Technical Report TR-CS-01-20

Service Composition in Ad-hoc Environments

Ph.D Dissertation Proposal

Dipanjan Chakraborty
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD 21250

{dchakr1}@cs.umbc.edu

December 21, 2001

Abstract

Service Composition refers to the technique of creating complex services with the help of smaller, simpler and easily executable lightweight services or components. The spurt of e-services on the web has recently increased the importance of service composition. There has been some research in trying to leverage the wide array of e-services available over the network to be able to provide more customized, complex services to e-customers. In the recent past, we have also seen the increasing use of mobile devices. We have seen the deployment of ad hoc networks like Bluetooth. We envisage that in the future, service composition will play a crucial role in providing access to complex services to a mobile device by utilizing resources available in its vicinity.

Currently existing service composition systems broadly address the problems associated with composing various services that are available over the fixed network infrastructure. In the future, along with services in the fixed network infrastructure, we would also be able to access services present in the various mobile devices in our vicinity. Ad-hoc networking technologies like Bluetooth will help in connecting these mobile devices to each other. The currently existing service composition systems do not work in the domain of composing services over ad-hoc networks.

We propose to investigate different ways in which services can be composed in an ad-hoc environment. In particular, the research aims at proposing and implementing principles and architectures of service composition in ad-hoc environments. The research aims at identifying key issues related to the efficient implementation of a composition platform in a dynamic environment. We also propose to test and judge the suitability of the principles in various kinds of ad-hoc networks.

Contents

1 Introduction

Service Composition refers to the technique of composing the functionalities of relatively simpler services to produce a "meaningful", arbitrarily complex service. A "service" may be defined as any software program, which accepts appropriate input data or query and produces an output. Ample such services can be found over the Internet. For example, the Google search engine may be thought of as a "service" that takes one or more keywords as its input and produces as its output a list of web pages where those keywords occur. We can consider mapquest.com to be another service that provides a plethora of cartographic information. In the world of software agents [69, 53, 43], we find examples of agent-services which can provide customized information to users. These agents can also be modeled as services capable of doing specific work for a user. Apart from services that can act as information providers, there may be other types of services too. There may be services that offer certain computation facilities [48]. Examples of such services are statistical problem solvers (e.g., a service that determines the median of a certain numeric distribution) or data stream modifiers. Systems like Legion [84], Globus [32] or SciencePad [24] offer good examples where computation nodes are distributed across the network.

Services thus are capable of providing information (may be customized) as well as computation facilities to a client. Recent years have seen the proliferation of various types of wireless handheld devices such as PalmPilots and cell phones. To realize the dream of ubiquitous information access, we need to provide the PDAs access to the different heterogeneous services in a uniform and unhindered manner. Typical handheld devices are significantly resource poor compared to their wired counterparts. They are capable of obtaining data from different service providers using the wireless networking infrastructure. However, in situations where the client has to apply some computation function after getting data from a service to obtain the required result, such devices lack the requisite computational capability most of the times. They have to find and use different "computation" services to compute the additional information they want.

Along with the development of different mobile devices, we have also seen the concomitant development of different networking technologies. Bluetooth-like systems that provide short range, moderate bandwidth connections and Wireless LANs (IEEE 802.11b) have been deployed in the industry and academia.

Another important perspective, which affects future research directions in wireless technology, is the study of the usage patterns of different mobile devices and how they change with the advent of new technologies. In the first phase, mobile devices were used as personal organizers and their basic functionality was very limited. PalmPilots were used to keep personal information and cell phones were used to make phone calls only. In the second phase, we observe that apart from being used as personal organizers, the PDAs are also used to access the Internet and carry out electronic transactions. Different e-commerce applications that take advantage of wireless networks are being developed. We are familiar with ads of people buying commodities via their cell phones from the beach. Cell phones or PDAs have effectively become mobile storefronts for e-tailers. This approach essentially treats palmtop devices as consumers of goods or information. The information or goods come from servers on the wired side. This approach is based on a client-proxy-server model [13, 86, 11, 33, 49, 29] and has been developed by the academia over the last five to six years in contexts such as web access from mobile platforms. Applications of this change in the usage pattern of mobile devices have percolated into the industry. However, from the research perspective, it is interesting to note that interaction between a "client" and a "service" is carried out on a one-to-one basis and the interaction is limited to these two entities.

Following the logical path, we observe that that the mobile user may require cooperation of multiple services residing in the wired infrastructure at the same time to satisfy its request. It is possible that the information required by a mobile user may not be available per se but needs to be computed dynamically using various resources (software, data or hardware services) distributed over the network. This is referred to as "service composition in the wired infrastructure". In section 3, we describe the current work in service composition including our own work [68].Thus, to provide ubiquitous information access to the mobile users, the different static web-services need to be composed as and when necessary. We explain the scenario with an example.

Consider a situation where you are traveling in a car with a GPS enabled navigation system (which is a service) to provide location information. Your friend (the navigator) suddenly becomes ill. You decide that you have to take him to the nearest hospital. You want to find out the route to the nearest local hospital. The navigation system has the necessary resources to provide you with a route to the hospital. However, the traffic conditions on the road are really bad. Under these conditions you might be interested in taking the least crowded road to the hospital. The GPS service in your car has the capability of providing the coordinates of the current location of the car. The navigation system inside cars these days usually has the ancillary data required to provide you with the shortest route to any particular place. However, it definitely would not be able to provide you with the optimal route taking into consideration such dynamically changing parameters like the traffic condition on different roads leading to the hospital. Even if we discovered one service (an information provider) willing to give automatic updates of traffic conditions on different roads, the system in the car may or may not have the necessary compute power or software service to calculate the route. To further emphasize the point, consider that your car does not have a navigation system nor is it GPS enabled. You have a PDA with a GPS service capable of giving you the location information. Thus you cannot count on the support of the navigation system to provide you with the route to the local hospital. If we technically analyze the query, we observe that to compute the optimal route to the hospital with respect to traffic conditions, we need the following services:

- Service that provides dynamic traffic information on all roads leading to the hospital from where the car is
- Service that provides a road map of the area
- Service that provides GPS locations of the car and the hospital
- Service that provides a computation platform to compute the optimal route to the hospital with respect to the traffic information on different roads

The standard network-based solution one can think of is to implement a single powerful centralized service provider to compute the result of such a query. The service would be able to obtain dynamic traffic updates from different streets with the help of sensors or other hardware/software components. It would have a database of road maps of the nearby area. It would also have the software component required to compute the optimal route to the destination with respect to the traffic conditions. Apart from the usual problem of scalability in such a solution, we also assume the existence a complex service capable of doing such a specific task, to be available in the network. Wireless link related problems like intermittent disconnection or high latency in transferring data from the fixed network infrastructure may also prevent you from getting the data at the right time. Moreover, in the fixed network infrastructure the probability of finding such a complex service serving a customized function is very low. However, we have a much better chance of finding different services capable of doing smaller generic tasks, like providing the road map of the nearby locality or traffic information on certain streets. This is because the applicability of such "atomic" services ranges from being used as a single service to being part of a complex service in conjunction with other "atomic" services too. The present-day Internet clearly lacks the ability to compose these heterogeneous software (and hardware) services to collaboratively provide information/computation services to the mobile user.

There has been considerable work in addressing the issues involved in composing different services that are available in the fixed network infrastructure. We shall describe them in section 3. The essential technique of composing e-services is to have a "centralized" mediator sitting in between the client and the different services. The mediator is also referred to as the "broker" or "middleware" in some literature. The client discovers a mediator using standard wired infrastructure-based service discovery mechanisms. The client submits a composite query to the mediator. The mediator uses state-of-the-art service discovery techniques such as Jini [3], Salutation or UPnP [71] to discover the different services required to execute such a request. The mediator helps in the collaboration of the different services and uses sophisticated mechanisms to optimize the cost required to execute the composite request. This enables these entities to co-operate with each other using this system to dynamically compute information, which is not a priori available to the mobile device.

Current service composition platforms have been designed with the inherent assumption that the e-services are resident in the fixed network infrastructure. Thus, they are running on a relatively stable platform and can access each other using relatively high bandwidth communication channels. The composition platforms are based on a centralized manager or broker which manages the various issues like service path creation [23], delegation of service discovery to the proper discovery manager, appropriate combination of different services and management of the information flow between service components. The client querying the system could be a client that is wired to the Internet. It could even be a wireless client too. In the event of the client being a wireless client, several other factors like bandwidth, delay, disconnection, display attributes of the mobile device or resources on the mobile device need to be taken into consideration by the mediator. Our previous work in the field of service composition [68] centers around providing a mobile device access to composite services in the wired network with the help of a "mobile aware" broker. In a mobile environment where the different hosts are connected together by ad-hoc networks and are available for a limited time, we cannot assume the presence of a fixed broker service that will be available in all ad-hoc environments to help us utilize the services in our vicinity.

We now paint a vision picture of the future by extrapolating the usage pattern of the mobile devices a bit further. We envisage that these mobile devices (and many others still to come into the market) would be anywhere and everywhere. The trend suggests that in the future, each person will have multiple wirelessly connected devices on them (e.g. a Bluetooth cell phone and a PalmPilot and a Infrared enabled wrist-watch). These mobile devices would not only be able to communicate with each other using ad-hoc networking or point-to-point connections, but would also be able to collect necessary information from adjacent devices physically present on some other person. These devices would be heterogeneous and would have varied information on each of them. Multiple services would exist on each of these devices. These services would be capable of doing different specific tasks. Using wireless/ad-hoc network connectivity, every mobile device would try to utilize these services available in its vicinity to its maximum advantage. In the earlier example, we assumed that the PDA of the driver would have a connection to the fixed network infrastructure. Due to certain drawbacks explained earlier, it might not be a good proposition to obtain this information from the wired Internet. A connection to the fixed network infrastructure might not be available too. On the other hand, in the resource-rich neighborhood of the future, it may be possible to obtain such information from the mobile devices in the vicinity of the driver. For example, the PDA inside a car coming from the opposite direction might have a service that is willing to give information about any important landmark (a hospital) it has seen on the way. The navigation system (another service) inside another car passing by might be able to provide the map of the local area. These two services can be combined to locate a hospital in the neighborhood. The computation service required to compute the information may be available in yet another mobile device (a ThinkPad) that is lying at the back seat of the car. Notice that in this situation, the driver could not compute the "optimal route" to the "nearest" hospital. However, the purpose of the driver is served since he could get an approximate answer from the services in the vicinity. There might also be situations when multiple services in the vicinity of the car have information about different hospitals in the neighborhood. In such an event, the computation service could be used to find the nearest hospital. Information about traffic could also be obtained from the information services inside the mobile devices of cars passing by to derive some "approximate" idea about the least crowded path to a hospital. Moreover, the correctness of this information available from services in the vicinity may be more than the correctness of a fixed information provider in the Internet. We begin to see how service composition in this ad-hoc environment may leverage the maximum benefit from this rich environment.

To the best of our knowledge, service composition has not been explored in such an ad-hoc networking domain. It will however play a very crucial role in utilizing the resources and computation platforms available in the vicinity of a mobile device. It would also help in the proliferation of peer-to-peer mobile commerce and m-transactions in future. Thus, since this "vicinity" will become increasingly resource-rich with time, it is worthwhile exploring different ways of service composition in such an environment.

The rest of the proposal is organized as follows. In section 2 we present a precise description of the problem and the key terms involved in it. In section 3 we present the literature on service composition and other related technologies involved in an ad-hoc environment. In section 4 we present the preliminary design of three different techniques for ad-hoc service composition. In section 5 we present a very brief summary of the initial thoughts on implementation.

2 Problem Boundary Definition

In this section, we define the problem domain and specify its boundary. We try to come up with a functional definition of the problem and direction of the research:

Fixed broker-based or fixed central-entity based service composition techniques 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 proposed research intends to study different approaches that enable service composition in a dynamic ad-hoc environment. The research aims at assessing the computational and technical feasibility as well as sustainability of these approaches in a dynamic, ad-hoc and ubiquitous environment.

At the outset it is essential that we lay out a clear formalism of the key terms involved in the research to avoid further ambiguity. The following operational definitions have been designed in order to narrow down the scope of this research.

At its lowest level of granularity, the term *service* can be defined as any hardware or software entity that resides on any mobile device or platform and has distinct functional attributes that characterizes it. Multiple services can combine themselves to produce a qualitatively different new service. The functionality of the service should be expressible in a semantic manner [55, 31]. The service could be accessible to other systems using agent-based message passing paradigm [77] or Remote Procedure Call [80] mechanisms. The service itself could also be a piece of mobile code in a distributed environment.

The term *ad-hoc* can be visualized from multiple point of views. However, we will stick to a narrow definition of ad-hoc environment especially in the context of mobile computing. An ad-hoc environment is defined as a confederation of heterogeneous mobile systems that is established on the fly and is capable of sustaining itself in a dynamically changing environment. As the environment changes, the interconnections between the nodes in the confederation may change. The ad-hoc environment maintains the confederation by continuously reorganizing itself.

Service composition can be defined as a dynamic integration of multiple services available in an ad-hoc environment in response to a request from a client.

3 Background and Related Work

The original concept behind service 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 [75]. There has been work in upgrading these software components dynamically at run-time [27] 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, 39, 8] 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 Composition in ad-hoc environments is one way of re-using the software that is available in vicinity of a device. However, in an ad-hoc environment, the rigid structural framework of software re-use fails when the multiple components are loosely defined and is in a state of continuous change.

There has been considerable research work in the field of service composition in the context of sedentary networks to solve composition related issues associated with the fixed network infrastructure model. In this section, we shall try to present a comprehensive survey of the associated literature and critically evaluate the design models used in various composition systems with respect to their applicability in an ad-hoc environment. An inherent assumption behind the success of service composition in ad-hoc networks is that there is a relatively stable and reliable ad-hoc networking infrastructure available to the composition platform. We present a brief overview of the recent ad-hoc networking technologies that will be essential in providing the required communication infrastructure to interconnect heterogeneous mobile devices.
Service Composition inherently assumes that there is an efficient mechanism to discover the services that are being used to compose a complex service. In this section we shall endeavor to present an overview of the currently existing commercial and research-level service discovery systems that have been implemented for the wired and wireless networking infrastructure. After discussing the prominent service composition systems, we present an overview of the literature in multi-agents, semantic expression languages and other areas related to this field.

3.1 Ad-hoc Network Technology

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 [72] in this environment can be categorized as:

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

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)[65] is a table-driven algorithm based on the Bellman-ford routing mechanism [50]. 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) [19] 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) [59] is another example of a table-based ad-hoc routing protocol.

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) [66] 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) [47] 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) [64], Associativity-Based Routing (ABR) [83], Signal Stability Routing (SBR) [26] 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.

As far as the development of commercial ad-hoc networking systems are concerned, Bluetooth [63] 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.

Broadcasting in ad-hoc networks has been a research issue due to the various problems associated with the movement of nodes, bandwidth considerations, fast deployment, fault-tolerance and the broadcast storm problem [73]. Researchers have been trying to address the various issues in this domain. Basagni [7] present a deterministic mechanism for delivery of broadcast packets in ad-hoc networks which is mobility-transparent and also guarantees the delivery of packets without the need to recompute transmission schedules when the topology changes. Pagani and Rossi [61] have described a reliable and fault-tolerant way to deliver broadcast packets in ad-hoc networks. The scheme ensures that every host in the network delivers the same set of messages to the upper layer.

Ad-hoc networks can be categorized based on the mobility of the different participating nodes. When heterogeneous mobile devices are connected to each other by the virtue of their physical proximity with each other, an ad-hoc network is formed. Thus network formed between different wearable computers on the body of a person is an ad-hoc network. However the different nodes in such a network are stationary in relation to each other and hence the network is stable even though they are connected wirelessly. The network formed between devices on different persons who are passing by in the street is a very fast moving and unstable ad-hoc network. Similar examples of relatively stable or unstable ad-hoc networks can be drawn from daily life. Network formed between mobile devices of persons in a train or a bus is a stable ad-hoc environment. The network formed between mobile devices in a shopping plaza is a relatively unstable ad-hoc environment.

3.2 Service Discovery Systems

A service on a mobile device needs to discover the presence of other mobile services in its vicinity to carry out transactions. This is the problem of service discovery. In this section we present an overview of state-of-the-art systems.

3.2.1 Service Location Protocol (SLP)

The Service Location Protocol (SLP) [38] 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.

3.2.2 Jini

Jini [71]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.

3.2.3 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) [71] 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.

3.2.4 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.

3.2.5 Ninja Secure Service Discovery System

The Ninja Secure Service Discovery System (SDS) [41] 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.

3.2.6 DReggie: The Jini-based Semantic Service Discovery System

The project DReggie [16] is an attempt to take Jini and similar service discovery systems beyond their simple syntax-based service matching techniques and enable semantic matching capabilities by adding highly expressive service description facilities. Semantic service matching widens the scope of a certain service discovery request by being able to locate services, which "approximately" match the given request. Service matching can be carried out based on the functional description of the service. Thus even though a service, which exactly matches a given request, may not be available in the environment, semantic discovery has the ability to find out services which have been advertised in a different way but carries out the same or at least similar functionality. DReggie enables such discovery by adding the facility of rich service description using DAML [55] and introducing intelligent reasoning modules to carry out the semantic matching process. At the heart of DReggie is an enhanced Jini Lookup Service (JLS) that enables smart discovery of Jini-enabled services. DReggie also helps clients discover services in a manner unchanged from the existing Jini Lookup and Discovery infrastructure. In addition, it allows services to advertise their capabilities, constraints and system features in a well-structured, descriptive format using DAML.

Any service using the DReggie infrastructure is described using DAML in terms of its capabilities, requirements and service attributes. When a service registers with the enhanced Jini Lookup Server (JLS), it registers a DAML description entry as well. This object (DAML DOM object) is stored in the lookup server and updated whenever the service registers again. When a client wants to use a service, the client creates a DAML DOM object that describes the desired service along with its constraints. The DOM object provided by the client contains "approximate" information about the type of service the client is attempting to discover. The DReggie lookup server contains two different modules to carry out the matching process of the service request with descriptions of the advertised services: a simple java-based matching module and an advanced prolog-based reasoning module. In our initial implementation of the infrastructure, the simple java-based reasoner has been integrated with the JLS. The java-based module compares the two DOM objects and finds either exact or approximate service matches. The other advanced reasoning module, the prolog engine service is capable of more sophisticated matching based on DAML service profiles. In our future work, we aim to integrate the Prolog engine as a module with the Jini Lookup Server to discover m-services more efficiently and successfully.

3.2.7 Other Service Discovery Systems

The Bluetooth [76] 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 [51]. 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 [60] authentication. The architecture is based on a client-registry-service type model.

The Universal Description, Discovery and Integration (UDDI)[22] specification is a company initiative in creating a global, platform-independent, open framework to enable businesses to discover each other, define how they interact over the Internet and share information in a global registry that will more rapidly accelerate the global adoption of B2B eCommerce. The architecture uses SOAP [67] that allows one program to invoke service interfaces across the Internet in a language independent and distributed manner. SOAP is built on the Extensible Markup Language (XML) also defines a simple way to package information for exchange across system boundaries. SOAP also defines a way to make remote procedure calls between systems in a manner independent of programming language or operating systems. The architecture follows a centralized model where all business services register to a common UDDI registry service. This enables business clients to find businesses of their choice with greater flexibility and ease.

3.3 Existing Service Composition Systems

In this section, we discuss some ongoing research work in this area of dynamic service composition.

3.3.1 Dynamic Service Composition architecture at Carleton University, Canada

Work on dynamic software and service composition has been going on at the Systems and Computer Engineering Department, Carleton University, Canada [58]. Their work on service composition is a successor of their research in the field of dynamic software component upgradation at runtime called software hot-swapping [27] and componentization of software for high-availability systems. They have targeted the networked e-services as the software components that can be meaningfully composed to provide complex and advanced services by reusing the present resources. Their research tries to identify key issues related to service composition in an environment of distributed networked services. They have identified two different ways to carry out composition of heterogeneous services.

- Interface fusion/Composite Service Interface In this method, multiple service components communicate with one another as separate independent entities to provide an enhanced service that can be accessed by a common interface. This new interface will properly reflect the appropriate services that can be composed at a particular instant. An extended Facade Design pattern [35] is used to create such a dynamically changing interface. The interface also controls the routing of appropriate requests to the proper services and acts as a central coordinator. Different network components are available using the Jini Component discovery architecture [3]. This method of composite interface formation identifies a good way of exposing the necessary available services to a user. The primary advantage of this technique is the speed at which a composite service can be created and exported as an interface. This is because a new component does not need to be constructed dynamically. Rather, only the interface of that component needs to be known to construct the composite interface.

- Standalone Composite Service To prevent the necessary burden of arbitrating over the composite service interface and dynamically updating it, the other obvious path to follow is to create a standalone composite service by using the services available. However, this type of service creation is only necessary when speed of aggregation is not an important factor and the service to be composed has a widespread use. A single monolithic service can be composed by dynamic assembly of service components in a way analogous to the assembly of pipes and filters. In this case, all the service components remain independent. In the second approach, service code or logic of each component is assembled into the new composite service. Only methods specific to the composite service are imported and assembled together. The primary advantage of such composite service creation is that the service code can be reused and composed easily with other services. Here, all the code executes at the same location. This increases efficiency and response time since it does not require the costly message transmission and coordination between distributed services. They have used XML to appropriately describe the capabilities, constraints, input, output and dependencies of each service. The service component discovery is facilitated using the Jini service discovery framework.

3.3.2 eFlow: HP laboratories

eFlow [15] 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. The eFlow platform provides the necessary flexibility to service composition with the help of the following properties:

- Adaptive Service Process: eFlow engine offers the facility of being able to plug in new services offering the same type of service even when the engine is processing a composite service. eFlow enables such adaptability with several features like dynamic service discovery, multi-service nodes and generic nodes. Dynamic service discovery layer and service brokers find out new services as they come up. This enables plugging in new services to a running process if the new service suits the running process better than the one it is replacing. This ties in well with the characteristics of the rapidly changing Internet environment. Each service node specifies a service selection rule. This rule is compliant with the language understood by the service broker. In return eFlow expects the service broker to return an XML document providing the necessary details of the service, e.g. input/output information, URI to contact the service, billing and payment information. Multi-service nodes come into action when there is need for invoking multiple threads of the same service with different input parameters. eFlow provides definition of such multi-service nodes in its service definition. Personalization is supported with the help of generic service nodes. Generic service nodes are not statically bounded to a particular set of services. Instead they can be configured to consist of any set of services either at process instantiation time or at runtime.

- Dynamic Service flow modification: eFlow engine has the capability of dynamically changing the way a composite service is produced. Thus the process schema (definition of a composite service) can be changed while the service is already running. This involves complex state migration of the running process instances. This functionality is required when complex e-commerce transactions need to be modified in the course. Two types of service process modifications introduced are ad-hoc changes and bulk changes. Ad-hoc change is applied to a single running process instance and bulk changes are applied to a subset of instances of a service process. Such modification is possible with the help of a migration manage, which, apart from doing various other functions, also verifies whether the behavioral consistency is maintained between the original service execution pattern and the newly changed one.

3.3.3 Ninja Service Composition Platform: UC, Berkeley

Researchers at UC, Berkeley have developed a platform to enable dynamic service composition in the Ninja Project [23, 52]. 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. The composition architecture involves the interplay of the following entities.

- Internet Service Providers: Internet Service Providers are the main resource providers providing the information requested by a query. A service is an Internet accessible application that is residing on a high-end machine. These services are strongly typed.

- Network Service Providers: Network Service Providers provide connection to the various devices through varying type of networks.

- Automatic Path Creation (APC) Service Providers: This service is the central service involved in coordinating the composition of the different strongly typed services. Once a request comes to this service, it figures out the different components required to execute it, discovers the physical services over the wide area, sends the required information and query to the respective service, monitors and regulates the execution flow of those services. In the Ninja platform, the APC service enables service access from diverse clients, dynamic service composition, data flow optimization and data path adaptability.

- Service Discovery Service Provider: Service Discovery Service Provider [41] is a directory type service, similar to the Jini lookup server or the SLP [38] directory service. Different Internet services advertise their capabilities in terms of structural and semantic information about their inputs and outputs to the service discovery service. The service discovery service also has authentication and access control features to prevent unauthorized access of an user to a service. It helps the APC services to find out physical services in the network.

- End Clients: The end clients are either fixed machines connected to the internet or mobile devices with access to the Internet. These devices have varying form factors, display types, processing power etc.

Central to the architecture is the concept of a service 'path'. When a APC service gets a request for a composite service execution, it figures the set of 'operators' (services which operate on the data flowing) and 'connectors' (network services which help in transporting the data from one operator to another that will have to be plugged in to the data path between the client and the actual service. The APC creates a logical path by searching over the graph of operator space using shortest path strategy. Then it creates a physical path where it finds the actual services to do the task. The APC is also responsible for path instantiation, execution and maintenance. Each APC establishes a 'control path' between the nodes and itself to coordinate each service. The connectors enable different type of network optimizations. Other operators involved in the system are Data format translation operators, Protocol conversion and Content transformation operators. APC also pays attention to the placement of operators when the actual service path spans over a wide area. The system ensures fault tolerance by using cluster of workstations to run the APC services and constant remote monitoring of the different operators executing in a certain path. In an earlier work, Katz and Hodes [42], defined the concept of making local services available to the client in an ad-hoc manner and also mentioned how different complex behaviors can be implemented using underlying composable objects. However, the system considered discovery of interfaces and inducting transduction protocols in an unix pipe-type manner to carry out the complex function. The system was targetted towards fixed network infrastructure also.

3.3.4 Dynamic Broker-based Service Composition Platform at UMBC

Our work [68] in the field of service composition centers on the main objective of enabling ubiquitous access to all sorts of information from a mobile device. Research in this area has focused on eliminating different wireless link related problems like bandwidth, disconnection etc. The information is also assumed to be available from static information providers residing in the fixed network infrastructure. We developed a broker-based layered service composition platform to enable a mobile host to access information which cannot be obtained 'as is' from a fixed information provider, but requires the collaboration of multiple services. The system has been developed using the Ronin Agent Framework and uses an enhanced Jini service for agent discovery.

The Ronin Agent Framework [17] is a Jini-based agent development framework that is designed to aid in the development of next generation smart distributed mobile systems. The Ronin Agent Framework

introduces a hybrid architecture, a composition of agent-oriented and service-oriented architectures, for deploying dynamic distributed systems. Ronin contains a number of features that distinguish it from comparable frameworks, including an Agent Communication Language (ACL) and network protocol independent communication infrastructure. It also includes an agent description facility that allows agents to discover and lookup each other, an agent proxy architecture that provides agent mobility behavior and customizable agent communication scheme. Ronin framework models Jini services as agents. Central to Ronin architecture is the notion of Ronin agent and it's corresponding agent deputy. An agent deputy acts as a front-end interface for the other agents in the system to communicate with the Ronin agent it represents.
We have used XReggie: an enhanced Jini Service Discovery Framework [18]to discover the different information agents in our system. The standard Jini infrastructure finds out services using strict interface or attribute based matching. We enhanced this rudimentary service discovery mechanism to do the discovery in a semantically richer way. We used XML (Extensible Markup Language) to describe a service by defining its capability, system features on which it is running, cost and whether the service is mobile. The enhanced Jini Lookup service finds out a service matching the request by comparing the XML attributes of each service and hence enables a wider range of service selection.
The architecture is divided into 4 distinct layers, viz. network layer, uniform agent communication layer, broker layer and dynamic service discovery layer (Figure 1). The Network layer is the lower most layer in

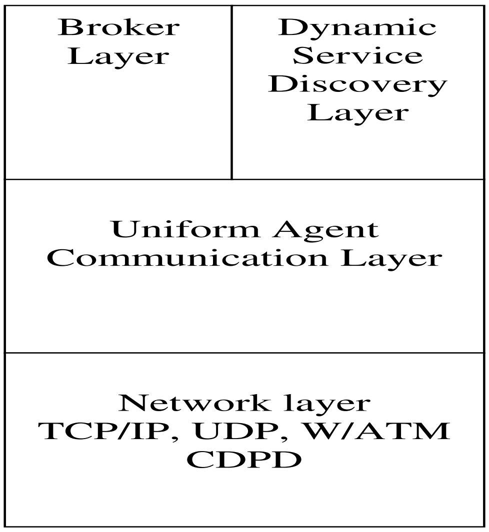

Figure 1: layered Architecture

our architecture. This layer represents the underlying communication infrastructure being used to communicate between different Ronin Agents in our system. The Ronin Framework allows the use of different protocols to transfer its messages. The network layer thus can encapsulate any communication mechanism and allows seamless integration of different network protocols in our system.
The Uniform Agent Communication Layer manages delivery of information to the different ronin agents in the system. This layer uses a proxy-based uniform agent communication mechanism and can efficiently handling message transport for the agent. The agent communication layer has the potential to offer different type of communication services to the agent above. Examples are simple store-and-forward messaging to handle network disconnection, message multiplexing, message filter or even message translators for mes-

sages coming from agents following a different language.

The Broker Layer is the core engine, which finds the individual services required in computing required information and manages the distribution of computation. This layer has the knowledge base of what services should interact to produce the required information to the end-user. The broker is responsible for task disintegration, hierarchical service screening, and calculation of the optimal computation path and distribution of computation. Task Disintegration refers to breaking down a complex job into smaller executable subcomponents. Right now, we maintain a database mapping a request or query to the various subcomponents that can be possibly run parallelly or in some order to execute the query. Hierarchical Service Screening is a technique that the broker applies to screen out the services that are incapable of executing subcomponents of the request at hand. Once the various services have been found out, the broker tries to find out an optimal way of executing the query. The broker uses the XML descriptions of the different service components (obtained from the XReggie lookup server) to decide on an optimal choice. Once the path has been figures out, the broker also handles distribution of computation by invoking the services of the different agents containing the subcomponents and managing the information flow. The Dynamic Service Discovery Layer essentially contains the XReggie system.

Very recently, we have enhanced the XReggie system to the DReggie system [16] where we use the DARPA Agent Markup Language (DAML) to describe services and their capabilities. The main advantage of our composition system is its ability to find an optimal way of executing a particular request. The optimization criterion could be monetary cost of executing a particular request, bandwidth usage for exchanging data or response time to execute a query. The broker considers code mobility as a factor to be considered along with data mobility. Thus it is able to optimize the bandwidth usage in situations where a huge amount of data is to be operated upon by a relatively small service component by dynamically relocating the component to the data platform. The enhanced service discovery mechanism also allows more flexibility in finding out different types of services and reasoning about their capabilities based on their XML descriptions. The broker distributes the computation amongst the components by considering several factors like CPU usage at the current platform, memory availability, monetary cost of executing the service etc.

3.4 Other Related Areas

Even though there has been very little work done in the field of service 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 [69] 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 [46] 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 [14] 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. in [14], an API and framework for mobile environments have been defined and implemented. The implementation addresses the above-mentioned issues of a mobile environment. The API is flexible so that features can be plugged in and out of any collaborative system. However, 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 [6] model multi-agent coordination based on a stochastic version of cellular automata. Treur et. al in [12] 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.

Some interesting work has been going on at Lockheed Martin Advanced Technology Lab on developing an autonomous agent collaboration and information dissemination framework [53]. The primary focus has been to develop a system on which mobile agents can interoperate with each other and share information. The system effectively minimizes bandwidth usage and cost of running mobile agents by dynamically modifying mobile agent itineraries based on the current information of the distribution of data in the system. The primary manager agent present in each node in the distributed system is the ATL Postmaster agent. Each Postmaster agent maintains state information about the itineraries of all the agents in the system, the different tasks they are supposed to perform, the location where the results of the execution of the tasks would be cached and also maintains link state information about the bandwidth between each host in the system. Whenever an agent comes to any machine, it first 'checks in' with the Postmaster agent and provides a description of its itinerary. The Postmaster agent then utilizes the knowledge it has, to determine the new itinerary, which might optimize the net cost for the agent to traverse through its itinerary and would also minimize bandwidth usage. The Postmaster also passes some additional information left back by previous agents and hence helps in agent collaboration. The Postmaster agent also has the capability of sending periodic updates on any change of the data to a remote agent upon request. The Postmaster Agent System assumes the existence of an agent system underneath itself and only helps in making the system more efficient in terms of bandwidth usage and cost of services.

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 [45] and rescue operations to psychology. Teamwork can be viewed in a hierarchical manner as well as in a distributed manner. Wooldridge and Jennings [85] have characterized team goals by terms like "joint intentions" [20]. Tambe [81] shows how the mental states can be coordinated using communication protocols. Teamwork can also be accomplished using sharedPlans [37] 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 [79] 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) [30, 28] 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 [78] 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 [9], LEAP [10] and JACKAL [21] 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 [57] 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 have presented two leader election algorithms for mobile ad hoc networks, the functioning of which is based on TORA [64] (Temporally Ordered Routing Algorithm). In section 4.3.1, we have described the dynamic central entity selection mechanism that endeavors to find a node capable of acting as the broker for a certain service composition. This mechanism 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 [36] 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 an ad hoc service composition environment, we want to take the maximum benefit from the resources and agents already available in our environment.

E-Speak [40] 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 [82] 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) [62] is unsuitable for ad-hoc environments due to its client-server type of architecture. Batista and Rodriguez [8] 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 [44].

Awduche, Gaylord and Ganz [4] have done some work on resource discovery in distributed systems with mobile hosts. The concept is similar to discovering services in mobile networks. They have tried to model the problem of discovering resources in a distributed system by a statistical bayesian framework. They have tried to find out the optimal way to locate resources taking wireless network related issues like disconnection and low bandwidth into account. The concept in this work however can't be 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 have addressed Service Composition from the theoretical point of view in [54]. Their work explores the use of action systems [5] 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.
Formal description techniques are required to give confidence and to prove correctness and safety of distributed systems. Action systems are such formal description mathematics. The paper discusses action systems and also proposes representation of services using a variation of action systems, viz. guarded variables. The two main development techniques used by action systems are Refinement and Decomposition. Refinement refers to the specification of abstract variables in a more concrete form and representation. Decomposition refers to the breaking up of action systems into sub-systems by partitioning the state variables and actions. The paper described guarded composition: a new modularization operator for action systems. Guarded Composition separates the coordination aspects with the computational aspects by moving the coordination aspects to the guards of the guarded composition. This is similar in theory to communicating sequential processes in distributed operating systems terminology. In our proposed research, we wish to study and implement ways in which service composition can be enabled in the ad-hoc networking domain.

Queloz and Villazon [70] have talked about composing the functionalities of mobile codes in the context of active networks. Mobile code can be made to migrate to a remote machine and execute 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 have not considered 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. Moreover, the mobile code that is used to as a service in executing a complex job is generated by the source of the request, an assumption that can be made about services existing in resource-poor mobile devices.

It is evident that in order to achieve dynamic service composition, the services have to be described in a machine interpretable and semantic manner. The DReggie system describes a way of discovering services by using semantic reasoning logic. There is a need for an ontology by which a standard service can be described in a machine interpretable and semantic manner. In [16], we have created a service ontology using DAML where the description of a service focuses on lower level systems oriented issues like inputs/outputs, cpu or memory requirements, platform dependencies etc. DAML-S [31] is a DAML based web service ontology developed by the combined effort of DARPA and W3C. It supplies the web service providers with a core set of ontologies to describe a service and its properties in a human as well as machine interpretable format. The ontology addresses four specific facets to be considered to describe a web-service. It provides support to enable automatic web service discovery, automatic web service execution, automatic web service composition and interoperation and automatic web service execution monitoring. The automatic web service composition further describes the pre and post conditions of the execution of a service that helps a machine in dynamically associating it with any other service. The DAML-S has a fair potential of being used as a good representation language to describe services on mobile devices.

The Web Services Description Language (WSDL) by W3C [1] is an XML format for describing network services as a set of endpoints operating on document-oriented or procedure-oriented messages. The operations and messages are described abstractly and bound to a concrete protocol later on. E.g. a message may be bound to the SOAP protocol. However, the current version of WSDL only supports message bindings with SOAP 1.1, HTTP GET/POST and MIME. The Web Services Flow Language (WSFL) [56] designed by IBM is an XML-based flow specification for web service composition. WSFL describes the interaction pattern of a collection of web services and also defines the usage pattern of each service so that the resulting composition describes how to achieve a particular business goal. Microsoft's Biztalk Framework [34] is an XML framework for application integration in e-commerce. It specifies a framework for business-to-business (B2B) document exchange within the enterprise and over the Internet.

3.5 Critical Evaluation of Existing Composition Techniques

We judge the suitability of the different service composition techniques with respect to their general architecture, fault tolerance and scalability factors, ability to cope with heterogeneous services and adaptivity to the dynamically changing environment of e-commerce. We observe that all most all the existing fixed infrastructure based service composition architectures follow a centralized model for managing and executing the composition. Figure 2 depicts the main design technique used in the architectures. These types of architectures in general make no assumption about the position of the client. The client thus can be a mobile client too. The services that are used in the composition are assumed to be available in the fixed network infrastructure.

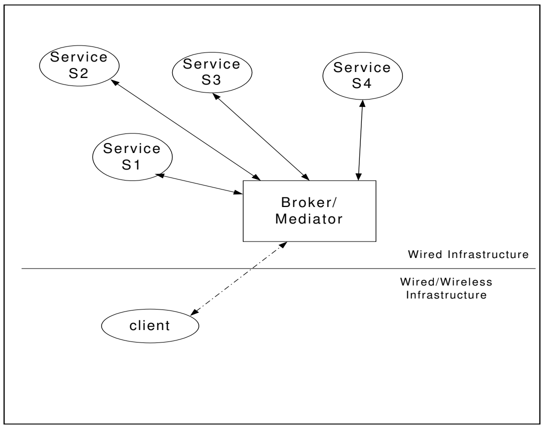

Figure 2: Central Entity based Service Composition Architecture

Agent-based collaboration platforms like that of the ATL postmaster system has the potential to make good service composition platforms. The Postmaster system helps agents to share and transfer information. The Postmaster system helps in delivering a partially executed query to another agent for further processing. Hence multiple agents may cooperate with each other to provide a composite functionality with the help of the Postmaster system.

The system also has some limitations. The state information of each and every agent has to be maintained by each and every postmaster agent to be able to make judiciary decisions. This limits the usage of such a system to only high-end machines with enough volatile memory capable of supporting such memory intensive processes. The agent state information also has to be disseminated to each postmaster in the system. This involves information broadcast and poses a scalability problem as the scope of such systems goes beyond the local area network.

The system is not fault-tolerant to node failures. This is because, if a node fails with some agents residing in it, the agent is destroyed and its state information is also lost. This might result in other postmaster agents being misinformed about the presence of some information at a particular node, when such information has already been lost. The Postmaster platform functions well in a distributed system, but such a mobile agent collaboration requiring global information state maintenance will not work in an ad-hoc networking domain where multiple nodes might be linked together only for a short amount of time.

The eFlow engine provides a good foundation and a leading path towards the science of composition of different e-services. The engine is strictly targeted towards fixed infrastructure type services and is particularly suitable for long drawn complex e-commerce transactions. Dynamic service flow modification may not be possible if the system was used to carry out short-term e-transactions. eFlow has a centralized architecture. The eFlow engine is at the crux of the whole process where all the composite services are defined, the service nodes and graphs are initialized and all actions and process flows are coordinated. Thus the architecture is very much dependent on the successful running of the eFlow engine and has a central point of failure. This also leads to scalability issues, once the engine needs to process thousands of service composition requests. The system is also focused towards only being able to compose complex services from current services and doesn't focus on optimizing the composition with respect to parameters like bandwidth usage for the information flow, usage of the machines on which the services are residing, etc.

Carleton's Service Composition architecture mentions of two different ways of implementing composite services. The concept of interface fusion is unique and has good applicability in composing services if the end client is capable to do the required computation. The advantage of creating a single composite service by combining the required components into a single object has applicability in software reuse. Thus, once this service is created, it doesn't need to be broken down and can go on serving customers. This is contrary to the concept of 'service path' in the Ninja Platform, where the path is broken down once the query has been executed. Both the approaches have got its advantages and foibles. The single composite service may be efficient when there are multiple requests requesting the same type of composition. However, a monolithic composite service adds redundancy and duplication of components to the system.

The Interface fusion mechanism does not consider services that might follow the agent paradigm and may be communicated using an agent language. With the increasing growth of agent technology, a heterogeneous composition platform needs to take this into account. Creation of a dynamic interface is carried out at the client's machine that tends to use the interface. Also, the client device needs to coordinate the messages between the different component services. This may be computationally expensive for thin clients. This type of dynamic interface composition is definitely not scalable over a wide area where there are numerous services available.

The standalone composite service paradigm assumes that the code of the service is serializable, or in other words, the code and the application can be imported from the service provider and compiled and assembled elsewhere. This is not a good assumption since these days, typical service providers are usually static in nature and communicate with the interface or message passing paradigm and are unwilling to lend out

code or executable to be run on another platform. The other disadvantage of composing services using the standalone composite approach is that, it limits the execution of this monolithic service to only resource rich clients. The assembly of components, filtering the code and compilation is also time consuming and is particularly suitable for off-line service composition.
The service composition platform at UC, Berkeley was developed with the primary goal of making the composition system fault tolerant, scalable and fully automated. The system uses service discovery service to find out existing services and hence is adaptable to the changes of the environment. The fulcrum on which the whole architecture is based is the APC service. The APC service creates the graph of operator space, decides on a logical path, finds out physical services to execute the components and even monitors the system for faults. Thus if one service in a path fails, the APC detects it soon and replaces it by another service. The system thus achieves fault tolerance with respect to services. The architectural concept of the system is not immune to a centralized point of failure. The APC service centrally monitors several activities and the success of the path composition depends on the successful functioning of the APC service. The system however addresses this problem by running APC services over a cluster-computing platform and leverages various advantages from the platform. Examples include high availability through redundancy, fault tolerance, and incremental scalability. This makes the implementation resistant to single point of failures. This however makes the architecture vulnerable to the above-mentioned failures in the absence of such a computationally rich platform. Service composition will play a critical role in providing enhanced functionality to mobile users in the future. The composition platform should be able to take advantage of the services existing in the vicinity of a mobile device by leveraging the ad-hoc networking communication protocols being developed. We can't assume the presence of a cluster-computing environment or similar other systems with good processing power to be able to coordinate such composition. The Ninja service composition architecture optimizes network bandwidth usage by introducing efficient connectors between two service nodes. However, the basic system follows the design of sending the data from one service node to another and get it processed. It does not consider other ways of optimization by relocating the service component to the source of the data in the event that the dataset is huge. In section 3.3.4 we have discussed how our service composition environment tackles this problem.

Our work in the field of service composition, even though uses an enhanced Jini service discovery mechanism, suffers from the drawback of scalability imposed by the basic Jini architecture. The service composition platform concentrates in combining different services existing in the fixed network infrastructure to provide information to the mobile user. The composition is dependent on the broker and hence the architecture is centralized. However, a centralized architecture is well suited where the services are fixed and reside in the Internet. Our system would be able to cater to mobile clients due to the advantages of the underlying Ronin system. However, the design approach does not fit into the realm of service composition in an ad-hoc domain where the individual services are also mobile and we cannot guarantee the presence of a fixed broker system.

4 Proposed Research

In this section, we present an overview of the plan of our proposed research. In particular, we describe three different techniques for service composition in ad-hoc environment that we propose to design, implement and test. We describe the general architecture and assumptions of our service composition system. We also lay out the different problems that we would try to address in the course of the research.

4.1 Taxonomy of Composite Services

There are several ways of classifying service composition. In this section, we classify the different types of service composition that are feasible over the Internet. There are several ways of classifying service composition. In this section, we classify different service composition techniques. It is worthwhile mentioning here that the different techniques identified below are not orthogonal to each other and that overlaps are possible.

4.1.1 Proactive Composition and Reactive Composition

Proactive composition refers to the offline composition of available services to form new services. Thus, these types of compound services are pre-compiled before even a user has made a request to execute this composite service. The services that may be used for proactive service composition are usually stable and constantly running on a resource-rich platform. This composite service may integrate the different subcomponents into a single service on a single platform or may employ techniques to communicate with the different components residing on different hosts. An example of proactive service composition may be that of an online business process of reservation of air tickets. Any ticket reservation service carries out three distinct functions: ticket availability check, credit card check and updation of the required database to reserve a ticket for the customer. This composite service is used at a very high rate over the Internet. Thus it would be wise to integrate these three services together and serve those high volume of requests rather than trying to create the service each time a customer arrives with a request.

Reactive composition refers to the process of creating a compound service on-the-fly. These types of services are created only when there is a request from a customer for the deployment of such a service. Thus Reactive composition is a lazy composition technique. Reactive composition requires a component manager that takes the responsibility of collaborating with the different subcomponents to provide the composite service to the user. Reactive composition is particularly suitable when there are few requests for that particular composite service and the different service components are also not stable. Reactive composition may be a combination of proactive composite services. An example of reactive service composition is that of a "Business Tour Manager". A Business Tour Manager in charge of arranging a tour for a customer has to collaborate with the Ticket Manager, Itinerary Manager, Hotel Manager and Car-rental Manager to organize a tour. The interaction cannot be predefined and depends on certain dynamic situations. As an example, suppose that after issuing a request to the tour manager, the client found out that he does not want a hotel accommodation. The client intimates the information to the tour manager. The service composition has to adjust itself with this new constraint imposed on it. Reactive composition is better able to exploit the current state of services and make certain runtime optimizations based on real-time parameters like bandwidth, cost of executing the various subcomponents etc.

4.1.2 Mandatory-composite services and Optional-composite services

Mandatory-composite services refer to those classes of composite services, where all the subcomponents must participate for the proper execution and output of the resulting composite service. These types of services will be dependent on the successful execution of each subcomponent to produce a satisfactory result. If the different components are distributed over the network as is usually expected, the reliability of the different components will multiplicatively contribute to the reliability of the composite service. An example of such a service is the composite service that is able to calculate the averages of stock values of a certain company. This composite service will consist of a service capable of retrieving the stock history of a certain company and another service capable of filtering the relevant information to calculate the average of the values over the time interval. It may be worth noting here that the set of proactive services is not mutually exclusive with the set of mandatory-composite services.

Optional-composite services on the other hand do not necessarily need the participation of certain subcomponents or services for the successful execution of a query. Examples of such services may be efficient connectors between two service components that might efficiently optimize data transfer through a variable bandwidth line. In this case, the two service components might anyway be able to communicate with each other without the optimizer in between. Addition of the optimizer may save valuable bandwidth.

4.2 Key Issues in Service Composition

In the following section, we identify some of the important issues to be considered to develop a comprehensive service composition platform irrespective of whether it is in a mobile system or fixed workstation environment.

1. Service Discovery
   Service Discovery is one of the most important issues to be considered in order to enable wide area Internet Service composition. e-services available in the Internet follow a wide range of definitions. Examples include interface-based service invocation or services that are agents and follow a particular agent language. There might be other services which have a web-based front-end and can be invoked by only filling in appropriate markup forms. An efficient service discovery architecture should be able to discover all services conforming to a particular functionality, irrespective of its method of invocation. This requires that there should be a uniform way of describing the functionality of different e-services. The DARPA DAML initiative [55] is such an attempt to describe the services with the help of a semantic language. A good service discovery architecture should be able to perform semantic level reasoning on the service description to discover out appropriate services. Strength of a good service discovery infrastructure becomes more important in a mobile domain due to the heterogeneity and scarcity of services that exactly match the request of a client. The discovery framework should also scale across the wide area network. Current industry standard service discovery frameworks like Salutation, UPnP, Jini [71] or Bluetooth SDP [63] individually address only a subset of the above mentioned characteristics. Our work in the area of service discovery [16] attempted to add more semantic matching capability to Jini-based service discovery systems.

2. Service Coordination and Management
   Every service composition platform must have some entity coordinating the different services involved in the composition. As we have seen, in all the above service composition platforms, the efficiency and design of the platform depends on the way the central entity or manager handles the different services. The problem of coordination and management becomes difficult when the entities are distributed across the network and require different methods to be communicated. The central entity needs to ensure that the composite service is tolerant to failures, available and efficient to a reasonable extent. The composition manager should also be able to efficiently manage the network resources and apply certain optimization criteria to reduce the cost of the composite service. We can roughly enumerate the work of the central entity as follows:

   (a) Decide on what services will be able to create a particular composite service

(b) Discover the appropriate services for a certain composition

(c) Manage the order of execution

(d) Make the composite service tolerant to failures of the individual 'minute' services

(e) Optimize the different factors that can subjectively decide the quality of the particular composition.

(f) Ensure the transparency of the end-client to underlying computations

These tasks remain the same, irrespective of whether the central entity is static and is dealing with static services of whether it is situated in a dynamic environment and is dealing with mobile services. It is evident that in an ad-hoc environment, to enable dynamic composition, we cannot assume the presence of a "central entiry service" to be present in our vicinity all the time. This would make the success of the composition prone to the availability of the platform on which the central entity resides. We can envisage brokers having the capability of performing a reactive or a pro-active service composition.

3. Uniform Information Exchange Infrastructure
We need different services following different information exchange mechanisms to operate together to realize a heterogeneous service composition platform. Examples of such mechanisms include services that follow the message-passing paradigm to communicate with clients, services that follow the remote method invocation mechanism or agent-based services that follow a certain agent language. A good service composition platform should be able to communicate with all the different services and utilize them appropriately for any composition with other services not following the same type of information exchange paradigm.

4. Fault Tolerance and Scalability
Fault Tolerance is a very important attribute to be considered in designing a service composition architecture. This is because a composite service is dependent on many distributed networked services for its proper functioning. Hence, if a network service breaks down, the architecture should be able to detect this and resort to fault control mechanisms to ensure that the processing of the query is not hindered. The composition platform should degrade gracefully as more and more services become unavailable. Scalability is also an important aspect in this domain. If we considered a centrally monitored service composition system, it requires a lot of CPU cycles to process a single composition request, especially if the components are not co-located in the same system. In the future, dynamic composition of e-services has the potential to unleash a lot of new applications. Hence scalability would be an important factor to the success of such systems when there would be numerous users concurrently making composite service requests.

5. Adaptability
The world of e-services changes rapidly. e-services are coming up and going down frequently. There are also examples of long-standing e-services (e.g., stock quote providers). With the induction of mobile devices, the chances of non-persistent services being available for sometime are increasing. A good composition platform should be able to adapt its composition by taking maximum advantage of the currently available services. This increases the chances of availability of the composite service in a dynamically changing environment.

6. Service Sub-tasking
A composite service is a conglomeration of different smaller services. An important question to be addressed in this area is: how does the broker decide break up of a generic task into several subtasks. The broker should not have information about the different sub-tasks that need to be done to

execute a particular task. How can this information be inferred from the query of a particular service composition request? The generic problem of splitting a task into generic or optimal sub-tasks is very complex and goes into the domain of planning [25] in AI. We might need expressive description facilities for the broker to calculate such information.

4.3 Service Composition Architecture for Ad-hoc Environments

In this section, we describe a general layered architecture that we believe, will enable service composition in an ad-hoc environment. We consider that the request for a composite service would originate at a mobile device. The goal of this architecture is to leverage the maximum benefit from the services and computation platforms existing in the vicinity of the device. The services themselves would be present on different heterogeneous devices. We assume that these devices would be connected to each other using ad-hoc networking protocols. Figure 2 depicts the environment where we trying to make such a composition feasible.

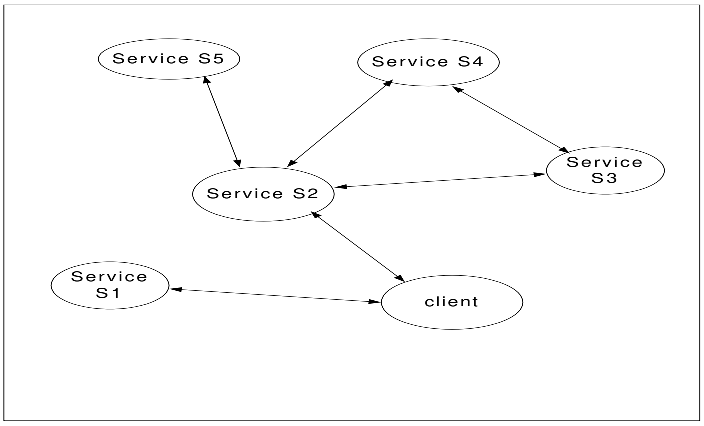

Figure 3: Ad-hoc Service Composition Environment

We observe that all the services (S1 to S5) are existing on different mobile devices. All these mobile devices are not interconnected to each other. But they form a scatternet [63] type of network. Suppose the client wants a service S that is a combination of services S1 to S4 in some order. Let us assume that S=S1+S2+S3+S4
The symbol '+' depicts that there is a linear dependence of these services when they would be actually executed. There are two ways in which the composition can be achieved. First, if any of the mobile devices (or the client itself) is connected to the wired network infrastructure, then this device can access the services available in the Internet. Considering the multitude of services available on the Internet, it is reasonable to assume that the above mentioned services would also be available there. So, the client needs to use some standard "wired-infrastructure-based" service composition broker to obtain the information it wants. However, if such a facility is not available, then the client would be deprived of the information it needs, even though the rich environment in its vicinity had the required resources to provide it with a reasonable solution. Moreover, utilization of services available in the vicinity of a device might turn out to be beneficial as far as the efficiency and speed of the composition are concerned.

A totally pure pro-active composition is impossible in such an environment. This is primarily due to the short life span of devices in the vicinity and the dynamically changing nature of the environment. We aim to develop a pseudo-proactive service composition technique that can be deployed in such an environment. We will also develop two reactive composition techniques suitable for such environment. We aim to study the various factors affecting the platform efficiency, the different mobility scenarios in which each composition technique can be best used and utilized, the primary differences between these two platforms in terms of efficiency, adaptability to the changing environment, response time and throughput. Figure 3 explains the different layers and modules in the architecture. This architecture is targeted towards

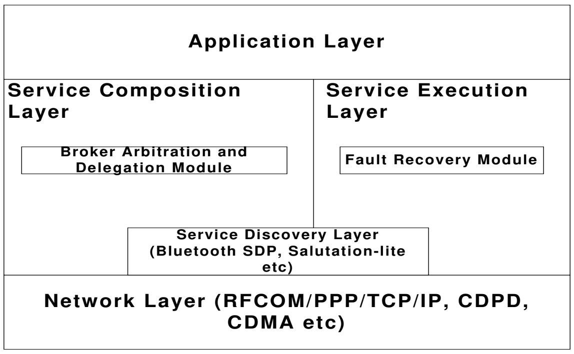

Figure 4: General Architecture for ad-hoc service composition

providing the functionality of composite services to mobile devices by taking advantage of the resources available in their vicinity. The architecture assumes:

1. There is a service discovery layer present in each mobile device that can discover heterogeneous services residing on different mobile devices in the vicinity. By “vicinity” we mean the mobile devices to which communication can be established using ad-hoc and wireless networking protocols. Note that a mobile device may be connected to the Internet backbone and may be capable of acting as a gateway to the millions of services residing in the wired Internet. The success of the composition depends on the “radius” over which the composition can be achieved. This depends on the success of the service discovery layer in discovering services which are multiple hops away. “Radius” of service composition is defined as the number of hops from the client from where a service component was obtained.

2. The services can be invoked in a certain way and the service discovery layer would be able to provide information on how to invoke the service.

3. The functionality of a broker remains the same as described in section 4.2.

4. The implementation of the architecture would exist as a layer in each mobile device that participates in service composition.

5. Any mobile device having an implementation of the architecture has the potential of becoming a broker for a particular service composition.

This architecture, although primarily targeted towards completely ad-hoc environments, is nevertheless applicable to fixed network infrastructures too. The architecture is targeted for an environment where different mobile devices with varying capabilities exist in the vicinity of one another. Each device is connected to the PDAs in its vicinity with the help of wireless ad-hoc network. The devices are mobile, have a short "switched-on" time and hence are unreliable. Each device has one or more services that it can export to other users.

A client is a mobile device from where the service composition request originates. A broker is a mobile device that coordinates the different components to calculate the result. The services may reside on a mobile device or may reside on the fixed infrastructure if that is accessible through a mobile gateway. In the latter case, the mobile device that has got access to the services existing in the Internet is supposed to act as a proxy for the different 'wired' services that it can access. The architecture consists of five distinct layers that help in the composition of services.

1. Network Layer
   The Network Layer forms the lowest most layer in the architecture. The network layer encapsulates different network level protocols that are used to establish wireless connectivity with multiple mobile devices. For example, we may consider the network layer to provide TCP/IP connectivity over RFCOM for a Bluetooth radio frequency network. We also assume that the network layer is capable of routing data from one node to another over multiple hops. Thus ad-hoc routing protocols like TORA or DSDV [64, 65] would also be a part of the features provided by this layer. The network layer might offer connection less or connection-oriented paths for data transmission.

2. Service Discovery Layer (SDP Layer)
   The service discovery layer is very important for the proper functioning of the composition platform. There is a direct dependence of the success of the composition techniques on the underlying service discovery mechanisms. This layer encompasses the protocol used to discover the different services that are available in the vicinity of a mobile device. Examples of service discovery systems very well suited for the wired network infrastructure are Jini, UPnP, SLP etc. Industry standards for dynamic discovery of services in an ad-hoc environment are Bluetooth SDP and the Salutation-lite protocol. In section 4.4, we lay out the functionalities that a composition platform would want from the underlying service discovery systems. We also try to analyze the feasibility of the current service discovery systems to be able to provide the composition layer with such functionality.

3. Service Composition Layer
   This layer is responsible for taking up the functionality of a service broker and carrying out the whole service composition process. This layer assumes that it has a priori information about the services required to satisfy a particular service composition request. The requester of the service may supply this information to it. The task of breaking down an arbitrary task into composable subtasks is a very complex problem in the domain of AI. However, the main purpose of our system is to provide the facility of being able to compose and utilize services in the vicinity. In the future, it is possible to have another service will do this work for the broker. That service may be implemented by a third party and may employ complex knowledge representation techniques to solve the problem. Hence for our purpose, it is reasonable to assume that the broker has a priori information about the different services required to compose a complex service. In our implementation of this layer, we would have different techniques for composing services incorporated into the system. These techniques are fundamentally different from the techniques used in static 'wired' service composition. We propose to implement two different types of reactive service composition techniques:

- Dynamic Central Entity Selection Technique
- Distributed Brokering Technique

In section 4.3.1 we describe the techniques in detail. We also propose to implement a pro-active service composition technique and test its suitability in ad-hoc domains. We describe this technique in section 4.3.2.

4. Service Execution Layer
The service composition layer provides a feasible order in which the services can be composed and also provides information about the location of the service(s). It also determines whether the current environment has the capability to compose a complex service. The Service Execution Layer is responsible for carrying out the execution of the different services. There are various ways in which the execution layer might optimize the cost of executing a certain complex service. As an example, the execution layer might want to optimize the bandwidth required to transfer data over the wireless links and hence execute the services in such a manner that minimizes the bandwidth utilization. Since the mobile devices are assumed to be of limited strength in terms of memory and CPU capability, several run-time optimizations, as far as load distribution is concerned, are possible. The service execution layer implements the functionality to carry out such optimizations. This layer has a module called the "Fault Recovery Module". The function of the fault-recovery module is to ensure that the complex service degrades gracefully due to failures. The inherent property of ad-hoc environments is that any node might fail or become disconnected with a node at any point of time. The 'Fault-recovery module' employs different techniques to guard against failure of the composed service. We shall explain these mechanisms as we explain the pro-active and re-active composition techniques. The Service Execution Layer and the Service Composition Layer would be tightly coupled with each other due to their dependence on each other.

5. Application Layer
The application layer embodies any software layer that utilize our service composition platform. The application layer encompasses different GUI facilities to display the result of a composed service and provides the functionality to initiate a request for a composite service.

4.3.1 Re-active Service Composition Technique

In this composition technique, the service composition is triggered only when there is a request for such a service. This enables the composition layer to make proper utilization of the currently available services in its vicinity. We further subdivide this technique into essentially two different approaches depending on the way the services are composed. Irrespective of the techniques used, it is the job of the assigned broker to perform the re-active composition. We describe them in detail in the following section:

1. Dynamic Central Entity Composition Technique
It is very evident that in an ad-hoc environment, to enable dynamic composition with a broker, the broker cannot reside on a particular host. This would make the success of the composition prone to the availability of the platform on which the broker resides. In this technique the broker performs a re-active composition. Hence it does not keep any information beforehand about the different services present in the environment. This approach centers on a procedure of dynamically selecting a mobile device to be a broker in the environment. We assume that the mobile devices coordinating with each other have the necessary functionality (or software to be specific) to act as a broker or coordinator of a service composition. In the following section, we describe three distinct features of the Dynamic Central Entity Composition Technique.

(a) Broker Arbitration and Delegation

This module is primarily responsible for guaranteeing that the architecture can be successfully applied over a mobile domain. When a request for service composition arrives at this module in a mobile device it finalizes a platform that is going to carry out the composition and monitor the execution. Once the platform has been chosen, the device is intimated of its responsibility in composing and executing a particular request. The mobile device acting as the broker is responsible for the whole composition process for a certain request. The selection of the host that performs the brokerage may be dependent on several parameters like: power of the platform (battery power left), number of services in the immediate vicinity, stability of the platform, secondary resource availability of a platform etc. We might even decide to make the originator of the request to be the broker for that particular composition. The research aims at determining the appropriate parameters that should be considered for such a decision. We intend to implement a distributed algorithm that can perform the required arbitration based on the different parameters. We summarize some of the important questions to be addressed here:

- How does the requester come up with a reasonable platform that agrees to do the composition?
- What are the different parameters that need to be taken into account to select a broker platform?
- How can the reliability of the platform be judged?
- How efficiently can this information be circulated to the different nodes participating in the arbitration?
- What kinds of parameters need to be checked to see whether the composition is at all possible in such an environment? For example, the ad-hoc network might be changing too fast for any such algorithm to work.
- What are the performance metrics that can appropriately evaluate the efficiency of composition in such environments?
- Do we need different algorithms for different kinds of ad-hoc networks? (e.g., fast moving vs. slow moving ad-hoc network) How can the nature of ad-hoc network be taken into account?

The primary goal of this module is to provide the request with the most appropriate environment possible so that the cost of composition also may be minimized. Each request may be assigned a separate broker. This makes the architecture immune to central point of failure and the judicious choice of brokerage platform has the potential of distributing the load appropriately within the mobile hosts, thus avoiding the problem of swamping of a host by numerous requests. It is worthwhile to mention here that even though it might appear that the source of the request could easily act as the broker of that request thus pre-empting any need for such arbitration, the architecture would not be able to optimize the cost of the request with respect to the factors described above. At this juncture, we do not know which would be the best policy to use. The research aims to find out a reasonable answer.

(b) Service Composition and Information Retrieval

Once the brokerage has been determined, the request is sent to that broker platform residing on the mobile host. The broker residing on the mobile host now takes up the burden of carrying out the functionalities of a standard broker. The broker's first job is to discover the services from its vicinity. The broker progressively increases its "radius" of search to discover all the different services necessary for the composition. Once it figures out the availability of the different services, it carries out its functionality as the intermediary to compose the different services. Note

that the services may be present multiple hops away from the broker. The broker returns failure if it fails to discover all the required services. Once the broker has determined the composition sequence, it should pass on the control to the execution layer that takes the responsibility of executing the different miniature services in the order determined. The execution layer has the capability of making many run-time optimizations to minimize the cost of the request. Depending on the ordering of the services and their inter-dependence, the execution layer might decide to execute some services parallelly or might decide to re-orient the ordering, provided the initial set of constraints on the ordering of the services is maintained.
However, in such an ad-hoc domain, the device on which the broker is residing is prone to failure due to many reasons like shutting down of the device or network failure due to mobility. The main aim of the platform is to guarantee maximum availability of the different services. Service availability is not enough to guarantee maximum availability in such an environment. The presence of the broker is also equally important to leverage the maximum benefit from them. If the broker suddenly breaks down, the composition has to be started anew. This layer prevents such an occurrence by continuously updating the source of the request on the progress of composition. Since a composite service is discrete, the broker sends back the partial reply and the completion state of the request after a subtask is complete. The source of the request keeps a cache of this partial result obtained so far. As we shall see, this forms the basis of making the platform fault tolerant to broker failures. The research aims at finding out the suitability of such an approach for different ad-hoc environments. We summarize some the issues that need to be addressed in this section:

- How much information needs to be made available to the composition layer for it to come up with a reasonable solution?
- What are the factors that need to be taken into consideration to select a service when there might be multiple instances of the same service in the vicinity?
- How can service discovery layer be instructed to control its 'radius' of search?
- How can the services be searched in the most optimal manner?
- In what type of ad-hoc networks is this type of a mechanism successful?

(c) Fault Recovery

An ad-hoc environment is very error-prone. There may be several reasons due to which the service composition may fail in the middle of its execution. Some of them are listed below:

- A service that is taking part in a composition becomes unavailable due to certain reasons (network partition, low battery power, shutting down of the device etc). Thus the broker would not be able to execute a complex service that requires the service. If there are multiple such services in the environment, then the broker might initiate another series of service composition request to incorporate a similar service (service performing the same functionality) into the composition.
- The platform that performs the functionality of the broker breaks down or becomes unavailable.
- Network partition prevents the result of a composed service from being transmitted back to the originator of the source node. This is similar to a failure since the end result is that the result could not be intimated to the requester of the composite service.

The standard solution to this problem is to make the requester initiate a new request for the composite service. This is very inefficient and not applicable in such an environment due the high probability of the occurrence of such a failure. Thus, if the composition has to start afresh, it makes the architecture prone to faults, especially faults that hamper the working of the broker. We propose to employ fault-tolerant techniques to guard against such faults to a reasonable extent. To prevent this, the partial completion state of the request might be periodically updated to the source node of the request. The broker does this. If the broker platform fails, the source node detects the unavailability of updates. It then starts the fault-recovery process. In this module, the source of the request reconstructs the query that is still left unsolved by the broker. This request is now treated as a different service composition request in the environment. This technique uses a greedy-type approach in hoarding as much information as was obtainable from the partially executed query. This way, the architecture guarantees fault-tolerance against node failures. We would want to look into the efficiency of such a mechanism in a disconnection prone ad-hoc network.

If a network partition prevents the updates from being propagated to the source node, the source node might employ either a random-retransmit or binary exponential backoff technique to probe the current broker for updates. It should fail after it has tried sufficient number of times. However, this method might prevent the source node from requesting re-execution of the unchanged request in the event that the broker node could reconnect back to the source node even after a temporary network partition. We would want to look into the issue of how much tolerance in delay the source node is going to accept and whether the delay can be somehow predicted from the characteristics of the ad-hoc network.

Each request in this architecture might be assigned a different broker due to the broker arbitration technique described earlier. This makes the system distributed and tolerant to single point of failure. If a node currently acting as the broker of a request fails, then the architecture adapts itself to select other brokers dynamically. This increases the system availability. However, a single request is still prone to "single point of failure" of the broker platform. In the research, we propose to determine how the different fault-tolerant mechanisms can compromise such faults.

2. Distributed Brokering Technique
   The key idea in this approach is to distribute the brokering of a particular request to different entities in the system by determining their 'suitability' to execute the partial composition of the request. A composite service may be considered to be a combination of n (n>=1) services s1 to sn. In the following section we explain the basic features employed by this approach.

   (a) Broker Arbitration
   This module would perform the simple functionality of selecting the broker of the initial set of services required to execute a complex transaction. The client would be responsible for deciding the broker who is going to take over the composite service. The criteria for selection of the broker would be similar to the criteria used for dynamic central entity selection mechanism. However, in this technique, more emphasis would be placed on certain attributes like, the vicinity of services required 'immediately' w.r.t the services that may be required after the composition has advanced certain steps. Thus in the earlier example (where S=S1+S2+S3+S4), if S1 and S2 are currently available in the vicinity of the requester, the requester will try to compose and execute them first. It will try to discover a platform who can potentially do this limited composition and execution. The service S3 and S4 might be available multiple hops away. However, the primary concern of this approach is to utilize the resources in the immediate vicinity. Thus, a single broker would not be responsible to do the whole composition. Rather, it would complete only 'as much' as it can. This module is also responsible for handing over the composite query to the respective broker residing on a device.

   (b) Service Composition and Information Retrieval
   The broker is responsible for composing the services S1 to Sn. Contrary to the dynamic central

entity selection mechanism, the broker here is not responsible for the whole composition. The broker decides on a 'radius' in which to search for services (radius: number of hops). The composition is carried out amongst services discovered within this radius. This has a few advantages as opposed to the Dynamic Central Selection mechanism.

i. The broker has to deal with a subset of the whole composition and hence its workload is reduced.
ii. The architecture allows some flexibility for services that may be available some time after the request has been made.
iii. The system allows more optimization in composing the service w.r.t the physical distribution of the services or the topology of the network.
iv. The system guarantees more fault tolerance than the Central Entity Selection mechanism.

Suppose a broker determines that it has services S1 to Si available in its vicinity(within radius r). It goes ahead and carries out the partial composition and execution. Then it carries out two actions

i. It intimates the requester (source node) about the 'current state' of the execution.
ii. It uses the 'Broker Selection and Arbitration Module' to select another broker which has the ability to carry out a subset or whole of the remaining composition. While selecting the next platform to delegate the brokerage to, the obvious stress would be on the availability of the remaining services in the vicinity of the next platform. The current broker delegates the brokerage of the remaining composition to that particular broker/device. In this manner, the composition hops from one node to another till the final result is obtained. Then the current broker returns the final answer of the composition to the client. The important research issues that need to be addressed here are:
    - How does this technique compare with the "Dynamic Central Entity Selection Mechanism"?
    - How is the load on the service discovery layer affected by these two techniques?
    - In what type of ad-hoc networks is such a composition technique helpful?
    - Is there any way to cache the composition results and help in the development of a knowledge base in each device? This knowledge base may help in finding information about the components that need to be executed for an arbitrary service later on.
    - What should be the optimum radius above which a broker should not go? How does it compare to the radius being used in the earlier scheme? What are the performance gains?
    - How much is the reliability of such a distributed technique in an ad-hoc environment?
    - Can the composition technique to be invoked be predicted based on certain parameters of the ad-hoc network (E.g. topology) ?

(c) Fault Recovery
Multiple brokers take part in composing a particular request. The design of the system is such that at a certain point of time there is always a positive probability that the service composition and execution will be successful if the resources are present. The system also degrades gracefully with node failures. Our initial intuition tells that the fault tolerance mechanism in this architecture is similar to that used in the previous one. The failure of a broker to communicate partial information back to the client is an indication that the platform has become unavailable or some other disconnection or network partition related problem has cropped up. However, in this technique, the partial information would be propagated from different broker platforms.

The broker platform keeps track of the source of the source of the request and the client also keeps track of the current broker. If the client node does not obtain the information after reasonable pursuits, it issues a new service composition request with the subset of the composition that is still to be completed.
Both these architectures assume that the client who has issued the request is available (switched-on) all the time. This is a valid assumption to make in this context since it is very natural for a client making the request to keep the mobile device switched on at that time.

4.3.2 Pro-active Service Composition Technique

This technique centers on the idea of pre-configuring the information on the different compose-able entities in the vicinity and pre-calculating the different composition subsets possible in the environment. A pro-active service composition would intuitively suit a relatively stable environment with restricted node movements. E.g. scenarios are where the topologies of the mobile devices are relatively stable (passengers in a train, or embedded devices on the body of a person). In this method, each mobile device calculates the 'richness' of its environment periodically. It also discovers the set of services that are available to it and figures out whether valid compositions can be done using those services. Thus whenever a request for a composite service comes to a mobile device, it knows before hand whether such a composition is possible. The negative answer can be returned very quickly. Even if such a composition is possible, the system does not have to spend time in doing the service composition. It can start the execution of the components.

A common question is that such a system might not be scalable with increasing number of services. We are not expecting the number of devices and services available in the vicinity of a person or device to be so high as to make this method unfeasible. In this type of composition, each mobile device would maintain a cache of the different compositions possible at that instant by taking help of the devices/services available in the surroundings. This cache shall be updated whenever a device moves out of the vicinity or a device comes in. Some of the research issues to be addressed in this context are:

- How do you describe the services appropriately so as to be able to find out the end result of combining two or more arbitrary services? (two services combined might not yield any valid composed service at all)
- How do you efficiently maintain a cache of the service compositions and what policy you use to update it?
- How much radius (number of hops) do you use to do the pre-calculation of the composable services? How does it depend on the number of services currently available and how can you regulate it?
- How do you ensure that the algorithm will degrade gracefully with increasing number of services?
- If a device in the vicinity has connection to the wired infrastructure, then this node may be considered to have millions of services available to it. How will you model the services so as to take such factors into consideration?
- How does this type of service composition technique compare to the re-active techniques in different ad-hoc environments?

4.4 Critical Evaluation of the requirements imposed on a Service Discovery protocol by the service composition architecture

In this section, we outline some of the implementation related issues that can be sketched out at this point of the research. The inherent changeability of an ad-hoc network calls for a critical examination of how the current service discovery mechanisms can be used to discover services in mobile devices. Moreover, for service composition to take place in such an environment, the service discovery mechanism needs to satisfy some requirements. It is evident that the success or efficiency of the service composition techniques will be pruned by the capabilities of the underlying service discovery layer. In the following section, we lay out some of the important requirements of a service discovery system in an ad-hoc environment.

- There should be a flexible way of discovering services existing in peer mobile devices. The service discovery mechanism should have support for the following types of service discovery capabilities in their decreasing order of importance:

  1. Light weight semantic service matching capability: The semantic service matching technique should not incorporate heavy semantic reasoning services which would further compromise the processing power of the low form-factor mobile devices.
  2. Interface based simple service matching capability
  3. Attribute based simple service matching capability
  4. Unique Identifier based service matching capability

  We stress the importance of existence of all these service matching capabilities since this helps in dealing with the heterogeneity of service discovery platforms in a dynamic environment.

- The service discovery should happen in a network independent manner.

- The service discovery architecture should be language independent.

- Service Discovery Protocol should provide a means or handle for the client to invoke or access the service.

- The Service Discovery Protocol should have a control on the number of hops across which it should search for services.

- The Discovery architecture should be de-centralized and based on peer-to-peer networking model.

- Service Discovery Protocol should have the ability to forward SDP requests to nearest neighbors and should implement schemes to handle duplicate SDP requests. It should also handle other factors contributing to the overloading of the network.

- Service Discovery Protocol should handle security issues (impersonation, man-in-the-middle type attack) involved in accessing services present on other mobile devices.

We try to analyze how much the existing service discovery platforms are able to satisfy the above mentioned requirements. Service Location Protocol is a language independent protocol capable of peer-to-peer service discovery. However, it uses multicasting to contact other peers which is a problem in ad-hoc networks. Moreover, SLP does service matching based on simple attributes and does not have any semantic level discovery technique. Jini is language dependent and has got a centralized architecture. Moreover, it employs strict interface or attribute-based service matching and has got a heavy footprint also. It does not handle security issues too. UPnP is based on TCP/IP networks and uses a very simple service discovery matching mechanism. It uses XML to describe the services and its controls. UPnP uses multicast and is based on a decentralized peer-to-peer architecture. The UDDI is targeted towards web services and has got a very centralized model for service discovery and registration. The Ninja SDS has a centralized architecture and also employs a fixed central entity to address the security issues. None of these protocols defines a way to restrict service discovery across multiple hops in an ad-hoc network. The Salutation protocol uses attribute based service matching capability only. It uses Sun's Remote Procedure Call mechanism to communicate across multiple Salutation managers residing in different systems. It has a de-centralized architecture and its ability to forward service requests to adjacent salutation managers could be utilized to do service request forwarding in an ad-hoc network. The architecture is network and language independent. The service matching in salutation is however attribute-based and does not employ semantic matching techniques. The Bluetooth SDP is very restricted due to its UUID based service matching approach. The Bluetooth SDP also does not provide information on how to invoke a service. It only provides information about the particular service. The SDP also lacks the ability to do multicast to query multiple devices for a service. From the above discussion it is evident that any particular service discovery mechanism cannot be used 'as is' to satisfy the needs of a service composition environment. However, it might be possible to modify one or more of the above mechanisms to make it suitable for ad-hoc environments. As an example, the DReggie [16] service discovery mechanism incorporates semantic level service matching capabilities to the existing Jini service discovery protocol. The Jini Service discovery protocol however is unsuitable for ad-hoc environments due to its centralized architecture for service registration and lookup. It is however possible to scale down the specification of a Jini Lookup Server and implement it in a light weight manner so that it can be installed in small form factor devices. Ideally, it is possible to have a reduced version of the Jini lookup server in each mobile device participating in the ad-hoc network. We want to investigate further into the area of service discovery to find out a suitable service discovery mechanism for ad-hoc networks. We might also enhance any of the existing protocols and make it suitable for ad-hoc environments.

5 Conclusions

We intend to carry out an exhaustive simulation of the service composition techniques mentioned in section 4.3 using any suitable ad-hoc networking simulator. We are aware that the network simulator (ns2) initially developed for DARPA [74] offers a suitable platform for networking research. However, we plan to investigate the simulator and judge its suitability to simulate dynamic ad-hoc networks. We might create our own simulator if our findings about ns2 are not affirmative. The questions that have been posed in this research can be best answered if the whole environment is simulated. To provide full functionality to the service composition techniques, we might implement a service discovery protocol which addresses the critical requirements of service discovery in ad-hoc networks. We might enhance one of the 'nearest matching' existing protocols too.

We also intend to carry out a practical implementation of one or more service composition techniques using up-to-date commercial ad-hoc networking protocols. Initial thoughts about a practical implementation point to a scenario where mobile nodes would be a combination of laptops and iPAQs, both of which would be Bluetooth enabled. We intend to carry out experiments on the sustainability of the different composition techniques on such a practical ad-hoc network. We intend to validate or compare our theoretical simulation results with those obtained from the practical implementation.

References

[1] Web Services Description Language 1.1. World Wide Web, `http://www.w3.org/TR/wsdl#_wsdl`.

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

[3] Ken Arnold, Ann Wollrath, Bryan O'Sullivan, Robert Scheifler, and Jim Waldo. *The Jini specification*. Addison-Wesley, Reading, MA, USA, 1999.

[4] Daniel O. Awduche, Arthur Gaylord, and Aura Ganz. On resource discovery in distributed systems with mobile hosts. In *Proc. MOBICOM. Rye. Newyork.*, 1996.

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

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

[7] S. Basagnia, D. Bruschi, and I. Chlamtac. A mobility-transparent deterministic broadcast mechanism for ad-hoc networks. *IEEE/ACM Transactions on Networking. 7/6*, 1999.

[8] Thais Batista and Noemi Rodriguez. Dynamic reconfiguration of component-based applications. World Wide Web, `http://www.citeseer.nj.nec.com/416208.html`.

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

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

[11] Harini Bharadvaj, A. Joshi, and Sansanee Auephanwiriyakyl. An active transcoding proxy to support mobile web access. In *Proc. IEEE Symposium on Reliable Distributed Systems*, October 1998.

[12] 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.

[13] C. Brooks, M. S. Mazer, S. Meeks, and J. Miller. Application-specific proxy servers as http stream transducers. In *Proc. WWW-4, Boston, http://www.w3.org/pub/Conferences/WWW4/Papers/56Application-Specific*, May 1996.

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

[15] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and dynamic service composition in eflow. Technical Report, HPL-200039, Software Technology Laboratory, Palo Alto, CA, march 2000.

[16] Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, and Anupam Joshi. Dreggie: A smart service discovery technique for e-commerce applications. Accepted for presentation at the workshop to be held with 20th Symposium on Reliable Distributed Systems, october 2001.

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

[18] Harry Chen, Dipanjan Chakraborty, Liang Xu, Anupam Joshi, and Tim Finin. Service discovery in the future electronic market. In *Working notes of Seventeenth National Conference on Artificial Intelligence, Eleventh Innovative Applications of AI Conference. Austin. TX*, july 2000.

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

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

[21] 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.

[22] Universal Description Discovery and Integration platform. World Wide Web, `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.PDF`.

[23] UC Berkeley Computer Science Division. `http://ninja.cs.berkeley.edu`.

[24] T. Drashansky, S. Weerawarana, A. Joshi, R. Weerasinghe, and E. Houstis. Software architecture of ubiquitous scientific computing environments. *ACM - Baltzer Mobile Networks and Nomadic Applications. 1:4.*, pages 421–432, 1996.

[25] K. Erol, J. Hendler, and D. Nau. Htn planning: Complexity and expressivity. In *Proc. AAAI.*, 1994.

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

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

[28] Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, 1997.

[29] Tim Finin, Anupama Potluri, Chelliah Thirunavukkarasu, Don McKay, and Robin McEntire. On agent domains, agent names and proxy agents. In Tim Finin and James Mayfield, editors, *Proceedings of the CIKM '95 Workshop on Intelligent Information Agents*, Baltimore, Maryland, 1995.

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

[31] DARPA Agent Markup Language for Services. World Wide Web, `http://www.ai.sri.com/daml/services/daml-s.pdf`.

[32] Ian Foster and Carl Kasselman. Globus: A metacomputing infrastructure toolkit. *International Journal. Supercomputer Applications*, pages 115–28, 1997.

[33] A. Fox, I. Goldberg, S.D. Gribble, D.C. Lee, A. Polito, and E.A. Brewer. Experience with top gun wingman: A proxy-based graphical web browser for the usr palmpilot. In *Proc. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, 1998.

[34] Biztalk Framework. World Wide Web, `http://www.biztalk.org/home/default.asp`.

[35] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Elements of reusable object-oriented software. In *Addison-Wesley*, 1995.

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

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

[38] E. Guttman, C. Perkins, J. Veizades, and M. Day. "rfc2068: Service location protocol, version 2". ftp://ftp.isi.edu/in-notes/rfc2608.txt, 1999.

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

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

[41] T. Hodes and Randy Katz et. al. An architecture for a secure service discovery service. In *Proc. Fifth International Conference of Mobile Computing and Networks, Seattle WA*, August 1999.

[42] T.D. Hodes, R.H. Katz, E. Servan-Schreiber, and L. Rowe. Composable ad-hoc mobile services for universal interaction. In *Proc. MOBICOM. Budapest. Hungary*, 1997.

[43] Marcus J. Huber. Jam: A bdi-theoretic mobile agent architecture. In *Autonomous Agents. Washington. Seattle*, 1999.

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

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

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

[47] D.B. Johnson and D.A Maltz. The dynamic source routing protocol for mobile ad-hoc networks. *Mobile Computing. Imielinksi and H. Korth. Eds. Kluwer.*, pages 153–181, 1996.

[48] A. Joshi, S. Weerawarana, T. T. Drashansky, and E. N. Houstis. Sciencepad: An intelligent electronic notepad for ubiquitous scientific computing. In *Proc. of Intl. Conf. on Intelligent Information Management Systems*, pages 107–110, June 1995.

[49] Anupam Joshi. On proxy agents, mobility and web access. *ACM/Baltzer Journal of Mobile Networks and Applications*, 2000.

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

[51] Lalana Kagal, Vlad Korolev, Harry Chen, Anupam Joshi, and Timothy Finin. Centaurus: A framework for intelligent services in a mobile environment. In *International Workshop on Smart Appliances and Wearable Computing (IWSAWC). Phoenix. Arizona*, 2001.

[52] 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.

[53] Jennifer Kay, Julius Etzl, Goutham Rao, and Jon Thies. The atl postmaster: A system for agent collaboration and information dissemination. In *Proc. Second International Conference on Autonomous Agents*, 1998.

[54] Joost N. Kok and Kaise Sere. Distributed service composition. In *Technical Report No. 256. Turku Centre for Computer Science. Finland.*, march 1999.

[55] DARPA Agent Markup Language. World Wide Web, `http://www.daml.org`.

[56] Web Services Flow Language. World Wide Web, `http://xml.coverpages.org/wsfl.html`.

[57] Navneet Malpani, Jennifer L. Welch, and Nitin 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.

[58] David Mennie and Bernard Pagurek. An architecture to support dynamic composition of service components. Systems and Computer Engineering. Carleton University, Canada.

[59] 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.

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

[61] E. Pagani and G.P Rossi. Providing reliable and fault-tolerant broadcast delievery in mobile ad-hoc networks. *Mobile Networks and Applications. Volume 4. No. 3*, 1999.

[62] Meta Object Facility Resource Page. World Wide Web, `http://www.omg.org/technology/cwm`.

[63] Bluetooth White Paper. World Wide Web, `http://www.bluetooth.com/developer/whitepaper`.

[64] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proc. IEEE INFOCOM.*, april 1997.

[65] C.E Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing(dsdv) for monbile computers. *Comp. Commun. Rev.*, pages 234–44, October 1994.

[66] C.E. Perkins and E.M Royer. Ad-hoc on-demand distance vector routing. In *Proc. 2nd IEEE Workshop. Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[67] Simple Object Access Protocol. World Wide Web, `http://msdn.microsoft.com/workshop/xml/general/SOAP_White_Paper.asp`.

[68] Chaitanya Pullela, Liang Xu, Dipanjan Chakraborty, and Anupam Joshi. Component based architecture for mobile information access. In *Workshop in conjunction with International Conference on Parallel Processing (ICPP).*, August 2000.

[69] David V. Pynadath, Milind Tambe, and Nicolas Chauvat. Building dynamic organizations of distributed, heterogeneous agents. In *Proc. Agents. Barcelona. Spain*, 2000.

[70] Pierre-Antoine Queloz and Alex Villazon. Composition of services with mobile code. In *Proc. First International Symposium on Agent Systems and Applications Third International Symposium on Mobile Agents. Palm Springs. California.*, 1999.

[71] Rekesh John. UPnP, Jini and Salutaion - A look at some popular coordination framework for future network devices. Technical report, California Software Labs, 1999. Available online from.

[72] Elizabeth M. Royer and Chai-Keong Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications.*, april 1999.

[73] Y. Chen S. Ni, Y. Tseng and J. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proc. MOBICOM. Seattle. Washington*, 1999.

[74] The Network Simulator. World Wide Web, `http://www.isi.edu/nsnam/ns/`.

[75] Ian Sommerville. Software engineering. In *Addison-Wesley Pub. Co.*, 1982.

[76] Bluetooth Specification. World Wide Web, `http://www.bluetooth.com/developer/specification/Bluetooth_11_Specifica%tion_Book.pdf`.

[77] FIPA ACL Message Structure Specification. World Wide Web, `http://www.fipa.org/specs/fipa00061/`.

[78] FIPA Agent Management Specification. World Wide Web, `http://www.fipa.org/specs/fipa00023`.

[79] FIPA Communicative Act Library Specification. World Wide Web, `http://www.fipa.org/specs/fipa00037`.

[80] Remote Procedure Call specification. Sun Microsystems, Inc. RPC: Remote Procedure Call Protocol Specification Version 2, June 1988. RFC 1057.

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

[82] 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.

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

[84] Brian S. White, Michael Walker, Marty Humphrey, and Andrew S. Grimshaw. Legionfs: A secure and scalable file system supporting cross-domain high performance applications. In *Proc. Supercomputing. Colorado. Denver.*, November 2001.

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

[86] B. Zenel. *A Proxy Based Filtering Mechanism for The Mobile Environment*. PhD thesis, Department of Computer Science, Columbia University, N/A.