Technical Report TR-CS-02-02
A Reactive Service Composition Architecture for Pervasive Computing Environments

Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Timothy Finin, Yelena Yesha
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD 21250
{dchakr1,fperic1,joshi,finin,yeyesha}@cs.umbc.edu

March 6, 2002

Abstract

Technological advances in semiconductor processing and design as well as wireless networking are leading us towards the vision of Pervasive Computing. We envision that in the (near) future, devices all around a person, either embedded as a part of smart spaces, or being carried by other people in the vicinity, will provide an array of services that she might want to use. Development of customized services by integrating and executing existing ones has received a lot of attention in the last few years with respect to wired, infrastrutcure based web-services. However, service discovery and composition in web-based environments is performed in a centralized manner with the help of a fixed entity. Moreover, wired infrastructure-based service discovery and composition architectures do not take into consideration factors arising from the possible mobility of the service providers. In this paper, we present a distributed, de-centralized and fault-tolerant design architecture for reactive service composition in pervasive environments. The design of the architecture is based on a peer-to-peer model. We introduce two reactive techniques for service composition in our design. We also present the Anamika system, an initial implementation of our design architecture. We present experiments to show the functioning of our design and implementation.

1 Introduction

Service Composition can be defined as the process of creating customized services from existing services by a process of dynamic discovery, integration and execution of those services in a planned order to satisfy a request from a client. Research in the area of service discovery[2, 23, 4, 9, 27] and service composition[8, 26, 17, 24, 6, 21] has focused on trying to leverage the wide array of e-services available over the network to provide customized services to e-customers, for example planning a business trip for a person. A business trip manager could integrate existing services like a car rental service, an airline ticket booking service, and a hotel room reserving service to provide the user with a complete planned business trip. There has been a sharp increase in these types of wired infrastructure-based services in the last few years. Existing service composition systems [21, 8, 17] broadly address the problems associated with composing various services that are available over the fixed network infrastructure. They primarily rely on a centralized composition engine to carry out the discovery, integration and composition of web-based e-services.

However, computing today is becoming increasingly pervasive. We can see advancements in both computational capabilities of commonly used devices, like cellular phones and handhelds, and their ability to wirelessly communicate with one another. Sensing and computational capabilities will also increasingly be embedded in engineered artifacts, from household appliances to roads to even clothes we wear. We envisage that in the near future, these mobile and embedded devices will also be capable of providing customized information, services and computation platforms to peers in their vicinity. People will need the cooperation of services available in their resource-rich vicinity to satisfy their information needs. Wired service composition architectures do not address the problems associated with composing services in such "pervasive" environments. We explain by digressing a bit into the future.

Philip is traveling in a car with a built-in Global Positioning System (GPS) device and has a IEEE 802.11/Bluetooth-enabled personal digital assistant (PDA). His colleague in the car, Mark, suddenly falls ill and must be taken to a hospital immediately. Philip has to find the route to the nearest hospital and the traffic conditions on the road are really bad. Philip does not want to simply find any route to the hospital, but one that consists of least crowded highways, roads and streets. This cannot be done by route generation systems presently found in cars.

The traditional infrastructure oriented mobile systems solution to this problem is to ensure that Philip's PDA establishes a connection to some central server that can provide directions, maps and some traffic information. One can assume that Philip's cellular phone would provide near-continuous connectivity to the Internet, thus allowing the PDA to pose certain queries to the server and obtain answers so that it can perform the required tasks. The implicit assumption here is that any arbitrary service we might need will be provided as a packaged, monolithic entity on the wired side. This assumption is questionable from an economic perspective since the user base for some services will be very small (sometimes 1 person!). Even if we stick to technical issues only, we believe that this solution is not the most efficient. One obvious problem is the scalability of the centralized system that handles service requests. With a large number of wireless devices attempting to connect to it to request different services, the traffic/route server quickly becomes a bottleneck. Another problem is that dynamic information such as current traffic conditions is unlikely to be up-to-date on the server. We note that Philip in this scenario does not benefit from the knowledge that the outer loop of the DC Beltway is "slow" from Silver Spring to the American Legion Bridge (a stretch of 10-15 miles); however he would benefit immensely from knowledge that a particular portion of the highway (270-Beltway merge) or some major intersection has suddenly become congested perhaps due to traffic

volume. Yet another problem is the latency of the connection between the PDA and the server since WAN wireless connections tend to have latencies and disruptions. In a dynamically changing environment, information that takes too long to reach the PDA might well be useless. If, for example, the PDA instructs Philip to "take exit 33" after he has passed it, that information is stale and useless.

Consider an alternate approach. The neighborhood of Philip's car is information-rich because of presence of devices in vehicles around it, and the vehicles themselves are continuously engaging in conversations with their neighbors thus obtaining (and perhaps storing) useful information. Philip's PDA might try to obtain the necessary information about the shortest route to the nearby hospital that has the minimum traffic by using the information and computation sources available in its ad-hoc neighborhood. It decides that in order to satisfy the goal, it needs to compose the services of a dynamic traffic information service provider, a road map service provider and a route calculating service provider. First, the PDA determines its current location by connecting to the car's built-in GPS device. It then discovers the presence of a Road Map Atlas CD-ROM at the neighboring car, provides it with the (current location) and (any hospital) as the end points, and asks for several alternate routes. Once it knows which roads it could use, it discovers current traffic condition information from cars traveling in the opposite direction. This information, along with the location and map information, is given as input to an on-board route calculating service in a nearby high-end BMW to finally obtain the *best route to the nearest hospital.*

Let us consider a simpler example, without the complications induced by high motion rates of the participating entities. Bob is chatting with his friends at his university's student union when his cell phone beeps indicating that he has received an email. Bob reads the email from his cell phone and finds that the email contains an attachment that the sender has urged him to read as soon as possible. Unfortunately the attachment is in MS Word, which Bob can neither download nor open using his cell phone. Bob looks around for a computer terminal but cannot locate one in the vicinity. In such situations, the traditional solution would be to have a proxy service on the wired Internet that would convert the attachment and stream it to Bob's cell phone either as text or as voice (see for instance the transcoding proposed by IBM's Content Adaptation Framework). For reasons discussed earlier in this section, we envision an alternate scenario where Bob might be able to use existing services in his vicinity.

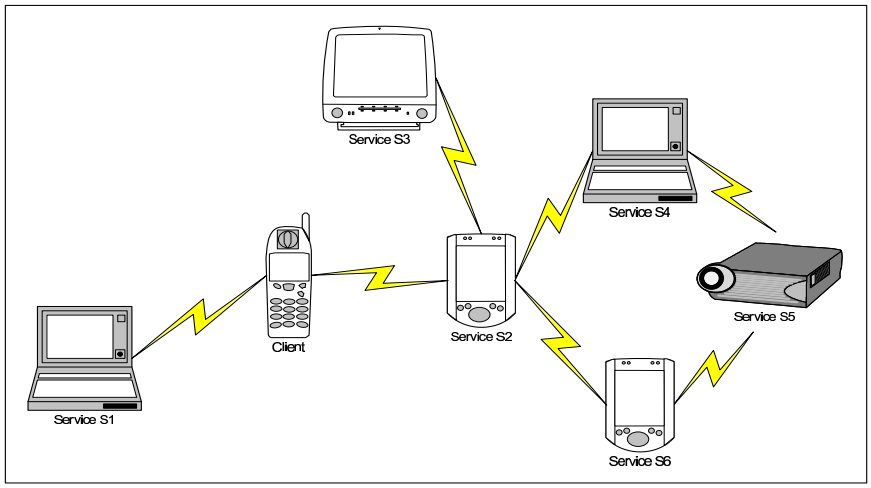

Figure 1: Ad-hoc Service Composition Environment

Suppose one of his friends has a visor with 802.11b springboard module and another friend has an Bluetooth-enabled iPaq. Bob uses the network connection (service) provided by the visor to download the file. The file then can be sent over to the pocket-word viewer (service) on iPaq over Bluetooth to let Bob view it. Suppose that Bob's friend does not have an iPaq, but one of the people sitting on a nearby table has a Bluetooth-enabled laptop that has Word (or at least a word viewer). The owner of the laptop is busy working and would not want an arbitrary window popping up on his monitor which would allow bob to come and look at his attachment. Bob's Bluetooth-enabled cell phone discovers that there is a Bluetooth-enabled HP Postscript printer on the next floor. Bob's cell phone uses the "Internet" service on the visor to download the file, gets the word file converted to postscript by the "word processing" service on the laptop and has it printed out. Thus we see that by intelligently discovering and integrating

the functionalities of different services in the vicinity in a dynamic manner, we are able to satisfy an user's request.

Service Composition systems for the pervasive computing environments exemplified above need a different design approach than those developed for wired services. This is because many of the assumptions of standard composition architectures of wired services are no longer valid in such dynamic environments. Service Composition architectures in wired infrastructure assumes the existence of a centralized composition entity that carries out the discovery, integration and execution of services distributed over the web. They also need a tighter coupling with the underlying network layers. For instance, Bluetooth service discovery protocol is limited to discovering services within a single radio hop. A blind use of Bluetooth SDP[27] would lead to a restriction of "vicinity" in ad-hoc environments that is counterintuitive. The vicinity should involve devices that are in geographical proximity even though they are multiple hops away in network terms because of the nature of the underlying ad-hoc network.

We have designed a distributed architecture to perform service composition in pervasive computing environments. Central to our system is the concept of a distributed broker that can execute at any node in the environment. An individual broker handles each composite service request, thus making the design of the system immune to central point of failure. A broker may be selected based on various parameters such as resource capability, geometric topology of the nodes and proximity of the node to the services that are required to compose a particular request. Current prototype of our system has been implemented over Bluetooth.

The rest of the paper is organized as follows: In section 2 we discuss the present state-of-the-art research in the field of service composition. In section 3 we present our design of a distributed architecture for service discovery and composition. In section 4 we present the Anamika reactive service composition system: our initial implementation of the design framework. We conclude in section 5.

2 Related Work

Service discovery and composition is an important and active area of research [7, 11] and has been studied widely in the context of web-services. Research in this area has forked along three main branches: service process model description languages, service discovery platforms and service composition architectures.

Service description languages like Web Services Development Language (WSDL) [1], Web Services Flow Language (WSFL) [19] and DARPA Agent Markup Language for services(DAML-S) [13] have been developed to describe web services in a flexible manner. 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 Web Services Flow Language (WSFL) [19] designed by IBM is an XML-based flow and usage specification for web service composition. The DAML project by DARPA and the W3C focuses on standardizing DAML as the language to use to describe information available on any data source, in order that the information may be understood and used by any class of computers, without human intervention. We have used DAML-S to encode composite services in our work.

Service Discovery Architectures like Jini [4], Salutation and Salutation-lite [2], UPnP [25], Service Location Protocol [14], have been developed over the past few years to efficiently discover wired infrastructure based services from wired as well as wireless platforms. However, most of these service discovery infrastructures have a central lookup server type architecture for service registration and discovery. Service matching mechanism used is also restricted to unique identifiers, attribute, interface-based matching or XML-based static string matching. The Bluetooth Service Discovery protocol [27] is a peer-to-peer service discovery protocol that can be used over ad-hoc environments. However the service matching in Bluetooth is very rudimentary and based on unique identifiers. Our prior work in this area includes developing a Jini-based framework for flexible service discovery [9] where service matching can be done using semantic reasoning features provided by DAML-S. We also enhanced the Bluetooth service discovery protocol to include service description-based reasoning [5] using Prolog. Service discovery in pervasive computing environments also need to have a robust and flexible way of discovering services following a peer-to-peer model.

Most of the research in realizing service composition systems for web-based services have a centralized architecture for service integration and execution management. eFlow [8] and CMI [26] are service composition engines based on the centralized model of service management. A central composition engine composes and monitors composite service executions. Service Discovery models used in these engines are also lookup-server based. These engines are targeted toward fixed infrastructure type services and particularly suitable for long drawn complex e-commerce transactions. Service Composition in ad-hoc environments need a distributed and decentralized approach. The Ninja Service Composition architecture [17] is a scalable architecture that supports multiple service integration based on input-

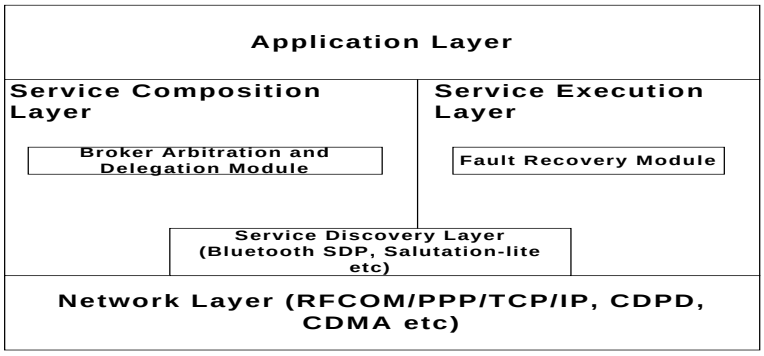

Figure 2: General Architecture for ad-hoc service composition

output matching of services. Central to the architecture is the automatic path creation 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. The architecture handles central point of failure by replicating services on multiple workstations. However the important distinction between Ninja and our work is that in our architecture, service composition is based on a peer-to-peer model and and there is no distinction between clients and composition managers. The architecture is geared to take maximum advantage of currently available platforms. Sheng, Ngu et. al [6] have developed a framework for declarative web services composition using statecharts[15] and communities where a composition is executed in a decentralized way. In their composition framework, different states of a particular request are assigned to different coordinators and the coordinators invoke the wrapper service for a particular state whenever all the preconditions to execute that particular state has been met. The execution of a composite service is distributed in this manner.However, the coordinators of a particular composite service are preassigned and statically determined. In dynamic environments, we have to accomodate the availability/unavailability of these coordinators and take into account factors that decide whether a particular participating platform can be a coordinator or not. Our prior work [24] in the field of service composition centers on enabling ubiquitous access to all sorts of information from a mobile device. We developed an agent-based middleware platform for mobile information access by integrating the functionalities of multiple infrastructure-based services to provide information to a mobile user. Our middleware platform took into account mobility related issues like disconnections, bandwidth and resource constraints on mobile devices while trying to process a composite query from the device. Service Composition is also related to workflow management [20, 22] and teamwork related theories, agent-based business management [16, 29, 28, 10]. However, the usual assumptions about resource availability and reliability do not work in pervasive computing and ad-hoc environments.

3 System Architecture and Design Principles

In this section, we describe a general layered architecture that enables service composition in pervasive computing environments described earlier. The architecture is targeted for an environment where different heterogeneous devices with varying capabilities exist in the vicinity of one another. Figure 1 depicts such an environment. The devices are connected to others in their vicinity with the help of short-range ad-hoc connections. Many of 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. We consider that the request for a composite service would originate at one such device.

Our architecture introduces two distributed **reactive** techniques to carry out service composition in purely ad-

hoc environments. It also specifies techniques that the service discovery layer utilizes to carry out efficient scalable service discovery. Reactive service composition refers to the type of composition that is executed only upon request. Our composition architecture primarily deals with the discovery, integration and execution of the components of a composite request. The problem of splitting a task into sub-tasks is complex and goes into the domain of planning [12] in AI which is outside the scope of our present work. Current implementation of our architecture assumes that the process model of the composite task would be provided to it. It is a straightforward exercise to plug in an external planning system into the design that will provide the system with a process model of execution for a composite service. 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 2 depicts the different layers and modules in the architecture. A client is a device from where the service composition request originates. A broker is a device that coordinates the different components to calculate the result. Some devices might have connectivity to the fixed wired infrastructure and hence be able to act as proxy for wired services. The architecture consists of five distinct layers that help in the composition of services.

3.1 System Components

Network Layer: The Network Layer forms the lowest layer in the architecture and encapsulates networking protocols that provide wireless/ad-hoc connectivity to peer devices in the vicinity. For example, we may consider the network layer to provide TCP/IP connectivity over a Bluetooth radio frequency network or over Ad-Hoc WLAN (802.11) networks. For our initial implementation of the architecture, we have chosen a connect-transmit-disconnect mode of network connectivity between peer devices utilizing the RFCOMM [27] protocol over Bluetooth network.

Service Discovery Layer: The service discovery layer is required 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. Our design of the service discovery mechanism is primarily based on the following principles:

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

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

3. *Semantic description-based service matching:* The service discovery mechanism has support for semantic service matching using descriptions of services in DAML+OIL[18]. We used DAML+OIL to describe services since the language is being standardized by World Wide Web Consortium to represent metadata of any information or resource on the Internet. The service discovery mechanism is able to reason with the description of services and provide high flexibility in the matching mechanism. Apart from this, the service discovery protocol also incorporates Interface-based, attribute-based and unique identifier based service matching capabilities. These service discovery mechanisms combined together increases the efficiency of the service discovery layer to discover heterogeneous services in the environment. The service discovery mechanism also provides a means and knowledge for a client to invoke the discovered service.

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

Service Composition Layer: This layer is responsible for carrying out the process of managing the discovery and integration of services to yield a composite service. The process model of the composite service is supplied as input to this layer. In our current implementation, we have used the 'compositeprocess' definition of DAML-S to describe a process model.

Both the reactive techniques that we have designed are decentralized, immune to central point of failure and take maximum advantage of the mobility of the nodes, topology of the environment and currently available resources in the vicinity. Both these techniques carry out a broker arbitration that decides the platform that takes the responsibility of becoming a broker for a certain request. Thus, each request may be assigned a different broker platform. The broker arbitration and delegation module and the techniques employed within it thus play a vital role in making this architecture fundamentally different from the techniques used in static 'wired' service composition. We describe the two techniques in detail in sections 3.2 and 3.3.

Service Execution Layer: The Service Execution Layer is responsible for carrying out the execution of the different services. Prior to this, the service composition layer provides a feasible order in which the services can be composed and also provides location and invocation information of the service(s). There are various ways in which the execution layer might optimize the cost of executing a certain composite service. As an example, the execution layer might want to optimize the bandwidth required to transfer data over the wireless links between different services and hence execute the services in an order that minimizes the bandwidth utilization. Since the mobile devices are assumed to be of limited strength in terms of memory and processor 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", which is responsible to guard against node failures and service unavailability. We explain the fault-tolerant techniques in section 3.2 and 3.3. The Service Execution Layer and the Service Composition Layer are tightly coupled with each other due to their dependence on each other.

Application Layer: The application layer embodies any software layer that utilizes 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.

3.2 Dynamic Broker Selection Technique

This approach centers on a procedure of dynamically selecting a device to be a broker for a single request in the environment. In the following section, we describe three distinct features of the Dynamic Broker Selection Technique.

Broker Arbitration and Delegation: When a request for service composition arrives at the service composition 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 informed of its responsibility. The mobile device acting as the *broker is responsible for the whole composition* process for a certain request. The selection of the broker platform may be dependent on several parameters: power of the platform (battery power left), number of services in the immediate vicinity, stability of the platform, etc. The brokerage arbitration might make the originator of the request to be the broker for that particular composition. We intend to implement a distributed algorithm that can perform the required arbitration based on the different parameters. Each request thus 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 different devices. This avoids the problem of swamping the central composition entity by numerous requests. It may appear that the requesting source could easily act as the broker of that request thus pre-emptying any need for such arbitration; however, the architecture would then become unable to take a full advantage of other resource-rich platforms from where the broker functionality could have been performed more efficiently with respect to services in the near vicinity and topology of the ad-hoc environment.

Service Integration and Execution: The assigned broker's first job is to discover the services from its vicinity. The broker progressively increases its search "radius", a number of devices that it can reach by asking other devices in its radio range to forward service request, to discover all of the different services necessary for the composition. The broker returns failure when it fails to discover all of the required services. Service discovery and integration

is followed by service execution. 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 in parallel or might decide to re-orient the ordering, provided the initial set of constraints on the ordering of the services is maintained.

Fault Recovery: Faults in ad-hoc environment may occur due to a service failure, due to a sudden unavailability of the selected broker platform, or due to network partition. The standard solution to this problem is to make the requester to initiate a new request for every composite service. This is very inefficient and not applicable in our environment due the relatively high occurrence probability of the above failures. The fault-tolerance module in the architecture employs check pointing to guard against such faults. The broker for a particular request sends back checkpoints and the state of the request to the client of the request after a subtask is complete. The client keeps a cache of this partial result obtained so far. If the broker platform fails, the source node detects the unavailability of updates. It then starts the fault-recovery process. 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 is obtainable from the partially executed query. If a network partition prevents the updates from being propagated to the source node, the source node employs either a random-retransmit or binary exponential back-off technique to probe the current broker for updates, and fails after a fixed number of unsuccessful attempts. If a node currently acting as the broker of a request fails, then the architecture adapts itself to select other brokers dynamically.

3.3 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 a part of the composite request. A composite service may be considered as a combination of n servers (n>=1), denoted as S1, S2, ... Sn. In the following section we explain the basic features employed by this approach.

Broker Arbitration: This module performs the simple functionality of selecting the broker of the initial set of services required to execute a complex transaction. The client (request originator) decides on the first broker who becomes in charge of the composite service. The criteria for broker selection may be similar to the criteria used for dynamic broker selection mechanism. However, in this technique, more emphasis is placed on certain attributes like, the vicinity of services required 'immediately' with respect to the services that may be required after the composition has advanced certain steps. For example if the composed service consists of executing four services, S1 to S4, and 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 discover a platform which 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. *A single broker is not responsible for performing the whole composition.* Rather, it completes only 'as much' as it can.

Service Integration and Execution: The broker is responsible for composing the services S1 to Sn. The broker decides on a service search "radius". The composition is carried out among services discovered within this radius.

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 integration and execution. It performs out two actions:

1. It informs the requester (source node) about the 'current state' of the execution.
2. It uses the 'Broker Arbitration' Module to select another broker which has the ability to carry out a subset or whole of the remaining composition. 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.

Fault Recovery: Multiple brokers take part in composing a particular request. We use source-monitored fault-tolerance to detect faults. Current broker of a composition request sends back checkpoints to the source that contain the partial execution reply. However, in this technique, the partial information would be propagated from different

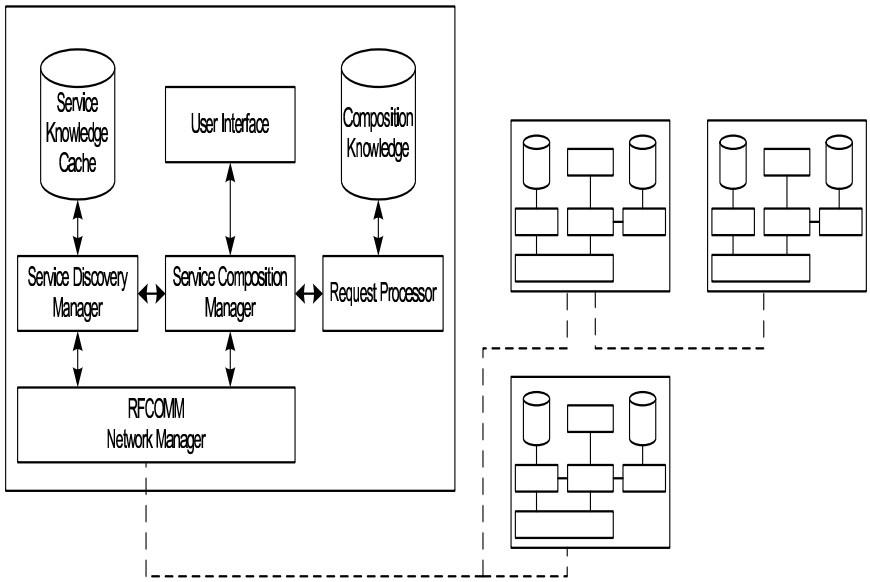

Figure 3: System Components in Anamika Reactive Service Composition Architecture

broker platforms. Each broker platform keeps track of the source of the request and the source also keeps track of the current broker. If the client node does not obtain the information after reasonable number of pursuits, it issues a new service composition request with the subset of the composition that is still to be completed. Both techniques assume that the client issuing 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 Implementation and Experiments

In our initial implementation of the design architecture, we have developed a reactive service composition system called Anamika. The individual components of the Anamika system existing in participating mobile devices are described in Figure 3. Current prototype of the architecture has been implemented over Bluetooth [27]. Composition knowledge is described using DAML-S[13] in terms of subset of individual services that might be able to satisfy a composite request. Service discovery is done in a peer-to-peer manner using semantic description of services using DAML-S and our light-weight reasoning engine present on participating devices. Anamika implements dynamic broker selection mechanism and decides the best platform to carry out the composition based on a combination of the processor power of the platform and number of services that the platform has. We explain the details of the different components of the system in the following sections.

Network Manager: The Network Manager implements the API required for higher layers to reliably communicate to neighboring peers over Bluetooth. Efficient implementation of the design architecture necessitate the network layer to have support for broadcasting. However, broadcasting in Bluetooth is restricted to messages used for device discovery. In order to exchange application level messages, a device must first establish a link-level connection with its peers. In addition, every communicating device must either be a *master* or *slave* thus making multiple simultaneous links between two devices impossible. We have used the connect-transmit-disconnect mode of sending Anamika messages between peers. We implemented the networking level API over RFCOMM [27] protocol. We used segmentation and reassembly of packets each of 64 bytes size to prevent large chunks of data overflowing the receiver buffer in a Bluetooth peer. The Network Manager provides APIs for clients to connect to other peer devices in the vicinity as well as APIs for devices to listen for incoming messages and send acknowledgments or replies back to the sender. We implemented the Network Manager by using IBM's BlueDrekar transport driver [3] and protocol stack for Linux kernel 2.4.2-2.

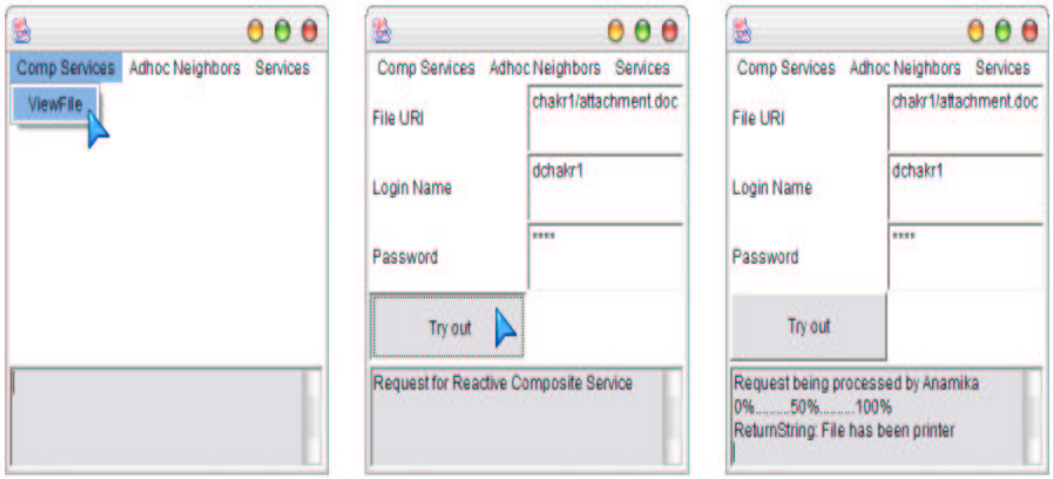

Figure 4: User Interaction with Anamika System

Service Discovery Manager: Service Discovery Manager provides the functionality to local Composition Manager to discover services in Bluetooth peers. Services are described using DAML-S based on inputs, outputs, functionality classification, functional similarity to other services and invocation mechanisms. The service description also incorporates platform specific information like processor type, speed etc. We developed this ontology by extending the ontology developed in the DReggie [9] system, a Jini-based semantic service discovery framework. The ontology can be obtained from `http://daml.umbc.edu/ontologies/dreggie-ont.daml`. A request for service discovery also follows the same ontology and the discovery mechanism uses light-weight semantic matching of service descriptions to discover matching or 'nearly' matching services. We have modified the light-weight reasoner in the DReggie system [9] to reason with the service descriptions. The service discovery mechanism returns description of the service discovered, invocation information and other platform specific details (like processor power, memory availability) to the requesting device. We show an example of the information that the semantic service discovery mechanism obtains for a a printer service.

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

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

Service Composition Manager: The Service Composition Manager is the principal component that is responsible for service composition and management. Peer Service Composition Managers collaborate with each other to implement different techniques of service composition as discussed in section 3. The current implementation of the Composition Manager supports the "Dynamic Broker Selection Technique" for service composition. The Composition Manager receives a request from the Application Layer. It uses the Request Processor module to parse a composite service description. The Request Processor module consults the Composition Knowledge base to find out suitable sets of services that might be combined to provide an answer to the query. Request Processor returns multiple process models to carry out a particular composite service. The Composition Knowledge base has been modeled in DAML-S. Below we give an example of the decomposition of a generic query to view a certain URI or a file (in case the client machine does not have the capability to do so) by using the resources of the vicinity:

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

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

When a request for a composite service comes to the composition engine, it takes help of the underlying semantic service discovery mechanism to obtain information about the availability of the necessary services in the vicinity. Note that some of the required services might be locally present on the device itself. The composition engine decides on the best available platform to carry out the composition based on number of services local to that platform that would be utilized by this composition request and its processor power. Currently we carry out a broker arbitration between the platforms that has at least one service that is participating in the current composition. The Client Composition Manager decides on a Composition Manager that is going to perform the task based on the number of usable services that device has for the composite request. This information is obtained during service discovery. Each request in this environment may potentially be assigned a separate platform to carry out the composition. We have implemented two different techniques to carry out the discovery and execution of services that take into account

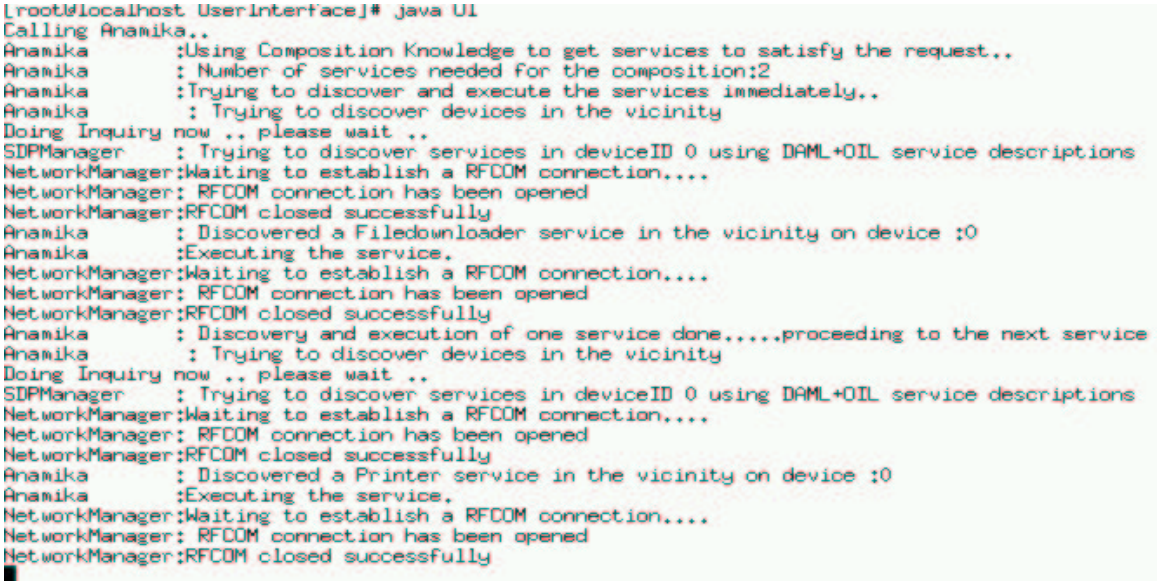

Figure 5: Log of activities performed by Anamika system at client for Experiment 1.

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

Experiments: We carried out various experiments to validate the proper functioning of the Anamika Reactive Service Composition system and test different features of the "Dynamic Broker Selection Technique". In our experimental setup, services reside on different laptops. These services are registered to the local composition managers residing on the machines. Bluetooth modules provided by Ericsson and IBM's BlueDrekar software stack [3] and transport driver are used by the Network Manager to communicate between devices. Some of the laptops also have 802.11b connectivity to the Internet. Every Anamika system displays a graphic user interface for clients to access composite services formed by the services available in the vicinity. Figure 4 shows the user interactions with the Anamika system to view a file by composing services available in the vicinity.

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

Experiment 1: In this experiment, we make the request originator the composition manager for that composite request. The Client Composition Manager carries out the discovery followed by the immediate execution of the corresponding service in the sequence prescribed by the Request Processor. We present a snapshot of the system activity in figure 5. We observe in the log that the Composition Manager at the client tries to determine what services can be combined to satisfy the user request. It then initiates device discovery followed by peer-to-peer semantic service discovery to all the devices in its vicinity. The Composition Manager aggressively executes a service

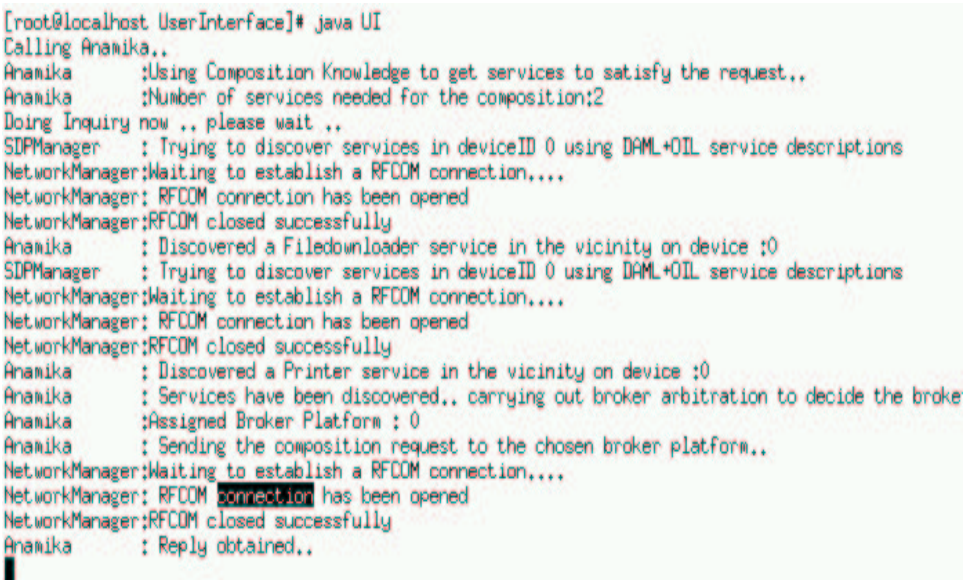

Figure 6: Log of activities performed by Anamika system at client for Experiment 2.

immediately after it is discovered. In this case, it executes the "Filedownloader" service. It then proceeds on to search for a printer service. It discovers a printer service and sends the file to the printer and prints a user-friendly message to the user interface (Figure 4). We also observe the connect-transmit-disconnect mechanism used by the Network Manager over RFCOMM to transmit service discovery and execution related Anamika messages. If a printer service is unavailable, the Composition Manager will return to consult the Composition knowledge (via the Request Processor) about other alternate services that can be used (for example a word processor) to let the user view the content of the file.

Experiment 2: We carry out this experiment for the same request. However, this time we enable the "Dynamic Broker Selection Technique". We also move the "Filedownloader" and the "Printer" service to the same machine. The Client Composition Manager performs the discovery of all the required services first. Client Composition Manager now carries out a broker arbitration to decide the broker platform based on the number of services residing on a platform and its CPU power. The algorithm processes the service descriptions to extract out the resource details (CPU power and CPU type) and runs in linear time with respect to number of components/services required for a composite request. Since both the services are residing on a remote device, the brokerage is transferred to the platform and the composite request is sent to the remote Composition Manager. The remote Composition Manager carries out the composition locally since both the services are locally available. We show the activity log of the client Composition Manager in Figure 6.

We also carried out several experiments to test the fault-tolerance of the system with respect to service and network unavailability. We carried out experiments where the "Printer" service shuts itself down after it has already been discovered by a composition manager to execute a composite request. The composition manager executing the composite request appropriately detects this and tries to execute the "view an attachment" request using other available services (an word processor in this case). However, due to space limitations we are unable to present those results.

5 Conclusions

In this paper, we have introduced a novel design approach for service composition in pervasive computing environments. Our architecture for service discovery and composition is distributed, decentralized and fault-tolerant to service and network unavailability. Service Discovery is done in a peer-to-peer mode rather than a centralized mode, and service descriptions are cached for scalability. We use a combination of least-recently-used and aging

policies for cache replacement. We introduce two *reactive* techniques, "Dynamic Brokerage Selection" mechanism and "Distributed Brokerage technique" to accomplish service composition in dynamic environments. Our approach enables any device participating in the composition to act as the broker, making the design immune to single point of failure. We use a source-monitored fault-tolerance mechanism using checkpoints and rollbacks to the last completed service. This enables us to gracefully recover from commonly expected failures such as disconnection in a mobile environment.

We have also presented the Anamika system, an initial implementation of our design architecture. Anamika has been implemented over Bluetooth using RFCOMM as the network layer. Current implementation of Anamika models composite processes using DAML-S and other services are described using an extension of the DReggie ontology. Anamika supports peer-to-peer based semantic service discovery and matching, and implements "Dynamic Brokerage selection" mechanism, where we use processor speed and number of local services needed for a particular composite request to decide a broker platform for a composite request. We have presented experimental validation of the working of our system. Our future work includes implementing the "Distributed Brokerage" mechanism in Anamika, perform assessment of the different mechanisms with respect to factors like mobility of the environment, availability rate of the services.

References

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

[2] The Salutation Consortium Inc 1999. Salutation architecture specification (part 1), version 2.1 edition. World Wide Web, `http://www.salutation.org`.

[3] IBM alphaworks. BlueDrekar protocol driver. World Wide Web, `http://www.alphaworks.ibm.com/tech/bluedrekar`.

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

[5] S. Avancha, A. Joshi, and T. Finin. Enhancing the Bluetooth Service Discovery Protocol. Technical report, University of Maryland Baltimore County, August 2001. TR-CS-01-08.

[6] B. Benatallah, M. Dumas, Q. Sheng, and A. Ngu. Declarative composition and peer-to-peer provisioning of dynamic web services. In *18th International Conference on Data Engineering.*, February 2002.

[7] F. Casati, D. Georgakopoulos, and M. Shan editors. Special issue on e-services. *VLDB Journal*, 2001.

[8] 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.

[9] 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.

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

[11] G. Weikum. Editor. Special issue on infrastructure for advanced e-services. *IEEE Data Engineering Bulletin, 24(1)*, March 2001.

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

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

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

[15] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, pages 293–333, October 1996.

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

[17] 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.

[18] DARPA Agent Markup Language and Ontology Inference Layer. World Wide Web, `http://www.daml.org/2001/03/daml+oil.daml`.

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

[20] A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler. The wise approach to electronic commerce. In *Journal of Computer SYstems Science and Engineering*, September 2000.

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

[22] CrossFlow Project Web Page. World Wide Web. `http://www.crossflow.org`.

[23] UPnP White Paper. World Wide Web, `http://upnp.org/resources.htm`.

[24] 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.

[25] 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.

[26] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Proc. Intl. Conference on Advanced Information Systems Engineering, Sweden.*, June 2000.

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

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

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