one-to-one manner and does not handle one-to-many collaboration.

Agent Coordination also has been explored from the theoretical point of view. Barfoot et. al in [9] model multi-agent coordination based on a stochastic version of cellular automata. Treur et. al in [15] models the knowledge, interaction and coordination of complex tasks and reasoning capabilities in agent-based systems. Their work focuses on presenting a formal model of a compositional agent system that handles simple as well as complex tasks. The complex tasks are decomposed into simpler sub-tasks by a tree structure and the distribution patterns of the tasks to the respective agents have also been statically modeled. However, in an ad-hoc environment, various other environment and platform related issues have to be considered to enable task distribution.

Service Composition may also be linked to teamwork literature in the agent world. Teamwork in the agent world has been look at in detail and their application range from business management [60] and rescue operations to psychology. Teamwork can be viewed in a hierarchical manner as well as in a distributed manner. Wooldridge and Jennings [110] have characterized team goals by terms like “joint intentions” [32]. Tambe [102] shows how the mental states can be coordinated using communication protocols. Teamwork can also be accomplished using sharedPlans [52] where subsets of the intentions are shared between agents.

The central aspect of these agent teamwork based systems is that the team has a common goal. Services can be modeled as agents in the field of service composition. But teamwork related techniques might not directly apply in this field since the agents over here do not share a common goal. The agents residing on devices in an ad-hoc environment might not even know about the ultimate goal for which it is executing a subtask.

Over the years, Foundation for Intelligent Physical Agents (FIPA) has developed various standards for the agent community. The FIPA Communicative Act Library Specification [101] describes the protocol to be followed to negotiate with an agent before asking it for some help with some service (call for proposal, Accept Proposal, disagree, confirm etc). Knowledge Query Markup Language (KQML) [45, 47] was developed as a protocol to support communication between intelligent agents. KQML associates "cognitive" states of the agent with the use of the language's primitives. FIPA's agent management specification [100] describes the different functions (agent registration, life cycle management, migration management, agent delegation etc) of an agent management system residing on an agent platform. These functions can be used to model the broker management systems in service composition environments. However, the broker management has to consider other issues like task scheduling, task cost optimization, fault-tolerant techniques apart from some of the standard management functions. Agent development platforms like JADE [12], LEAP [13] and JACKAL [33] have also been developed with different design goals to facilitate deployment of agents. All these platforms are java-based and hence platform independent. LEAP however targets small form-factor devices and JACKAL specifies a communication infrastructure for agents and multi-agent systems.

Malpani, Welch and Vaidya [74] address the problem of trying to elect a unique leader in a mobile ad hoc network. The problem is well studied in the graph literature, but different factors like graph partition and merging, efficiency of the algorithm needs to be taken into consideration in an ad-hoc network. They present two leader election algorithms for mobile ad hoc networks, the functioning of which is based on TORA [80] (Temporally Ordered Routing Algorithm). Our broker arbitration techniques introduced in chapter VI should not be confused to be equivalent to leader election techniques in ad-hoc networks. This is because here we do not need to guarantee that there is a single broker in the whole network. Infact, for multiple concurrent compositions there may be multiple brokers in the same network. The broker is also selected by applying a rich set of constraints on the capabilities of the different nodes. This is not taken into consideration in leader election algorithms.

Hercock, Collis and Ndumu [56] describe an approach to dynamic distributed parallel processing using mobile agent-based infrastructure. In their problem domain, the initial request is taken from an user and is

decomposed into optimum level of task granularity. A central ‘Hive Agent’ is responsible for wrapping the tasks with mobile code and send it out to different mobile nodes for execution. In essence this is very similar to service composition. The architecture however assumes the presence of the centralized ‘Hive Agent’ that manages the distribution of the agents. Moreover, the different agents performing the subtasks are also created at the centralized site. This is an unreasonable assumption in our environment. In pervasive environments, we want to take the maximum benefit from the resources and agents already available in our environment.

E-Speak [57] is the well-known product from HP laboratories in the field of web service discovery. E-Speak is a language and operating system independent platform for hosting e-services. Each organization might interact with a Service Registry in order to publish a web service to be offered as a component in some higher-level composite Web service, or to locate the published web service. The different service repositories are interconnected so that the query is not restricted to the local server only. The e-Speak services defines a uniform services interface (API’s) and uniform services interaction (e-Speak engine) that allows e-services to dynamically interact to discover, negotiate, broker and compose themselves to solve a business to business or business to consumer service request.

The Common Object Request Broker Architecture (CORBA) has the potential to use different objects for interoperation. But the architecture is repository-oriented where all the objects have to register their server skeletons to be accessible to clients. Thompson, Pazandak et. al [105] have attempted to unify the web and this distributed object component architecture with the help of an Intermediary Architecture. The Intermediary architecture accesses the Object Request Broker and hence makes the functionalities of those objects web-accessible. The ORB model even with the induction of Meta Object Facility (MOF) [78] is unsuitable for ad-hoc environments due to its client-server type of architecture. Batista and Rodriguez [11] have described a CORBA-oriented way to dynamically reconfigure component-based applications. Their work handles reconfiguration of CORBA components running their environment called LuaSpace with the help of a dynamically typed language called Lua [59].

In [5], Awduche, Gaylord and Ganz present their work on resource discovery in distributed systems with mobile hosts. The concept is similar to discovering services in pervasive environments. They model the problem of discovering resources in a distributed system by a statistical bayesian framework. They calculate the optimal way to locate resources taking wireless network related issues like disconnection and low bandwidth into account. The concept in this work even though interesting, cannot be directly applied to ad-hoc domains. The work was done considering that the mobile hosts are connected to a wired network

using a wireless LAN. Moreover, there is an assumption that the location of the resources and the movements of the nodes are stored in centralized repositories. The theory however can be applied in modeling the search cost involved in finding services in a wireless LAN environment.

Kok and Sere addresses Service Composition from the theoretical point of view in [68]. Their work explores the use of action systems [7] for modeling distributed systems as a collection of services. The action system formalism is a state-based approach to distributed computing. Distributed systems may be seen as a complex combination of a set of simple rudimentary services. Services can be composed in a sequential manner where they may share variables and hence do information flow. The services may also be executed in a parallel manner. The paper explores how action systems can be used to model such service composition. Reasoning can also be done using refinement calculus.

Queloz and Villazon [89] introduce the concept of composing the functionalities of mobile codes in the context of active networks. Mobile code migrates to a remote machine and executes some task there. In this way, a mobile code can also help in installing custom software, which can be used by the local machine to interpret data. Communications using mobile code doesn't follow the network stack paradigm. Thus multiple mobile codes may be used to execute a common goal. Their work is based on the movement of mobile code in the fixed network infrastructure. Hence they do not consider factors like loss of mobile agents while they are being transmitted or bandwidth related considerations. The movement of mobile code may not always be a good idea in a disconnection-prone ad-hoc environment.

Chapter III

ARCHITECTURE AND PROTOCOLS

This chapter describes the integrated architecture that enables distributed service discovery and composition for pervasive environments. Layers in the architecture encapsulate various protocols implementing functionalities of each layer. This chapter details the broad functionalities of the various layers as it is related to pervasive environments. We also present a semantic classification of services and related ontologies that has been used in modeling protocols in the layers. We recall from chapter I that we shall use the term *mobile environments (ME)* to mean *infrastructure less mobile environments*.

III.A Key Issues in Discovery and Composition for Infrastructure-less Pervasive Environments

We describe issues that are of concern in the design of our architecture for ME.

- **Distributed Coordination Model:** Service discovery protocols for ME need a registry-less distributed architecture. Protocols like Jini [3], Salutation and Salutation-lite [96], UPnP [62], UDDI [108], Service Location Protocol [53] have been developed to facilitate applications to discover remote services residing on stable networked machines in the wired network. These architectures are primarily centralized/semi-centralized, registration-oriented and have an implicit assumption that the underlying network is stable and is capable of providing reliable communication. Clearly, service discovery in ME requires a decentralized design approach where a node should not be depending on some other node to advertise/register services. Each service should be autonomous and at the same time other applications should be able to discover them.

Service Composition involves the use of a coordinating entity (referred to as broker) that manages the discovery, integration and execution of a composite service. Coordination and management of a composite service in mobile environments (ME) essentially involve discovering the necessary component services, integrating the information obtained from the discovery, invoking the individual services, managing the order of execution and handling faults. Coordination and management need to be primarily adaptable to the topology changes of the mobile environment and handle mobility, network disconnections and other types of node or service failures. However, unlike infrastructure-based composition architecture, ME cannot presuppose the existence of a centralized powerful machine. A centralized composition manager might not be able to access all the necessary services and secondly, a centralized composition manager might become *unreachable* from a device in the ME. Moreover, such architecture is prone to single point of failure. Hence, composition architectures for ME need a distributed coordination model.

- **Resource Heterogeneity and Awareness:** Devices with varying resources and capabilities exist in the geographical vicinity of one another. A service discovery or composition protocol should be able to utilize these varying resources available to it. Resources refer to either services or computation power that are available and vary amongst devices. An efficient discovery and composition architecture should ideally be able to compose a service with the most efficient use of the surrounding environment and with minimum network overhead. It should also be able to utilize the spatial distribution of the services (refered to as service topology) and computation resources in the vicinity. In chapters IV and VI, we show how our protocols are able to utilize the service topology for efficient composition.

- **Mobility Management and Adaptability:** Protocols participating in service discovery and composition should be able to able to adapt themselves to the changing neighborhood. In particular, it should be able to utilize newly arrived services that were not available before. A composition protocol should be able to select a possible set of services based on their mobility patterns, platform battery life, how long they would be in the vicinity etc. Mobility changes the underlying service topology. Composition protocols for ME should be able to adapt their execution model to suit the changing service topology.

- **Fault Tolerance and Reliability:** Mobile device often have a short *switched-on* time and often are unable to process remote requests due to system overload or network level disconnections. A robust discovery and composition architecture should have protocols that are tolerant towards such failures and

degrades gracefully with increasing service/resource unavailability. Faults may range from network level disconnections, service discovery failures to node failures and service execution failures. The composition architecture should be resilient towards these failures and apply appropriate fault control mechanisms to ensure the processing of the composite request.

- **Efficient Network Utilization:** Network bandwidth is a very important parameter to be considered while designing any distributed discovery and composition architecture for ME. By default, service discovery and composition protocols are usually broadcast-based and essentially impose a relatively high network load. An efficient discovery or composition protocol should try reducing network wide broadcasts. However, reducing network wide searches may reduce the chances of a service being discovered and hence reduce composition efficiency. Our architecture follows a strategy of greedy localized searches to compose services and tries to conserve network bandwidth.

III.B System Model and Device Roles

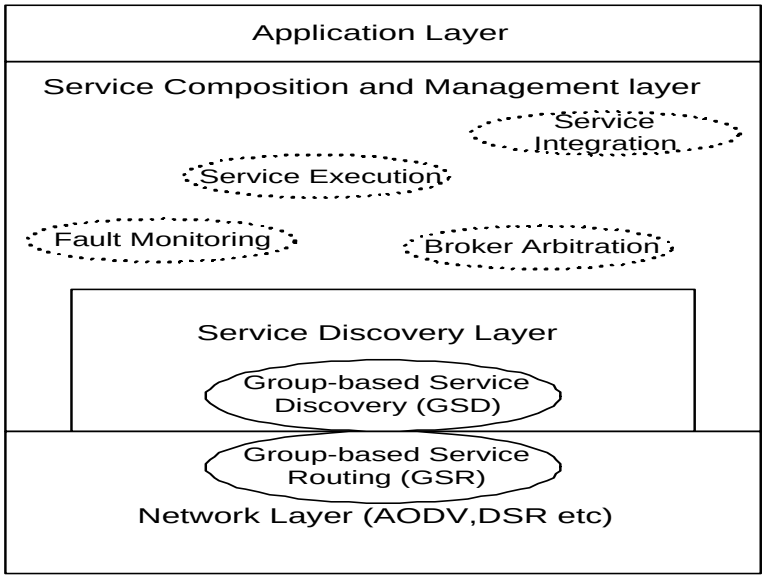

Figure III.1: System Architecture

Our system consists of mobile nodes each providing one or more services that can be invoked by peer devices. We assume that these devices are connected to each other using ad-hoc networking protocols. Examples of such network connectivity range from current day protocols like 802.11(a,b,g), Bluetooth [14],

Infra Red etc to ad-hoc routing protocols like AODV [85], TORA [80], DSDV [84], GSR [36]. Resources like computation power, battery life vary from one device to the other. We do not distinguish between a client and a service provider. Composite requests are sent out by multiple nodes. A node sending out a composite request can itself participate in another composition. Figure III.2 depicts the system environment. Each device may assume various roles as part of this architecture. The various roles each device may assume are enumerated below:

1. **Request Source (RS):** Mobile device from where a particular composite request originates. Note that a node is referred to as the *Request Source* only with respect to its request.
2. **Service Provider (SP):** Mobile device that contains services that are accessible from other peer nodes.
3. **Composition Manager (CM):** The device that manages the discovery, integration and execution of a composite request. This node can itself be the *Request Source* or a *Service Provider* for another composite request.

We also define some key terms that we use through out the remaining part of the thesis.

1. **Description-level Service Flow (DSF):** Declarative description of a composite service or a composite request. We use DAML-S for specifying a composite service. Our specification consists of a list of service descriptions along with the desired flow of execution that constitutes the composite service.
2. **Execution-level Service Flow (ESF):** A complete specification of the composite service with execution-level details required to invoke the services in the SPs.
3. **Atomic Service:** A service that resides on a single SP and can be invoked from other devices. This service may consist of further components, but the components must reside on the same SP in order to make the service *atomic*.
4. **Composite Service Length:** Number of distinct atomic services that constitute a composite service.

Figure III.1 shows an overview of the architecture and components that are resident on each mobile node in Figure III.2. We briefly describe the functions of the various layers. A protocol at a higher layer leverages the features provided by its lower layer. A notable design-wise advantage of our protocols is that although the protocols in the upper layers utilize the protocols in the lower layers (e.g. composition protocols utilizing network and service discovery layers), our protocols can also run on other standard routing and discovery protocols for ME.

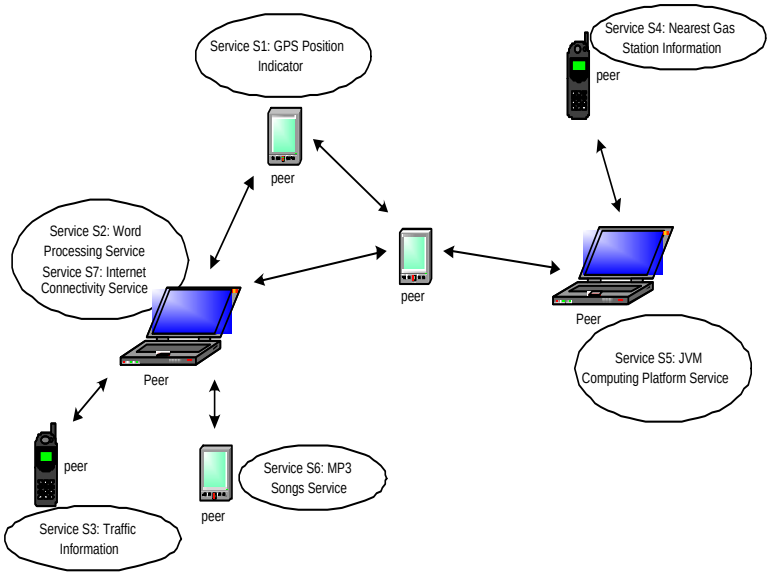

Figure III.2: Infrastructure-less Service Composition Environment

III.C Semantic Classification of Services

We have chosen OWL to define ontology to describe services/resources in a ME. There are couple of reasons for choosing an ontology-based approach to describe services. (1) The semantics of OWL can be used to describe services in different nodes and also to enable semantic matching support with those service descriptions. Any resource or service is described in terms of *classes* and *properties*. In addition, OWL provides rules for describing further constraints and relationships among resources including cardinality, domain and range restrictions as well as union, disjunction, inverse and transitivity. These axioms can be easily exploited to create an ontology describing services and service groups. (2) OWL, which is based on eXtensible Markup Language (XML) [115] and Resource Description Framework [72] is also being used as a standard to describe information/service on the wired infrastructure and the Web. This makes our description interoperable with other semantic web infrastructures.

III.C.1 Drawbacks of Existing Service Matching Techniques

We believe that existing service discovery architectures like SLP, Jini, UPnP and Salutation, although popular, have a few limitations which makes them unsuitable for wide deployment in mobile environments especially infrastructure-less pervasive environments. We briefly discuss and analyze these limitations.

- **Lack of Rich Representation:** Services in the eMarket as well as pervasive environments are heterogeneous in nature. These services should be defined in terms of their functionalities and capabilities. The functionality and capability descriptions of these services should be used by the service clients to discover them. The existing service discovery infrastructures lack expressive languages, representations and tools that are good at representing a broad range of service descriptions and are good for reasoning about the functionalities and the capabilities of the services [27].

- **Lack of Constraint Specification and Inexact Matching:** In the existing service discovery infrastructures, it is impossible to find services which require a specific attribute value that can change based on the dynamic content of the environment. In addition, service functionalities are described at the syntax level or object level. This makes it difficult to apply approximate matching rules. For example, if the client is attempting to discover a black and white printer, an approximate matching rule would succeed on finding a color printer.

- **Lack of Ontology Support:**

  Services need to interact with clients and other services across enterprises. Service descriptions and information need to be understood and agreed among various parties. In other words, well-defined common ontology must be present before any effective service discovery process can take place. Common ontology infrastructures are often either missing from or are not well represented in the existing service discovery architectures. Architectures like Service Location Protocol, Jini and Salutation do provide some sort of mechanisms to capture ontology among services. However, these mechanisms like Java class interfaces and ad-hoc data structures are difficult to be widely adapted by the industries to become standards. In the Universal Play and Plug (UPnP) architecture, service descriptions are represented in XML (eXtensible Markup Language), which provides a good base foundation for developing extensible and well-formed ontology infrastructure. However, service descriptions in UPnP do not play a role in the service discovery process.

III.C.2 Service Ontology

Our ontology describes services in terms of its capabilities, inputs, outputs, platform constraints, and device capabilities of the device on which it is residing etc. Using the class/subClassOf axiom of OWL, we incorporate a preliminary grouping of different possible services in a ME primarily based on service functionality.

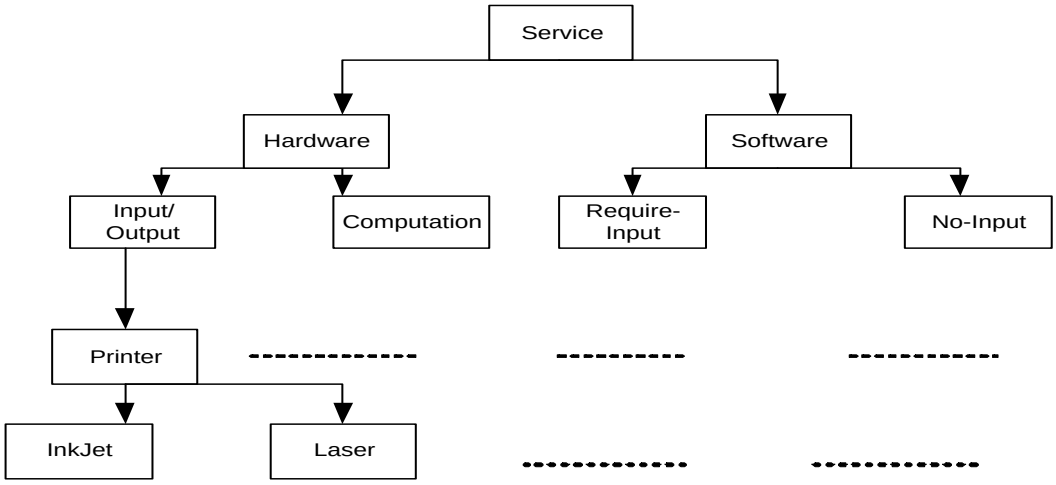

Figure III.3: Hierarchical Grouping of Services

A significant advantage of our architecture is that the ontology is extensible and one can modify it without altering the mechanisms used in the various protocols. The discovery and composition protocols would take into account the modification.

The generic class *Service* is functionally classified into two main sub-groups: Hardware and Software Service. Each sub-group is further classified in this manner till we reach a very specific service. For example, a *color printer* service may be classified under *Service/ Hardware/Input-output-type-Service/ Printer-Service*. Figure III.3 shows the functional hierarchy. Our discovery protocol GSD utilizes this information in selectively forwarding a service discovery request to the node that might contain the service requested. Service descriptions developed using this ontology are also used to carry out semantic service matching when a service request is matched with a service description. This eliminates strict syntactic matching based on unique identifiers or interfaces and provides us with the flexibility to do loosely coupled matching amongst heterogeneous services. Appendix A provides the service ontology. We have recently upgraded it to conform to the Web Ontology Language (OWL). However, at the time of the development of this architecture, the ontology was in DAML and hence the prototype implementation described in chapter VII utilizes the DAML version of the ontology.

III.D Network and Routing Layer

The Network Layer encapsulates networking protocols that provide wireless or ad-hoc connectivity to peer nodes in the vicinity. Examples of such network connectivity range from protocols like 802.11(a,b,g) in ad-hoc mode, Bluetooth [14], Infrared to ad-hoc routing protocols like AODV [85], TORA [80], DSDV [84], GSR [36]. We have developed a Group-based Service Routing Protocol (GSR) [36] for providing efficient routing support below service discovery applications. We have used GSR as our routing protocol. GSR is on-demand and utilizes the path used during service discovery for end-to-end data transmission. We have shown in [36] that integrating routing with service discovery in mobile ad-hoc environments increase system efficiency. We report the results in chapter V. Our routing protocol provides end-to-end session management that detects disconnections, performs session packet buffering, handles retransmissions and facilitates service-centric routing.

III.E Service Discovery Layer

The service discovery layer provides the system with the protocols required in order to discover services residing on nodes in ME. Our composition protocols use an efficient distributed group-based service discovery protocol (dubbed GSD), described in chapter IV to discover services.

Standard protocols for service discovery in mobile environments are either request-broadcast based or advertisement-broadcast based [55, 24, 96]. A typical request-broadcast protocol broadcasts a service discovery request globally to all nodes in the network. Conversely, an advertisement-broadcast-based protocol broadcasts advertisements of the local services in each node to all other nodes in the network. These advertisements are cached by the nodes. Apart from the fact that these protocols are inefficient in terms of bandwidth and resource usage (as the request has to be processed by all the nodes including those with limited processing capability and battery power), they impose a huge load on the network [95].

Our discovery protocol reduces global broadcasts to a large extent and efficiently uses network bandwidth for discovering services (if present). GSD is based on the concepts of *peer-to-peer caching of service advertisements in limited vicinity* and *group-based selective forwarding of discovery requests*. We use an ontology developed using DAML [24, 70] to effectively describe services. Services are classified into a hierarchical group based on their functionalities. We describe the classification and semantic description of services in section III.C. Each SP (Service Provider) advertises its services to other peer nodes in a *limited* vicinity.

```
<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml= "http://www.daml.org/2001/03/daml+oil#"
  xmlns:process="&DamlProcess;#">

<daml:class rdf:ID="ViewFileComposite">
 <rdfs:subClassOf rdf:resource="&DamlProcess;#CompositeProcess" />
 <rdfs:subClassOf>
   <daml:Restriction>
      <daml:onProperty rdf:resource="&DamlProcess;#composedOf" />
        <daml:oneOf rdf:parseType="daml:collection">
          <daml:Class>
              <daml:intersectionOf rdf:parseType="daml:collection">
                  <daml:Class rdf:about="process:Sequence" />
                  <daml:Restriction>
                    <daml:onProperty rdf:resource="&DamlProcess;#components" />
                        <daml:toClass>
                           <daml:Class>
                             <daml:listOfInstancesOf rdf:parseType="daml:collection">
                                <daml:Class rdf:about="#Filedownloader" />
                                <daml:Class rdf:about="#Printer" />
                             </daml:listOfInstancesOf>
                           </daml:Class>
                        </daml:toClass>
                  </daml:Restriction>
            </daml:intersectionOf>
        </daml:Class>
        <daml:Class>
          ......
</daml:Class>
      </daml:oneOf>
   </daml:Restriction>
 </rdfs:subClassOf>
</daml:class>
</rdf:RDF>
```

Figure III.4: Description-level Service Flow (DSF) to compose a Printer service and a File Download service

Advertisements also include a list of the several service groups that the SP has seen in its vicinity. This information is used to selectively forward discovery requests to SPs. GSD reduces network-wide broadcasts and achieves increased efficiency in discovering services.

III.F Service Composition layer

This layer encompasses protocols responsible for carrying out the process of managing the discovery and integration and execution of composite services. We introduce two distributed **reactive** protocols for service composition in ME. Reactive service composition refers to the type of composition that is executed on-demand. These two protocols are the primary contributions of this paper. Both reactive protocols are decentralized, immune to central point of failure and take maximum advantage of the mobility of the nodes, service topology and currently available resources in the vicinity. Both protocols carry out a broker arbitration that decides on the Composition Manager (CM) for a certain request. Thus, each composite request may be assigned a different CM.

The composite request consists of a Description-level Service Flow (DSF) of the composite service formulated using DAML-S. DSF of a composite service contains high-level description of the services needed for the composition and the execution flow. Figure III.4 presents a sample of a DSF to view a PDF file (by first downloading it and then printing it) in case the client does not have the capability to do so.

CM on receiving a composite request performs the process of service integration where it discovers the required services and constructs an execution-level service flow (ESF). ESF contains specific information (device address, invocation mechanism among others) used to invoke various component services. The service integration is followed by service execution where each service participating in a composition is executed. This layer manages the execution order of the multiple services and monitors faults. The broker arbitration module, the service integration and execution module and the techniques employed within it plays a vital role in making our architecture fundamentally different from static infrastructure-based service composition. We describe our composition protocols in detail in chapter VI.

III.G Application and Presentation Layer

The Application and Presentation layer embodies any software layer that utilizes our service discovery and composition architecture. This layer encompasses different GUI facilities to initiate service discovery, service invocation requests. It also displays the result of a composed service and provides the functionality to initiate a composite request.

Chapter IV

GROUP-BASED SERVICE DISCOVERY PROTOCOL

This chapter describes our distributed service discovery protocol for infrastructure-less pervasive environments. The protocol dubbed GSD (Group-based Service Discovery Protocol), is based on the concept of peer-to-peer caching of service advertisements and group-based intelligent forwarding of service requests. It does not require a service to register to a registry or lookup server. Services are described using an ontology based on the Web Ontology Language (OWL). We exploit the semantic class/subClass hierarchy of OWL to describe service groups and use this semantic information to selectively forward service requests to respective nodes. OWL-based service description achieves increased flexibility in service matching. We present simulation results of our protocol. The findings show that our protocol achieves increased efficiency in discovering services by efficiently utilizing bandwidth via controlled forwarding of service requests.

We envisage that in the near future, static, mobile, and embedded devices will provide customized information, services and computation platforms to peers in their vicinity. The primary goal of applications for pervasive computing environments is to perform the task given by the user by exploiting the resources or services that are present in the neighborhood. Some requests need a single service, which is directly available in the vicinity, whereas some other requests need multiple services or information sources to be integrated to obtain the desired result.In either case, we need a flexible service discovery infrastructure that is tailored to pervasive environments[22].

Chapter III emphasizes the need for distributed service coordination in infrastructure-less pervasive environments. There has been considerable academic and industrial research effort in service discovery in the context of wired as well as partly wired/wireless networked services. Two important aspects of service discovery are the *discovery architecture* and the *service matching mechanism*. Protocols like Jini [3], Salutation

and Salutation-lite [96], UPnP [62], UDDI [108], Service Location Protocol [53] have been developed to facilitate applications to discover remote services residing on stable networked machines in the wired network. Some of these protocols can also be used by mobile devices to discover networked services using wireless networking technologies like 802.11 (a, b or g). The general architecture followed by all these protocols is like this: A service advertises and registers itself to a service register that keeps track of the networked services. Services can de-register at any point of time. Most of the communication happens over IP-type networks, and thus relies on multicasts and broadcasts for important elements such as the discovery of the registry itself. In summary, these architectures are primarily centralized/semi-centralized, registration-oriented and have an implicit assumption that the underlying network is stable and is capable of providing reliable communication. Clearly, service discovery in pervasive computing environments requires a decentralized design approach where a node should not be depending on some other node to advertise/register services. Each service should be autonomous and able to advertise its presence. Moreover, the discovery should also adapt itself to reflect the changes in the vicinity and be able to utilize the underlying network efficiently.

Existing *service matching* techniques described above use simple matching schemas. They use interface descriptions(e.g. Jini) or attributes or even unique-identifiers (Bluetooth SDP[14]). Service matching is done at a syntactic level. However, syntactic level matching and discovery is inefficient for pervasive environments due to the autonomy of service providers and the resulting heterogeneity of their implementations and interfaces. For example, we can have the same service implement different interfaces which could result in the failure of a syntactic match if the service query does not match with any interface. To alleviate this problem, there has been considerable work to develop languages [72, 16, 41] to express service requirements and facilitate flexible semantic-level service discovery [24, 113, 37].

IV..1 Discovery Architectures in Infrastructure-less Pervasive Environments

Service *discovery architectures* [55, 5, 4] in pervasive environments are either global-broadcast based or advertisement-based. In a broadcast-based [1] solution, a service discovery request is broadcast through out the network. If a node contains the service, it responds with a service reply. The protocol, under ideal conditions of fully-connected network without message losses offers high reliability in discovering a service. However, it suffers from disadvantages. Global broadcast scales poorly with increasing network diameter and network size. Secondly, it utilizes resources and computation power on all nodes of the network including nodes that

[1] Broadcast-based protocol is also referred to as Request-broadcast based protocol in some parts of the paper

do not even have the service or nodes that may not even fall in the path to the desired service. This extra processing is essentially redundant. Thirdly, it utilizes tremendous network bandwidth (since the request traverses to all nodes through all paths possible) and hence creates a huge load on the network. The other solution is for the services to advertise themselves to all the nodes. Each node interested in discovering services cache the advertisements. The advertisements are matched with service requests and a result is returned. In this solution, the cache size increases with the number of services. Many of the nodes have limited memory and are unable to store all the advertisements. Soon the cache gets full. This is also inefficient in terms of bandwidth usage, since the whole network has to be flooded with advertisements every subsequent time interval.

IV..2 GSD Design Theme

Solutions thus far have mostly considered the *service matching* and the *discovery architecture* (as emphasized in chapter III) as two decoupled fields. GSD combines the fields by utilizing semantic service descriptions used in service matching to develop an efficient, distributed, highly scalable and adaptive service discovery architecture for pervasive computing environments. Our protocol is based on the concept of peer-to-peer caching of service descriptions, bounded advertising of services in the vicinity and efficient selective forwarding of service discovery requests using functional group information (described in chapter III) being propagated with service advertisements.

We introduced functional grouping of services based on their semantic descriptions in chapter III. This enables GSD in encompassing a broad range of discovery techniques ranging from simple broadcasting to directed unicasting, thus making it highly adaptable to the requirements of the network. The group information is used to selectively forward a service request to other devices where there are greater chances of the service being discovered. Semantic grouping of services is not uncommon in the service matching research and has been used to enable functionally similar or “near” matches [24]. We use the information to enable semantic matching and build a highly integrated yet distributed efficient discovery infrastructure.

IV.A GSD Protocol

GSD is based on the concepts of (1) Bounded advertising of services in the vicinity (2) Peer-to-Peer dynamic caching of service advertisements (3) Service group-based selective forwarding of discovery requests. Our

protocol also has multiple user-controlled parameters that determines the extent of bounds for advertising, service caching and discovery request propagation. In this section we describe these key aspects of our protocol in detail.

IV.A.1 Service Advertisements and Peer-to-Peer Caching

Each Service Provider (SP) periodically advertises a list of its services to all the nodes in its radio range. An advertisement message consists of the following fields:

<Packet-type, Source-Address, Service-Description, Service-Groups, Other-Groups, Hop-Count, Life-time, ADV_DIAMETER>

A monotonically increasing identifier called *broadcast-id* along with the *source-address* uniquely identify a broadcast and detects duplicate advertisements. Please note that this identifier is different from *source sequence numbers* maintained by nodes in traditional ad-hoc routing literature. Sequence numbers refer to a single message identifier whereas *broadcast-id* refers to a broadcast event that may generate multiple messages. The *Service-description* and *Service-groups* contain information about the local service(s) and their corresponding service groups.

Additionally, each node receiving the advertisement can forward it to all other nodes in its radio range. The field ADV_DIAMETER determines the number of hops each advertisement travels. Each node increments the Hop-Count when it forwards an advertisement that is in turn used to compute whether the advertisement can be forwarded any further. Figure IV.1 shows the pseudo code for sending advertisements.

```
Function SendAdvertisement(..) :-

1.  After each ADV_TIME_INTERVAL period do {
2.    Initialize Adv_Message;
3.    Adv_Message[Service-Description]=GetLocal_ServiceInfo(Service_Cache);
4.    Adv_Message[Service-Groups]=GetLocal_ServiceGroupInfo(Service_Cache);
5.    Adv_Message[Other-Groups]=GetVicinity_GroupInfo(Service_Cache);
6.    Adv_Message[Hop-Count]=0;
7.    Adv_Message[Lifetime]=ADV_LIFE_TIME;
8.    Adv_Message[Adv_Diameter]=ADV_DIAMETER;

9.    Transmit Advertisement to all nodes in the radio range;
10. }
```

Figure IV.1: Pseudo Code of the Process of Advertising Services in the Vicinity

Each node on receipt of an advertisement stores it in its *Service Cache*. Each entry in the Service Cache contains the following fields:

<Source-Address, local, Service-Description, Service-Groups, Other-Groups, Lifetime>

Apart from storing advertisements, a Service Cache also stores descriptions of local services in the node(identified by the *local* field in each cache entry). The field *Other-Groups* contain a list of the groups that the corresponding *Source-Address* (sender of the advertisement) has seen in its vicinity. We follow a lowest-remaining-lifetime replacement policy to replace entries when the cache is full. However, we are aware of work in predictive cache modeling [25] and profile-driven caching [82, 29] that can be used in GSD to model the cache replacement strategy. However, since cache replacement policies are not the focus of this protocol, we chose a simple uniform cache replacement strategy for all the protocols. Figure IV.2 displays the pseudo code of the peer-to-peer caching and advertisement forwarding process.

```
Function P2PCacheAndForwardAdvertisement(..) :-

1. if (Duplicate(Adv_Message))
2.   then discard Adv_Message;
3. else {
4.   Serv_Cache=Initialize_Entry_in_Service_Cache(..);
5.   Serv_Cache[Source-Address]=Adv_Message[Source-Address];
6.   Serv_Cache[local]=0;
7.   Serv_Cache[Service-Description]=Adv_Message[Service-Description];
8.   Serv_Cache[Service-Groups]=Adv_Message[Service-Groups];
9.   Serv_Cache[Other-Groups]=Adv_Message[Other-Groups];
10.  Serv_Cache[Lifetime]=Adv_Message[Lifetime];

11.  if (Adv_Message[Hop-Count]<Adv_Message[ADV_DIAMETER]) {
12.    Increment_HopCount (Adv_Message);
13.    Retransmit_Advertisement (Adv_Message);
14.   }
15. }
```

Figure IV.2: Pseudo Code for Peer-to-Peer Caching and Forwarding of Service Advertisements

The advertisement frequency, advertisement diameter and advertisement lifetime are user-controlled parameters that enables GSD to be adapted to the necessities of the device and the environment. Thus, devices in relatively static environments may choose to have a low advertisement frequency with a high advertisement diameter whereas the reverse can be applied towards highly mobile scenarios where devices have low availability. We follow the policy of *passive pushing* of advertisements rather than *active pulling* of descriptions from nodes. Passive pushing enables a device to detect changes in the environment by the receipt of a new advertisement, thus making the detection process simple, efficient and localized to the device. Active pulling

of information on the other hand has greater chances of collisions of messages at the receiving node.

IV.A.2 Advertising Service Groups

Apart from advertising its own services, GSD also uses the same advertisements to advertise functional group information of services a node has seen in its vicinity. The field *Other-Groups* in an advertisement contains an enumerated list of the service groups of all the *non-local* services seen by sender node. This information is obtained from the advertisements stored by the node in its service cache (line 5 in Figure IV.1). Figure IV.3 shows the pseudo code for the function that computes this information.

```
Function GetVicinity_GroupInfo(Service_Cache) :-

1. Other-Groups={};
2. For each Entry S in the Service_Cache do {
3.   If (S is not local){
4.    for (each group Gi belonging to S[Service-Groups] or S[Other-Groups]) {
5.         if (Gi is not in Other-Groups) then
6.                 Add Gi to Other-Groups
7.    }
8.   }
9. }
10. return Other-Groups;
```

Figure IV.3: Algorithm to Determine the Service Groups Present in the Vicinity of a Device

We observe that this service group information gets propagated from one node to another and may potentially cover the whole network (if the network is partition free). Functional group information provides a good abstraction to represent services and are enough to divert a discovery request towards the appropriate region. They also provide a good measure to aggregate the service descriptions and hence save on network bandwidth.

Figure IV.4 shows an example of propagation of service advertisements and the associated service group information for a simple ad-hoc network. We note that with an increase in the diversity of services in a pervasive environment, the different functional groups of services would also increase. Each device has a maximum limit of the number of service groups it keeps for a certain neighboring node. Currently, the limit is set to the size of the hierarchical tree. However, for memory constrained devices, our protocol allows having a lower value of the maximum number of service-groups that can be stored. In section IV.A.3, we describe the actions taken by GSD when a node does not have enough group information to forward a discovery request.

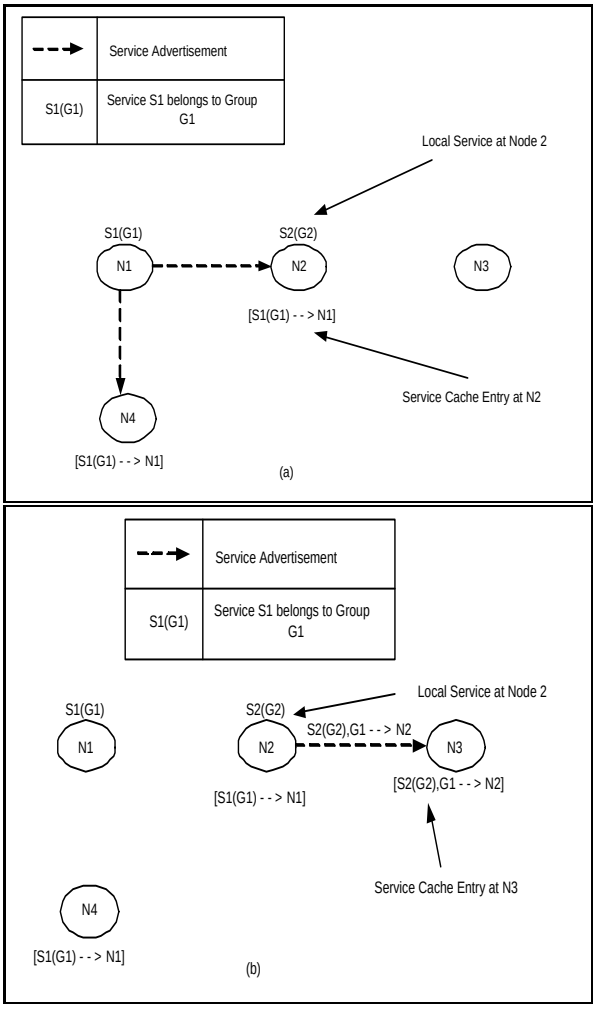

Figure IV.4: Service Advertisements and propagation of service group information. Figure IV.4a: Advertisements being sent by node N1. Figure IV.4b: Service Group information being propagated by node N2 during its advertisement phase.

IV.A.3 Request Routing

A service discovery request originates from a Request Source (RS) whose application layer requests the service. A request consists of our OWL ontology based description of the service requested that also optionally includes descriptions of service groups to which the requested service belongs. The request is matched with the services present in the local cache of the RS (that might also be a SP). A service discovery request is formed on a local cache miss. A service discovery request contains the following fields:

<Packet-type, broadcastId, Service-Description, Request-Groups, Source-Address, Last-Address, Hop-Count>

The field *Request-Groups* contains the service group(s) to which the requested service belongs. *Hop-Count*, a user-controlled parameter specifies the maximum propagation limit of the request. We use the information regarding *Other-Groups* present in the service cache of each node to selectively forward a discovery request in case of a local cache miss. We recall from the previous subsection that each entry in the service cache of a node contains a field *Other-Groups*. Thus, if the request belongs to one of those groups, then there is a chance that the requested service might be available near the node that sent the advertisement. Consequently, instead of broadcasting the request GSD selectively forwards the request to those nodes.

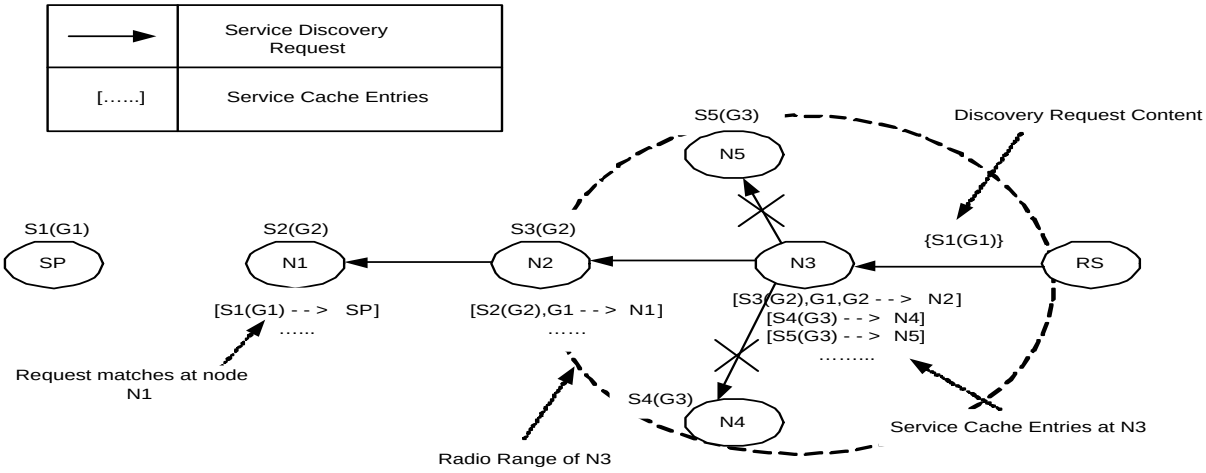

Figure IV.5: Group-based Selective Forwarding of Service Discovery Request

The selective forwarding process is explained in Figure IV.5 for a simple ad-hoc network. It shows a sequence of nodes connected to each other with RS being the requesting source and SP being the service provider where the requested service (S1) is available. For the sake of simplicity, we only display a linear connection of nodes and do not show other nodes that might be present in the vicinity. We do not show the exchange of advertisements in the figure. Assuming that each node has advertised its own services and other remote service groups, Figure IV.5 shows the partial service cache entries in each node. For example, the entry

$$S2(G2), G1 -> N1$$

means that node N1 has service S2 belonging to group G2 and it has seen a service belonging to group G1 in its vicinity. When a request belonging to group G1 comes to N3, then instead of rebroadcasting it to all

nodes in its vicinity (N4, N5), N3 selectively forwards it to node N2. This is because only N2 claims to have seen a service belonging to group G1 in its vicinity. This process continues in all other nodes until the request has reached N1 where it finds a direct match of the requested service (present in the service cache of N1).The request is by default broadcast to other nodes when the algorithm fails to determine a set of nodes to selectively forward the request to. Figure IV.6 shows the pseudo code of the selective forwarding process.

```
Function Selective_Forward(..) :-

1. if (Hop-Count of Discovery_Message >0) then {
2.  Request-Groups=Discovery_Message[Request-Groups];
3.   for (each entry S in Service_Cache} do {
4.     If any group Gi in S[Other-Groups] belongs to Request-Groups then {
5.       Node N=S[Source-Address];
6.       Decrease the Hop-Count of the packet by 1;
7.       Forward the Discovery_Message to N;
8.       }
9.   }

10.  if (the request was never forwarded) then {
11.     Decrease the Hop-Count of the packet by 1.
12.     Broadcast the request to the neighboring nodes.
13.   }
14. }
```

Figure IV.6: Algorithm showing the Selective Forwarding Process in GSD

We observe from the above algorithm, that when a node does not have enough information to selectively forward a request, it broadcasts the request to its neighboring nodes. As a practical example, a Service Request for a Printer Service could specify its Request-Group to be <NULL>, or <Input/Output>, or <Input/Output, Hardware>, or <Input/Output, Hardware, Service>. Thus depending on the amount of Request-Group information, the request would be selectively forwarded (or broadcast) to other nodes.

We observe that the selective forwarding process might also result in *false forwards*. The request might be forwarded to a region where the service is no longer available (due to mobility of nodes). This might result in the failure to discover a service that simple broadcasting of the request would have succeeded in discovering. In section IV.B we explain how our protocol can be adapted to reduce false forwards. Moreover, our experiments show that the decrease in efficiency is in fact insignificant.

IV.A.4 Reverse Routnig of Service Reply

A service reply is generated from the node that matches a service discovery request. There are couple of approaches to route the reply back to the RS. (1) One can use any standard ad-hoc routing protocol like

AODV [85], TORA [80], DSDV [84] to route the reply back to the RS. (2) The path traversed by the discovery request could be retraced back by the reply using reverse routing mechanism. Standard routing protocols try discovering a new route to the destination that involve steps like route discovery or broadcasting link-state information that generate additional network load. On the other hand, using the existing route traversed by the request could easily reduce this additional load. It has been argued in [6] and we have shown in a separate report [36] that integrating routing with service discovery increases system efficiency. Hence, we use the concept of reverse routing to route the service reply back to the RS. However, reverse routing fails if the route becomes stale or some of the nodes in the previously established path moves away. We detect such failures and resort to traditional routing using Ad-hoc On-Demand Distance Vector protocol (AODV) to route the reply from the point of failure to the RS. The node upstream in the path detects the failure to transmit the reply to the next hop. We illustrate the concept in Figure IV.7.

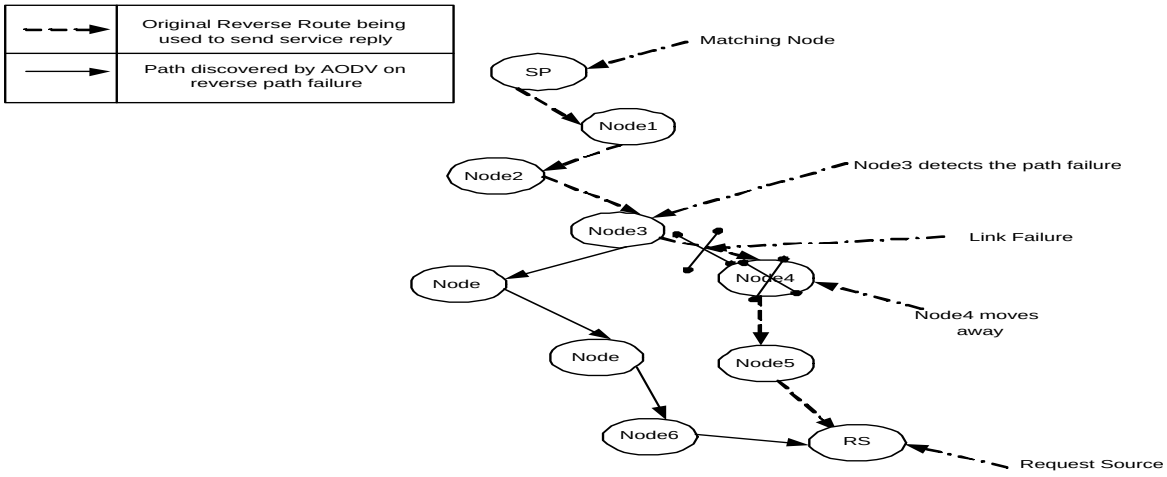

Figure IV.7: Reverse Routing of Service Reply

Each Request Packet contains a *Last-Address* field, which contains the address of the node from which a request is coming. Each node in addition to maintaining the Service Cache also maintains a Reverse-Route table. Each entry in the Reverse-Route table contains the following fields:

<Source-Address, BroadcastId, Previous-Address>

An entry is added to the table at the time of forwarding the discovery request. The entry is kept for REV_ROUTE_TIMEOUT time units. When a service reply corresponding to a request reaches this node, the table is consulted to determine *Previous-Address* in the path to the RS to forward the reply to. The

Source-Address and *BroadcastId* uniquely identifies a service reply that corresponds to a particular service request.

Determination of the shortest route in a graph could potentially require computation intensive graph traversal algorithms. However we avoid such algorithms due to the use of broadcast and secondly the non-stringent requirement to be able to find the shortest path.

IV.A.5 Service Matching

Service matching is important in enabling flexibility and richness in the discovery process. Apart from representing services using our functional hierarchical groups, our OWL ontology also provides constructs to describe services in terms of input/outputs, functional similarity, service capabilities, device/resource requirements etc. Additionally, each node in our architecture contains a *service matching* module that encapsulates functionalities for matching a service discovery request with a service description. We inherit various semantic features from OWL (class/subClassOf, unionOf etc) to match services with multiple request types. This allows the request to be specified in a flexible manner. For example, the same query can be represented using different requirements to match a certain service. More details of the service matching algorithm and the ontology can be obtained in [24].

We have augmented the service matching module to extract service group related information from a service advertisement. This is used by the protocol to store service group information separately in the service cache of each node and facilitate the selective forwarding process.

IV.B Salient Protocol Features

This section discusses some salient features and presents some theoretical evaluations of GSD that we believe would help in better understanding the benefits of our protocol. These include enabling a broad set of discovery mechanisms, adaptability to different pervasive environments, scalability and network-wide reachability, dynamic self-starting property and network load analysis.

IV.B.1 Enabling a Broad Range of Discovery Mechanisms

GSD by virtue of its hierarchical grouping of services can enable a broad set of discovery mechanisms ranging from *broadcast* to *directed unicast* mechanisms of the discovery requests. Service discovery requests contain

information regarding the group(s) to which the service belongs. Thus, at its limit, this could represent a leaf node group in the hierarchical tree (refer to Figure III.3). If the number of selective forwards at each intermediate node is one, then this results in a directed unicast of the discovery request.

However, as described in section IV.A, directed unicast in mobile environments may result in *false forwards*. The hierarchical grouping of services allows the discovery request to specify parent-groups (that are higher up in the functional hierarchy in Figure III.3). This increases the range of nodes to which the request is selectively forwarded. This is because, higher the service group is in the tree, the more is the chance of nodes having seen a similar service. At its limit, the request is in fact broadcast if the service-group specified is the root of the hierarchical tree. Broadcast-based discovery suits some constrained pervasive environments like office space or environments where most devices are at one hop distance.

Additionally, by varying the service-group information in the request, GSD also can control the chances of the protocol in discovering a *nearly-matching* service. For example, a discovery request looking for a LaserJet color printer with a service-group value of *LaserJet printer* would not be able to discover (or reach) an *InkJet printer* service. However, a service-group value of *printer* (that is the parent of the class *LaserJet printer* might be able to discover an *InkJet color printer* instead since it belongs to the same parent group of services called *printer*.

IV.B.2 Adaptability

GSD offers users to control several aspects of the protocol like advertisement diameter, maximum hop-count of discovery requests and advertisement frequency. This enables our protocol to easily adapt to the needs of users and pervasive environments. for example, an office environment can enforce a policy on the devices that the advertisements be broadcast only till 1 hop. GSD does not impose any restriction on the minimum number of entries in the service cache of devices. This makes our protocol well-suited for heterogeneous devices with varying memory constraints. GSD by virtue of its registry-less structure makes a service and a device autonomous. This is very important in pervasive computing environments since dependence on other mobile lookup servers/registries makes the protocol prone to faults due to failure of such registries/lookup servers. Services announce themselves when they come to a new environment. Services are expunged from the service caches passively if the advertisement has not been renewed for a certain time. The registry-less nature of our architecture makes it highly adaptable to changes in the vicinity due to mobility as well as device unavailability.

IV.B.3 Scalability and Network-wide Reachability

Request-broadcast based protocols can theoretically cover the whole network. Hence, under ideal conditions of non-partitioned network and no message losses, request-broadcast based protocol can guarantee the discovery of a service (if present). However, this protocol trades off network load to increase its discovery space. The network load due to discovery requests increases tremendously with increase in the network size. GSD on the other hand, can theoretically discover any service in the network with bounded broadcasts.

Consider the network (G) in Figure IV.8. Let RS= Request Source that is looking for a service S, SP= an arbitrary service provider having the service S. Let us also assume it is the only instance of S present in the network.

- **Request-broadcast protocol:**. Let D= broadcast diameter. Hence, this protocol can only cover the nodes within D hops of RS (marked by the circle with RS at its center in Figure IV.8). Let N=set of nodes that this protocol can cover. Clearly if SP does not belong to N, then this protocol would fail to discover S.

- **GSD protocol:** Let P= an arbitrary node lying on the edge of the network formed by the broadcast diameter D from RS. Then, assuming that the network does not have any partition, there will be at least one path leading from P to SP. This further means, that due to service advertisements, the group information of the service S will eventually reach the node P through the path. Thus, in GSD, if the discovery request reaches P, it will be selectively forwarded towards SP and would eventually be able to discover the service. Thus, GSD would essentially cover the whole network under identical conditions.

This makes our protocol highly scalable with respect to large-scale ad-hoc networks and high request load. It might appear that advertising increases the total load of our system. Our experiments show that even with bounded advertising, our protocol scales much better than broadcast-based service discovery. We validate through our experiments that GSD in fact performs much better in terms of network load for large networks.

IV.B.4 Dynamic Self-starting Property

GSD has a dynamic self-starting property and is not dependent on any bootstrap mechanism or fixed hosts for startup. Neither is it dependent on the topology or the mobility of the nodes for its stability. Each node

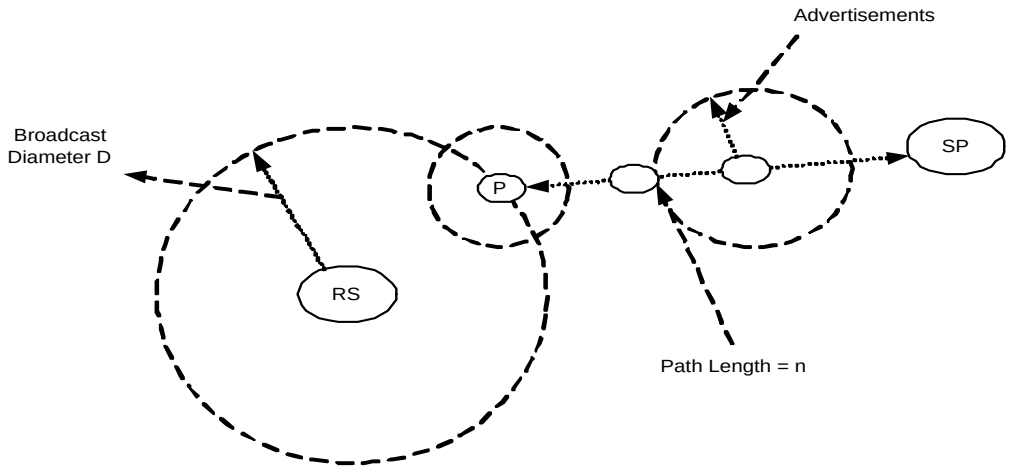

Figure IV.8: Network-wide Reachability study of GSD

maintains a soft state of the services present in its vicinity and hence on failure, does not need to do any fault-recovery during start-up. It passively collects the information by listening to advertisements.

IV.C Network Load Analysis

It might appear that GSD with bounded advertisements and selective forwarding of requests may impose greater network load (in terms of number of messages) than simple global-broadcast based protocol. A global broadcast-based protocol does not have any advertisements. However, it broadcasts the requests to all nodes in the network. In this section, we layout simple equations that determine the network load for each of these protocols for a bounded network.

Let N=number of nodes in the network G. Let us consider that all nodes send out advertisements in GSD.

Let b= total number of nodes that generate service discovery requests

Let T= total time of observation.

- **Broadcast-based Protocol:** Let R_f= Request Frequency (number of requests/second). All requests are broadcast to the whole network. Let m= total number of messages generated in the system due to a single service request being broadcast in the network. Thus, in time T, the total network load generated by Broadcast is

$$M_{BCast} = R_f * m * T * b \tag{IV.1}$$

- **GSD Protocol:** Let A_f= Average Advertisement Frequency (number of advertisements/second) across all the nodes N in G. Let n= total number of messages generated in a single bounded advertisement from a single node in G. Thus total number of messages generated by advertisements in time T by all nodes in G is $M_{Adv} = n * N * A_f * T$.

  Let p=average number of messages generated in the system due to a single discovery request in GSD. Thus p$\leq$m since at its worst case, GSD discovery request would be broadcast through out the network. Total number of messages generated in the network due to requests in time T is $M_{Req} = p * R_f * T * b$.

  We observe that total number of messages generated in GSD is a sum total of the request messages and the advertisement messages. Thus, M_{GSD}= total network load in G due to GSD is given by

$$M_{GSD} = n * N * A_f * T \quad + \quad p * R_f * T * b \tag{IV.2}$$

  We also note that for GSD to have lesser network load than Broadcast, $M_{BCast} \geq M_{GSD}$, or

$$R_f * (m - p) * b \geq n * N * A_f \tag{IV.3}$$

IV.D Experimental Evaluation

We simulated the GSD protocol using the ad-hoc network simulator Glomosim [114]. We primarily compare various discovery mechanisms of GSD with simple broadcast-based discovery that has been predominantly used so far to discover services in ad-hoc/pervasive environments. We note again that in a broadcast-based discovery protocol (dubbed as BCast), a service request is globally broadcast to other nodes in the network until the required service has been discovered. There are no advertisements and the broadcast request dies down after all nodes have received the request once.

Clearly, the worst case performance of GSD (in terms of network load) is when the service request is broadcast to other nodes. This happens when enough service group information to do selective forwarding is unavailable. We call this protocol GSD-B. We also compare average case performance of GSD when GSD performs selective forwarding of request. We call this GSD-S. We also compare performance of the protocols with varying advertisement diameter. We do not compare GSD with global advertisement based protocol, since it generates n times the load generated by request broadcast-based protocol (assuming the request rate

is same as the advertisement rate) and hence is a very inefficient solution for large scale networks. We observe that performance of GSD will deteriorate as the average advertisement diameter is increased. Our experiments show that an advertisement diameter of 1 provides best results.

We assume a pessimistic evaluation strategy and compare GSD in environments less favorable to it. A pessimistic evaluation strategy helps us better justify the effectiveness of GSD in more conducive environments. We impose the following restrictions on the simulation environment:

1. *Request Source Restriction:* The number of request sources sending discovery requests is restricted to 1. This makes b=1 in equation IV.3. This reduces the additive effect formed due to multiple request sources and makes it more difficult for the equation to be true, thus favoring bCast.

2. R_f/A_f *Ratio:* In equation IV.3, since the values of m,p,n are not known beforehand, we observe that a low value of R_f and a high value of A_f would make BCast more favorable as far as network load is concerned whereas the vice versa would make GSD more favorable. Hence in our experiments, we have varied ratio of R_f/A_f from 0.25 to 2.0 to favor BCast for one part and favor GSD for the other part.

3. *Density of Matching Services:* The more the number of SPs, the greater is the chance of either protocols discovering the service. Hence in our experiments, only 10% of the SPs contain the service desired by the discovery request. The initial placement of the matching services were at the edge of the network.

IV.D.1 Experimental Model and Evaluation Metrics

Our experimental model consists of mobile service providers (SP) containing one or more services connected to each other using an ad-hoc network. The mobility of the nodes was assumed to follow random-waypoint [63] pattern. We used an application layer packet generation function to generate service requests at regular time intervals. For the purposes of the simulation, we used representative services S0 to S99 to represent actual services and groups G1 to G10 to represent service groups with G10 being equivalent to the parent service group called “Service” at the root of our hierarchical tree.

All our experiments were carried out with a fixed node density so as to appropriately simulate the effect of increased network size. The results shown correspond to averages over experiments run for 3 different randomization patterns for a total time of 75 minutes with the value of R_f ranging from 1 request/minute to 8 requests/minute. Thus, the plots are averages over a minimum of 225 data points to a maximum of 1800

data points. Figure IV.1 represents the various experimental parameters used and varied in our simulations.

Duration	4500 seconds
Network Area (x,y)	(145 X 145m) to (200 X 200m)
No. of Nodes	50,100, 200
Network Diameter	10, 14, 19
Tx Range (Transmission Range)	30m
Tx Throughput	20kbps
Advertisement Interval	15 seconds
Advertisement Timeout	40 seconds
broadcast jitter	10 milliseconds
Mobility	Random way-point with 2 m/s speed and 5 s stoppage time
Initial topology	uniform topology with nodes equally spaced out in (x,y)
MAX_RETRIES to discover a service	4
Advertisement Diameter	1,2
R_f/A_f	0.25 to 2.0

Table IV.1: Experimental model parameters for Group-based Service Discovery Protocol (GSD)

We evaluated the protocols with respect to several metrics like average response time, average response hops, discovery efficiency, average network load, average message processing per node and other metrics that provide statistics regarding the usage of service groups in GSD. We present the results in the next subsection.

IV.D.2 Simulation Results

This subsection details studies of various metrics like response time, response hops, average network load, discovery efficiency for GSD in our simulated environment in Glomosim. Plots are measures of average values of data sets measured in our experiments. Standard deviations of the observed data are reported in appendix B.

Average Response Time for discovery requests is referred to as the time taken from the instant the request is sent out to the instant a service reply is obtained. We observe in Figure IV.9 that the average response time of BCast is at least 2 times higher than the average response times observed in GSD-S and GSD-B. We also observe in Figure IV.10 that the average Response Hops or average number of hops traveled by the response is greater for BCast. Moreover, average response hops in GSD-S seems to be marginally lower than GSD-B. This shows that our protocol performs better than BCast in terms of response time and average response hops. We believe that the increase in response time is mostly due to the average response hops being about 2 times greater in BCast. The average response hops decrease since each request in GSD travels only till the node where it discovers a matching service description (due to a previous advertisement). The discovery request

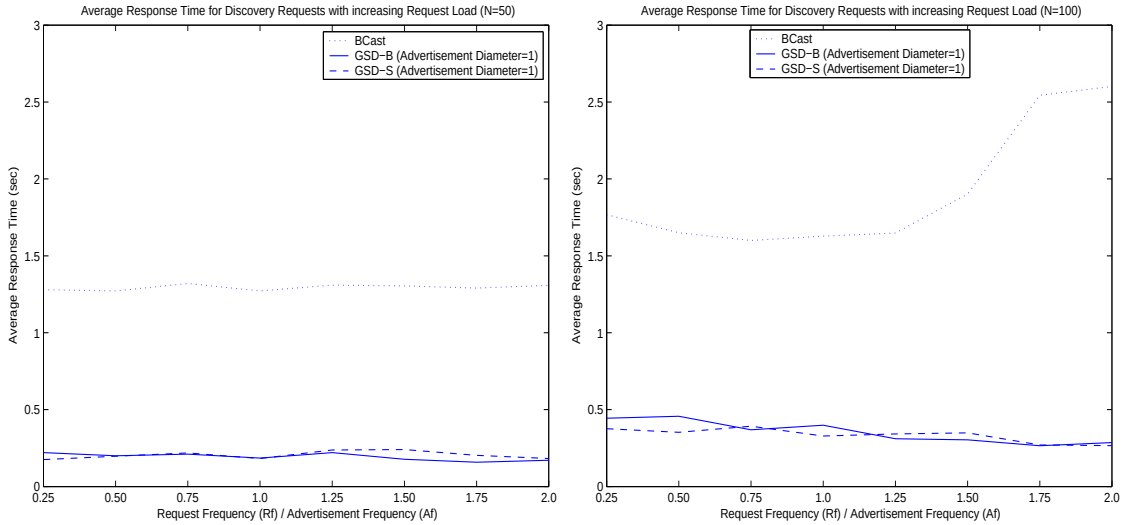

Figure IV.9: Average Response Time statistics of GSD and BCast

does not need to reach the actual service provider (as explained in section IV.A.3).

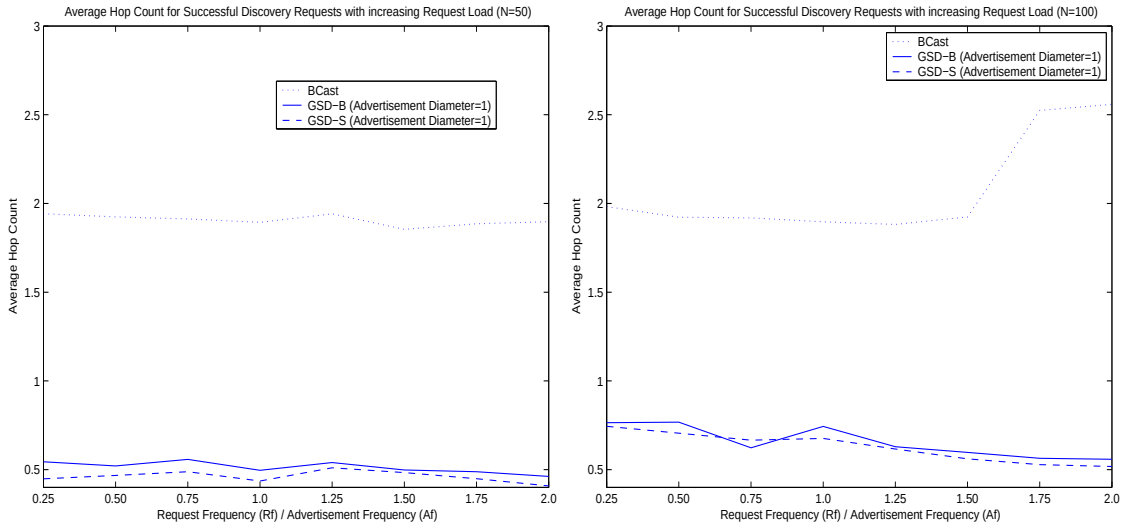

Figure IV.10: Average Response Hops comparison of various versions of GSD with BCast

Figure IV.11 shows the amount of network load generated by the various protocols. Average network load is defined as the average number of messages (advertisements and discovery requests) processed per node. We observe that network load of GSD-S and GSD-B increase very slowly with increasing request load. We also observe that BCast performs better for a low value of R_f/A_f. This is intuitive since according to equation IV.3, a low value of R_f/A_f favors BCast. However, for values of $R_f/A_f \geq 0.50$ for advertisement

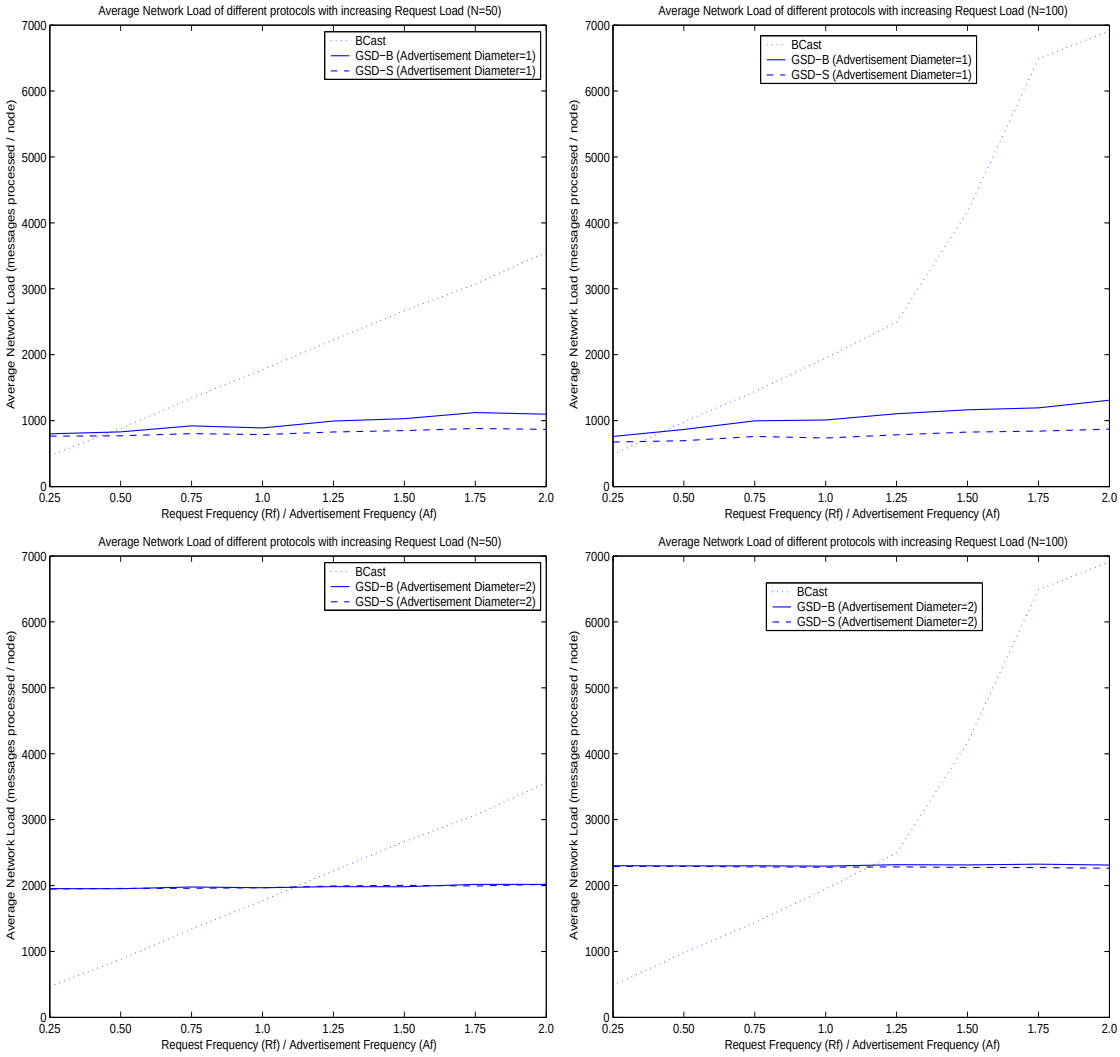

Figure IV.11: Average Network Load comparison of various versions of GSD with BCast

diameter=1, GSD starts performing better. We also notice similar performance improvements of GSD for advertisement diameter=2. This shows that our protocols are very scalable with respect to increasing request load as well as network size.

Understandably, GSD (both GSD-S and GSD-B) generates greater network load with increasing advertisement diameter. (in terms of average number of messages processed per node). However, the increase in the load on the network with increasing request load is very low. Our experiments suggest that GSD-S with an advertisement diameter of 1 provides best results as far as network load and response time statistics are concerned. We also observe that the gradient of increase in the load is much higher in BCast for N=100. This further proves that BCast scales poorly with increasing network size.

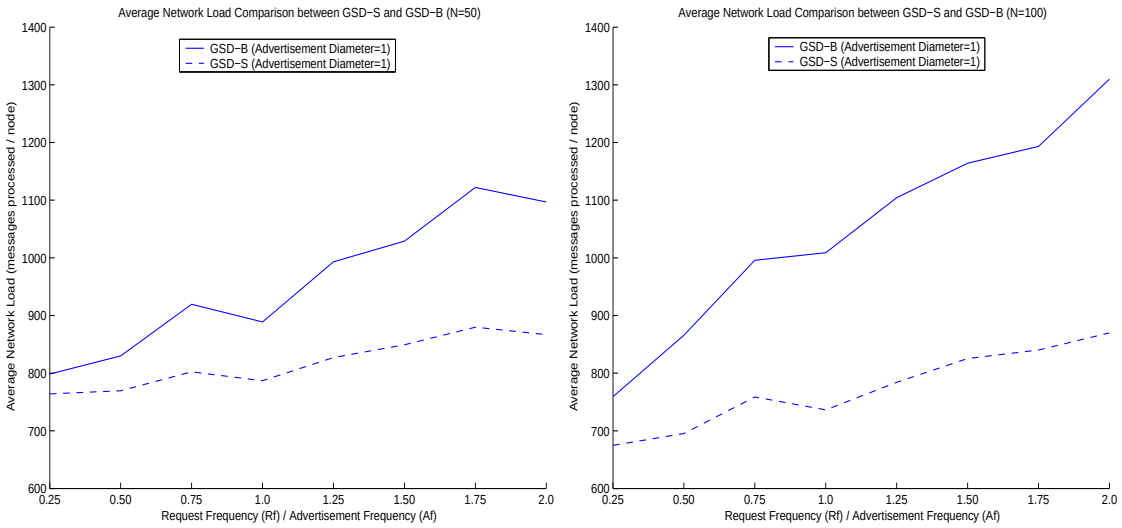

Figure IV.12: Comparison of GSD-S and GSD-B in terms of Network Load

Discovery Efficiency is defined as the fraction of discovery requests that are successful in discovering the required service. One important tradeoff between BCast and GSD-S is that GSD-S might generate *false forwards* leading to a discovery failure. Thus, intuitively, BCast should have a greater discovery efficiency, especially in mobile environments. Figure IV.13 shows the various discovery efficiencies we observed for BCast, GSD and GSD-S. The efficiencies are remarkably similar for N=50. This shows that our protocol performs almost as well as BCast but with much efficient usage of the network thus making it a much more scalable solution.

The efficiency of BCast drops drastically for a greater network (N=100) with high request load. We believe that this is mostly due to the huge network load generated due to broadcasting of all the requests due to which many of the service requests/responses are dropped or lost due to collisions. We could not calculate the number of messages being dropped in case of broadcasts, since Glomosim silently discards broadcast messages if there are collisions. However, Figure IV.14 gives us a comparison of the increase in the number of discovery requests processed per node for the various protocols that further corroborates our argument.

It might seem from Figure IV.11 that GSD-S and GSD-B perform similarly. However, this is not true. Selective forwarding brings about 50% reduction in total network load as we observe in figure IV.12. The difference is not evident due to compression of the plots in figure IV.11. Moreover, from Figures IV.9,IV.10 and IV.13, we observe that GSD has this performance gain without any significant loss in terms of response time, response hops and discovery efficiency. We do not however observe such drastic differences for advertise-

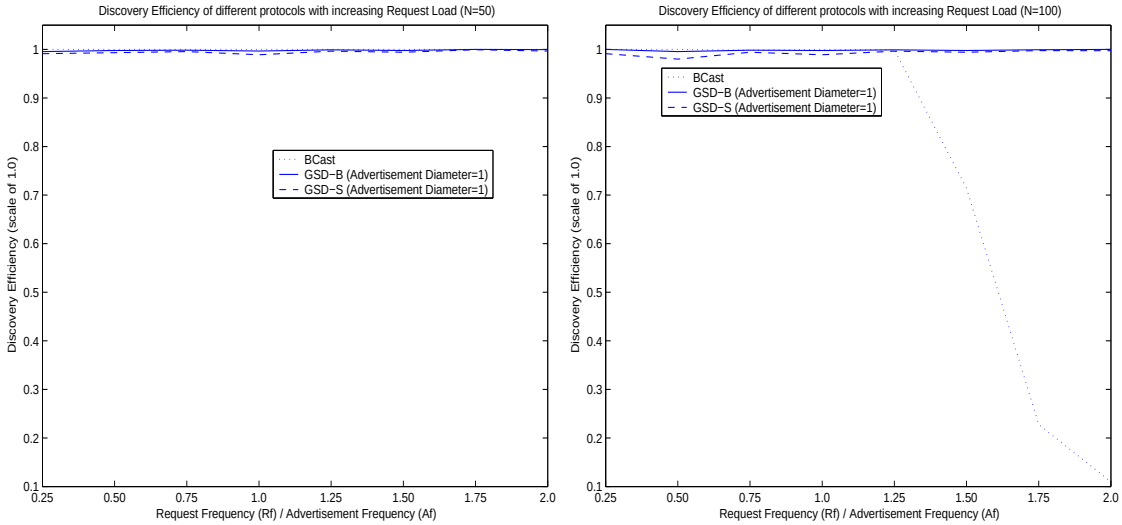

Figure IV.13: Discovery Efficiency comparison for various versions of GSD with BCast

ment diameter=2. We think it is mostly because a higher advertisement diameter replicates the same service information across a greater number of nodes thus reducing the number of effective selective forwards.

Figure IV.15 provides an estimate of the decrease in the average number of selective forward events in the nodes due to an increase in the advertisement diameter in GSD-S. We observe that that GSD-S with an advertisement diameter of 2 performs better in reducing the amount of selective forwards. This follows from our protocol since an increase in the diameter would cause the service to be replicated in greater number of nodes thus increasing its chances of being discovered with lesser number of selective forwards. However, we would still advocate that GSD-S with advertisement diameter of 1 performs better since it generates much less overall network load (Figure IV.11).

IV.E Chapter Summary

This chapter describes a novel protocol (GSD) for service discovery in pervasive computing environments. Service Discovery is done in a peer-to-peer mode rather than a centralized mode, and we use advertisements to disseminate service information. We use an ontology based on OWL (detailed in Chapter III) to describe services and use the Class/SubClass hierarchy of OWL to group services based on their functionality. We use this group information to intelligently route service requests. GSD is scalable to request load and network size and highly adaptable to various pervasive computing environments. We present exhaustive experimental

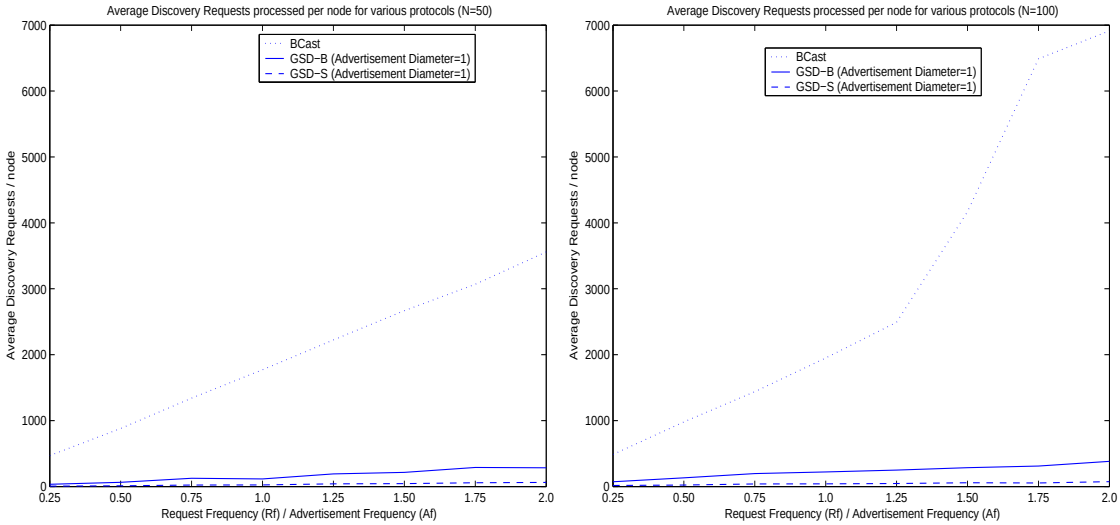

Figure IV.14: Average discovery requests processed per node for various versions of GSD with BCast

results of performance of GSD in mobile environments for various kinds of request load and network sizes. Our results show that GSD scales very well with increasing request load and network size whereas standard broadcast-based solutions used so far for service discovery in ad-hoc networks do not.

Moreover, our protocol provides the same standards of efficiency in discovering services when compared to Broadcast-based solutions. In fact, for large networks and high request loads, broadcast performs worse than GSD in terms of discovery efficiency.

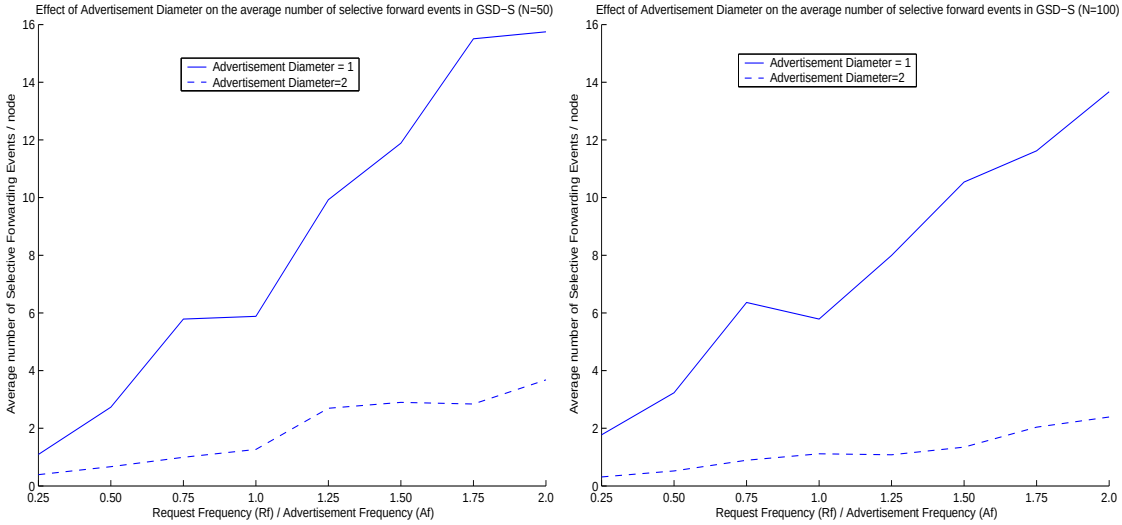

Figure IV.15: Average Selective Forward events processed per node for GSD-S

Chapter V

GROUP-BASED SERVICE ROUTING PROTOCOL

V.A Introduction

This chapter describes Group-based Service Routing protocol(GSR): a new routing and session management protocol for ad-hoc networks as an integral part of a service discovery infrastructure. Traditional approaches place routing at a layer below service discovery. While this distinction is appropriate for wired networked services, we argue that in ad hoc networks this layering is not as meaningful and show that integrating routing with discovery protocols increases system efficiency. Central to our protocol is the idea of reusing the path created by the combination of a service discovery request and a service advertisement for data transmission. This precludes the need to use separate routing and discovery protocols. GSR also combines transport layer features and provides end-to-end session management that detects disconnections, link and node failures and enables service-centric session redirection to handle failures. This enables GSR to accommodate service-centric routing apart from the traditional node-centric routing. We compare GSR with AODV in terms of packet delivery ratio, response time and average number of hops traveled by service requests as well as data. GSR achieves better packet delivery ratio with a minor increase of the average packet delivery delay.

Service invocation is carried out after service discovery and involves sending of service invocation data to the desired service. Service invocation primarily utilizes underlying ad-hoc routing protocols [85, 63, 84, 80] for its operation. Most prior work in the area of service discovery and invocation assumes that the process of service discovery and routing are only loosely coupled. To the contrary, it has been argued in [6, 67] that cross layer integration of protocol stacks, especially in dynamic environments improve system efficiency. There has also been some initiatives in utilizing service-centric data to route packets [8, 18] for wired networks. AODV

[85] defines a service extension to its routing protocol to incorporate discovery. However, as discussed in chapter II, the extension considers only bit-level addressing of services and is primarily based on the broadcast-driven nature of the AODV protocol. We argue that an efficient service discovery protocol can provide further efficiency to an integrated discovery and routing protocol.

Apart from the benefits pointed out in [6], integration of the service discovery with routing in ad-hoc networks provides the following benefits: (1) **Usage of available routes:** The discovery infrastructure while trying to discover a service discovers multiple possible paths to reach a service too. Typically, a discovery infrastructure discards this information. While this is not needed in wired networks (since network topology is fixed and there are very few route changes), it could be effectively used by ad-hoc routing protocols; (2) **Service-centric Route Enablement:** Multiple instances of the same service may potentially exist on different ad-hoc nodes. If needed, the integrated layer can use the information in the discovery infrastructure to route the invocation data to a *service instance* instead of a *node address*. This makes the integrated protocol *service-centric* instead of the traditional *node-centric* approach towards routing; (3) **Resilience to Service-Node Failures:** Moreover, all routing protocols are node-centric (they route based on the *node address* or IP address) and hence prone to failure of that node. Service-node failure leads to the service being unavailable leading to a service failure. Ideally, we would like service discovery and invocation to be immune to service-node failure since multiple instances of the same service could be existing on different nodes. We achieve this by combining the service discovery and routing layers. We borrow the notion of *path-repair* which is widely used in optical networks where the switching fabric is aware of multiple paths from source to destination. However, instead of multiple paths, the service discovery layer is aware of multiple instances of a specific type of service and the route to that service. In the event of a service-node failure, this new integrated layer can rediscover another instance of the service and deliver data to it. (4) **Reduced Routing Overhead:** Reuse of the discovery infrastructure to route invocation data results in reduced overhead since certain mandatory actions of standard routing protocols like *route discovery*, *path maintenance* can potentially be integrated with the discovery infrastructure. Typically, ad-hoc service discovery protocols involve local or global broadcasting of service advertisements and service requests until the requestor has discovered the desired service. Once a service is discovered, any standard *on-demand* or *link-state/distance-vector* based routing protocol is used for service invocation. These routing protocols, which reside below the service discovery layer, perform route discovery (in case of on-demand protocols) and link-maintenance (in case of link-state/distance-vector protocols) using a flooding-based approach. The overhead is essentially redundant

because these steps could be combined with the broadcasts done during service discovery.

The GSD service discovery protocol described in Chapter IV utilizes available underlying ad-hoc routing protocol for service invocation and data transmission. Our routing protocol namely Group-based Service Routing protocol (GSR) is based on the concept of reusing the path created by a service discovery request and a service advertisement to enable data transmission. A service request matches a service advertisement at an intermediate node. The route between the source and the destination service is formed by the combination of the path traversed by the service request and the path traversed by the advertisement. We call the path that is actively participating in data transfer as the *ACTIVE_PATH*.

Traditional routing protocols do not encapsulate transport layer features like end-to-end network state maintenance, session management etc. We have augmented GSR to provide the transport layer features of end-to-end session management during service invocation since GSR offers upper layer applications with features of reliable communication with end applications and services. We call the augmented protocol Group-based Service Routing protocol with Session management (GSR-S). A session is maintained for each invoked service at each node in the *ACTIVE_PATH*. Each session handles various kinds of failures (service-node and link failures), buffers ongoing data transmissions at the intermediate nodes and performs service-centric session redirection depending on session-specific preferences from the source of the request. GSR defines two kinds of sessions: *Service-Consistent session* and *Node-Consistent session* to offer various kinds of service guarantees to the end user.

We note that GSR also supports standard node-centric routing. We dont attempt to override the approach of keeping routing at a separate layer in the network stack. We argue that when the upper layer is essentially a service discovery protocol, then integrating the discovery with the routing protocol yields better efficiency.

We present simulation results comparing our integrated routing protocols (with and without session management) with a version where we have our service discovery protocol running over standard Ad-hoc On-demand Distance Vector (AODV) [85] routing protocol. We compare average packet delivery ratio, average service response time, average packet delay and average service response time and packet hops (data and request packets). Our integrated protocol gives almost 100% packet delivery ratio with a minor increase in the average packet delay. This is much better than the standard performance of AODV as a routing protocol in such environments. We also observe that reusing service discovery paths often results in reduced data path length. Predictably, one drawback of our protocol is in the increase in the average packet delay for GSR-S. This is mostly due to the buffering and retransmission caused due to session redirection by GSR-S. However,

an analysis of delay distributions shows that majority of the packets have delays comparable to AODV packet delay. We chose AODV as a baseline for comparison over other protocols because link-state/distance-vector protocols [80, 84] have sufficiently more routing overhead than on-demand protocols (AODV, DSR).

V.B GSR Protocol Key Definitions

In this section, we define key terms and concepts associated with GSR and GSR-S. GSR uses GSD as the service discovery protocol and incorporates routing support to it by enabling service invocation and data transmission. Some of the definitions in this section are intuitive and well-known. We define them for the purpose of clarity with respect to our work.

- **Request Source (RS):** Node from where a particular service discovery and invocation request originates. Note that a node is referred to as the *Request Source* only with respect to the request that it has originated.
- **Service Provider (SP):** Nodes that contains services that are accessible from other peer nodes.
- **Intermediate Node (IN):** For a particular discovery request, a node where the discovery request finds out a matching service.
- **ADVERTISEMENT_PATH:** Path traversed by a service advertisement starting from a particular SP to an IN. It is measured in *number of hops*.
- **REQUEST_PATH:** Path traversed by a service discovery request starting from the RS. It is also measured in *number of hops*.
- **RESPONSE_PATH:** Path traversed by a service reply that is generated in response to a service discovery request. It is measured in *number of hops*.
- **DATA_PATH:** Path formed by combining an ADVERTISEMENT_PATH and a REQUEST_PATH that meet at an IN. The IN could as well be the SP or the RS (in which case the length of either the ADVERTISEMENT_PATH or the REQUEST_PATH would be 0).
- **ACTIVE_PATH:** It is defined as the DATA_PATH actually employed to transmit service invocation data[1] to the discovered SP. Out of multiple DATA_PATHs, a single path is chosen to be the AC-

[1] we use the term *service invocation data* and *service data* interchangeably in the rest of the paper.

TIVE_PATH for a service invocation.

- **Service-Consistent Session**: It refers to a session that requires all data to be sent to a particular service but does not require it to be sent to a particular node. In case of service-node or link failures, such sessions could be redirected to another node hosting the same service.
- **Service-consistent Discovery Request:** Service discovery request that just specifies the service description and does not contain any node specific information.
- **Node-Consistent Session**: A Node-Consistent Session requires service invocation data to be sent to a particular service at a particular node.
- **Node-Consistent Discovery Request:** Service discovery request that contains node specific information apart from the service that it is trying to discover.

V.C GSR Protocol Model

GSR uses the discovery infrastructure of GSD to support service invocation and transmission of service data instead of using a separate ad-hoc routing protocol. The central idea in GSR is to utilize the ADVERTISEMENT_PATH and the REQUEST_PATH (or the RESPONSE_PATH) to transmit service invocation data. Like AODV [85], we assume link reversals on the underlying ad-hoc connection. Hence if a node A is reachable from node B, then node B is also reachable from node A under identical conditions. GSR creates several DATA_PATHs after a service has been discovered by appropriately combining the ADVERTISEMENT_PATH and the REQUEST_PATH. It selects a suitable DATA_PATH for transmission of service data. This becomes the ACTIVE_PATH. It also maintains end-to-end session over the ACTIVE_PATH to detect link failures or service-node failures. We employ the technique of partial path reconstruction that enables session redirection to a different node or reconnection to the same node through a different path depending on service guarantee requirements. We explain the various salient components of GSR in the following subsections.

V.C.1 Dependence on GSD

Current implementation of GSR depends on GSD for its successful operation. However, the design of the routing protocol is not dependent on GSD. GSR could as well be integrated with any other ad-hoc service discovery protocol. This is because the only condition that GSR imposes on the underlying discovery protocol

is the ability to discover a service. However, using GSD as the driver protocol enables GSR with the following advantages: (1) **Efficient usage of network bandwidth:** GSD performs selective forwarding (instead of broadcasting) and efficiently discovers routes to the discovered service. This is again mostly due to the hierarchical grouping of services in GSD.(2) **Semantic Session Redirection:** GSD performs semantic service matching that enables loose or near matches of services. Thus, if a service request tries discovering service S1 belonging to groups G1, GSD enables the discovery request to match with a service S2 belonging to group G1 that matches most of the functional requirements of service S1. This lets GSR to redirect a session to another service having similar functionality in case of service-node failure. We note that, a hierarchical grouping of services could be replaced by a flat grouping (where there is only one level in the hierarchical tree). However, a hierarchical approach simply enhances the chances of GSD to discover services that are farther and farther away from the specified description and hence enables graceful degradation in case of service unavailability.

V.C.2 Data Path Setup

A service discovery request matches service descriptions at several intermediate nodes (IN). On a successful match, a service reply is generated by the IN and transmitted back through the REQUEST_PATH in the reverse direction. Each node in the REQUEST_PATH maintains a pointer to its previous hop in the path leading back to the RS. This is established at the time the discovery request reaches the nodes. However, REQUEST_PATH only contains the path from the RS to the IN. In GSD, service advertisements are broadcast. GSD maintains no information about the route from the IN to the SP. We have enhanced service advertisements in GSR where in each node receiving an advertisement maintains a pointer to its previous hop leading to the SP.

The service reply is dropped in case of disconnections or link failures. Link failures result when the node upstream in the path has either moved or shut itself down. RESPONSE_PATH refers to the actual path that the reply traverses to reach the RS from the INs. Each node in the RESPONSE_PATH maintains a forward pointer to the node leading to the IN. This is done when the service reply traverses the corresponding node. Note that these two paths could be different if a standard routing protocol was used to transmit the service reply back to the source. Our protocol reuses the path that was already traversed by the service request instead of having to discover another new one. A DATA_PATH is formed by combining the RESPONSE_PATH with the ADVERTISEMENT_PATH to establish a complete route from the RS to an SP. Figure V.1 shows the various steps in involved in the creation of the DATA_PATH.

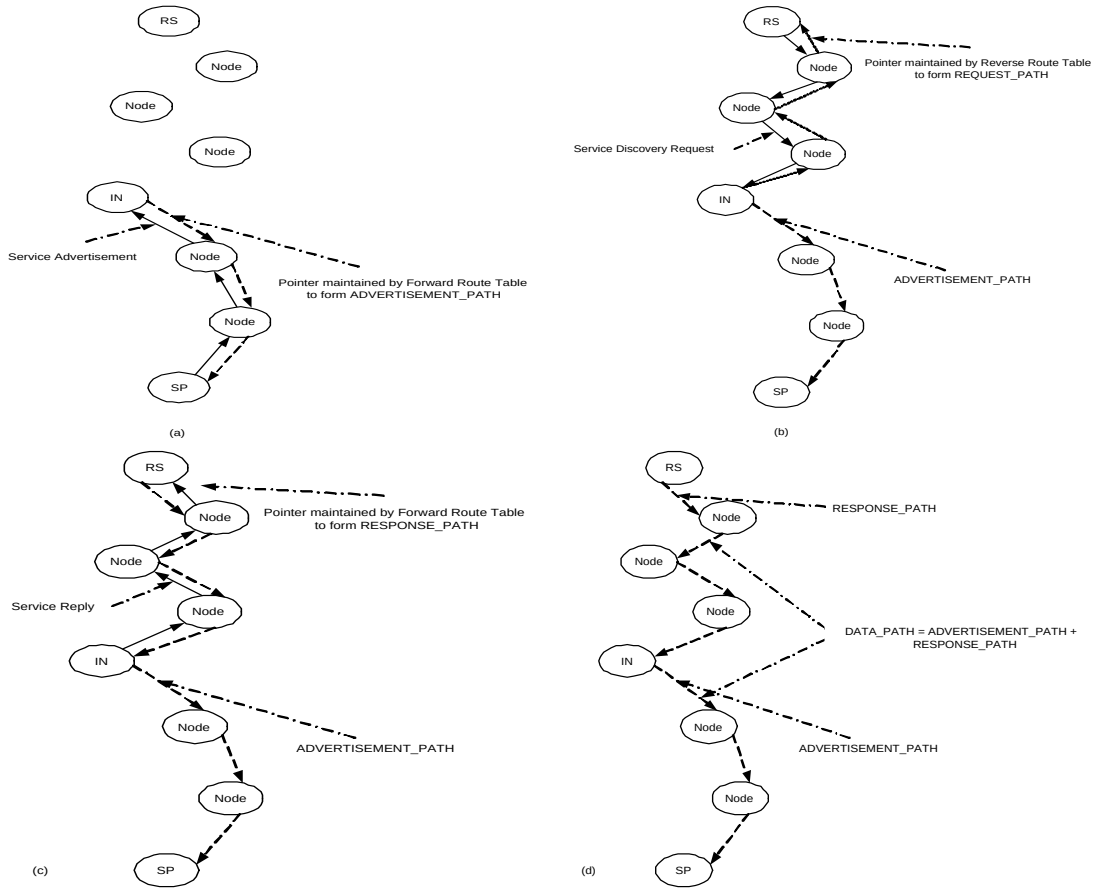

Figure V.1: Creation of a single DATA_PATH in GSR. The ADVERTISMENT_PATH is set up first when service advertisement is sent by the SP (Figure V.1a). The REQUEST_PATH is set up after that (FigureV.1b) . The service reply propagates back to the RS using the REQUEST_PATH and in turn sets up the RESPONSE_PATH (Figure V.1c). The ADVERTISEMENT_PATH and the RESPONSE_PATH form the DATA_PATH (Figure V.1d).

We note that for a certain discovery request, there could be DATA_PATHs established to different *similar* services or *multiple instances* of the same service. There could also be multiple DATA_PATHs to the same service through various INs. The IN bridges the RESPONSE_PATH with the ADVERTISEMENT_PATH. Each node in the paths maintain time outs corresponding to the paths. A DATA_PATH thus expires if either the ADVERTISEMENT_PATH or the RESPONSE_PATH times out.

V.C.3 Active Path Selection

We have observed in the previous section that various DATA_PATHs are formed after service discovery due to the presence of potentially multiple instances of the same service. Once the RS receives a service reply,

it determines the best service to perform the service invocation. The details of the best service selection can be found in [24]. In this protocol, we are primarily concerned with routing service invocation data to the selected service. However, multiple DATA_PATHs could also be formed with the same instance of the selected service. This is because:

1. Service Requests propagate through multiple outgoing links (since they are selectively forwarded) and could potentially be answered by the same INs leading to formation of multiple DATA_PATHs. Please note that an IN detects duplicate requests. However, the RS might send out multiple discovery requests for the same service.

2. There could be multiple instances of a single unique service cached on different INs. These INs could reply to a service request thus resulting in multiple RESPONSE_PATHs and hence multiple DATA_PATHs.

One of the DATA_PATHs is chosen to be the ACTIVE_PATH for the corresponding service invocation. Currently, an ACTIVE_PATH is chosen from available DATA_PATHs based on *minimal hop count* from the RS to the SP. This information is obtained from the service reply from the IN. The service reply while traversing the RESPONSE_PATH computes the length of the RESPONSE_PATH (in hops). However, it does not know the length of the ADVERTISEMENT_PATH. To accommodate for this, we enhanced each service advertisement to compute the length of the current path and store it in each IN it traverses. An IN, while replying to a discovery request conveys the ADVERTISEMENT_PATH length to the service reply. This is used to compute the final DATA_PATH length by the RS.

Once an ACTIVE_PATH has been selected, the RS uses it to transmit service invocation data. All other DATA_PATHs constructed during the discovery phase time out if they are kept unused and all route information is deleted.

V.C.4 End-to-end Session Maintenance

GSR uses the selected ACTIVE_PATH to transmit service invocation data. Service invocation in ad-hoc networks require various kinds of service guarantees. For example, the request might require that all the data be sent to one instance of the discovered service. One example of such service invocation could be data streaming applications in sensor networks where all data of a particular type needs to be sent to one instance of a service only. On the other hand, the request could be also specify that the data could potentially go to

multiple instances of a particular service. An example of such service could be audio streaming applications where the audio is sent to the music service nearest to a person. Service guarantees could also be relaxed to service groups where the data could potentially go to any service belonging to a particular functional group. Example of such services could be music streaming service where the music could potentially be sent to any ‘”speaker” in a given location. GSD protocol by virtue of its semantic service matching features is capable of discovering services based on required service guarantees. Furthermore, we have incorporated session management into GSR to provide various levels of service guarantees during service invocation too.

GSR supports two kinds of session, namely: *Service-Consistent session* and *Node-Consistent session*. A Node-Consistent session (as defined previously) requires that all data in a particular invocation be sent to *one instance* of the discovered service. A Service-Consistent session on the other hand only requires that the data be sent to the particular service. It does not impose any restriction on the service instance. In subsection V.C.5, we discuss how these strict and relaxed guarantees are enforced in case of failures.

Each node in the ACTIVE_PATH maintains a session for an open connection. A session is initiated by the RS at the time of sending service invocation data. The RS specifies the type of session it desires. Each node in the ACTIVE_PATH maintains a *Session_Handler* for each connection going through it. A *Session_Handler* keeps the following information for each connection:

<Service_Description, Current_SessionId, ACTIVE_PATH_Source, ACTIVE_PATH_Destination, Next_Hop, Previous_Hop, Session_LifeTime, Session_State, Session_Strictness, Session_Buffer>.

Session_Buffer is used to buffer packets during failures. A *Session_Handler* is initiated when a service invocation data packet is received by a node. Thus *Session_Handlers* are initiated at different times at different nodes in the ACTIVE_PATH. Each data packet piggybacks along with it session related information. Session related information include *SessionId, Service_Description* of the service to which this session belongs to, *ACTIVE_PATH_Destination, Session_LifeTime* and *Session_Strictness*. The parameter *Session_Strictness* specifies whether the session has to be node-consistent or the session could be service-consistent. *ACTIVE_PATH_Destination* refers to the address of the selected SP.

V.C.5 Path Disconnections and Session Redirection

End-to-end session management in GSR provides it with connection monitoring to detect link failures and node failures. A *Link Failure* might happen due to disconnections or mobility of nodes in the ACTIVE_PATH. The node that is upstream in the ACTIVE_PATH detects this when it fails to transmit data packets to the next hop. A *Service-Node Failure* on the other hand refers to the failure due to shutting down of the SP or the SP becoming unreachable. We detect this at the node just ahead of the SP in the ACTIVE_PATH when it fails to transmit data packets to the SP.

GSR takes corresponding actions depending on the type of session. Intermediate nodes are not able to distinguish between a Link Failure and a Service-Node Failure. This is because, both the failures are triggered from failure to transmit data packets successfully. Thus, for a Node-Consistent session, the intermediate node that detects the failure uses the discovery infrastructure to discover another route to the required SP. A Node-Consistent discovery request is sent out during this type of failure. All packets coming into the node during this period are buffered in the corresponding *Session_Buffer*. If an alternate route is discovered, then all buffered packets are transmitted through the new path that becomes augmented to the already existing ACTIVE_PATH. Figure V.2 describes the redirection of the ACTIVE_PATH. The session is dropped if the node fails to discover a route to the required SP.

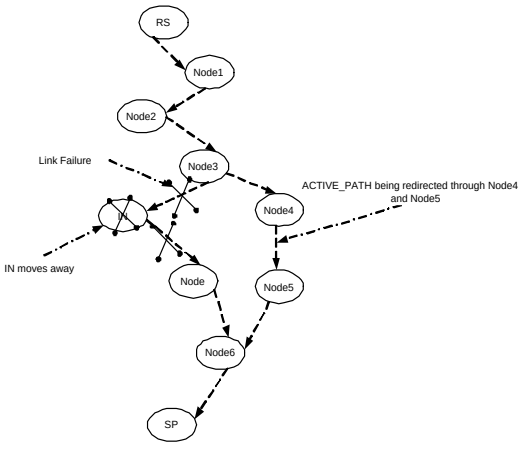

Figure V.2: ACTIVE_PATH Redirection in GSR

GSR resorts to service-centric session redirection in case of link or service-node failures for a Service-Consistent session. The node that detects the failure tries using GSD to discover a service that has the same specification as of the service in the ongoing session. The discovery request sent out during this period is

Service-Consistent. On a successful discovery, the session is redirected to the new SP. Note that the node detecting the failure redirects the session. This does not impose any load on the RS. Moreover, the part of the ACTIVE_PATH that is usable remains intact. The ACTIVE_PATH from the RS to the new SP is established by combining the old ACTIVE_PATH from the RS to the node that detects the failure and the new ACTIVE_PATH formed from that node to the new SP.

This novelty of our routing protocol enables service invocation data to be routed based on service descriptions, thus enabling GSR to be service-centric. Currently, GSR tries discovering services exactly matching the description of the service in the ongoing session. However, we could also redirect a session to a service belonging to the group of the service in the session by utilizing the semantic service matching features and the hierarchical service groups. Figure V.3 describes session redirection in GSR. By buffering packets and retransmitting them, GSR tries to improve the reliability of the protocol. However, it does not guarantee delivery of all the packets in the service invocation data. We show in section V.E that GSR performs reasonably well compared to other routing protocols in terms of data delivery.

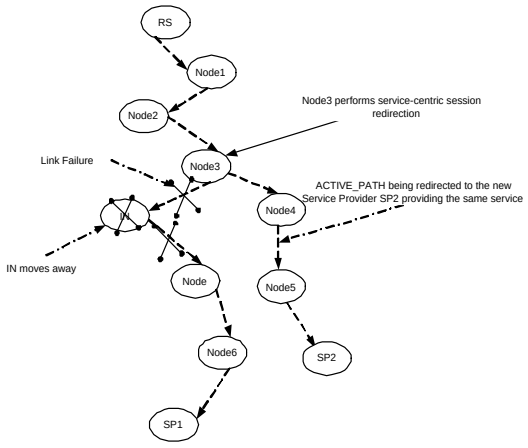

Figure V.3: Service-centric Session Redirection in GSR

V.D GSR Implementation Components

We have employed the use of several existing and new techniques to augment our discovery infrastructure to enable service-centric data routing. We employ the use of *Forward Route Tables* and *Reverse Route Tables* to maintain path specific information during service discovery and invocation. Additionally, we also maintain a *Session Table* that stores information about ongoing sessions through a node. A *Session_Handler* in each

node maintains session data corresponding to each session. A *Session_Handler* could belong to any of the four session states, *Session_Discovery, Session_Active, Session_Dropped* and *Session_Finished*. In this section, we describe the various implementation level components of GSR.

V.D.1 Reverse Route Table

Each node in addition to maintaining a *Service Cache* (for GSD) also maintains a *Reverse Route Table*. It is a node-centric route table indexed by the *source address* of a node. Important fields in the Reverse Route Table (RR Table) are:

<Source-Address, Previous-Address, Life-Time, Hops-To-Source>

Reverse Route Table performs the function of reverse routing a service reply back to the source of the request using the REQUEST_PATH. This makes the REQUEST_PATH same as the RESPONSE_PATH in GSR but in the reverse direction. The RR table is updated during the time a service request arrives at a node. Each service request packet contains a *Last-Address* field. Each node seeing a service request changes this field to reflect its own address before forwarding it to other nodes in its vicinity. This value is stored in the RR Table (as *Previous-Address*)of each node the request traverses. The RR Table is used to send a service reply back to the RS. Our protocol offers updating of the RR Table based on shortest route or recency of the request.

- *Shortest Route Updation*: The RR Table for a source address is updated only when it has received a request that has lesser value of *Hops-To-Source* than the existing entry (if there is one).
- *Recency-based Updation*: The RR Table is updated whenever a request received from a source is more *recent* than the previously stored value. Recency is determined using the timestamp of the service request and the *Life-Time* of the entry.

These actions are performed only after the discovery layer has handled duplicate requests and replies. The entry in the table is kept for REV_ROUTE_TIMEOUT time units. The REV_ROUTE_TIMEOUT value thus determines the time limit for which intermediate nodes maintains the REQUEST_PATH. Shortest Route Updation could be used for relatively stable ad-hoc networks where as recency-based updation could be used for fast-moving ad-hoc networks where the recency of the route is very important.

V.D.2 Forward Route Table

It is a node-centric route table indexed by the *Destination Address* of nodes. Important entries in the Forward Route Table (FR Table) are:

<*Destination-Address, nextHop-Address, Life-Time, Hops-To-Destination*>

FR Table performs the role of forming the DATA_PATH by combining RESPONSE_PATH and ADVERTISEMENT_PATH. Service Invocation Data is routed through the selected DATA_PATH using FR Table. FR Table determines the *nextHop-Address* when data packets reach a certain node in the path. FR Table is updated in when a service reply or an advertisement arrives a node.

- *Service Reply based Updation*: A service reply packet contains two fields (apart from others) namely: *Last-Address* and *Service-Source-Address*. The *Last-Address* identifies the next hop required for a data packet to reach *Service-Source-Address*. These values are stored in the FR Table. Each node, before forwarding a service reply changes the *Last-Address* field to reflect its own node address.

- *Advertisement based Updation*: The FR Table is also updated when a service advertisement reaches a node. The *Source Address* of the advertisement and the *Last-Address* from where it has been forwarded are stored in the FR Table. This forms the ADVERTISEMENT_PATH. Thus each node on the ADVERTISEMENT_PATH knows the next hop neighbor in case it has to forward data to the source of the service advertisement.

FR Table updation is based on shortest route to destination as well as recency of advertisement and service reply. This is analogous to the updating on RR Table. Each entry is kept for FOR_ROUTE_TIMEOUT time units and that determines the duration of the time the DATA_PATH remains idle. The DATA_PATH is destroyed if it has not been used within this time interval. Apart from the above mentioned updations, the timeout value of entries in the FR Table that participate in an ACTIVE_PATH is updated whenever a session is initiated in an ACTIVE_PATH.

V.D.3 Session Maintenance

Session_Handlers at each node manage sessions for each connection through the node. Data transfer is performed by forwarding the data packets through the ACTIVE_PATH using the FR Table. After a Ses-

sion_Handler is initiated, it constructs an entry in the Session Table. It obtains the required information from the session-related information piggybacked in an invocation packet (explained before) and from the FR and RR Tables. As mentioned before, the Session_Handler can possibly have four states. A session after being initiated belongs to either *Session_Discovery* or *Session_Active* state. The states of the Session_Handler and the corresponding actions performed it are explained in detail in this subsection.

- *Session_Discovery*: This is the state when the required service has not yet been discovered and the *Session_Handler* is waiting for the requested service. *Session_Handler* on entering this state initiates a service discovery using GSD. The *Session_Handler* also buffers incoming data packets in the *Session_Buffer*. A *Session_Handler* can go into this state from either *Session_Active* state (explained below) or right at the inception of the session at the source of the request. During the inception of the session, the discovery request sent out by the *Session_Handler* is always *Service-Consistent*. However, if this state has been reached from *Session_Active* state, the discovery request sent out depends on the type of the currently existing session. The state goes either into *Session_Active* state if a service meeting the session specifications was discovered, or goes into *Session_Dropped* in case of a discovery failure after repeated trials.

- *Session_Active*: This is the state when a *Session_Handler* engages in transferring data packets in an ACTIVE_PATH. A *Session_Handler* monitors the outgoing link for a certain connection. Whenever, it receives a data packet, it checks the FR Table to determine the next hop for it. It then forwards the data packet to the next hop. In the event of a link failure or node failure, the outgoing packet is buffered. The *Session_Handler* immediately goes back to the *Session_Discovery* phase on detecting this. This state can be reached from either a *Session_Discovery* state or during the creation of the session at a node. A node other than the source node or the destination node, but belonging to the ACTIVE_PATH can go into this state right at the creation of the session. This is because the node already belongs to a DATA_PATH and hence knows a route to the particular service/destination. Hence it can actively participate in data transfer without having to go into the *Session_Discovery* state. It transitions to the *Session_Finished* state in case of a successful transfer or to the *Session_Dropped* state in case of an unsuccessful transfer.

- *Session_Dropped*: This is the state when either the *Session_Handler* has failed in transmitting all the packets to the destination. This may result due to the failure of a node to discover the required service.

All session-related information is purged from the participating nodes. Pending data packets for a dropped session are dropped too. A *Session_Handler* that goes from the *Session_Discovery* to this state, has not succeeded in discovering the required service. A session buffer overflow may lead to a session being dropped too. This state transitions into the *Session_Finished* state at the end.

- *Session_Finished*: This is the end state reached by the *Session_Handler* once it has finished a successful or an unsuccessful session. It is reached either from the *Session_Active* or *Session_Dropped* state.

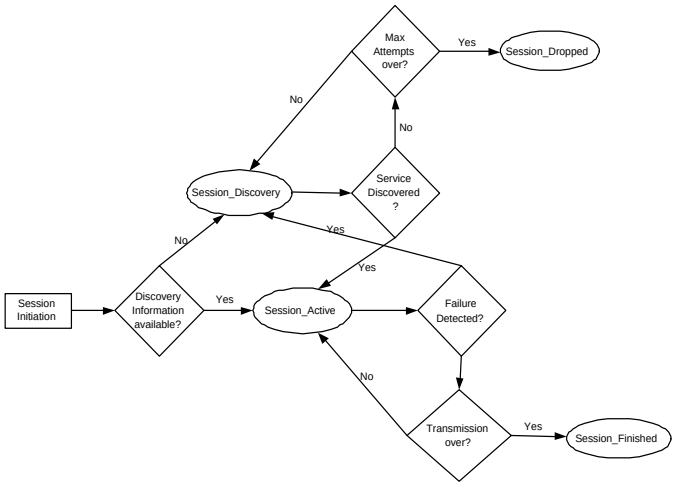

Figure V.4: Session State Transition Diagram in GSR-S

Figure V.4 depicts the session state changes. Our session maintenance and failure handling detects most failures and resorts to appropriate fault handling mechanisms depending on the session type (as explained in section V.C). However, there are certain instances where the node failure might go undetected. For example, if a node is in *Session_Discovery* state, and it goes down after having buffered all incoming packets, it wont be detected by its upstream neighbor in the *Active_Path*. However, if some packets were still being transmitted by the upstream node, it would be able to detect the failure due to packet drops and take necessary actions.

V.E Experiments

We implemented GSR on the ad hoc network simulator Glomosim [114] under various mobility conditions and different node topologies. We compared GSR with our basic model where we had our service discovery architecture as a layer on top of AODV. We had two versions of our integrated protocol. The basic version

does not do any session management. We call this GSR (Group-based Service Routing). The version of our protocol that performs end-to-end session management is termed as GSR-S (Group-based Service Routing with Session Management). We compare these two versions with the basic version where the discovery layer uses AODV for all routing purposes. We call this GSD+AODV (Group-based Service Discovery+AODV).

We implemented the integrated protocol as a routing layer in the Glomosim protocol stack. We used CBR (Constant Bit Rate) messages from the application layer as triggers to invoke discovery and routing in our protocol. Simulation Environment consisted of node topologies ranging from a topology with 25 nodes to a topology with 64 nodes. In this paper, we present results for the two extremes (25 nodes and 64 nodes). We used a random way-point mobility pattern for all the nodes. The initial setup followed a grid structure where all the nodes were distributed in a grid over the terrain. For the purposes of the simulation, we used representative services S0 to S64 to represent actual services and groups G1 to G10 to represent service groups with G10 being equivalent to the parent service group called “Service”. Through out all the experiments, we used a pessimistic evaluation strategy by keeping the requested service only on 8% of the nodes. This is because, intuitively with the increase in the availability of the service, the performance of the protocol will improve. All the simulation parameters including the various timeouts used for different tables and link and radio layer parameters have been enumerated in Figure V.1.

Duration	600 seconds
Network Area (x,y)	(110 x 110m), (250 X 250m)
Initial Topology	Grid topology
Network Diameter	7, 11
Tx Range	30m
Tx Throughput	20kbps
No. of Nodes	25, 64
Advertisement Interval	10 seconds
Advertisement LifeTime	5 seconds
Reverse Route Timeout	12 seconds
Forward Route Timeout	8 seconds
broadcast jitter	10 milliseconds
GSR-S Session type	Service-Consistent

Table V.1: Experimental model parameters for GSR and GSR-S

We compared GSR, GSR-S and GSD+AODV with respect to packet delivery ratio, average response time for request, average response hops, average packet delay and average packet hops. We define mobility of nodes by P(Sm, SMax) where :

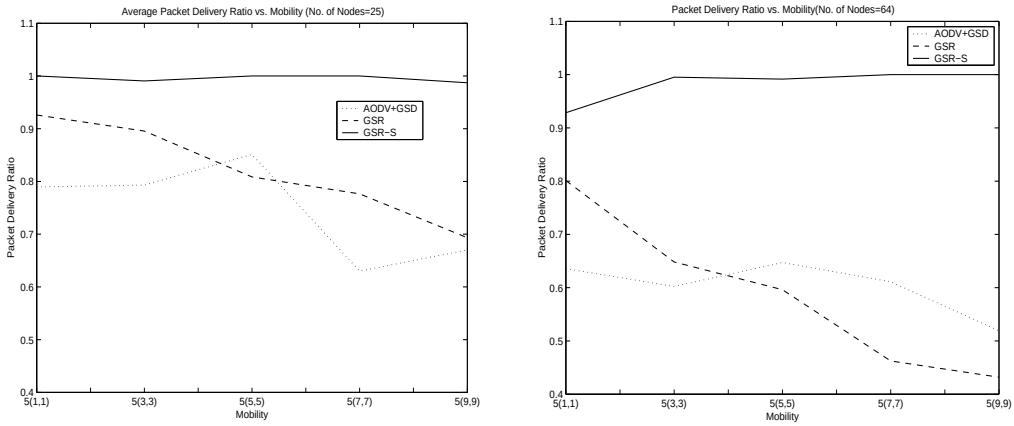

Figure V.5: Average Packet Delivery Ratio Comparison graph of GSD+AODV, GSR and GSR-S

P= Pause (in seconds) after the node has moved to a new position

Sm= Minimum speed (meters/sec) of movement of the node

SMax= Maximum speed (meters/sec) of movement of the node

The node moves with a speed within the range (Sm, SMax). In our simulations, we have varied node speed ranging from 1 meters/second to 9 meters/second with a pause time of 5 seconds for all the experiments. All the time calculations (packet delay, response time) are in seconds.

- *Experiment 1: Packet Delivery Ratio vs. Mobility*

  Average Packet Delivery Ratio is defined as the number of data packets received by the final destination as a fraction of the number of data packets transmitted by the source. In Figure V.5 we plot the average packet delivery ratio as a function of mobility. GSR-S quite predictably shows a packet delivery ratio of almost 1 which is significantly higher than either AODV+GSD or GSR. This is mainly due to the end-to-end session management. This shows that integrating session management along with routing provides with improved efficiency in packet delivery. However, it is also noticed that packet delivery ratio of GSR is more than AODV+GSD in low mobility situations. However, AODV seems to be performing somewhat better in high mobility situations (9 meters/second) for networks consisting of large number of nodes (64 nodes in the graph). This is because often times in high mobility scenarios, the reverse routes break due to node mobility contributing to the decrease in the delivery ratio. AODV, on the other hand discovers a fresher route each time it transfers service data.

- *Experiment 2: Average Data Delay vs. Mobility*

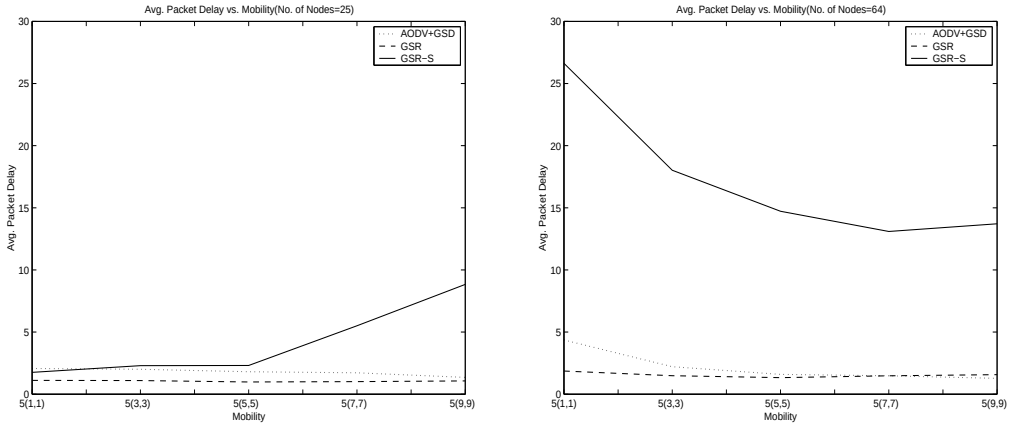

Figure V.6: Average Packet Delay of GSD+AODV, GSR and GSR-S w.r.t varying mobility

We calculated the Average Packet Delay for data received in Experiment 1. Logically, packet delay should increase in GSR-S since GSR-S buffers dropped packets, and again retransmits them after the session has been re-established. The results have been plotted in Figure V.6. As expected, we see that packet delay in GSR-S is greater than delay in GSR or AODV+GSD. However, we also observed that the distribution of delay for GSR-S shows a high proportion of packets (about 75%) being delivered with delays almost similar to AODV or GSR. The average data delay is strongly affected by a small fraction of the packets which take a very long time to be delivered. These are packets that had been buffered by a session during a node or link failure. Figure V.7 shows the delay distribution. We also observe that GSR performs consistently better than AODV in terms of packet delay. This is because GSR uses one of the DATA_PATHs to transmit data whereas AODV transmits data only after having finished its route discovery process. This result along with results from Experiment 1 shows that reusing the path already obtained during discovery would result in faster data transmission with a minor decrease in the delivery ratio. The delivery ratio on the other hand drastically improves with session maintenance.

- *Experiment 3: Average Data Hops vs. Mobility*

  The average hops traveled by data packets gives an idea of the efficiency of the routing protocol in terms of discovering shorter routes. Our results show that GSR performs best in terms of average data hops both in low mobility as well as high mobility situations with small and large node topologies. GSR-S performs a little worse than GSR. This is again attributed to the fact that some of the data packets are redirected via a different path (these packets are dropped in GSR), resulting in increased

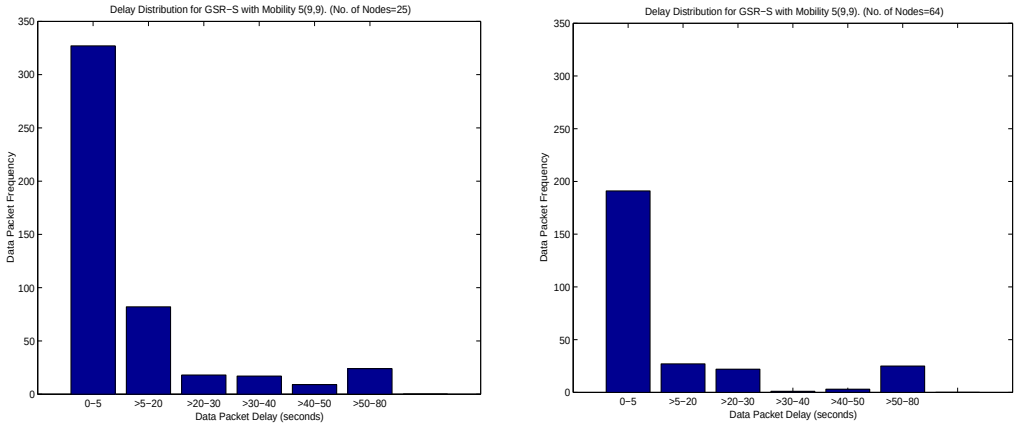

Figure V.7: Delay Distribution for GSR-S with Mobility 5(9,9)

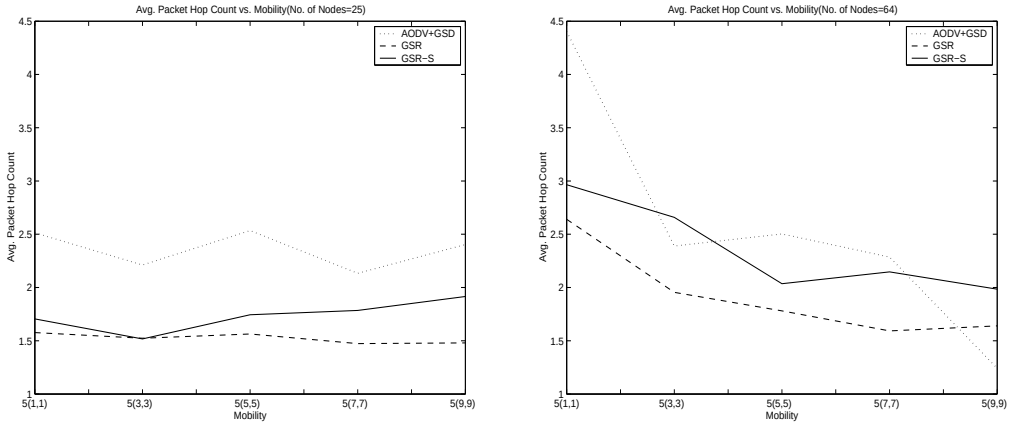

Figure V.8: Average Data Packet Hop Count registered in GSD+AODV, GSR and GSR-S w.r.t varying mobility

hop count. AODV+GSD performs even worse than GSR-S. However, under high mobility conditions (9 meters/second) in a large node topology (64 nodes), AODV performs comparatively better than both the other protocols. We believe this is because high mobility often breaks the reverse routes. AODV, in such situations discovers fresher routes from the RS to the SP. GSR-S, on the other hand employs partial path reconstruction from an intermediate node. In high mobility scenarios, this often results in GSR-S discovering a longer path than what AODV discovers from RS to the SP. The results have been shown in Figure V.8.

- *Experiment 4: Average Request Response Time vs. Mobility*

  Request Response Time refers to the time taken for a service reply to reach the source after a service

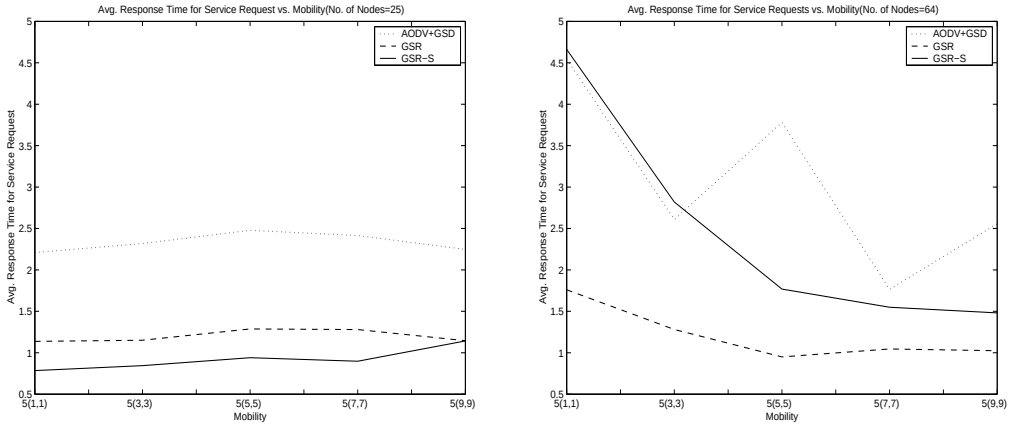

Figure V.9: Average Request Response Time in GSD+AODV, GSR and GSR-S w.r.t varying mobility

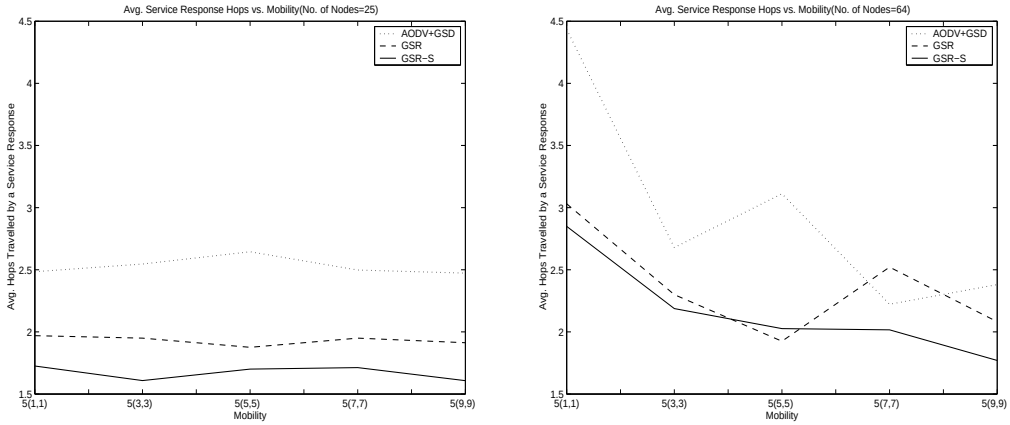

Figure V.10: Average Response Hops in GSD+AODV, GSR and GSR-S w.r.t varying mobility

request had been sent out. This result gives us an estimate of how long AODV takes to transmit a service reply back to the source vis-a-vis our protocol where the reply is reverse-routed back to the source. In Figure V.9, we observer that with less number of nodes (25 nodes), GSR/GSR-S performs significantly better than AODV. We also observe that with 64 nodes, the response time decreases with increasing mobility. This is corroborated in Figure V.10 where we see a decrease in the average response hops (thus resulting in decreased response time) for large nodes under high mobility conditions. However, with small number of nodes (25) there is no significant decrease in the average response time. GSR/GSR-S, as expected is more efficient in utilizing a shorter route than AODV.

V.F Chapter Summary

We present the design of an integrated service discovery and routing protocol (GSR). Our integrated protocol has the capability to do both node-centric as well as service-centric routing. Service Discovery is a key component for the development of distributed applications. In this chapter, we show that an integration of service discovery with routing in ad hoc networks offers greater system efficiency.

Integrating different layers of the protocol stack has well-known shortcomings. The integrated protocol becomes non-modular and difficult to upgrade. Correctness checks also become complicated and code reuse becomes difficult. While the benefits offered by modularity in wired networked systems outweigh the need for integration, in ad hoc networks, integration of the two layers offers greater justification in terms of the benefits obtained.

Chapter VI

BROKER-BASED DISTRIBUTED SERVICE COMPOSITION PROTOCOLS

In this chapter, we describe two distributed protocols for service composition in mobile environments that take into consideration mobility, dynamic changing service topology and device resources. The composition protocols are based on distributed brokerage mechanisms and utilize a distributed service discovery process over ad-hoc network connectivity. We present simulation results of our protocols, and compare them with a centralized service composition protocol traditionally used for wired-infrastructure environments. The results show that our approach clearly outperforms the existing centralized approaches, and that our protocols are able to adapt and better utilize the changing service topology and resources in a mobile environment.

In chapter II, we provide a categorization of research directions followed by the academia and the industry in the area of service composition. Our protocols contribute towards the development of distributed architectures for mobile and pervasive environments. We leverage our prior work [24, 28, 37] and the emerging DARPA Agent Markup Language for Services (DAML-S) [40] standard to provide the semantically rich declarative specification of a composite service.

Current service composition protocols [20, 66, 75] have been designed with the inherent assumption that the services are resident in the wired infrastructure. Thus, services are assumed to be on stable nodes/devices and connected to each other over high bandwidth reliable communication channels. Composition protocols are centralized and consist of a preconfigured *composition manager* or *broker* residing on high-end machines. The *composition manager* performs *service coordination and management* that involves managing issues like service path creation [104], delegation of service discovery to the proper discovery manager, appropriate

combination of different services and management of the information flow between service components. Such protocols do not work well in mobile environments, especially ad-hoc mobile environments. Some of the notable limitations are:

- **Central Point of Failure:** Centralized design approach of wired-infrastructure based composition protocols is prone to single point of failure in a mobile environment.
- **Mobility:** Mobility changes the topology of a mobile environment. This changes the *service topology* (distribution of services on various mobile nodes). Current composition protocols lack support for mobility.
- **Fault Management:** Mobile nodes are prone to various kinds of faults. Faults range from network level disconnection, service discovery failures, and service execution failures to node failures. Composition protocols for mobile environments need to be adaptive and resilient to these failures.

We focus our work on infrastructure-less mobile environments (discussed in chapter II since we believe that (1) Infrastructure-based mobile environments are just a special case of the infrastructure-less environment where some devices are fixed, resource-rich and have infrastructure support. (2) Solutions for this environment can be as easily adapted to infrastructure-based mobile environments. For the remaining of the chapter, we shall use the term *mobile environments (ME)* to mean *infrastructure less mobile environments*.

Our distributed architecture for service discovery and composition introduced in chapter III primarily supports two distributed broker-based protocols for service composition in ME. We leverage our work in service discovery (described in chapter IV) and service-centric routing (described in chapter V) to support discovery and routing underneath our composition protocols. The composition protocols are based on the concept of efficient selection of brokers to handle composite requests. Salient features of our protocols are:

- **Decentralized Control:** Our composition architecture is immune to central point of failure.
- **Mobility-Dependent Dynamic Composition Model:** Our composition protocols utilize the dynamically changing service topology to select the appropriate *atomic services* in a composite request.
- **Efficient Utilization of Mobile Resources:** Mobile resources include mobile services, computation power on the devices, memory and battery life. Our composition protocols distribute the composite requests amongst devices in the mobile environment.

VI.A Design Objectives

We describe issues that were of concern in the design of the distributed protocols for service composition.

- **Distributed Coordination Model:** Service Composition involves the use of a coordinating entity (referred to as broker) that manages the discovery, integration and execution of a composite service. Coordination and management of a composite service in mobile environments (ME) essentially involve discovering the necessary component services, integrating the information obtained from the discovery, invoking the individual services, managing the order of execution and handling faults. Coordination and management need to be primarily adaptable to the topology changes of the mobile environment and handle mobility, network disconnections and other types of node or service failures. However, unlike infrastructure-based composition architecture, ME cannot presuppose the existence of a centralized powerful machine. A centralized composition manager might not be able to access all the necessary services and secondly, a centralized composition manager might become *unreachable* from a device in the ME. Moreover, such an architecture is prone to single point of failure. Hence, composition protocols for ME need a distributed coordination model.

- **Resource Heterogeneity and Awareness:** Devices with varying resources and capabilities exist in the geographical vicinity of one another. A service composition protocol should be able to utilize these varying resources available to it. Resources refer to either services or computation power that are available and vary amongst devices. An efficient composition protocol should ideally be able to compose a service with the most efficient use of the surrounding environment and with minimum network overhead. It should also be able to utilize the spatial distribution of the services (refered to as service topology) and computation resources in the vicinity. In sections VI.C and VI.E, we show how our protocols are able to utilize the service topology for efficient composition.

- **Mobility Management and Adaptability:** Service composition protocols should be able to adapt themselves to the changing neighborhood. In particular, it should be able to utilize newly arrived services that were not available before. A composition protocol should be able to select a possible set of services based on their mobility patterns, platform battery life, how long they would be in the vicinity etc. Mobility changes the underlying service topology. Composition protocols for ME should be able to adapt their execution model to suit the changing service topology.

- **Fault Tolerance and Reliability:** Mobile device often have a short *switched-on* time and often are unable to process remote requests due to system overload or network level disconnections. A robust composition protocol should be tolerant towards such failures and degrade gracefully with increasing service/resource unavailability. Faults may range from network level disconnections, service discovery failures to node failures and service execution failures. The composition architecture should be resilient towards these failures and apply appropriate fault control mechanisms to ensure the processing of the composite request.

- **Efficient Network Utilization:** Network bandwidth is a very important parameter to be considered while designing any composition protocol for ME. By default, service discovery and composition protocols are usually broadcast-based and essentially impose a relatively high network load. An efficient composition protocol should try reducing network wide broadcasts. However, reducing network wide searches may reduce the chances of a service being discovered and hence reduce composition efficiency. Our protocols follows a strategy of greedy localized searches to compose services and tries to conserve network bandwidth.

VI.B System Model and Definitions

Our system consists of mobile nodes each providing one or more services that can be invoked by peer devices. We assume that these devices are connected to each other using ad-hoc networking protocols. Resources like computation power, battery life vary from one device to the other. We do not distinguish between a client and a service provider. Composite requests are sent out by multiple nodes. A node sending out a composite request can itself participate in another composition. Figure III.2 in chapter III depicts the system environment. We define some key terms **ERR:Check if terms have been defined before** that we would be referring to in the remaining part of the chapter:

1. **Request Source (RS):** Mobile device from where a particular composite request originates. Note that a node is referred to as the *Request Source* only with respect to its request.

2. **Service Provider (SP):** Mobile device that contains services that are accessible from other peer nodes.

3. **Composition Manager (CM):** The device that manages the discovery, integration and execution of a composite request. This node can itself be the *Request Source* or a *Service Provider* for another

composite request.

4. **Description-level Service Flow (DSF):** Declarative description of a composite service or a composite request. We use DAML-S for specifying a composite service. Our specification consists of a list of service descriptions along with the desired flow of execution that constitutes the composite service.

5. **Execution-level Service Flow (ESF):** A complete specification of the composite service with execution-level details required to invoke the services in the SPs.

6. **Atomic Service:** A service that resides on a single SP and can be invoked from other devices. This service may consist of further components, but the components must reside on the same SP in order to make the service *atomic*.

7. **Composite Service Length:** Number of distinct atomic services that constitute a composite service.

A composite request consists of a Description-level Service Flow (DSF) of the composite service formulated using DAML-S. DSF of a composite service contains high-level description of the services needed for the composition and the execution flow. Figure VI.1 presents a sampl of a DSF to view a PDF file (by first downloading it and then printing it) in case the client does not have the capability to do so.

VI.C Dynamic Broker Selection based Protocol

The Dynamic Broker Selection-based protocol is primarily based on the concepts of distributing composite requests to composition managers (CM) distributed in the ME and single-CM execution policy per request. The Request Source (RS) of each composite request carries out a *broker arbitration* to select a CM from the ME. Each mobile device is a potential candidate for CM. The *broker arbitration* consists of controlled broadcasting of solicitation requests to devices in the nearby vicinity of the RS. The *broker arbitration* module in each device replies with information regarding its potential to be a CM for the request. The RS elects the best possible CM (explained in subsection VI.C.1) from the available devices and sends the Description-level Service Flow (DSF) to it. The elected CM on receiving the request increments its load level (number of executing composite requests) and starts processing it. The CM performs the following tasks for processing the composite request:

1. It extracts the atomic services from the DSF and uses the underlying service discovery protocol (GSD) to discover the services.

```
<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml= "http://www.daml.org/2001/03/daml+oil#"
  xmlns:process="&DamlProcess;#">

<daml:class rdf:ID="ViewFileComposite">
 <rdfs:subClassOf rdf:resource="&DamlProcess;#CompositeProcess" />
 <rdfs:subClassOf>
   <daml:Restriction>
      <daml:onProperty rdf:resource="&DamlProcess;#composedOf" />
        <daml:oneOf rdf:parseType="daml:collection">
          <daml:Class>
              <daml:intersectionOf rdf:parseType="daml:collection">
                  <daml:Class rdf:about="process:Sequence" />
                  <daml:Restriction>
                    <daml:onProperty rdf:resource="&DamlProcess;#components" />
                        <daml:toClass>
                           <daml:Class>
                             <daml:listOfInstancesOf rdf:parseType="daml:collection">
                                <daml:Class rdf:about="#Filedownloader" />
                                <daml:Class rdf:about="#Printer" />
                             </daml:listOfInstancesOf>
                           </daml:Class>
                        </daml:toClass>
                  </daml:Restriction>
            </daml:intersectionOf>
        </daml:Class>
        <daml:Class>
          ......
</daml:Class>
      </daml:oneOf>
   </daml:Restriction>
 </rdfs:subClassOf>
</daml:class>
</rdf:RDF>
```

Figure VI.1: Description-level Service Flow (DSF) to compose a Printer service and a File Download service

2. It obtains invocation details of the services that has been discovered from GSD and stores them.

3. Once all services have been discovered, the CM constructs an Execution-level Service Flow (ESF) of the composite service.

4. It then invokes the individual services following the execution flow as specified in the ESF and stores the temporary results.

5. On completion of execution of all atomic services, the CM transmits the final result to the RS; it decrements its load and awaits further messages.

6. During the execution phase, the CM (apart from storing the temporary results) periodically transmits checkpoints consisting of the partial results back to the RS. The RS uses these checkpoints to determine faults.

The RS interprets the loss of checkpoints from the desired CM as an indication of fault. Faults can be triggered due to (1) loss of checkpoint messages; (2) failure to discover a desired service (discovery-level fault); (3) failure to execute a service after having discovered it (execution-level fault); (4) CM might have

shut itself down or become unreachable (node failure). Our current prototype signals a fault to the RS in case the above-mentioned events occur. The RS waits for a maximum time limit by which it should have received the final result after which it gives up and restarts the composition. We follow an optimistic strategy to address faults. The basic solution to address faults in composition is for the source to restart the whole process if any service has failed during the execution. This solution is unable to utilize the partial results. The battery life and bandwidth spent in computing the request is also lost. In our solution, RS computes another Description-level Service Flow (DSF) of the composite request by utilizing the partial results and pruning the part that has already been executed. The new DSF is treated as a new composite request in the system. The RS maintains a state of its composite requests and merges the results after the new DSF has been executed.

VI.C.1 Broker Arbitration

We control the broadcast of solicitation messages by specifying the maximum number of hops for a message. This maximum hop count is increased if an appropriate CM is not found. The solicitation message consists of an enumerated list of atomic services that the composite request needs for its execution. Each device computes a *potential value* for itself that determines its suitability to be the CM for the request and returns it back to the RS. The potential value of a device is calculated by considering its local resources and by estimating the service *richness* of its neighborhood. Service richness is judged by considering the number of remote service advertisements cached by the device. Local resources considered are (1) number of matching local atomic services; (2) battery life; (3) current processing load on the node. Some attributes affect the potential value positively while some affect it negatively (e.g. current processing load). Formally, we denote U(CM_i) as the potential value of each CM (CM_i) for a composite request S.

$$U(CM_i) = f(S_c(CM_i), S_m(CM_i), L(CM_i), J(CM_i))$$

where $S_c(CM_i)$ is number of service advertisements cached by CM_i; $S_m(CM_i)$ is number of services belonging to the composite request that are present in the cache of CM_i; $L(CM_i)$ is battery life of CM_i; $J(CM_i)$ is current number of requests being processed by CM_i. A Composition Manager, CM_i is selected based on the following equation:

$$\exists B_i \text{ such that } \forall B_j \ (U(B_i) \geq U(B_j))$$

We note that for an infrastructure-based mobile environment where the infrastructure may contain a centralized service manager on a server, the arbitration can be adapted by appropriately weighing the various parameters so that it selects the manager as long as it is *reachable* from the RS. This flexibility of our protocol makes it well suited for heterogeneous mobile environments. Figure VI.2 shows the flow diagram of this phase.

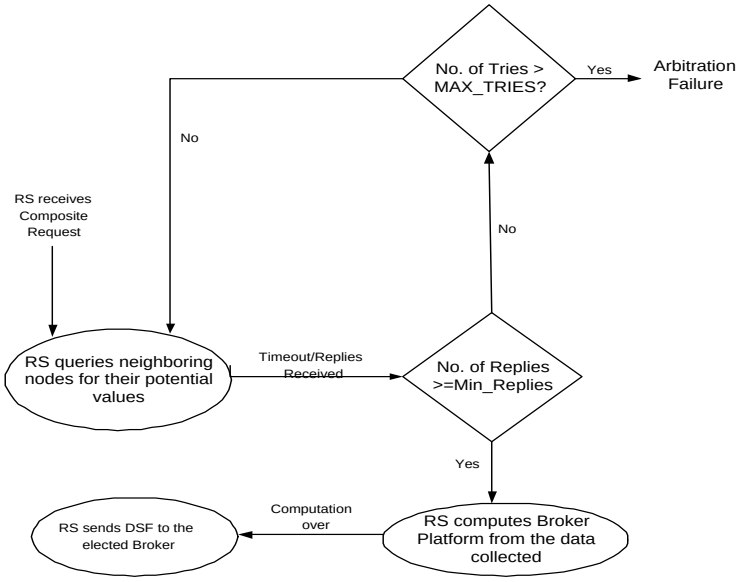

Figure VI.2: Flow Diagram of the *Broker Arbitration* phase in the Dynamic Broker Selection based Protocol

VI.C.2 Service Discovery and Integration

Service Discovery and Integration aids in generating an ESF from the given DSF of the composite request using the underlying GSD discovery protocol. Corresponding to each service description in the DSF, an actual atomic service is discovered. The network load during the discovery process is controlled by regulating the number of hops within which the service discovery is performed. There could be multiple instances of the same service existing in the environment. Our protocol currently selects the nearest available service. However, it can easily be extended to incorporate additional cost factors. An ESF is constructed to contain information on the actual services, its node binding and invocation details. It also contains the service flow (obtained from the DSF). This phase ends when all the required services have been discovered and a new ESF has been formed. Figure VI.3 describes the pseudo code of the operation of the protocol in this phase.

```
For each service Si in DSF {
   broadCast_Diameter=MIN_DIAMETER;
   service_discovered=FALSE;
   no_retries=0;
   while(!service_discovered && no_retries<=MAX_RETRIES) {
    Call GSD to discover Si;
    if Si has been discovered{
      ESF+=Invocation Details of S_i;
      service_discovered=TRUE;
      }
    else {
      broadCast_Diameter+=BROADCAST_INCREMENT;
      no_retries++;
      }
   }
}
```

Figure VI.3: Pseudo code of Service Discovery and Integration Phase

VI.C.3 Request Execution

The CM coordinates the execution of the services in the order specified by the ESF. The Broker uses the underlying GSR routing protocol [36] to transmit results received from one service to another during the execution. The partial results flow from one device to another through the CM. We call it *star-based* execution pattern. In future, we aim to incorporate *mesh-based* execution pattern where the result would directly flow from one service to the next service. Our checkpoint-based source monitored fault tolerance scheme imposes additional overhead of transmitting checkpoints in the network. We are currently analyzing the effect of our check pointing algorithm on the network bandwidth. We assume that the source node would not fail before successful completion of the composition. This is a reasonable assumption, since the source node would ideally want to keep itself turned on till it receives a final reply.

VI.D Limitations of Dynamic Broker Selection based Protocol

The protocol suffers from the following limitations:

- **Single-CM execution policy per request**: Single-CM execution policy per request assumes that a single CM is responsible to manage the entire composition. In ME, mobility often changes the service topology. This, along with other limitations like disconnections and network partition might make it impossible for the current CM to compose the whole request after having completed a partial execution. Ideally, the system should be resilient towards such failures.

- **Load on RS and CM:** The RS monitors for faults using checkpoints through out the composition process. Environments where the CM often fails to completely execute the service or environments

having high number of composite requests impose additional load on the individual RS. The CM has to maintain state information of the entire composite service and monitor the discovery and execution of the composite service. This overhead increases with the increase in the *composite service length*. Moreover, lengthy composite services are more prone to service topology changes and hence more prone to discovery and execution-level faults. This further results in an increase on the load of the RS as well (since it has to monitor more faults now).

VI.E Distributed Broker Selection based Protocol

Our Distributed Broker Selection based protocol relaxes the single-CM execution policy per request. The central idea behind this protocol is to distribute the task of composing a service to multiple CMs on an as-needed basis. A single composite request is initially assigned to a particular CM by the Requesting Source (RS). The assigned CM executes the request until it encounters a discovery or execution-level fault. In case of a fault, instead of directly notifying the RS, the CM enters a new phase called *arbitration control* phase. In this phase, the CM carries out the following steps:

1. It freezes the current process (*composition state freeze*) and extracts the un-executed part of the composite request and constructs a DSF from it.
2. It carries out a new *broker arbitration* to determine the best possible CM for the pruned request.
3. On successful selection of a new CM, the un-executed DSF together with the partial result of the composition is transferred to it (*state transfer*); it sends a notification to the RS with the address of the new CM.
4. The new CM resumes the usual composition process with the un-executed DSF.
5. The old CM sends a fault event to the RS only if it fails to either select an appropriate CM or fails to transfer the composition to the new CM. Figure VI.4 shows the state diagram of the *arbitration control* phase.

This protocol reduces the burden on the RS since the composition transfer is transparent to the RS. In the previous protocol, the RS would have to handle such faults. After the composition transfer, the RS starts receiving checkpoints from a different CM for the same request and figures out that the CM has changed. The new CM could communicate with RS via the old CM or use a different path as determined by the underlying

GSR routing protocol. We check for duplicate or spurious checkpoints to ensure consistency of information at the RS. Transparent composition transfer from one CM to another in the ME is inspired by the following observations:

- Service topology change: Discovery or execution level faults are mostly triggered by service topology change due to mobility. However, except for special cases, it is probable that the required services have not moved too far away from the CM. Thus, even though the CM might not be able to compose the remaining services, some CM in its vicinity might have a better chance of composing the remaining part of the request.
- Network bandwidth conservation: The old CM may find another CM more easily than the original RS. This is because, if the topology has not changed drastically, then chances of services being around in the vicinity are higher.

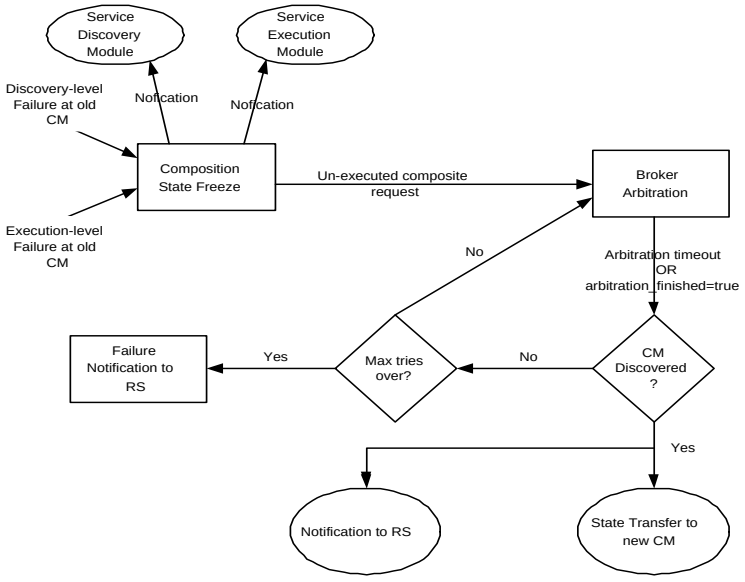

Figure VI.4: State Diagram of the *Arbitration Control* phase in the Distributed Broker Selection based Composition

VI.E.1 Broker Arbitration

We highlight some differences that are possible in the *broker arbitration* in the Distributed Broker Selection based protocol from the previous protocol. Suppose $\bar{S}$ denotes the DSF of the composite service. Our

distributed Broker Selection based protocol can exercise greedy alternatives for CM selection. As an example, the potential values of CMs can be calculated by giving more importance to services needed during the initial stages of the composition as opposed to services needed at a later stage. We recall from section VI.C that $S_m(CM_i)$ is number of atomic services belonging to the composite service that are present in the cache of Broker B_i. Unlike in the previous protocol, we envision that the new value of $S_m(CM_i)$=$\sum V_{S_i}$ where V_{S_i} is height of the node S_i in the task graph[10] of $\overline{S}$. The new potential value provides more importance towards the discovery and execution of services needed *immediately* as opposed to the services needed at a later stage of the composition. This is more appropriate for lengthy composite services. This would reduce the network load created due to the broadcast of control messages during this phase.

VI.E.2 Arbitration Control

Service discovery and integration in this protocol is same as in the Dynamic Broker Selection based protocol. The service execution triggers the *arbitration control* after encountering a discovery-level or an execution-level fault. The maximum number of delegations of a composite request is regulated by the *distribution limit* parameter. The current CM sends a failure notification to the RS if the distribution limit is reached.

Theoretically, the *arbitration control* can be triggered immediately after the execution of each atomic service. However, this generates unnecessary network traffic as the current CM (until it encounters a fault) is capable of executing the next atomic service in the ESF. We trigger the *arbitration control* only when the current CM encounters a discovery or execution-level fault.

- **Composition State Freeze:** The CM monitors the *composition state* of each request it receives. The *composition state* consists of information on the discovery state of each service in the request, execution state of each service, partial results and additional performance parameters like average hop distance of the services to the CM. On a state freeze, discovery and execution requests that had been issued for other services needed in the composition are invalidated. Current CM then constructs a discovery-level service flow (DSF) of the unexecuted part of the composite service.

- **State Transfer and Source Notification:** The CM transfers partial results of the executed part of the composite request as well as a DSF of the unexecuted part to the next CM. The CM notifies the source of the request with the address of the new CM. It invalidates the state that it had maintained for the composite service. The new CM on receiving the composite service creates a composition state by

combining the partial results and the un-executed DSF. It then continues composing the un-executed DSF.

The Distributed Broker Selection based Composition protocol offers us the following advantages over the Dynamic Broker Selection protocol. (1) **Better adaptability:** Our new protocol distributes the task of a single request over multiple CMs if needed. Change in the service topology may make a new CM more suitable to compose a request after the composition has started. Our protocol utilizes this and transfers the job to an appropriate CM.

(2) **Load reduction on CM for lengthy composite services:** A single CM does not need to compose the entire service. This imposes lesser load on a single CM since the load gets distributed amongst the participating CMs. (3) **Fault reduction for lengthy composite services:** The RS potentially receives fewer faults since the request transfer between CMs take place transparent to the RS. If the CM transfers the request, it has potentially discovered a better CM for the request. This design reduces the number of discovery or execution-level faults in the system.

VI.F Experiments

We implemented our protocols as part of the distributed event-based mobile network simulator Glomosim [114] (version 2.0). We present results comparing our protocols (Dynamic Broker-based Composition and Distributed Broker-based Composition) with each other using various service densities, node mobility and composite service lengths. We also compared our results with a broker-based centralized service composition protocol mostly used for wired networks or infrastructure-based mobile environments. In this protocol, all the composite requests are sent to a fixed composition broker or engine. We call it *Fixed Broker based composition*. For the purpose of our simulation, we placed this broker centrally in the network topology at the beginning of the simulation. We briefly describe the experimental model of the ME that we considered for our simulations.

VI.F.1 Experimental Model

Our ME model consists of mobile service providers (SPs) connected to each other using an ad-hoc network (190m x 190m area). Each SP can become a CM for a request. The RS is also one of these devices. The devices assume an initial grid-based topology before becoming mobile. We studied the behavior of our

protocols under various conditions of service topologies, service densities and node mobility. *Service density* in our model is defined as the percentage of SPs containing one or more of the atomic services required in a composite request. As an illustration, if a composite service consists of services belonging to the set [S1,S2,S3], then a service density of 60% would mean that 60% of the nodes in the system have either service S1 or S2 or S3 or a combination of them. Figure VI.1 enumerates the specific parameters of the ME experimental model that we used in our simulations.

Duration	3600 seconds (1 hour)
Space (x,y)	(190 x 190m)
Tx Range (Transmission Range)	30m
Tx Throughput	20kbps
No. of Nodes	64
Advertisement Interval	10 seconds
Advertisement Timeout	5 seconds
broadcast jitter	10 milliseconds
Mobility	Random way-point with 3 m/s speed and 5 s stoppage time
Initial topology	grid topology with nodes equally spaced out in (x,y)
Service density	20% to 100%
Composite service length	3,5,7
Broker Arbitration MAX_HOP_COUNT	1
MAX_RETRIES to discover an atomic service	3
MAX_CM_DELEGATIONS (*distribution limit*)	7

Table VI.1: ME experimental model parameters for distributed service composition protocols

VI.F.2 Evaluation Metrics

We define the following additional evaluation metrics for our simulations.

1. **Composition Efficiency:** The percentage of composite requests that were successfully composed (discovered, integrated and executed) by the system.

2. **Broker Arbitration Efficiency:** The fraction of composite requests for which a CM was assigned and the task was received by the assigned CM. Due to disconnections and network topology change in ME, a CM might fail to receive a task from the RS. This metric measures the efficiency of the *Broker Arbitration Phase* in deciding and actually successfully delivering the task to the CM.

3. **Broker Instantiation Time (BIT):** The amount of time taken by a source to decide on a CM and delegate the task to it. It includes the overhead for carrying out the *broker arbitration* and transfering the task to the CM.

4. **Composition Radius (CR):** The average number of hops needed by a CM (or a group of CMs) in order to discover the SPs and finally execute a composite service. A low value of composition radius means that most services were discovered in the nearby vicinity. This further implies that our *broker arbitration* selects the CM appropriately with respect to the service topology.

5. **Distribution Index (DI):** This parameter defined only in the context of Distributed Broker-based Composition. It is defined as the number of composite request delegations required to completely compose a service. It is restricted by the *distribution limit* of the system.

VI.F.3 Results

The experiments being reported correspond to a set of 100 composite requests for 3 different seeds of random way-point mobility pattern. Figure VI.5 shows the effect of service density on the Composition Efficiency. We observe that our distributed composition protocols clearly outperform the centralized service composition protocol as we expected based on our arguments in VI.A and the adaptability of our protocol. We also see that composition efficiency increases with increasing service density. This is because required services are more easily available to the protocol. Moreover, our protocols utilize the service topology to perform the composition efficiently.

We observe that composition efficiency of Centralized Service Composition decreases drastically with increase in the composite service length (observe the absolute values of the efficiencies in the 3 graphs in figure VI.5). Our protocols do not have appreciable decrease in the efficiency. We also observe that in general, the Distributed Broker Selection based Composition performs better than Dynamic Broker-based Composition in terms of composition efficiency. This is mostly because the former protocol does not have the single-CM execution policy and hence adapts itself better to the changing service topology. We also note that sometimes, due to the changing environment, the *broker arbitration* might not be successful in finding an appropriate CM (as indicated by the slight dip in the graph for composite service length=3 in figure VI.5).

This is because, the environment beside the CM might change by the time the task assignment has reached it.

Figure VI.6 shows the comparison of Broker Arbitration Efficiency between the three protocols. We observe that Arbitration Efficiency is in general much higher in our protocols. This is mostly because, the CMs are selected from nearby nodes in our protocols. Thus chances of a composite request not reaching its assigned CM are low. This contributes to the increase in the Arbitration Efficiency. In Centralized Service Composition, each composite request has to reach a fixed composition engine that is potentially multiple hops away. Hence chances of message losses are more. In terms of absolute measures, the efficiency observed in our protocols is well above 0.9. We also observe that generally the efficiency of Dynamic Broker Selection based Composition is fractionally higher than Distributed Broker Selection based Composition. We believe this is due to the network load generated due to greater number of *broker arbitrations* in the Distributed Broker Selection based Composition. We note that the broker arbitration efficiency does not show any particular relation to the service density in centralized service composition. This is expected since the service density does not affect the probability of a composite request reaching the selected CM.

Figure VI.7 shows how the Broker Instantiation Time (BIT) of different protocols compare with each other. We observe that the value of BIT is very high for Centralized Service Composition. This is attributed to the high end-to-end delivery delay across multiple hops in a ME. On the contrary, we observe that the BIT of our composition protocols is significantly lower. In our protocols, the RS selects CMs that are in the nearby vicinity and hence have much shorter routes. This signifies the need for composition protocols in ME to be able to utilize resources in the surrounding vicinity. We also see that the BIT does not show any definite relationship with respect to the increasing service density. This is because the service density only helps in deciding a CM. Since we have uniform service density, it does not have any implication on how far the CM is from the RS.

Figure VI.8 shows the comparison of the Composition Radius (CR) for the Dynamic Broker Selection based and Distributed Broker Selection based composition protocols. The CR observed during our experiments is in general lower for Distributed Broker Selection based Composition (except for composite service length of 7 where it is mostly higher). This is mostly due to our Brokerage Delegation mechanism. A CM transfers its brokerage to another (potentially better placed) CM on an unsuccessful discovery attempt. It does this instead of increasing its search span and trying again. This results in a lower CR for the Distributed Broker Selection based Protocol. We also observe contradictions where we see that the CR is fractionally higher for Distributed Broker Selection based Composition (composition length=7). We believe, it may be

due to “inefficient” decisions made by our Broker Arbitration Phase or due to the dynamic change in service topology. The service topology might have changed in an unfavorable way in the time gap between during the arbitration period and the actual delivery of the composite request to the CM. We believe that the composition radius for our distributed broker selection based protocol is higher for composite service length of 7 because the broker arbitration efficiency (figure VI.6) is lower. Thus many CMs actually increase their broadcast limit to compose required services.

We observe that the average composition radius decreases with increasing service density. This is because with increasing service density, hop count to discover services decrease. However, we observe an increase in the radius for Dynamic Broker Selection based protocol (by 0.1) with composition length of 3. This is due to a higher number of data points and some outliers in the data set for the service density of 40. Figure VI.9 shows the data distribution of the actual values, means of which have been plotted in figure VI.8. We observe that the median values (center lines at the boxes) and the range of data are very similar for densities of 20 and 40 % and decreases with further increase in the service density. This supports our claim that our protocol adapts itself with increasing service density to reduce the amount of network load due to service discovery.

We observe in figure VI.10 that the number of services that are discovered locally by the underlying GSD service discovery protocol is consistently high for Distributed Broker-based Composition. Thus the discovery protocol imposes lesser load on the network since more services can be discovered locally. This result shows that our Distributed Broker Selection based Composition achieves better utilization of the service topology. The Centralized Service Composition imposes maximum load on the network since most of its services are not available locally.

We measured the effect of service density on the average number of composite request delegations (viz. Distribution Index) for our Distributed Broker Selection based Composition. The results are plotted in figure VI.11. We observe that in a less dense service topology, the CM has to do a large number of request delegations. However, as the density increases, the number of such delegations decrease since more services are available near a CM. We also observe that as expected, the distribution index increases with increase in the composition length.

VI.G Chapter Summary

We presented distributed Service Composition protocols for mobile environments in this chapter. Our protocols are decentralized and utilize the service topology to compose services. Each composite request is independently assigned a Composition Manager (CM). The *Broker Arbitration* mechanism uses a controlled broadcast-based scheme for soliciting information from nearby nodes. Selection of a CM depends on a device-specific potential value and takes into account services present in the device, computation and energy resources and most importantly service topology of the surrounding vicinity. Service composition is carried out in a distributed manner utilizing the resources/services surrounding the assigned CM. However, our Dynamic Broker Selection based protocol is based on the concept of single Composition Manager per request policy. Our Distributed Broker Selection based protocol uses multiple CMs to execute a request. Both protocols are immune to central point of failure and do not require infrastructure support.

We compare our protocols to a centralized solution (used in wired and WiFi/802.11 environments) where requests are sent to a central broker in the system. Simulation results show that our protocols perform better in terms of composition efficiency and broker arbitration efficiency. Our protocols achieves better results in terms of composition radius and utilizing service topology. We also show that judicious placement strategy of the CMs in our protocols impose lesser load on the underlying service discovery and network layers. In general, the Distributed Broker Selection based protocol performs better than Dynamic Broker Selection based protocol in terms of composition efficiency and composition radius.

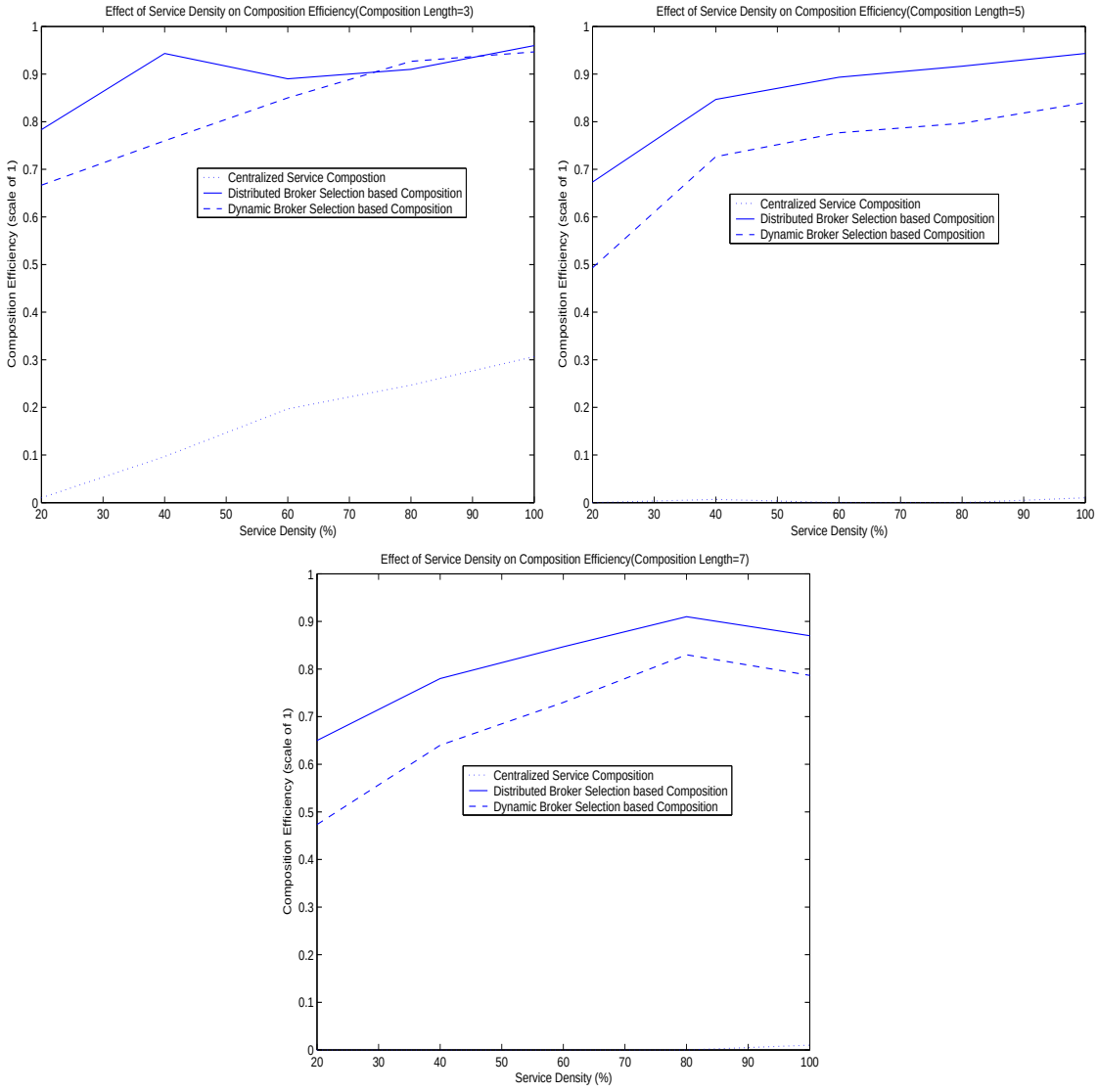

Figure VI.5: Comparison of Composition Efficiency of the different protocols

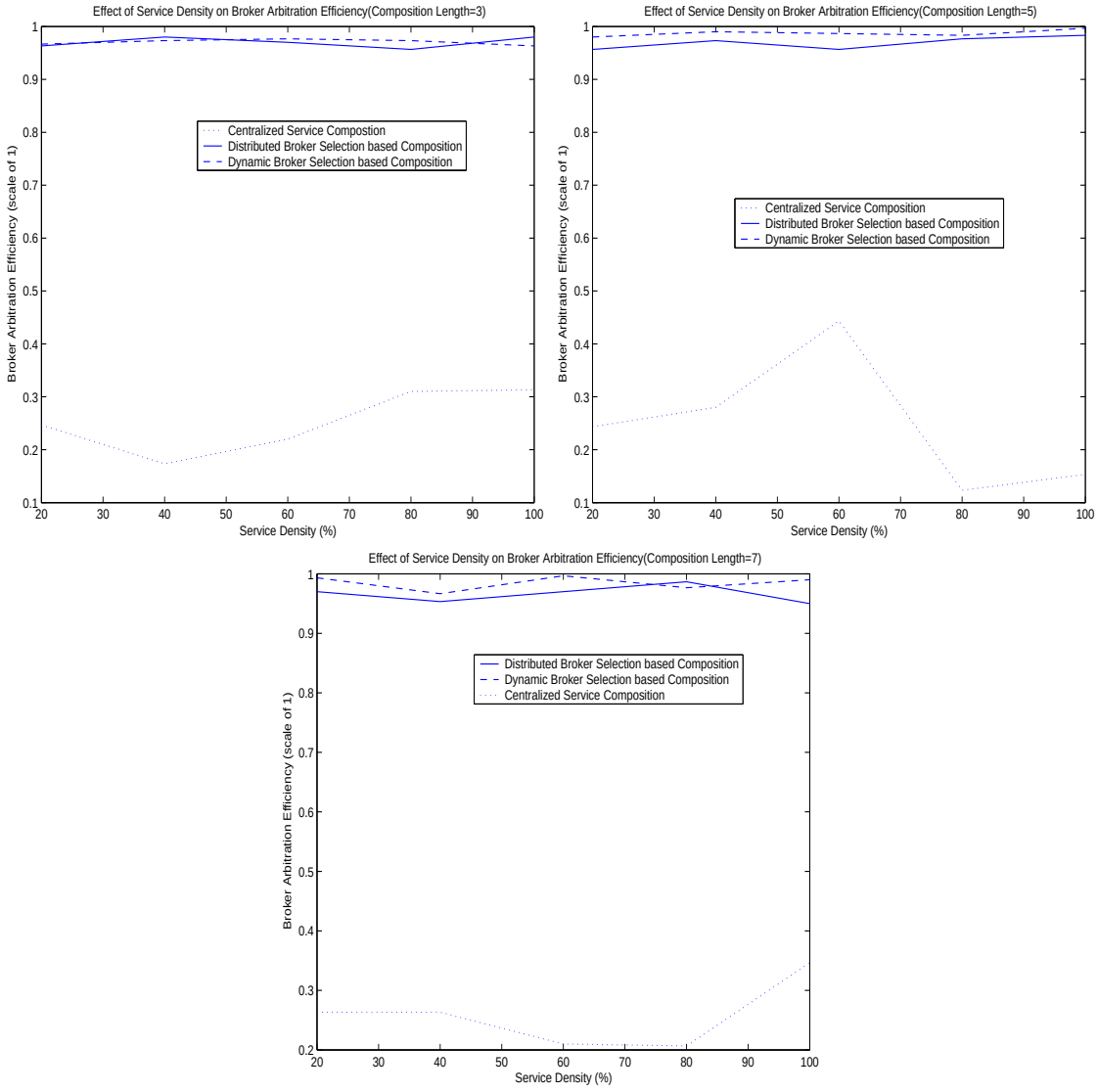

Figure VI.6: Comparison of Broker Arbitration Efficiency of the different protocols

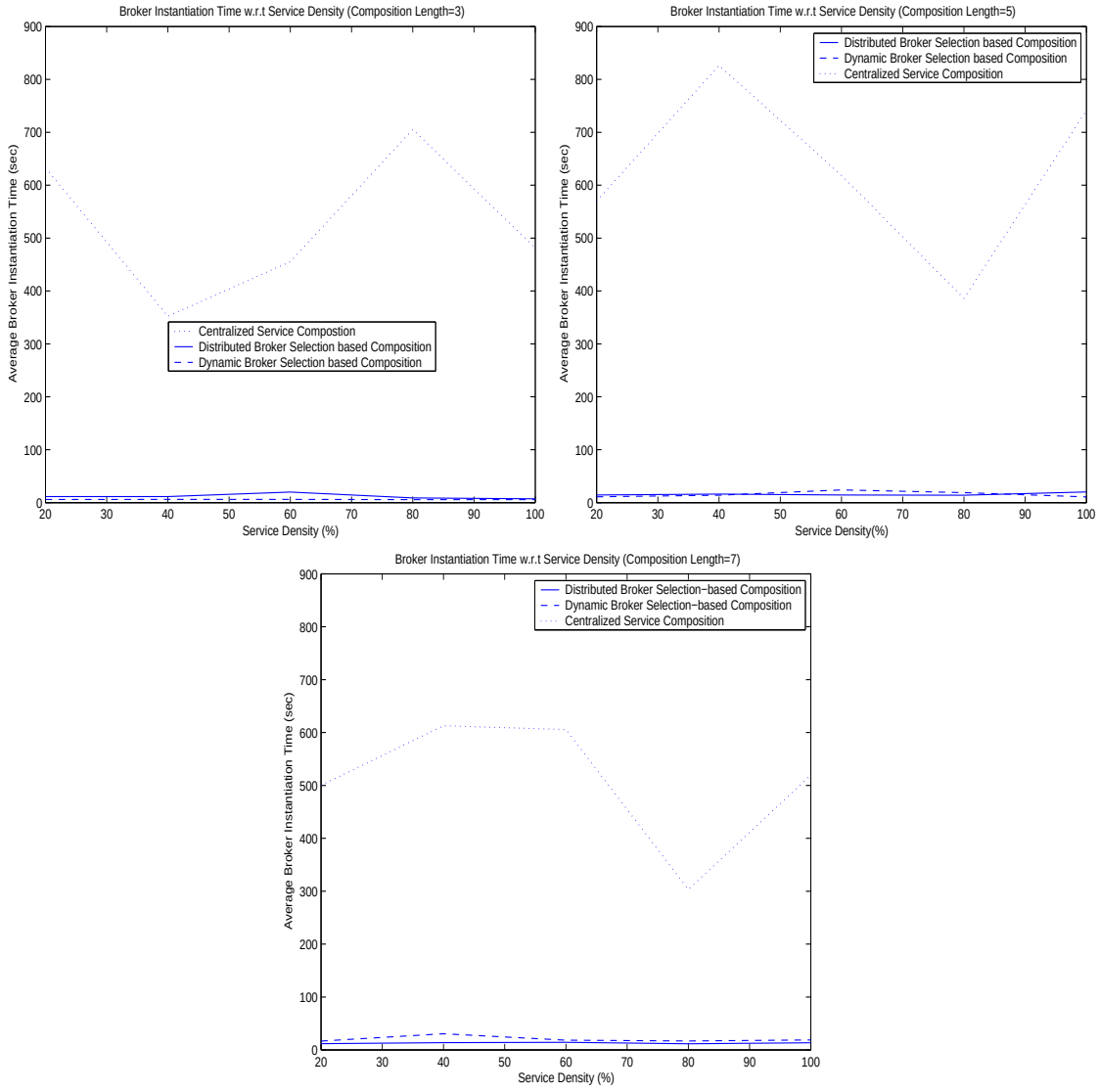

Figure VI.7: Broker Instantiation Time of the different protocols

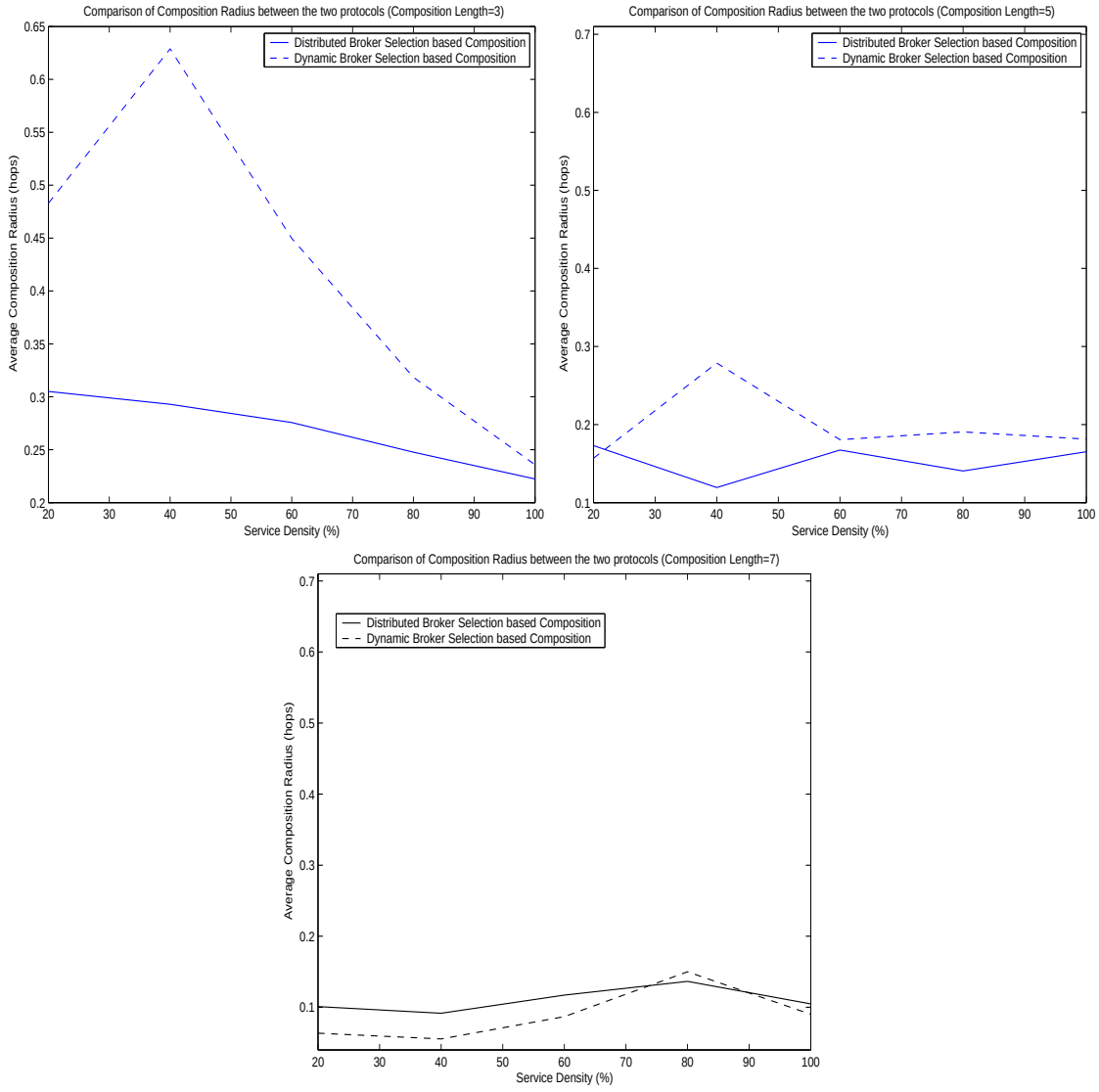

Figure VI.8: Comparison of Composition Radius of the different protocols

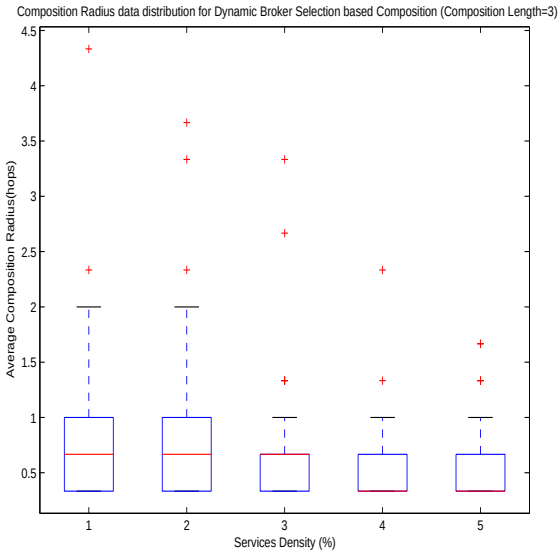

Figure VI.9: Data Distribution of composition radius to explain the distribution of mean composition radius for Dynamic Broker Selection based composition for composition length=3

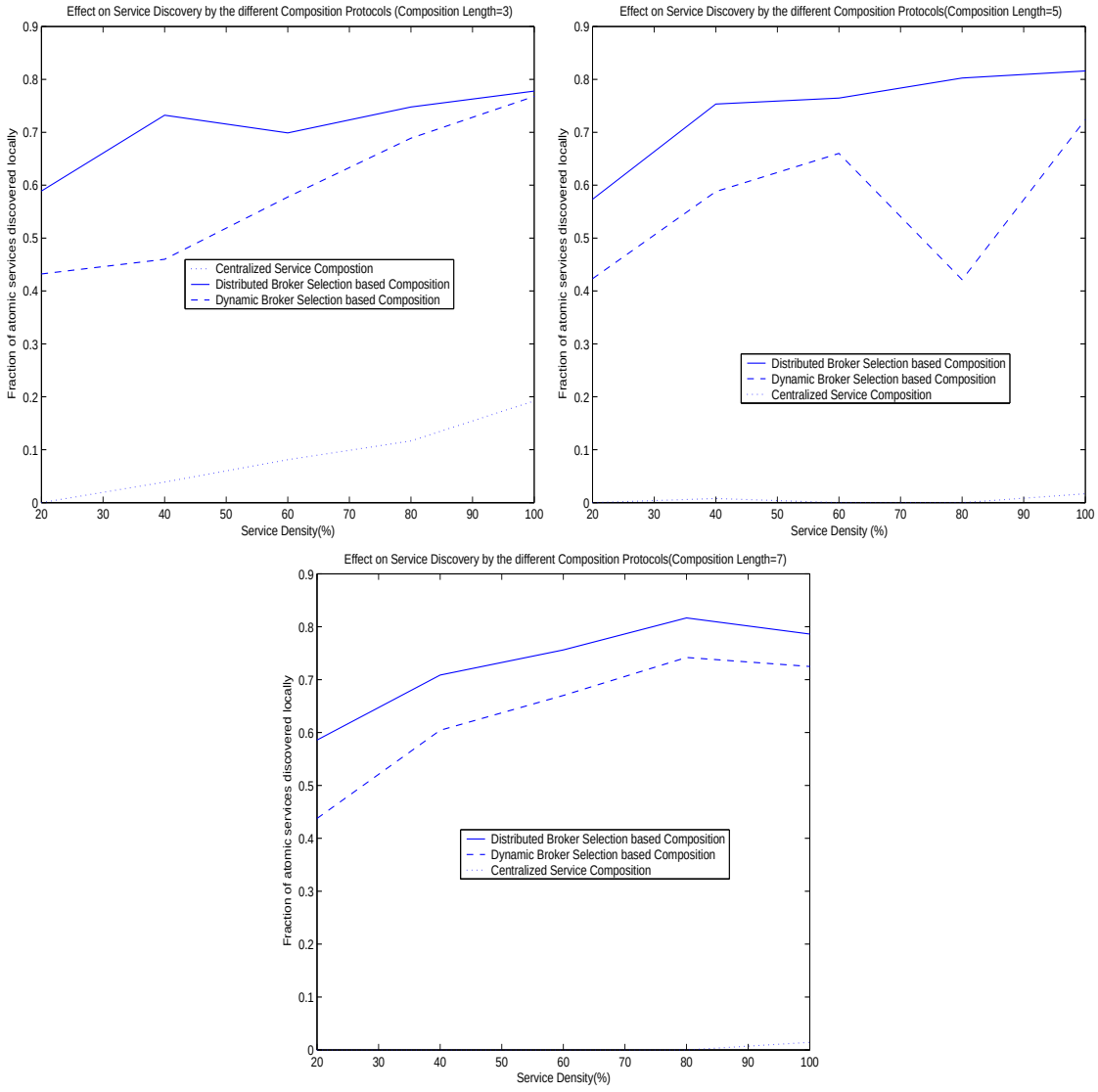

Figure VI.10: Efficiency of our protocols in terms of locally discovered services

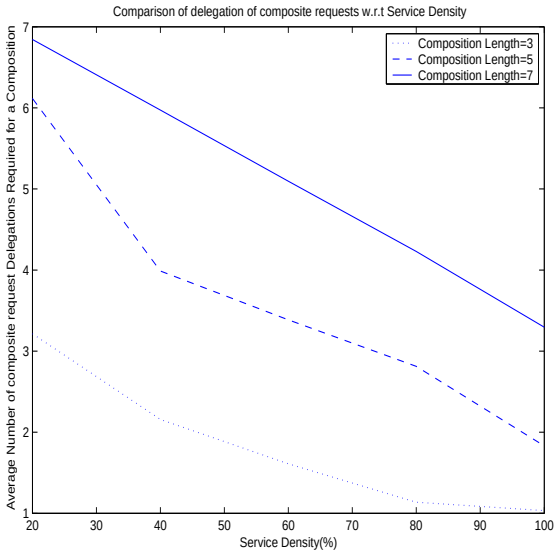

Figure VI.11: Effect of Service Density on the number of composite request Delegations for Distributed Broker-based Composition

Chapter VII

ANAMIKA: EXPERIENCES WITH A PROTOTYPE IMPLEMENTATION

This chapter describes our experiences in implementing parts of our architecture for service discovery and composition outlined in chapter III. We describe our experiences in implementing a version of our architecture on Bluetooth piconets using laptops. We implemented the distributed service discovery protocol in chapter IV and the Dynamic Broker Selection-based protocol in chapter VI. We leveraged support of semantic web tools and languages whenever possible. In particular, we used a DAML-based ontology (illustrated in chapter III) to describe services and DAML-S to describe composite requests. We also describe our efforts in creating various types of reasoners to analyze and match service descriptions with requests.

VII.A Implementation

In our initial implementation of the design architecture, we developed a reactive service composition system called Anamika. The individual components of the Anamika system existing in participating mobile devices are described in Figure VII.1. Current prototype of the architecture is implemented over Bluetooth [99]. Composition knowledge is described using DAML-S[40] in terms of subset of individual services that might be able to satisfy a composite request. Service discovery is done in a peer-to-peer manner using semantic description of services using DAML-S and our light-weight reasoning engine present on participating devices.

Anamika implements the Dynamic Broker Selection based protocol detailed in chapter VI. It decides the best device to carry out the composition based on a combination of the processor power of the platform and

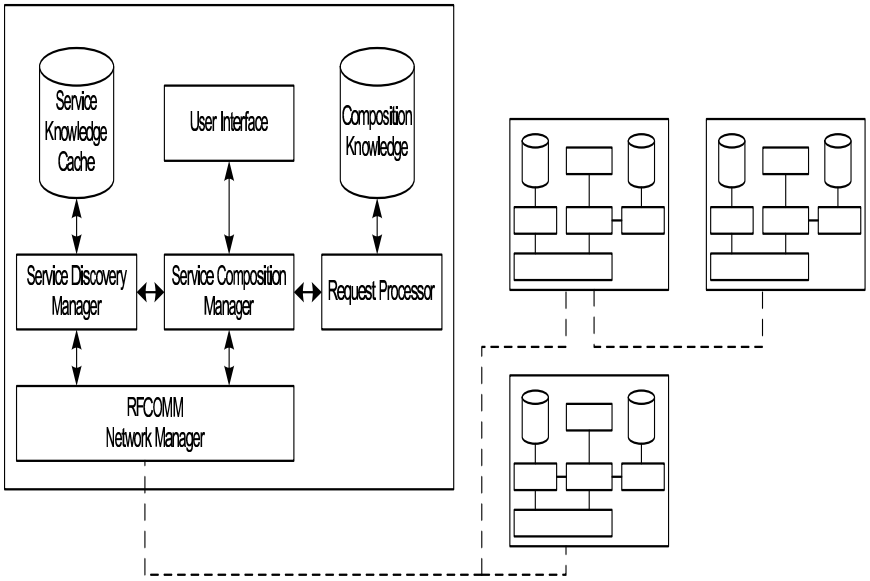

Figure VII.1: System Components in Anamika Reactive Service Composition Architecture

number of services that the platform has. We explain the details of the different components of the system in the following sections.

VII.A.1 Network Manager

The Network Manager implements the API required for higher layers to reliably communicate to neighboring peers over Bluetooth. An efficient implementation of the design architecture requires the network layer to have support for broadcasting (required for GSD advertisements). However, broadcasting in Bluetooth is restricted to messages used for device discovery. In order to exchange application level messages, a device must first establish a link-level connection with its peers. In addition, every communicating device must either be a *master* or *slave* thus making multiple simultaneous links between two devices impossible. Hence, we have used a *connect-transmit-disconnect* mode of sending Anamika messages between peers. We implemented the networking level API over RFCOMM [99] protocol. The Network Manager was implemented using IBM's BlueDrekar transport driver [1] and protocol stack for Linux kernel 2.4.2-2.

Connect-transmit-Disconnect Operation: Each device uses the well-defined APIs of the Bluedrekar protocol stack to initially discover a set of devices using *Device Discovery* [1, 99]. Once a device is discovered, an L2CAP connection is initially opened and an RFCOM connection is opened on top of L2CAP. The two devices between which an RFCOM connection has been opened thus share a connection for that span of

time. The RFCOM and L2CAP connections are terminated after the required data has been transferred. Each device thus follows a stateless but reliable mechanism to send multiple data segments from higher layers. During broadcasting of messages, this operation is repeated for all the neighboring devices discovered using Device Discovery.

We do segmentation and reassembly to limit the transmitted packets size to 64 bytes to prevent overflowing the receiver buffer in a Bluetooth peer. The Network Manager provides APIs for clients to connect to other peer devices in the vicinity as well as APIs for devices to listen for incoming messages and send acknowledgments or replies back to the sender.

VII.A.2 Service Discovery Manager

Service Discovery Manager provides the functionality to local Composition Manager to discover services in Bluetooth peers. Services are described using the ontology described in chapter III. Services are primarily described based on their inputs, outputs, functionality classification, functional similarity to other services and invocation mechanisms. The service description also incorporates platform specific information like processor type, speed etc.

Our design of the Service Discovery Manager is extracted from the GSD protocol and is based on the following principles:

1. *Peer-to-peer service discovery:* In an ad-hoc environment, services are discovered by directly contacting the device where the services are actually residing. We do not employ central lookup-server based service discovery and maintenance. Each device has a Service Manager where the local services register their information. Service Managers receive requests from remote Service Managers and send back replies based on a match.

2. *Dynamic caching of neighboring service descriptions:* Service Managers maintain a cache of descriptions of services that they have discovered in the course of their operation. Services are purged from the cache by using a combination of least-recently-used and incremental aging policies. Service descriptions of discovered services are cached on every device in this environment, which increases the overall efficiency of service discovery.

3. *Semantic description-based service matching:* The service discovery mechanism has support for semantic service matching using descriptions of services in DAML[70, 71]. We have subsequently up-

graded the ontology to be expressed in Web Ontology Language (OWL) [41]. The service discovery mechanism is able to reason with the description of services and provide high flexibility in the matching mechanism. Apart from this, the service discovery protocol also incorporates Interface-based, attribute-based and unique identifier based service matching capabilities. These service discovery mechanisms combined together increases the efficiency of the service discovery layer to discover heterogeneous services in the environment. The service discovery mechanism also provides a means and knowledge for a client to invoke the discovered service.

4. *Service Request Routing and propagation control:* Service Managers have the ability to forward service requests to neighbors and handle duplicate service requests. Service Managers also control the number of devices that are not in its radio range to which it sends service requests. This prevents flooding the network with service requests.

A request for service discovery also follows the same ontology and the discovery mechanism uses light-weight semantic matching of service descriptions to discover matching or 'nearly' matching services. The service discovery manager primarily implements the GSD protocol described in chapter IV. Each Discovery Manager consists of a light-weight reasoner. We have modified the light-weight reasoner in the DReggie system [24] previously developed by us to reason with the service descriptions. The discovery manager on receiving a request matches it with cached descriptions of services and returns description of the service discovered, invocation information and other platform specific details (like processor power, memory availability) to the requesting device. Figure VII.A.2 shows an example of the information mantained by the manager for a printer service.

The local Service Discovery Manager maintains a cache of these descriptions to increase the efficiency of future requests that are looking for the same service. The cache is purged by using a combination of least-recently-used and aging policies. The current prototype of the Service Discovery Manager does not do service request routing, thus limiting the propagation of service requests to one hop. However, caching of service descriptions provides capability to discover services which are 2 hops away.

VII.A.3 Service Composition Manager

The Service Composition Manager is the principal component that is responsible for service composition and management. Peer Service Composition Managers collaborate with each other to implement different

```
<?xml version="1.0" ?>
<rdf:RDF
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
xmlns:daml ="http://daml.org/2001/03/daml+oil#"
xmlns ="http://daml.umbc.edu/ontologies/dreggie-ont#"
>
<Component>
  <Description>
     <ServiceName>Filedownloader</ServiceName>
     <ServiceAlias>Internet</ServiceAlias>
     <ServiceAlias>WLAN</ServiceAlias>
     <Input>
        <ServiceInputType>Loginname</ServiceInputType>
     </Input>
     <Input>
       <ServiceInputType>Password</ServiceInputType>
      </Input>
     <Output>
        <ServiceOutputType>File</ServiceOutputType>
     </Output>
   </Description>
  <OtherInfo>/root/Anamika/source/services/Filedownloader-cache.daml</OtherInfo>
  <InvokeCommand>/root/Anamika/source/serviceExecs/Filedownloader/downloader</InvokeCommand>
 <Speed>
    <AmountUnit>
      <Amount>800</Amount>
      <Unit>Mhz</Unit>
    </AmountUnit>
   </Speed>
</Component>
</rdf:RDF>
```

techniques of service composition. The current implementation of the Composition Manager supports the Dynamic Broker Selection based protocol (refer to chapter VI) for service composition. The Composition Manager receives a request from the Application Layer. It uses the Request Processor module to parse a composite service description. The Request Processor module consults the Composition Knowledge base to find out suitable sets of services that might be combined to provide an answer to the query. Request Processor returns multiple process models to carry out a particular composite service. The Composition Knowledge base has been modeled in DAML-S. Figure VII.A.3 gives an example of the decomposition of a generic query to view a certain URI or a file (in case the client machine does not have the capability to do so) by using the resources of the vicinity:

When a request for a composite service comes to the composition engine, it takes help of the Service Discovery Manager to obtain information about the availability of the necessary services in the vicinity. Note that some of the required services might be locally present on the device itself. The composition engine decides on the best available platform to carry out the composition based on number of services local to that platform that would be utilized by this composition request and its processor power. Currently we carry out a broker arbitration between the platforms that has at least one service that is participating in the current composition. The Client Composition Manager decides on a Composition Manager that is going to perform the task based on the number of usable services that device has for the composite request.

```
<rdf:RDF
  xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml= "http://www.daml.org/2001/03/daml+oil#"
  xmlns:process="&DamlProcess;#"
>

<daml:class rdf:ID="ViewFileComposite">
 <rdfs:subClassOf rdf:resource="&DamlProcess;#CompositeProcess" />
 <rdfs:subClassOf>
   <daml:Restriction>
      <daml:onProperty rdf:resource="&DamlProcess;#composedOf" />
        <daml:oneOf rdf:parseType="daml:collection">
          <daml:Class>
              <daml:intersectionOf rdf:parseType="daml:collection">
                  <daml:Class rdf:about="process:Sequence" />
                  <daml:Restriction>
                    <daml:onProperty rdf:resource="&DamlProcess;#components" />
                        <daml:toClass>
                           <daml:Class>
                             <daml:listOfInstancesOf rdf:parseType="daml:collection">
                                <daml:Class rdf:about="#Filedownloader" />
                                <daml:Class rdf:about="#Printer" />
                             </daml:listOfInstancesOf>
                           </daml:Class>
                        </daml:toClass>
                  </daml:Restriction>
              </daml:intersectionOf>
        </daml:Class>
        <daml:Class>
          ......
        </daml:Class>
      </daml:oneOf>
   </daml:Restriction>
 </rdfs:subClassOf>
</daml:class>
</rdf:RDF>
```

This information is obtained during service discovery. Each request in this environment may potentially be assigned a separate platform to carry out the composition. We have implemented two different techniques to carry out the discovery and execution of services that take into account high mobility and short stability of services in highly dynamic environments.

In environments where the services can be expected to be available for a longer duration (e.g., a group meeting in a conference room), the discovery of all the component services is performed first, followed by execution. In highly mobile environments, discovery followed by execution may fail due to the high service presence instability. The service may become unavailable after it was discovered and before the execution request has been sent to the device. The composition in such environments is carried out by discovering and executing services in a sequential manner. Faults occur when the Composition Manager is executing a particular process model and a particular service in that model becomes unavailable. The Composition Manager adapts to these faults as well as the changing environment by trying out multiple different process models to execute a particular composite service.

VII.B Experiences

We carried out various experiments to validate the proper functioning of the Anamika Reactive Service Composition system and test different features of the Dynamic Broker Selection based protocol. In our experimental setup, services reside on different laptops. These services are registered to the local composition managers residing on the machines. Bluetooth modules provided by Ericsson and IBM's BlueDrekar software stack [1] and transport driver are used by the Network Manager to communicate between devices. Some of the laptops also have 802.11b connectivity to the Internet. Each device displays a graphic user interface for clients to access composite services formed by the services available in the vicinity. Figure VII.2 shows the user interactions with the Anamika system to view a file by composing services available in the vicinity.

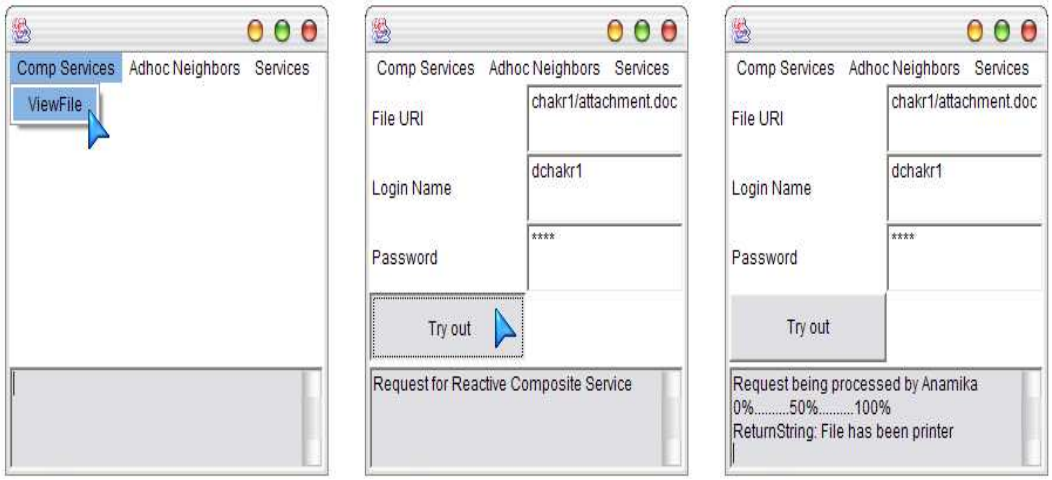

Figure VII.2: User Interaction with Anamika System

There is no central broker service in the system and all the participating systems can participate as brokers. We carried out experiments to test the proper functioning of the different mechanisms for service composition and under different states of the ad-hoc environment. The composite request we used was to "View a File in a server" using services available in the vicinity. The Client Composition Manager determines that viewing a file (the attachment is saved in a file at the server) can be only done when there is an "Internet Service" that can download the file from the location along with a "Printer Service" that can print a postscript file.

VII.B.1 Experiment with Request Source as the Broker

In this experiment, we make the request originator the composition manager for that composite request. The Client Composition Manager carries out the discovery followed by the immediate execution of the corresponding service in the sequence prescribed by the Request Processor. We present a snapshot of the system activity in Figure VII.3. We observe in the log that the Composition Manager at the client tries to determine what services can be combined to satisfy the user request. It then initiates device discovery followed by peer-to-peer semantic service discovery to all the devices in its vicinity. The Composition Manager aggressively executes a service immediately after it is discovered. In this case, it executes the "Filedownloader" service. It then proceeds on to search for a printer service. It discovers a printer service and sends the file to the printer and prints a user-friendly message to the user interface (Figure VII.2). We also observe the connect-transmit-disconnect mechanism used by the Network Manager over RFCOMM to transmit service discovery and execution related Anamika messages. If a printer service is unavailable, the Composition Manager will return to consult the Composition knowledge (via the Request Processor) about other alternate services that can be used (for example a word processor) to let the user view the content of the file.

VII.B.2 Experiment by enabling Broker Arbitration

We carried out this experiment for the same request. However, this time we enable broker arbitration in the system. We also moved the "Filedownloader" and the "Printer" service to the same machine. The Client Composition Manager performs the discovery of all the required services first. Client Composition Manager now carries out a broker arbitration to decide the broker platform based on the number of services residing on a platform and its CPU power. The algorithm processes the service descriptions to extract out the resource details (CPU power and CPU type) and runs in linear time with respect to number of components/services required for a composite request. Since both the services are residing on a remote device, the brokerage is transferred to the platform and the composite request is sent to the remote Composition Manager. The remote Composition Manager carries out the composition locally since both the services are locally available. We show the activity log of the client Composition Manager in Figure VII.4.

We also carried out several experiments to test the fault-tolerance of the system with respect to service and network unavailability. We carried out experiments where the "Printer" service shuts itself down after it has already been discovered by a composition manager to execute a composite request. The composition manager executing the composite request appropriately detects this and tries to execute the "view an attachment"

request using other available services (an word processor in this case).

```
[root@localhost UserInterface]# java UI
Calling Anamika..
Anamika        :Using Composition Knowledge to get services to satisfy the request..
Anamika        : Number of services needed for the composition:2
Anamika        :Trying to discover and execute the services immediately..
Anamika         : Trying to discover devices in the vicinity
Doing Inquiry now .. please wait ..
SDPManager     : Trying to discover services in deviceID 0 using DAML+OIL service descriptions
NetworkManager:Waiting to establish a RFCOM connection....
NetworkManager: RFCOM connection has been opened
NetworkManager:RFCOM closed successfully
Anamika        : Discovered a Filedownloader service in the vicinity on device :0
Anamika        :Executing the service.
NetworkManager:Waiting to establish a RFCOM connection....
NetworkManager: RFCOM connection has been opened
NetworkManager:RFCOM closed successfully
Anamika        : Discovery and execution of one service done.....proceeding to the next service
Anamika         : Trying to discover devices in the vicinity
Doing Inquiry now .. please wait ..
SDPManager     : Trying to discover services in deviceID 0 using DAML+OIL service descriptions
NetworkManager:Waiting to establish a RFCOM connection....
NetworkManager: RFCOM connection has been opened
NetworkManager:RFCOM closed successfully
Anamika        : Discovered a Printer service in the vicinity on device :0
Anamika        :Executing the service.
NetworkManager:Waiting to establish a RFCOM connection....
NetworkManager: RFCOM connection has been opened
NetworkManager:RFCOM closed successfully
```

Figure VII.3: Log of activities performed by Anamika system at client when the Request Source is the Broker

VII.B.3 Scalability Experiment

We carried out experiments to study the scalability of the discovery system by increasing the number of services required to be discovered in a single request. We calculated the average response time to discover all the services required for a composition. The average response time includes the time required to determine a server channel to use between the peer devices, setting up an RFCOM connection for the Service Discovery Managers to communicate and tearing down the RFCOM connection after the interaction. We increased the services from 1 to 5. The experiment was repeated 15 times. Our results are summarized in figure VII.5. The average response time increases approximately linearly with the increasing number of services. Note that the absolute values of the results depend on the type of Bluetooth hardware and software kit used.

VII.C Chapter Summary

In this chapter, we presented the the Anamika system, an implementation prototype of our design architecture in chapter III. Anamika has been implemented over Bluetooth using RFCOMM as the network layer. Current implementation of Anamika models composite processes using DAML-S and other services are described

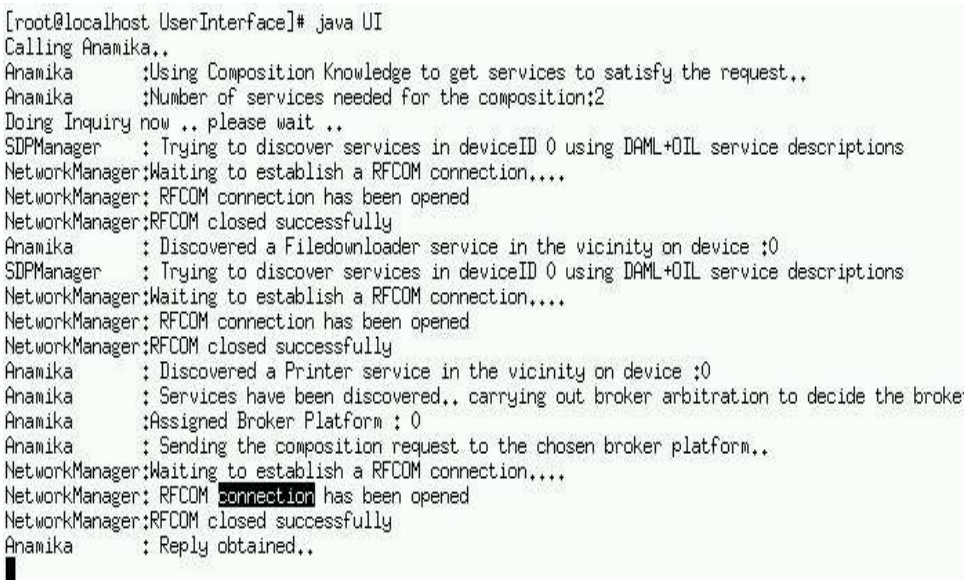

Figure VII.4: Log of activities performed by Anamika system at client when Dynamic Broker Selection based protocol is enabled

using an extension of the DReggie ontology. Anamika supports peer-to-peer based semantic service discovery and matching, and implements the Dynamic Broker Selection-based protocol, where we use processor speed and number of local services needed for a particular composite request to decide a broker platform for a composite request. We present experiences in running realtime queries in the system and demonstrate the working of our system. We also study the scalability of our system with respect to increasing number of services required for a composite request.

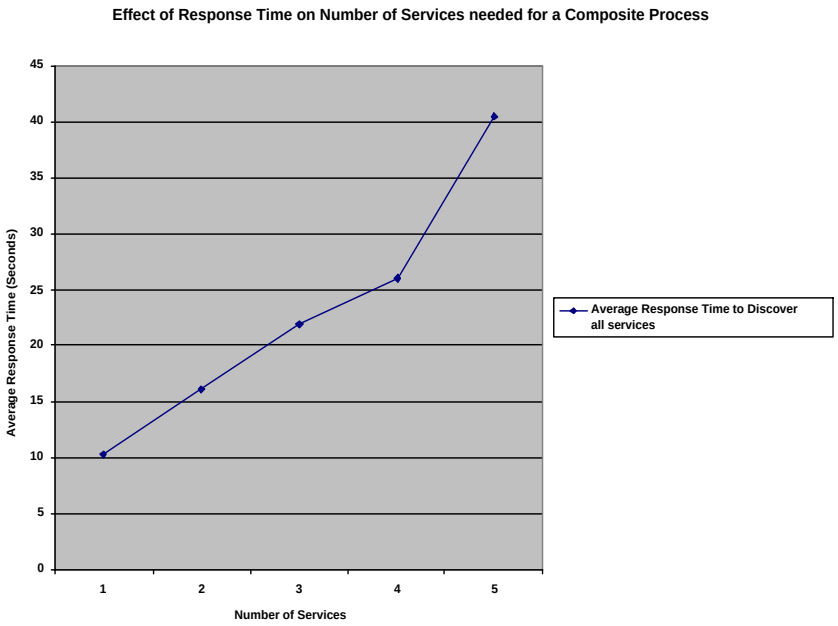

Figure VII.5: Effect on Response Time to discover services with increase in the number of services

Chapter VIII

PROACTIVE COMPOSITION AND FORMAL MODELING OF SERVICE CACHE

Proactive composition refers to the offline computation of possible compositions. Compound services are discovered and integrated before a user makes a request to execute the composite service. An example of proactive composition may be that of a personal agent on a mobile device precomputing the location of the nearest vietnamese restaurant whenever it detects it's presence in a new locality. Proactive computation has recently been researched in the context of data management in pervasive environments, especially in reference to modeling user activities through profiles [81, 83, 30]. The protocols described in chapter VI are reactive where the composition is trigerred upon user request. We also envision that mobile devices would be able to compute possible composite services by intelligently combining services present in the environment. A composite service might be made up of one (cardinality of composition=1) or many individual atomic services. Possible benefits of proactive composition are (1) faster responses for positive matches: Queries that match a pre-composed service will result in faster response time for service matches. This is because time for service discovery and integration is avoided; (2) Reduction in on-the-fly computation: On-demand computation incurred at the time of the user request is reduced. We leverage the idea of serindiptous computing [90] and claim that using device resources at a time when user requests are low leads to better utlization of run-time resources of a mobile device. (3) Effective monitoring of the environment: An office-type environment (a relatively stable environment) might need only one scan per hour to determine the set of possible compositions. Proactive composition thus might be more efficient in home or office type pervasive environments.

However, some of the notable issues to be addressed in order to enable a proactive computing platform for

pervasive environments are: (1) Services need to be matched intelligently to form logically feasible composite services. Combining a printer with a coffee maker does not make any sense in the real world. (2) Device Assumption: Proactive composition imposes assumptions on the resources (memory, CPU) of a device. For example, advertising and caching of services are very important in order to effectively calculate composite services in devices. Both these features impose asscumptions on the memory as well as on the underlying network. We need a service discovery protocol that utilizes the network layer efficiently. (3) User Request and Environment Assumption: Users requests vary from utilizing remote services to utilizing and combining data from remote services. Data can easily be cached (assuming memory is available) and combined to provide results. However, remote service invocation might not be possible in highly dynamic environments. This is because a component atomic service might not be available during the time of an user request even if the composite service has been pre-computed and is available locally in the device.

This chapter describes our effort towards analytically modeling protocols that enable proactive composition in pervasive environments. We follow the principle of bottom-up modeling and hence investigate in modeling service discovery protocols that are indispensible towards supporting proactive composition. In particular, we select the GSD protocol described in chapter IV and model the core features that are common to most service discovery protocols for infrastructure-less pervasive environments. We propose a bottom-up approach towards modeling essential features of distributed service discovery protocols. More specifically this chapter proposes a method to analytically model the service cache (SC) on an ad-hoc node. This provides us with a basic node-level model of the memory constraints of the protocol. This may be used to effectively determine memory requirements of the protocols in various environments (office, home or smart traffic environments).

We employ the use of stochastic process models and fundamental queuing theory to model the data present in the cache. We show using simulations how our model can be used to predict the service cache usage at the node level. Our model is based on the assumption of non-preemption of service descriptions from the service cache and that the system is at steady state. We assume that there are infinite number of services in the system. We also experimentally verify that our model predicts the system behavior reasonably well even when the assumption of infinite services does not hold.

VIII.A Target Protocol Summary and Motivation

Most service discovery protocols for ad-hoc networks have certain features in common. Some of these features are *advertising of a local service, caching of the received advertisements, broadcasting (or selective forwarding) of service discovery requests.* In this chapter, we have focused on a service discovery protocol in chapter IV that essentially has variants of all the above-mentioned features. The GSD protocol is summarized as follows:

Each node advertise their local service(s) to the nodes in their nearby vicinity (specified by number of hops). Each node caches a received advertisement in its service cache (SC) (provided it has space available). The advertisement timeouts and is expunged from the cache after a certain timeout period. When a node wants to discover a certain service, then it first checks its SC to see if it has information about the service. The service is considered "discovered" if an entry of the service advertisement is there in the SC. The node sends a service discovery request in the network in case of a cache miss by specifying the number of hops. Every node receiving this request checks its SC and replies with the information of the service if an entry corresponding to that service is present.

We present a queueing theoretic formalism to model the service discovery at the service node level by trying to model the ways in which a node is affacted by the discovery protocol. The service cache is the key component that influences the discovery process at the node level. This is primarily because :(1) Number of entries in a service cache (SC) determines the probability of a service request matching a given service description; (2) Number of entries in a SC offers a reasonable parameter to judge the density of services/frequency of advertisement in the environment (assuming some relaxed assumptions on the maximum memory of the node); (3) The information present in the SC is used to determine the routing policy of service requests. This chapter presents a M/G/c/c queuing theory-based model to predict the behavior of the SC and hence model the service discovery at the node level for ad-hoc networks.

Our motivation behind coming up with an analytical model is driven by the following factors:

- **Protocol Behavior Prediction**: Using an analytical model, we can predict the behavior of various components of the protocol. For example, we can predict the approximate number of service advertisements cached by a certain node in a given ad-hoc environment with given advertisement frequencies, advertisement timeout rates etc. In a broader sense, this model enables us to predict the memory requirements of a device to participate in service discovery when it is in different environments, viz.

malls, walkways, meeting rooms etc. Other examples include prediction of the amount of time each advertisement would be in a cache for a given value of the maximum cache size, prediction of the probability of the discovery of a service in a certain environment etc.

- **Theoretical Performance Bounds Calculation**: Our analytical model provides a theoretical performance bound on the working of various components of a service discovery protocol. We can determine how the network bandwith usage is being affected if the number of hops an advertisement travels is increased. It empowers us with the capability to estimate upper and lower bounds on the memory and network capacity usages for discovery protocols having variations of the features mentioned above.

- **Protocol Performance Optimization**: Certain aspects of the performance of the protocol can be optimized using theoretical bounds derived from the model. For example, we can maximize the probability of a service being discovered by optimizing on the network bandwidth expended to do advertising. For other systems, network bandwidth may have priority over discovery. We should be able to tune our protocol to provide maximum discovery probability by keeping the bandwidth usage to a minimum. The aim of the model is to provide a theoretical framework that will help us determine the appropriate values of the parameters that affect service discovery protocols (e.g. advertisement hops, SC units needed, discovery request hops etc).

To the best of our knowledge, there has been no work towards analytically modeling service discovery protocols. However, there has been considerable work in modeling services [70, 24, 111] and service interactions [112, 50] using descriptive constructs and logic. There has been some work in analyzing self-healing strategies [38] that enable service discovery systems to mantain consistency [39] during network failure. However, this work tries to use a event-based specification model to specify protocol interactions. They do not use any mathematical model to study and predict the behavior. We are not providing a separate section for related work since we have covered most of the concepts and existing theory in this section.

VIII.B Node-level Interaction Model

Our model is developed following a bottom-up modeling paradigm where we represent the Service Cache as the *atomic unit*. It is considered the atomic unit because it is the most basic unit that participates in the various interactions. All higher level interactions between nodes (e.g. forwarding of service advertisements, matching of service discovery requests, selective forwarding of service requests) either affect or are affected

by the service descriptions stored in the SC.

State Definition:

We define the state E_n of the SC in the following manner: *The SC in a node is in state E_n at time t if and only if it contains n remote service descriptions at time t.* Thus, formally:

$$\begin{aligned} E_n(t) &= \text{service descriptions present in an SC} \\ \text{at time t.} & \\ &= n(\text{according to our definition}) \end{aligned}$$

We consider the presence of one service description as one unit in the SC. The physical memory occupied by one service advertisement may be different from another (due to their varying sizes).

The state of an SC changes when a new advertisement arrives in the system. An advertisement is considered new if it contains a new service description in it. Let c= maximum number of advertisement units that can be stored in a cache. λ= mean of the arrival process of service advertisements. Thus, when the SC is in state E_n, then it may change to state E_{n+1} with the arrival of a new advertisement with mean rate λ. Similarly, the state of a SC also changes when an advertisement times out and is expunged from the cache. Let μ= mean of the timeout values (refered as timeout distribution or service time distribution in the paper). The state of SC changes from E_{n+1} to E_n when a service description is removed from the SC. We follow a *no preemption* policy when the SC is full. Thus, a new advertisement is discarded if the SC is full.

Figure VIII.1 shows the service cache in an ad-hoc node and the way we model it's contents using markov chains. We note that there are alternatives possible to the *no preemption* policy. However, we also observe that a cache gets expunged when its entry times out. Preemption policies lead us into cache replacement strategies when there is a new advertisement. This means, that our model has to encapsulate the cache replacement strategy as well. We intend to model that in the future to further enhance our model.

Another approach towards modeling service discovery interactions is to look at it from a top-down manner. In that approach, a *system state* comprises the combined states of the different nodes, the network channels, the messages, the service caches etc. A state change may occur due to the occurance of several events. Events range from network-level events like message transfer, message loss to application level events like advertisement timeout, advertisement entry etc. Such an approach is not only cumbersome, but would also be very difficult to model and yield meaningful results. On the otherhand, the bottom-up approach is simpler

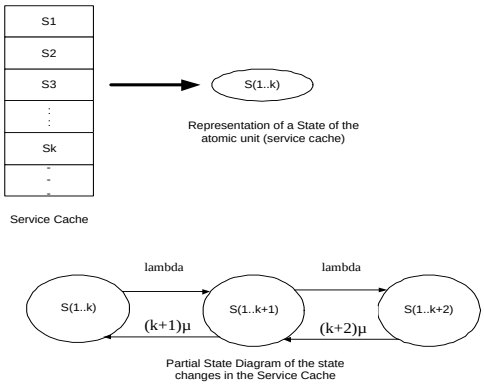

Figure VIII.1: Service Cache in a node and corresponding markov model

in this context and enables better understanding of the system too.

VIII.B.1 M/G/c/c Queueing System

We use the M/G/c/c queuing system [93] to model our service cache. A queuing system based on this model is in state E_n at time *t* if the number of jobs or customers in the system is *n*, that is, if N[*t*]=*n*. The arrival of a job into the system is considered a *birth* and the departure is considered a *death*.

A queuing system consists of one ore more servers and one queue. Servers are able to execute jobs. Jobs coming into the system are stored in the queue. A job is taken from the queue and executed in a server if it is empty. If all the servers are busy, then the job is queued and waits for a server to be free. Each server can execute atmost one job at a time. Following the analogy from the *Birth-and-Death* model, a queuing system is defined to be in state E_n at time *t* if N[*t*]=number of jobs being processed in the servers + number of jobs waiting in the queue. The arrival process of jobs coming into this system follow a stochastic distribution. The time taken for the jobs to be executed in the servers also follow a stochastic distribution. We denote λ as the mean of the arrival distribution and μ as the mean of the service time distribution.

An M/G/c1/c2 queuing system represents a system where:

- M denotes the distribution followed by the arrival process. A process denoted by M indicates that the interarrival time between two jobs follow an exponential distribution. Thus the arrival process follows a poisson distribution [93].

- G denotes the distribution followed by the service time of jobs. A process denoted by G indicates that it can follow any distribution (poisson, gaussian etc).

- c1 denotes the number of servers in the system.
- c2 denotes the maximum number of jobs that can be in this system at a given time *t*.

For the purposes of this paper, we focus on a system where c1=c2. We explain the reason behind such model in section VIII.C. Thus, there can be atmost c jobs (c=c1=c2) in the system. This in turn means that the effective queue length of this system is zero. This is because when a job comes into the system, there are two conditions possible:

- N[*t*]<c: One or more servers are free. Thus the job will be immediately allocated to a queue.
- N[*t*]=c: All the servers are busy executing jobs. Ideally, the job should be queued for a server to be free. However, since the maximum number of jobs in this system is also c, queuing of this job will violate this condition. Thus the job is thrown out of the system without being executed.

An M/G/c/c system is called a *loss system* because all arrivals when the system has exactly c entities are denied entry into the system. It has also been proved that all formulas given for M/M/c/c queueing system are also true for M/G/c/c queuing system [51]. This means that the distribution of the service time does not affect the results of a M/G/c/c system. Such queuing systems are called *robust systems*.

VIII.B.2 Formulae

This system is modeled as a *Birth-and-Death* process with the coefficients:

$$\lambda_n = \lambda \quad n = 0, 1, 2, ..., \tag{VIII.1}$$

$$\mu_n = n\mu, \quad n = 1, 2, ...c \tag{VIII.2}$$

λ_n and μ_n denotes the arrival rate and the service time rate when the state of the system is E_n. We denote a=λ/μ. a is defined as the traffic intensity of the system.

Let p_n = probability of system having *n* jobs

Thus p_n= probability of the system being in state E_n. It can be shown [93] that

$$p_n = \frac{\frac{a^n}{n!}}{1 + a + \frac{a^2}{2!} + \ldots + \frac{a^c}{c!}} \text{ for } n = 0, 1, ...c \tag{VIII.3}$$

Thus, we can get the probability of the system containing n jobs given that we know the arrival rae (λ) and the service time rate (μ) and the size of the system (c).

VIII.C M/G/c/c Model for Service Cache

We model the service cache in our bottom-up approach using the M/G/c/c queuing model. We use results obtained from M/G/c/c to predict the way in which the service cache works. The results discussed in section VIII.B.1 assume that the jobs are generated from an infinite pool of jobs and the results are at steady state of the system. Similarly, we have the following assumptions on our service cache:

1. **Infinite Services Assumption:** We assume that there are *infinite services* in the whole system. Thus, every service advertisement entering a service cache is considered unique. There are no duplicate advertisements in this environment. An advertisement is considered duplicate if it advertises about the same service. Note that by this definition, two advertisements from different nodes may be duplicates. In section VIII.D, we show our model performs well even if this assumption does not hold true.

2. **Steady State Assumption:** We assume *steady state* of the system. This work does not deal with transient states. A steady state in our ad hoc environment can be defined as that state when in a given unit area, the inflow as well as outflow of nodes is equal. From the point of view of the service cache, the system is in steady state when the arrival rate and the service time rate does not vary with time. Transient states are very difficult to model and moreover, most of the systems reach a steady state after a short time.

3. **No Preemption Assumption:** We assume that an incoming advertisement is discarded if the SC is full. One cache replacement strategy might be to replace an existing entry with the new one using some policy (LRU, MRU etc). However, in our model, a new advertisement is discarded if the SC is full. It does not preempt an existing advertisement.

We present an analogical proof showing that the various components of our service discovery protocol can be related using a one-to-one cardinality to the various components of the M/G/c/c queuing system. We also show how the state changes of the service cache is mapped to the state changes of the M/G/c/c system.

1. **Service Advertisements and Jobs**: We map a service advertisement in our system to a job in M/G/c/c. Thus arrival of a service advertisement into our system is considered equivalent to the arrival of a

job in M/G/c/c. However, one very important assumption of the theory behind M/G/c/c systems is that the inter-arrival time between jobs follow exponential distribution. In section VIII.D, we verify experimentally that the inter-arrival time between advertisements follow exponential distribution.

2. **Service Cache Entries and Servers**: Each free entry unit in the service cache is considered equivalent to a free server in the M/G/c/c system. The action of storing an advertisement in one free entry in the service cache is mapped to the action of a job being processed by a server in M/G/c/c. Thus equivalently, the service time distribution in M/G/c/c is mapped to the distribution of the timeout values of the individual entries in the service cache. Note that the service time can follow any distribution (property of M/G/c/c as explained in section VIII.B.1). We would like to mention over here that for the purpose of this paper, we assume each advertisement takes up one free entry in the cache. In practical systems, the memory required by individual advertisements might vary. We consider "one entry" of the service cache as the memory occupied by the biggest possible advertisement. In other words, the actual physical memory allocated in a node might vary depending on the total number of advertisements present in the cache. We consider the total number of advertisements present in the service cache as the total number of servers that are processing jobs in an M/G/c/c system.

3. **Maximum Cache Size** *c*: The maximum size of the cache (in number of units) can be considered equivalent to the number of servers *c* in the queuing model.

4. **Maximum number of entries** *c*: If the first three equivalence relationships hold, then we can deduce that the maximum number of jobs in the queuing model is analogous to the maximum number of advertisements present in a service cache at any time *t*. In fact, the service cache behaves exactly like the M/G/c/c queuing model in this regard. Under the assumptions stated in VIII.B, an advertisement is discarded if it arrives at the system when the service cache contains *c* entries. The arrival of a new job in the queuing model is treated in the same manner when the system contains *c* jobs.

Thus, with the above mappings, we model the service cache on a node in our environment using the M/G/c/c queuing model. Our service cache is in state E_n when there are *n* service descriptions in it. An arrival of a new advertisement is a birth (that changes the state) and the timeout of an advertisement can be considered to be a death. Since the model is also applicable to the M/G/c/c queues the distribution of timeouts is not important. Only the mean rate is important. Given an arrival rate λ and the timeout rate μ we can find the traffic intensity a (i.e. the ratio λ/μ) and the probability p_n that the system will contain *n* entries at steady

state.

VIII.D Experimental Evaluation

We implemented our test bed on the well-known network simulator Glomosim [114] under various mobility conditions and different node topologies. This test bed is part of the Group-based Service Discovery (GSD) protocol [22] discussed in chapter IV. GSD encapsulates the standard features of distributed service discovery protocols we are interested in modeling. Simulation environment consisted of node topologies ranging from a topology with 25 nodes to a topology with 100 nodes. We used a random way-point mobility pattern for all the nodes. Each node advertises its services across either one or more hops. For the purpose of the experiments, we considered services to be advertised across one hop only. Each experiment was ran for a simulation time of 1 hour over a space of (190m X 190m). We considered an uniform advertisement interval of 10s and uniform advertisement timeout of 5s. Nodes followed a random waypoint mobility pattern with 3m/s speed with a 5s stoppage time. All the simulation parameters including the various timeouts used for different tables and link and radio layer parameters have been enumerated in Figure VIII.1.

Duration	3600 seconds (1 hour)
Space (x,y)	(190 x 190m)
Tx Range (Transmission Range)	30m
Tx Throughput	20kbps
No. of Nodes	25, 64, 100
Advertisement Interval	10 seconds (uniform)
Advertisement Timeout	5 seconds (uniform)
broadcast jitter	10 milliseconds
Mobility	Random way-point with 3 m/s speed and 5 s stopage time

Table VIII.1: Experiment Parameters for M/G/c/c Modeling of Service Cache

We carried out experiments to observe the distribution of the inter-arrival time of service advertisements. This forms the first step towards the validation of the application of M/G/c/c model on the service cache. We also carried out probabilistic estimation of the service cache being in state E_n (n=0,1,2...c)(for a given λ and μ) and compared it to the values predicted by M/G/c/c model (using equation VIII.3). Finally, we performed experiments to determine how well M/G/c/c model approximates the behavior of the service cache when the **Infinite Services Assumption** does not hold. We discuss the results in the following subsections. Our simulation takes into consideration message losses, disconnections, delays and node mobility. Thus, this by far approximates a real-life ad hoc network.

VIII.D.1 Determination of Arrival Process of Advertisements

We observed that if the transmission of advertisements from individual nodes follow an uniform distribution (with a certain upper bound), the arrival process of advertisements follow a poisson distribution. Logically, if there are two nodes in the system, a uniform process of advertisement generation will result in an uniform arrival process (under ideal situations, i.e. no message loss, disconnections etc). However, this is clearly not the case when there are multiple nodes in the vicinity of a device and when the ideal situations do not hold true. We assumed that the transmission of service advertisements followed an uniform distribution. This is a most general assumption since it does not impose any restriction on the way in which the advertisement is generated. The inter-arrival time distribution of advertisements is plotted in figure VIII.2. We carried

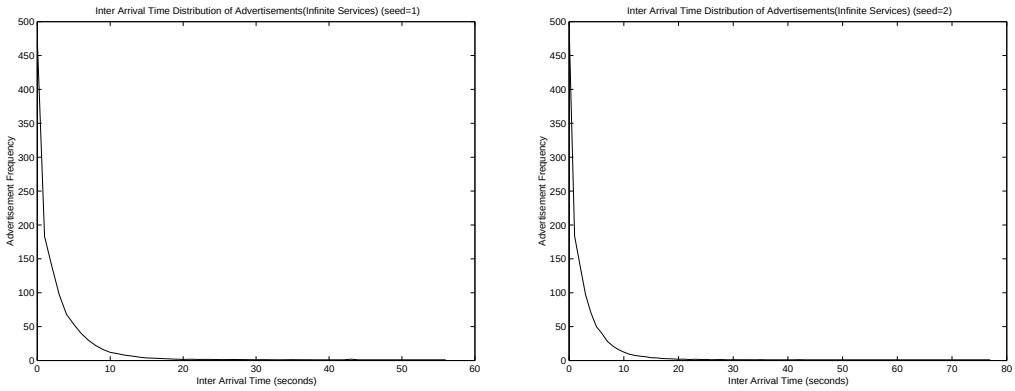

Figure VIII.2: Distribution of Inter-arrival times of service advertisements for Uniform advertisement transmission time distribution

out the above experiment for nodes ranging from 25 to 100 with different mobility patterns and different advertisement diameters. We carried out least square approximation on the experiment data and observed that the curve is very near to exponential. Thus, we conclude that for an uniform advertisement transmission distribution, the advertisement arrival process follows a poisson distribution for a real-life ad hoc network environment. This is an important result in the field of ad-hoc service discovery advertising. This result further substantiates the assumption about the distribution followed by the arrival process of jobs in a M/G/c/c queuing system.

VIII.D.2 Comparison of Cache Usage Probability

For a given λ and μ, equation VIII.3 provides a probabilistic measure of the system being in state E_n. Cache Usage is defined as the actual number of cache entries being used. We calculate the probabilistic distribution

of the service cache being in different states (state being given by the cache having certain entries) for a given λ and μ. We calculate λ from the arrival process distribution. We assume a constant service time distribution for the purpose of the experiments. Thus in our experiments the general distribution (G) is replaced by the deterministic distribution (D). We carried out these experiments assuming that all the incoming advertisements were unique. Thus we respected the **Infinite Services Assumption** in these experiments. We compared the probabilistic measures as obtained from experiments against the ones predicted by M/G/c/c theory. The results are shown in figure VIII.3.

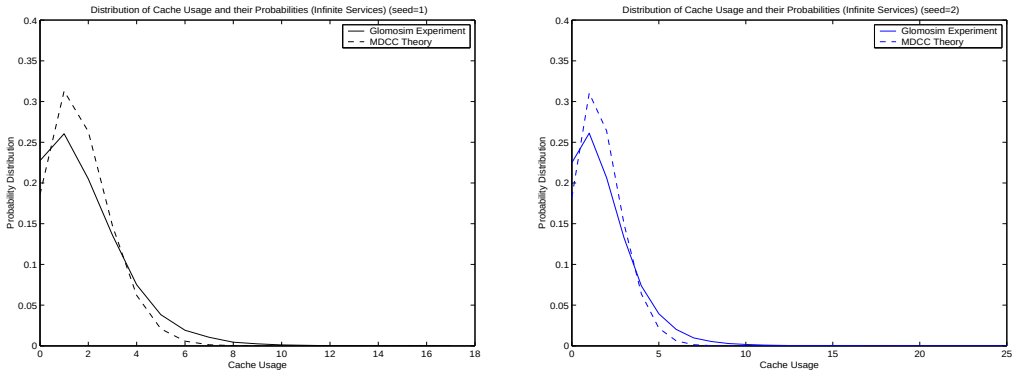

Figure VIII.3: Cache Usage Probabilistic Measure compared against prediction made using M/D/c/c model

We observe that the probability estimates follow each other very closely. In general, the M/D/c/c theory underestimates the probability right at the beginning, overestimates in the middle and underestimates at the end with marginal error bounds. We observe that the mean error ranges from 6% to 13%. Thus, we can conclude that M/G/c/c queuing model can be used to model the service cache of nodes in our bottom-up model towards service discovery. However, the predictions made by this model are subject to marginal errors but overall obtains good results in modeling the behavior of the service cache.

VIII.D.3 Comparison of Cache Usage Probability without the *Infinite Services Assumption*

The **Infinite Services Assumption** does not hold for practical ad-hoc networks. This is because there are finite number of nodes in an ad-hoc environment and it is infeasible to think of infinite services being present on a finite number of nodes in a real-life environment. The results of the M/G/c/c model are only valid when each job is unique. We carried out experiments to determine how well the model estimates the behavior of the

service cache in presence of a finite number of services. Two advertisements are duplicate if they contain the same service descriptions. We observed that the arrival process of advertisements follow a poisson process anyway. Our experiments showed that M/G/c/c is still a good model to apply even when the environment contains finite services. We carried out experiments with a service/node ratio ranging from 1 to 0.125. Figure VIII.4 displays the results. Thus, we can apply the model towards modeling service discovery even when there are finite number of services in the environment. However, the service/node ratio is important to determine how well the model will represent the actual working of the system. We have shown that the model is appropriate even with a service/node ratio of 0.125 which is a reasonable value for a real-life ad hoc network.

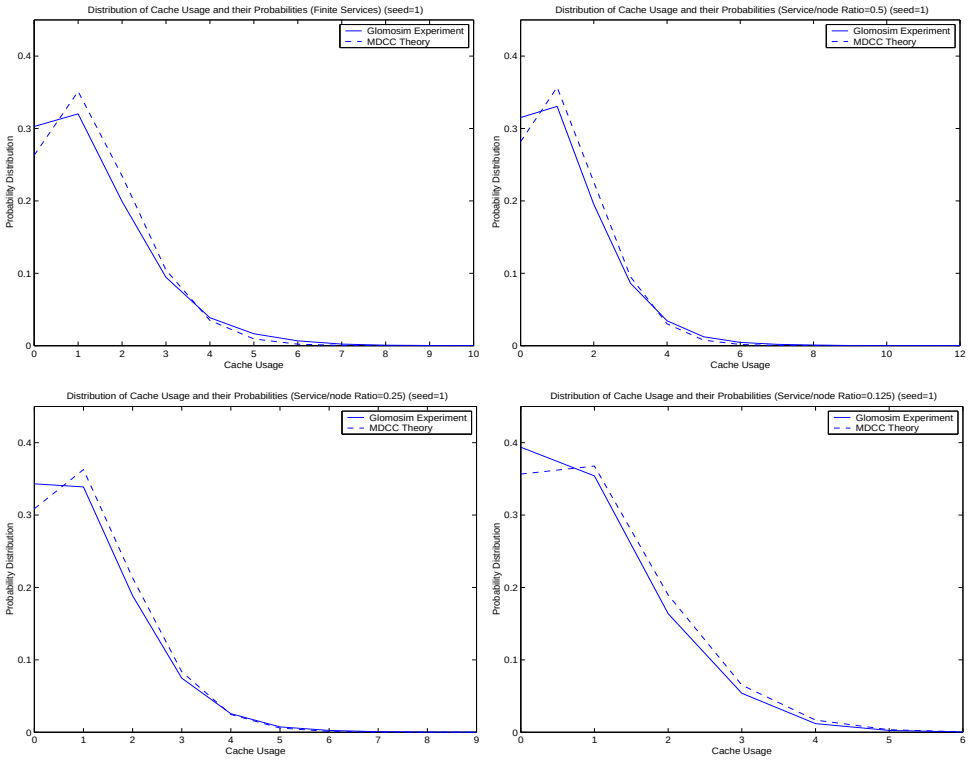

Figure VIII.4: Cache Usage Probabilistic Measure compared against prediction made using M/D/c/c model for finite services. Service/node ratio ranges from 1 to 0.125

VIII.E Chapter Summary

We analogically prove that M/G/c/c model can be used to predict the cache usage of a service cache (SC) in a mobile device participating in ad-hoc service discovery. We experimentally studied the arrival rate of advertisements (given uniform sending rate) at a certain cache. We observed that it follows exponential distribution

very closely. We experimentally proved that the cache usage can be predicted reasonably well using M/G/c/c queuing model. We showed that even for finite services M/G/c/c provides a good approximation of the predicted usage. This would enable practical ad-hoc environments to apply our theory as a good ”approximation model”.

Our bottom-up queueing theoretic model has got far-fetched applications in modeling various features of service discovery protocols in ad-hoc networks. We can determine the average timeout of a service description for a given the arrival rate, the size of the cache and some threshold probability of the required number of entries in the cache. We can also determine the optimal timeout rate for the system for a given advertisement arrival rate so as to achieve equilibrium. Equilibrium is defined as that state when the average number of entries in the SC do not fluctuate.

For example consider that the node has settled to some equilibrium value of timeout rate μ for the given arrival rate λ. Thus if a new node enters the vicinity of this node then the arrival rate will change which in turn will modify the timeout rate μ to maintain probability p_n that the cache contains n entries. Yet another application is to model the advertisement sending rate based on the received rate λ. Our model helps in predicting the corresponding rates for nodes to optimize their performance as well as networks to reduce their cumulative traffic. In future, we aim to model higher level interactions between these service cache units and eventually model protocol level components of ad-hoc service discovery applications.

Chapter IX

CONCLUSIONS

This chapter presents a summary of the dissertation and outlines three possible future directions of research.

IX.A Dissertation Summary

Internet has already ushered the information age revolution. One branch in Computer Science research is embarking on the immensely open world of wireless, mobile and pervasive computing to bring information to the finger tips of human beings *anytime and anywhere*. We see the growth of peer-to-peer networking technologies that offer immense possibilities and potential to utilize the information in one's vicinity in an ad-hoc manner. For example, we observe the development of WiFi hotspot environments and Bluetooth networks in Europe where people have started discovering capabilities of potentially unknown people around them through peer-to-peer networking technologies. Bluetooth headsets pair up with our laptops to stream music and do not constrain us to carry the music player or the laptop with us all the time. Managing and configuring all the necessary networks and services that can be accessed in this resource-rich environment will become increasingly difficult with the growth in the number of such devices and technologies. It is very important for sub-application technologies to address the problems of heterogeneity, scalability and management of these resources as well as computing components.

Modeling of objects (resources, data as well as computing elements) is very important in this environment and service-oriented modeling provides immense promise due to its modularity, flexibility and ability to model heterogeneous elements ranging from hardware devices to software computing platforms. Consequently, service discovery and composition form very important sub-application level technologies in such

environments.

This dissertation addresses the issues of service discovery and composition for *infrastructure-less pervasive* environments. It designs, develops, implements and evaluates a highly scalable, distributed and integrated architecture and associated protocols to enable service discovery and composition in infrastructure-less, dynamic and mobile environments. The key aspects of the protocols are their distributed-ness, immunity to central point of failure and the ability to utilize the resource-rich vicinity. The design of the architectures and the protocols therein offers several cross-layer optimizations that improve the overall performance of the architecture.

We developed a distributed peer-to-peer caching based novel protocol for service discovery that combines the traditionally de-coupled fields of service matching and discovery architecture. We also developed a data routing protocol with end-to-end session management for routing of service invocation data. The novel routing protocol enables service-centric session redirection on node failure and supports service-centric routing apart from standard node-centric routing. We developed distributed broker-based reactive service composition protocols for mobile environments and propose several experimental parameters to judge the efficiency of composition in mobile environments. We also proposed and evaluated a queuing theoretic model to formalize the concept of proactive composition in mobile environments. We implemented and evaluated the protocols in the proposed architecture. We utilized semantic web technologies as and when required in various aspects of the architecture. We developed an ontology grounded in Web Ontology Language (OWL), to describe services in mobile environments in terms of its inputs and outputs, platform constraints and device capabilities.

IX.B Future Research Directions

We identify three possible directions for future research in this section. However, there are several other areas where essential concepts of the protocols developed in this dissertation may be applied too.

- Formal Modeling of Discovery and Composition Architectures
- Integration of Security and Privacy Protocols
- Application of Cross-layer Optimization Techniques in Grid Computing Environments

IX.B.1 Formal Modeling of Discovery and Composition Architectures

This dissertation presents an initial formalism of a bottom-up queuing theoretic model for the concept of proactive composition in pervasive environments. Formal modeling of reactive discovery and composition protocols can lead to the theoretical predictions and approximation models of various components of discovery and composition protocols. For example, theoretic models can be used to predict the memory usage of devices in various environments (home vs. traffic environments) for participation in service discovery protocols.

IX.B.2 Integration of Security and Privacy Protocols

Security, privacy and trusting peer devices are very important in pervasive environments. This is more so because, chances of device identity falsifications and network snooping are much more in these environments. Developing a trust model over services/devices is as important as ensuring that the communication occurs in a secure manner. These are important problems to be addressed in the context of service-oriented modeling and computing in pervasive environments.

IX.B.3 Application of Cross-layer Optimization Techniques in Grid Computing Environments

Service-centric models for discovery and composition can also be mapped to resource grids in grid computing environments. Grid Computing platforms pose interesting challenges in the context of clusters of networked services that may be optionally connected through high bandwidth networks. Cross-layer optimizations between protocols might lead to more efficient discovery, integration and execution of queries.

Appendix A

SERVICE ONTOLOGY

```
<?xml version="1.0" ?>
<?xml-stylesheet href="http://www.w3.org/2002/06/rdfs2html.xsl"
                                    type="application/xml"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY rdf     "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <!ENTITY rdfs    "http://www.w3.org/2000/01/rdf-schema#">
   <!ENTITY xsd     "http://www.w3.org/2001/XMLSchema#">
   <!ENTITY owl     "http://www.w3.org/2002/07/owl#">
   <!ENTITY dreggie "http://daml.umbc.edu/ontologies/dreggie-ont#">
   ]>


<rdf:RDF
   xmlns:rdf  = "&rdf;"
   xmlns:rdfs = "&rdfs;"
   xmlns:owl  = "&owl;"
   xmlns:xsd  = "&xsd;"
   xmlns      = "&dreggie;"
   xml:base   = "&dreggie;">


<owl:Ontology rdf:about="">
```

```
 <owl:versionInfo>Filip Perich and Dipanjan Chakraborty</owl:versionInfo>
 <rdfs:comment>Anamika Primary Ontology</rdfs:comment>
</owl:Ontology>

<owl:Class rdf:ID=''Service''>
 <rdfs:label>Service</rdfs:label>
</owl:Class>

<owl:Class rdf:ID=''HardwareService''>
  <rdfs:subClassOf rdf:resource=''#Service''/>
</owl:Class>
<owl:Class rdf:ID=''SoftwareService''>
  <rdfs:subClassOf rdf:resource=''#Service''/>
</owl:Class>

<owl:Class rdf:ID=''InputOutputService''>
  <rdfs:subClassOf rdf:resource=''#HardwareService''/>
</owl:Class>
<owl:Class rdf:ID=''ComputationProviderService''>
  <rdfs:subClassOf rdf:resource=''#HardwareService''/>
</owl:Class>

<owl:Class rdf:ID=''PrinterService''>
  <rdfs:subClassOf rdf:resource=''#InputOutputService''/>
</owl:Class>

<owl:Class rdf:ID=''LaserJetPrinterService''>
  <rdfs:subClassOf rdf:resource=''PrinterService'' />
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID=’’hasComponentClass’’>
  <rdfs:domain rdf:resource=’’#Service’’ />
  <rdfs:range rdf:resource=’’#Component’’ />
</owl:ObjectProperty>


<owl:Class rdf:ID="Component">
 <rdfs:label>Component</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="DescriptionClass">
 <rdfs:label>Description Class</rdfs:label>
</owl:Class>

<owl:ObjectProperty rdf:ID="Description">
 <rdfs:domain rdf:resource="#Component" />
 <rdfs:range rdf:resource="#DescriptionClass" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="ServiceName" >
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="ServiceAlias" >
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="ClientName" >
```

```
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="Capability">
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="#CapabilityClass" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Requirements">
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="#RequirementsClass" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Cost">
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="#CostClass" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Mobility">
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="#MobilityClass" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Input">
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="#InputClass" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Output">
```

```
  <rdfs:domain rdf:resource="#DescriptionClass" />
  <rdfs:range rdf:resource="#OutputClass" />
</owl:ObjectProperty>

<owl:Class rdf:ID="CapabilityClass" />

<owl:ObjectProperty rdf:ID="ClientCapability">
  <rdfs:domain rdf:resource="#CapabilityClass" />
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ServiceCapability">
  <rdfs:domain rdf:resource="#CapabilityClass" />
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:Class rdf:ID="RequirementsClass" />

<owl:ObjectProperty rdf:ID="CPURequirement">
  <rdfs:domain rdf:resource="#RequirementsClass" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="MemoryRequirement">
  <rdfs:domain rdf:resource="#RequirementsClass" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="DiskRequirement">
  <rdfs:domain rdf:resource="#RequirementsClass" />
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="OSRequirement">
  <rdfs:domain rdf:resource="#RequirementsClass" />
  <rdfs:range  rdf:resource="#VersionName" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="SoftwareRequirement">
  <rdfs:domain rdf:resource="#RequirementsClass" />
  <rdfs:range  rdf:resource="#VersionName" />
</owl:ObjectProperty>

<owl:Class rdf:ID="CostClass" />

<owl:ObjectProperty rdf:ID="Local">
  <rdfs:domain rdf:resource="#CostClass" />
  <rdfs:range rdf:resource="#AmountUnit" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Remote">
  <rdfs:domain rdf:resource="#CostClass" />
  <rdfs:range rdf:resource="#AmountUnit" />
</owl:ObjectProperty>

<owl:Class rdf:ID="MobilityClass" />

<owl:ObjectProperty rdf:ID="ClientMobility">
  <rdfs:domain rdf:resource="#MobilityClass" />
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ServiceMobility">
```

```
  <rdfs:domain rdf:resource="#MobilityClass" />
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:Class rdf:ID="InputClass" />

<owl:ObjectProperty rdf:ID="ServiceInputType">
  <rdfs:domain rdf:resource="#InputClass" />
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ClientInputType">
  <rdfs:domain rdf:resource="#InputClass" />
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:Class rdf:ID="OutputClass" />

<owl:ObjectProperty rdf:ID="ServiceOutputType">
  <rdfs:domain rdf:resource="#OutputClass" />
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ClientOutputType">
  <rdfs:domain rdf:resource="#OutputClass" />
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Property">
  <rdfs:domain rdf:resource="#Component" />
```

```
  <rdfs:range rdf:resource="#PropertyClass" />
</owl:ObjectProperty>

<owl:Class rdf:ID="PropertyClass" />

<owl:ObjectProperty rdf:ID="ServiceProperty">
  <rdfs:domain rdf:resource="#PropertyClass" />
  <rdfs:range rdf:resource="#PropertyType" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ClientProperty">
  <rdfs:domain rdf:resource="#PropertyClass" />
  <rdfs:range rdf:resource="#PropertyType" />
</owl:ObjectProperty>

<owl:Class rdf:ID="PropertyType" />

<owl:DatatypeProperty rdf:ID="Name">
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="CpuType">
  <rdfs:range rdf:resource="#Type" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Availability">
  <rdfs:range rdf:resource="&owl;Thing" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Speed">
```

```
  <rdfs:range rdf:resource="#AmountUnit" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Size">
  <rdfs:range rdf:resource="#AmountUnit" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Memory">
  <rdfs:range rdf:resource="#AmountUnit" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="FileSystem">
  <rdfs:range rdf:resource="#AmountUnit" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="OperatingSystem">
  <rdfs:range rdf:resource="#AmountUnit" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Software">
  <rdfs:range rdf:resource="#AmountUnit" />
</owl:ObjectProperty>

<owl:Class rdf:ID="AmountUnit" />

<owl:DatatypeProperty rdf:ID="Amount">
  <rdfs:domain rdf:resource="#AmountUnit" />
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger" />
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="Unit">
  <rdfs:domain rdf:resource="#AmountUnit" />
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="VersionType">
  <rdfs:subClassOf rdf:resource="#Type" />
</owl:Class>

<owl:Class rdf:ID="VersionName" />

<owl:DatatypeProperty rdf:ID="Version">
  <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:Class rdf:ID="Type" />

</rdf:RDF>
```

Appendix B

OBSERVED STANDARD DEVIATION IN EXPERIMENTS

We enumerate the observed values for standard deviations of the results in various chapters in a tabular format. The units of data are reported along with the graphs. Range (R) refers to the range of data, the average of which are plotted in the graphs. We note that the range along with the standard deviation reflects the dispersion of the data. A high value of standard deviation for a small R means that the data was dispersed significantly whereas a low standard deviation for a high value of R means the opposite. We present both the values of range and standard deviation in the tables below.

Convention:

R=Range

σ=Standard Deviation

Protocols	R_f/A_f			
	0.25	0.50	0.75	1.0
BCast	R=3.28 σ=0.939	R=3.45 σ=0.924	R=9.10 σ=1.06	R=3.518 σ=0.912
GSD-B	R=4.07 σ=0.498	R=3.82 σ=0.449	R=3.54 σ=0.451	R=3.87 σ=0.423
GSD-S	R=2.45 σ=0.376	R=5.34 σ=0.481	R=4.69 σ=0.503	R=3.62 σ=0.42

Protocols	R_f/A_f			
	1.25	1.50	1.75	2.0
BCast	R=7.702 σ=0.989	R=13.126 σ=1.01	R=7.65 σ=0.997	R=10.269 σ=1.063
GSD-B	R=11.88 σ=0.563	R=3.315 σ=0.403	R=3.428 σ=0.357	R=9.367 σ=0.487
GSD-S	R=5.24 σ=0.514	R=4.88 σ=0.538	R=7.48 σ=0.494	R=3.94 σ=0.436

Table B.1: Observed standard deviation of data in graph IV.9 (No. of Nodes=50)

Protocols	R_f/A_f			
	0.25	0.50	0.75	1.0
BCast	R=4.59 σ=1.26	R=5.1 σ=1.23	R=4.74 σ=1.24	R=5.44 σ=1.24
GSD-B	R=5.88 σ=0.84	R=7.75 σ=0.88	R=6.34 σ=0.79	R=5.78 σ=0.73
GSD-S	R=3.75 σ=0.61	R=4.62 σ=0.61	R=3.77 σ=0.65	R=4.79 σ=0.58

Protocols	R_f/A_f			
	1.25	1.50	1.75	2.0
BCast	R=16.80 σ=1.43	R=15.38 σ=1.44	R=11.59 σ=1.84	R=16.55 σ=1.94
GSD-B	R=6.43 σ=0.64	R=9.23 σ=0.64	R=12.01 σ=0.64	R=8.84 σ=0.62
GSD-S	R=4.77 σ=0.61	R=5.76 σ=0.68	R=9.77 σ=0.63	R=7.81 σ=0.52

Table B.2: Observed standard deviation of data in graph IV.9 (No. of Nodes=100)

Protocols	R_f/A_f			
	0.25	0.50	0.75	1.0
BCast	R=6 σ=1.541	R=8 σ=1.536	R=7 σ=1.627	R=7 σ=1.49
GSD-B	R=4 σ=0.96	R=4 σ=0.93	R=5 σ=0.92	R=4 σ=0.91
GSD-S	R=4 σ=0.77	R=3 σ=0.83	R=4 σ=0.85	R=5 σ=0.79

Protocols	R_f/A_f			
	1.25	1.50	1.75	2.0
BCast	R=8 σ=1.60	R=7 σ=1.52	R=8 σ=1.54	R=8 σ=1.64
GSD-B	R=5 σ=0.95	R=6 σ=0.859	R=5 σ=0.78	R=6 σ=0.88
GSD-S	R=4 σ=0.84	R=5 σ=0.86	R=3 σ=0.77	R=4 σ=0.744

Table B.3: Observed standard deviation of data in graph IV.10 (No. of Nodes=50)

Protocols	R_f/A_f			
	0.25	0.50	0.75	1.0
BCast	R=7 σ=1.91	R=7 σ=1.74	R=9 σ=1.76	R=6 σ=1.73
GSD-B	R=5 σ=1.23	R=7 σ=1.33	R=6 σ=1.12	R=6 σ=1.17
GSD-S	R=4 σ=1.02	R=6 σ=1.03	R=5 σ=0.99	R=8 σ=1.04

Protocols	R_f/A_f			
	1.25	1.50	1.75	2.0
BCast	R=12 σ=1.913	R=6 σ=1.89	R=7 σ=2.3	R=6 σ=2.41
GSD-B	R=7 σ=1.05	R=5 σ=1.02	R=5 σ=1.02	R=7 σ=0.98
GSD-S	R=7 σ=0.97	R=4 σ=0.97	R=8 σ=0.93	R=4 σ=0.82

Table B.4: Observed standard deviation of data in graph IV.10 (No. of Nodes=100)

Protocols	R_f/A_f			
	0.25	0.50	0.75	1.0
BCast	R=109 σ=13.29	R=254 σ=17.86	R=303 σ=16.39	R=452 σ=18.72
GSD-B	R=123 σ=14.84	R=155 σ=11.78	R=183 σ=10.16	R=191 σ=10.23
GSD-S	R=133 σ=14.98	R=143 σ=10.66	R=135 σ=7.73	R=159 σ=7.722

Protocols	R_f/A_f			
	1.25	1.50	1.75	2.0
BCast	R=435 σ=20.05	R=615 σ=23.16	R=682 σ=24.87	R=754 σ=25.314
GSD-B	R=255 σ=10.69	R=237 σ=9.91	R=334 σ=12.05	R=295 σ=11.54
GSD-S	R=153 σ=6.98	R=154 σ=6.61	R=192 σ=6.66	R=182 σ=5.95

Table B.5: Observed standard deviation of data in graph IV.11 (No. of Nodes=50) (Advertisement Diameter=1)

Protocols	R_f/A_f			
	0.25	0.50	0.75	1.0
BCast	R=138 σ=17.99	R=290 σ= 24.88	R=365 σ=26.95	R=470 σ=30.92
GSD-B	R=179 σ=27.62	R=265 σ=24.35	R=341 σ=25.22	R=388 σ=24.40
GSD-S	R=179 σ=23.41	R=162 σ=18.23	R=246 σ=17.41	R=248 σ=14.95

Protocols	R_f/A_f			
	1.25	1.50	1.75	2.0
BCast	R=620 σ=36.72	R=630 σ=44.66	R=2032 σ=201.87	R=1954 σ=282.54
GSD-B	R=567 σ=25.96	R=597 σ=24.27	R=697 σ=25.01	R=808 σ=26.19
GSD-S	R=293 σ=15.84	R=361 σ=15.28	R=426 σ=14.51	R= 402 σ=14.59

Table B.6: Observed standard deviation of data in graph IV.11 (No. of Nodes=100) (Advertisement Diameter=1)

Protocols	R_f/A_f			
	0.25	0.50	0.75	1.0
GSD-B	R=265 σ=28.16	R=287 σ=20.78	R=288 σ=17.39	R=275 σ=13.54
GSD-S	R=285 σ=30.24	R=296 σ=20.76	R=265 σ=16.18	R=314 σ=14.60
Protocols	R_f/A_f			
	1.25	1.50	1.75	2.0
GSD-B	R=278 σ=13.55	R=282 σ=11.98	R=315 σ=12.51	R=286 σ=10.19
GSD-S	R=263 σ=12.93	R=293 σ=12.46	R=245 σ=11.19	R=307 σ=11.01

Table B.7: Observed standard deviation of data in graph IV.11 (No. of Nodes=50) (Advertisement Diameter=2)

Protocols	R_f/A_f			
	0.25	0.50	0.75	1.0
GSD-B	R=331 σ=45.80	R=371 σ=32.59	R=372 σ=26.24	R=352 σ=22.46
GSD-S	R=344 σ=41.42	R=296 σ=31.45	R=373 σ=26.13	R=366 σ=22.68
Protocols	R_f/A_f			
	1.25	1.50	1.75	2.0
GSD-B	R=490 σ=25.51	R=439 σ=19.28	R=433 σ=19.41	R=415 σ=19.00
GSD-S	R=381 σ=18.80	R=348 σ=18.84	R=290 σ=15.36	R=245 σ=16.15

Table B.8: Observed standard deviation of data in graph IV.11 (No. of Nodes=100) (Advertisement Diameter=2)

BIBLIOGRAPHY

[1] IBM alphaworks. BlueDrekar Protocol Driver. `http://www.alphaworks.ibm.com/tech/bluedrekar`.

[2] The Common Object Request Broker: Architecture and Specification (CORBA). `http://www.omg.org/technology/documents/specifications.htm`.

[3] K. Arnold, B. Osullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification (The Jini Technology)*. Addison-Wesley, Reading, MA, June 1999.

[4] S. Avancha, A. Joshi, and T. Finin. Enhanced Service Discovery in Bluetooth. *IEEE Computer*, 35(6):96–99, June 2002.

[5] D. O. Awduche, A. Gaylord, and A. Ganz. On Resource Discovery in Distributed Systems with Mobile Hosts. In *ACM Internation Conference on Mobile Computing and Networking (MOBICOM), Newyork, USA*, November 1996.

[6] P. Bhagwat B. Raman and S. Seshan. Arguments for Cross-Layer Optimizations in Bluetooth Scatternets. In *The 2001 Symposium on Applications and the Internet (SAINT)*, January 2001.

[7] R.J.R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing.*, 1983.

[8] M. Balazinska, H. Balakrishnan, and D. Karger. Instwine: A scalable peer-to-peer architecture for intentional resource discovery. In *International Conference on Pervasive Computing, Zurich, Switzerland*, August 2002.

[9] T.D. Barfoot and G.M.T.D'Eleuterio. Multiagent coordination of stochastic cellular automata. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI). Seattle. Washington*, August 2001.

[10] P. Basu, W. Ke, and T. Little. A Novel Approach for Execution of Distributed Tasks on Mobile Ad hoc Networks. In *IEEE Wireless Communicaitons and Networking Conference (WCNC). Orlando. Florida*, 2002.

[11] T. Batista and N. Rodriguez. Dynamic reconfiguration of component-based applications. In *5th International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'2000), IEEE, Limerick, Ireland*, June 2000.

[12] F. Belligemine and G. Rimassa. Jade-a fipa-compliant agent framework. In *PAAM '99. London*, pages 97–108, 1999.

[13] F. Bergenti and A. Poggi. Leap: A fipa platform for handheld and mobile devices. In *ATAL*, 2001.

[14] Bluetooth SIG. Specification. `http://bluetooth.com/`.

[15] F.M.T. Braizer, B.M. Dunin-Keplicz, N.R. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems, 6 (1)*, pages 67–94, 1997.

[16] D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0 - W3C Recommendation. `http://www.w3.org/TR/2000/CR-rdfschema-20000327`, 2000.

[17] D. Buszko, W. Lee, and A. Helal. Decentralized ad-hoc groupware api and framework for mobile collaboration. In *ACM International Conference on Supporting Group Work (Group'01).Boulder. Colorado*, September 2001.

[18] A. Carzaniga and A.L. Wolf. Content-based Networking: A New Communication Infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems. In conjunction with the International Conference on Computer Communications and Networks (ICCCN), Arizona, USA*, October 2001.

[19] F. Casati, D. Georgakopoulos, and M. Shan Editors. Special Issue on E-Services. *VLDB Journal*, 2001.

[20] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and Dynamic Service Composition in eFlow. In *Technical Report, HPL-200039, Software Technology Laboratory, Palo Alto, California, USA*, March 2000.

[21] D. Chakraborty and A. Joshi. Dynamic Service Composition: State-of-the-Art and Research Directions. Technical report, University of Maryland Baltimore County, December 2001. TR-CS-01-19.

[22] D. Chakraborty and A. Joshi. GSD: A novel group-based service discovery protocol for MANETS. In *IEEE Conference on Mobile and Wireless Communications Networks, Stockholm, Sweden*, September 2002.

[23] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. Service composition for mobile environments. In *Journal on Mobile Networking and Applications (MONET), Special Issue on Mobile Services*, March 2004.

[24] D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. DReggie: A Smart Service Discovery Technique for E-Commerce Applications. In *Workshop in conjunction with 20th Symposium on Reliable Distributed Systems*, October 2001.

[25] D. Chakraborty, A. Shenoi, A. Joshi, and Y. Yesha. Queuing Theoretic Approach for Service Discovery in Ad-hoc Networks. In *Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS), San Diego, California, USA*, January 2004.

[26] D. Chakraborty, Y. Yesha, and A. Joshi. A distributed service composition protocol for pervasive environments. In *IEEE Wireless Communicaitons and Networking Conference (WCNC), Atlanta, Georgia*, March 2004.

[27] H. Chen. Developing a Dynamic Distributed Intelligent Agent Framework Based on the Jini Architecture. Master's thesis, University of Maryland Baltimore County, January 2000.

[28] H. Chen, A. Joshi, and T. Finin. Dynamic Service Discovery for Mobile Computing: Intelligent Agents meet Jini in the Aether. *Baltzer Science Journal on Cluster Computing, Special Issue on Advances in Distributed and Mobile Systems and Communications*, 2001.

[29] M. Cherniak, M. Franklin, and S. Zdonik. Expressing User Profiles for Data Recharging. In *IEEE Personal Communications*, July 2001.

[30] M. Cherniak, E. Galvez, D. Brooks, M. Franklin, and S. Zdonik. Profile Driven Data Management. In *28th International Conference on Very Large Databases*, August 2002.

[31] C. C. Chiang. Routing in clustered multi-hop, mobile wireless networks with fading channel. In *Proc. IEEE SICON*, pages 197–211, April 1997.

[32] P.R Cohen and H.J Levesque. Teamwork. In *Nous, 25(4)*, pages 487–512, 1991.

[33] R.S. Cost, T. Finin, Y. Labrou, X. Laun, Y. Peng, and I. Soboroff. An agent-based infrastructure for enterprise integration. In *First International Symposium on Agent Systems and Applications (ASA '99). Palm Springs. California*, 1999.

[34] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria*, July 2002.

[35] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), California, USA*, August 2001.

[36] A. Joshi D. Chakraborty and Y. Yesha. An Integrated Service Discovery and Routing Protocol for Ad hoc Networks. *Ad Hoc Networks Journal, Elsevier Science, To Appear*, March 2003.

[37] T. Finin D. Khushraj and A. Joshi. Semantic Tuple Spaces: A Coordination Infrastructure in Mobile Environments. In *Second International Semantic Web Conference (ISWC), Florida, USA*, October 2003.

[38] C. Dabrowski and K. Mills. Understanding self-healing in service discovery systems. In *First ACM SigSoft Workshop on Self-healing Systems (WOSS), South Carolina*, pages 15–20, November 2002.

[39] C. Dabrowski, K. Mills, and J. Elder. Understanding consistency maintenance in service discovery architectures in response to message loss. In *4th International Workshop on Active Middleware Services, IEEE Computer Society*, pages 51–60, July 2002.

[40] DARPA Agent Markup Language for Services Specification Draft 0.5. `http://www.daml.org/services/daml-s/2001/05/`, May 2001.

[41] M. Dean, D. Connolly, F. V. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web Ontology Language (OWL) Reference Version 1.0. http://www.w3.org/TR/2002/WD-owl-ref-20021112/, 2002.

[42] G. Weikum. Editor. Special Issue on Infrastructure for Advanced e-Services. *IEEE Data Engineering Bulletin, 24(1)*, March 2001.

[43] K. Erol, J. Hendler, and D. Nau. HTN planning: Complexity and expressivity. In *International Conference on Artificial Intelligence (AAAI)*, 1994.

[44] R. Dube et. al. Signal stability based adaptive routing for ad-hoc mobile networks. *IEEE. Personal Communications.*, pages 36–45, February 1997.

[45] T. Finin et. al. *Draft Specification of the KQML Agent-Communication Language*. DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1993.

[46] N. Feng. Software hot-swapping technology design. Technical Report SCE-99-04, Systems and Computer Engineering, Carleton University, June 1999.

[47] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, 1997.

[48] P.C. Fishburn, A.M. Odlyzko, and R.C. Siders. Internet publishing and beyond: The economics of digital information and intellectual property. In *Cambridge, MA: MIT Press*, 1997.

[49] XLANG. Web Services for Business Process Design. `http://xml.coverpages.org/xlang.html`, 2001.

[50] BPEL4WS. Business Process Execution Language for Web Services. `http://xml.coverpages.org/bpel4ws.html`, 2002.

[51] D. Gross and C. Harris. *"Fundamentals in Queueing Theory, 2nd Edition"*. "Wiley", 1985.

[52] B. Grosz and S. Kraus. Collaborative plans for complex group action. In *Artificial Intelligence, 86*, 1996.

[53] E. Guttman, C. Perkins, and J. Veizades. RFC 2165: Service Location Protocol, June 1997.

[54] R. Hall. Agent-based software configuration and deployment, 1999.

[55] S. Helal, N. Desai, and C. Lee. Konark-A Service Discovery and Delivery Protocol for Ad-hoc Networks. In *Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans, USA*, March 2003.

[56] R.G. Hercock, J. C. Collis, and D. T. Ndumu. Co-operating mobile agents for distributed parallel processing. In *Agents. Washington. Seattle.*, 1999.

[57] Hewlett-Packard. *E-Speak Architectural Specification*, version beta 2.2 edition, December 1999.

[58] T. Hodes and R. Katz et. al. An Architecture for a Secure Service Discovery Service. In *Fifth International Conference of Mobile Computing and Networks, Washington, USA*, August 1999.

[59] R. Ierusalimschy, L. H. Figueiredo, and W. Celes. Lua- an extensible extension language. *Software: Practice and Experience, 26(6)*, 1996.

[60] N.R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. In *Artificial Intelligence 75(2)*, pages 195–240, 1995.

[61] N.R. Jennings and M. Wooldridge. Agent-oriented software engineering. In *12th Int Conf on Industrial and Engineering Applications of AI. Cairo. Egypt.*, 1999.

[62] R. John. UPnP, Jini and Salutaion - A Look at some popular Coordination Frameworks for Future Network Devices. Technical report, California Software Labs, 1999. http://www.cswl.com/whiteppr/tech/upnp.html.

[63] D.B. Johnson and D.A Maltz. The Dynamic Source Routing Protocol for Mobile Ad-hoc Networks. *Mobile Computing, Kluwer Academic Publishers*, pages 153–181, 1996.

[64] L.R Ford Jr. and D.R. Fulkerson. Flow in networks. Princeton University Press, 1962.

[65] L. Kagal, V. Korolev, H. Chen, A. Joshi, and T. Finin. Centaurus: A framework for intelligent services in a mobile environment. In *International Workshop on Smart Appliances and Wearable Computing (IWSAWC). Phoenix. Arizona*, 2001.

[66] R.H. Katz, Eric. A. Brewer, and Z.M. Mao. Fault-tolerant, Scalable, Wide-Area Internet Service Composition. Technical Report. UCB/CSD-1-1129. CS Division. EECS Department. UC. Berkeley, January 2001.

[67] V. Kawadia and P.R Kumar. A cautionary perspective on cross layer design. In *Submitted to IEEE Wireless Communications Magazine*, July 2003.

[68] J. N. Kok and K. Sere. Distributed service composition. In *Technical Report No. 256. Turku Centre for Computer Science. Finland.*, March 1999.

[69] T. Finin L. Kagal and A. Joshi. Trust-Based Security in Pervasive Computing Environments. In *IEEE Computer*, December 2001.

[70] DARPA Agent Markup Language. `http://www.daml.org`.

[71] DARPA Agent Markup Language and Ontology Inference Layer. `http://www.daml.org/2001/03/daml+oil.daml`.

[72] O. Lassila and R. Swick. Resource Description Framework. `http://www.w3.org/TR/1999/REC/rdf-syntax-19990222`, February 1999.

[73] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *16th ACM International Conference on SuperComputing, New York, USA*, June 2002.

[74] N. Malpani, J. L. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications. Boston. MA.*, 2000.

[75] D. Mennie and B. Pagurek. An Architecture to Support Dynamic Composition of Service Components. In *5th International Workshop on Component-Oriented Programming (WCOP), Sophia Antipolis, France*, June 2000.

[76] S. Murphy and J.J Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *ACM Mobile Networks and Applications Journal. Special Issue on Routing in Mobile Communication Networks*, pages 183–97, October 1996.

[77] B. C. Neuman and Ts'o Theodore. Kerberos: An authentication service for computer networks. *IEEE Communications*, 1994.

[78] Meta Object Facility Resource Page. `http://www.omg.org/technology/cwm`.

[79] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara. The daml-s virtual machine. In *Proc. 2nd International Semantic Web Conference (ISWC)*, October 2003.

[80] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Kobe, Japan*, April 1997.

[81] F. Perich. On peer-to-peer data management in pervasive computing environments. Ph.D. Dissertation. Computer Science and Engineering Department. University of Maryland, Baltimore County.

[82] F. Perich, S. Avancha, D. Chakraborty, A. Joshi, and Y. Yesha. Profile Driven Data Management for Pervasive Environments. In *13th International Conference on Database and Expert Systems Applications, (DEXA), Aix-en-Provence. France*, September 2002.

[83] F. Perich, A. Joshi, Y. Yesha, and T. Finin. Collaborative Joins in a Pervasive Computing Environment. *VLDB Journal*, December 2004.

[84] C.E Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. *Computer Communications Review*, pages 234–44, October 1994.

[85] 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, February 1999.

[86] C.E Perkins, E.M Royer, and S. Das. Ad hoc on-demand distance vector protocol. In *IETF Internet Draft. Version 12.*, November 2002.

[87] C. Pullela, L. Xu, D. Chakraborty, and A. Joshi. Component based architecture for mobile information access. In *Workshop in conjunction with International Conference on Parallel Processing (ICPP).*, August 2000.

[88] D.V. Pynadath, M. Tambe, and N. Chauvat. Building dynamic organizations of distributed, heterogeneous agents. In *Agents. Barcelona. Spain*, 2000.

[89] P. Queloz and A. Villazon. Composition of services with mobile code. In *First International Symposium on Agent Systems and Applications Third International Symposium on Mobile Agents. Palm Springs. California.*, 1999.

[90] A. Ranganathan and R. H. Campbell. Advertising in a Pervasive Environment. In *2'nd ACM International Workshop on Mobile Commerce*, pages 10–14, September 2002.

[91] O. Ratsimor, A. Joshi, T. Finin, and Y. Yesha. eNcentive: A Framework for Intelligent Marketing in Mobile Peer-To-Peer Environments. In *The 5th International Conference on Electronic Commerce (ICEC)*, October 2003.

[92] M. Ripeanu and I. Foster. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. In *1st International Workshop on Peer-to-Peer Systems*, March 2002.

[93] "S. Ross". *"Introduction to Probability Models, 8th Edition"*. "Academic Press", 2003.

[94] E. M. Royer and C. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications.*, April 1999.

[95] Y. Chen S. Ni, Y. Tseng and J. Sheu. The Broadcast Storm Problem in a Mobile Ad-hoc Network. In *MobiCom. Seattle. Washington*, 1999.

[96] The Salutation Consortium Inc 1999. Salutation Architecture Specification (Part 1), Version 2.1 Edition. `http://www.salutation.org`.

[97] J. Sauver. Percentage of Total Internet Traffic Consisting of Kazaa/Morpheus/FastTrack. `http://darkwing.uoregon.edu/joe/kazaa.html`, 2002.

[98] I. Sommerville. Software engineering. In *Addison-Wesley Pub. Co.*, 1982.

[99] Bluetooth Specification. `http://www.bluetooth.org/specifications.html`.

[100] FIPA Agent Management Specification. `http://www.fipa.org/specs/fipa00023`.

[101] FIPA Communicative Act Library Specification. `http://www.fipa.org/specs/fipa00037`.

[102] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 1997.

[103] D. Tang, C. Chang, K. Tanaka, and M. Baker. Resource Discovery in Ad hoc Networks. Technical report, Stanford University, August 1998. CSL-TR-98-769.

[104] Computer Science Division The Ninja Project. University of California, Berkeley. `http://ninja.cs.berkeley.edu`.

[105] C. Thompson, P. Pazandak, V. Vasudevan, F. Manola, G. Hansen, and T. Bannon. Intermediary architecture: Interposing middleware object services between web client and server. In *Workshop on Compositional Software Architectures. Monterey. California*, 1998.

[106] C.K. Toh. A novel distributed routing protocol to support ad-hoc mobile computing. In *IEEE 15th International Phoenix Conference on Computer and Communications*, pages 480–86, March 1996.

[107] J. Undercoffer, F. Perich, A. Cedilnik, L. Kagal, A. Joshi, and T. Finin. A Secure Infrastructure for Service Discovery and Access in Pervasive Computing. *Journal on Mobile Networking and Applications (MONET), Special Issue on Security in Mobile Computing Environments*, October 2003.

[108] Universal Description Discovery and Integration Platform. `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`, September 2000.

[109] E. Schwartz W. Adjie-Winoto and H. Balakrishnan. The Design and Implementation of an Intentional Naming System. In *In Proceedings of the Symposium on Operating Systems Principles, South Carolina, USA*, December 1999.

[110] M. Wooldridge and N.R Jennings. Intelligence agents: Theory and practice. In *Knowledge Engineering Review, 10(2)*, pages 115–52, 1995.

[111] WSDL. Web Services Description Language 1.1. `http://www.w3.org/TR/wsdl`, March 2001.

[112] WSFL. Web Services Flow Language. `http://xml.coverpages.org/wsfl.html`.

[113] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition Using SHOP2. In *2nd International Semantic Web Conference (ISWC), Florida, USA*, October 2003.

[114] R. Bagrodia X. Zeng and M. Gerla. GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks. *12th Workshop on Parallel and Distributed Simulations, Alberta, Canada*, 1998.

[115] XML. Extensible Markup Language. `http://www.w3c.org/XML/`.