Konark – A Service Discovery and Delivery Protocol for Ad-Hoc Networks.

Nitin Desai, Varun Verma and Sumi Helal
Computer and Information Science and Engineering Department
University of Florida,
Gainesville, FL-32611, USA
helal@cise.ufl.edu

Abstract — The proliferation of mobile devices and the pervasiveness of wireless technology have provided a major impetus to replicate the network-based service discovery technologies in wireless and mobile networks. However, existing service discovery protocols and delivery mechanisms fall short of accommodating the complexities of the ad-hoc environment. They also place emphasis on device capabilities as services rather than device independent software services, making them unsuitable for m-commerce oriented scenarios. Konark is a service discovery and delivery protocol designed specifically for ad-hoc, peer-to-peer networks, and targeted towards device independent services in general and m-commerce oriented software services in particular. It has two major aspects - service discovery and service delivery. For discovery, Konark uses a completely distributed, peer-to-peer mechanism that provides each device the ability to advertise and discover services in the network. The approach towards service description is XML based. It includes a description template that allows services to be described in a human and software understandable forms. A micro HTTP server present on each device handles service delivery, which is based on SOAP.

Konark provides a framework for connecting isolated services offered by proximal pervasive devices, over a wireless medium.

Keywords: Ad-hoc networks, service discovery protocols, pervasive computing, m-commerce.

I. INTRODUCTION

The last few years have seen a rapid increase in the usage of mobile devices such as laptops, cell-phones, and personal data assistants (PDAs) [HEL99]. The accompanied maturity in wireless technologies such as 802.11 has made wireless networks almost as ubiquitous as traditional wired networks. An important consequence of such developments has been the concept of ad-hoc networks. These networks are characterized by their lack of required infrastructure and ease of formation. As the “deployment” and usage of such networks increases, new paradigms begin to emerge that incorporate and utilize these new communication capabilities in ways that were previously not thought of or were impossible to implement. Some of the possible applications and scenarios for ad-hoc networks are as follows:

- Using your wireless enabled laptop computer, the ability to discover and use a nearby printer while sitting in an airport lounge.

- Using your wireless handheld device, the ability to not only locate a nearby restaurant serving your favorite Chinese food, but also reserve a table and order your food before walking to it.

- An application on your handheld device that packages your personal expertise of being a web-security consultant as a service, advertises this service in relevant places like technical conferences, and responds to searches for such services with more details such as your current location, your schedule, and how much you charge for a consulting session.

- The ability to share with others entertainment sources such as music and games available on your personal handheld device. Such a capability would be extremely useful while waiting for a flight in an airport lounge, or other similar situations of killing time in a public place.

In all the above-mentioned scenarios, the resources being shared or searched are packaged as services. As evident, services can be those offered by devices (e.g. printers, fax machines), or simply software services that are device independent. Even software services offer a myriad range of possibilities with opportunities related to an individual’s career, entertainment or daily chores like ordering food.

The feasibility of above scenarios requires not only the formation of ad-hoc networks and packaging of resource as services, but also a discovery and delivery mechanism suited to the needs of ad-hoc networks and geared towards m-commerce oriented services. Low-level technologies necessary to form a peer-to-peer, ad-hoc network are available. The missing link is a higher-level framework and protocol, which will enable devices to discover and advertise their services over ad-hoc networks.

There has been some work in the field of service delivery protocols. Sun Microsystems’ Jini [SUN99], Microsoft backed UPnP [UPN00], and Salutation [SAL97], are some of the frontrunners in this area. These were mainly developed considering the wired infrastructure. IBM has developed
DEAPspace [NID00, HER01] that addresses the discovery of devices in ad-hoc networks in a home environment.

As pervasive computing [WEI91] becomes a reality, it needs an important middleware for pervasive information access. To address the opportunities, issues, and requirements discussed, we present Konark [DES02, VER02], a middleware designed specifically for discovery and delivery of device independent services in ad-hoc networks. Konark is based on a peer-to-peer model with each participating device having the capabilities to store its local services, query the network for other available services, deliver its own services using a resident micro HTTP server, and use the services it discovered in the network. Since each device functions as an independent entity, events such as any participating device leaving or joining a network do not affect the functioning of the entire or a part of a network. Konark proposes a Christmas tree approach – a novel way of combining the way services are discovered and delivered with the way services and their metadata is stored in the devices.

An important distinguishing characteristic of Konark is the optimum human intervention support. While Konark provides for advertising and discovery to proceed on their own, it also enables the user to control these significant events. In the case of discovery, the user can specify when and what type of services should be searched for. He can also setup filters and notification events, features that give him complete control over the activity of his device. Similarly for advertising, the user can control which of his services to advertise and at what frequency. The usage of services is completely human driven as the services targeted are interactive and m-commerce based. Once a user discovers a service, he can view its properties via a stage called service description. If the user is satisfied by the description, he can move on to using the service by invoking the functions available. Thus the protocol itself allows user-interaction in certain components for better results if the user desires to do so.

We start this paper by discussing some of the issues we considered during the design phase of Konark. We then present the Konark design and implementation, followed by possible and necessary future work.

II. KONARK ARCHITECTURE

The computing environment defined by small handheld mobile devices with wireless connectivity will be vastly different from the traditional infrastructure based environment comprising of PCs with wired connections [BUS01]. While designing a service discovery protocol for ad-hoc networks, it becomes imperative to keep in consideration the challenges posed by these differences. A whole another set of issues is also raised by the type and range of services targeted.

One of the primary differences is the formation of the network itself. In the traditional wired networks, network formation can be considered to be systematic, with each participating node being assigned an identity by some administrator, or by another device in the network. Also, events such as a device joining or leaving a network are not very frequent and this provides a semblance of uniformity to the network. In ad-hoc networks, device participation is dynamic. It is not possible to assume that a controlling entity, whether an administrator or another device in the network, will be present to assign addresses to the nodes. To handle this issue, Konark assumes an IP level connectivity in the ad-hoc networks. It is a valid assumption considering most of the modern devices run operating systems that provide zero configuration techniques like Auto-IP [HAT01, CHE02]. Such an assumption also makes Konark independent of lower network layers, which could either be 802.11, IRDA, Bluetooth or any other protocol on which IP can be implemented.

Konark supports two types of networks – partially ad-hoc networks, and completely ad-hoc networks. Partially ad-hoc networks are those where some devices are static and others mobile. An example could be an ad-hoc network in an airport where a printer and a fax-machine are “fixed” while a traveler’s laptop is mobile. Another example could be a shopping mall where a shop’s server is static while a visitor’s PDA is mobile. This provides a client-server nature to the computing. Completely ad-hoc networks are those where each participating device is mobile and the network has been formed temporarily. For example, a network formed when a group of people comes together in a conference, in an emergency relief scenario, or in an airport lounge. In such cases, the devices would generally be small and thus resource-poor.

An important design factor in a service discovery protocol is the storage of services and their metadata, that is, information about available services. One possible approach is to have a centralized repository that keeps track of all available services in the network. Any network member offering a service would register its service with this repository, and all devices seeking any service would query it for available services. This approach is best suited for traditional wired networks where it is possible to assume that some device will always remain in the network. For highly dynamic networks, such as the examples provided in earlier paragraphs, such an assumption is fallible. It also might not be possible to have algorithms such as leader election to choose a registry server each time some node moves out of the network. Also, with devices being small and resource-less, it would be difficult for any device to maintain information about all services in the network. Konark uses a completely distributed peer-to-peer approach for to solve this problem. It specifies that each device will have a local repository that will maintain the local services being offered by that device.

Traditionally, service discovery protocols have focused on services provided by devices, such as printers, fax-machines, cameras, audio-systems, etc. With the increasing popularity of handheld devices, and the support for mobile commerce, we envision an entirely new set of possible services. These services would be device independent and could vary from resources such as games or music, personal information such as professional expertise, information-oriented services such as maps or weather, or commerce based services like tickets to a movie. Users would be able to package their own resources or information as services and offer them to others. This requires services to have simple and rich description capabilities along with support for cross-platform lightweight usability. Konark defines an XML-based simple and rich description language to
describe these kinds of services. This language is similar to WSDL and enables the description of a wide range of services.

Another important issue in service discovery protocols is of how to make service information available to other devices in the network. Konark supports both advertisement and discovery. The service providers can push their services into the network. The rate of advertising depends on a lot of factors. It can be on a periodic basis or can be stimulated by events such as a new device joining the network. Advertising can be also based on other factors such as geographical or temporal information, that is, location and time based advertising. We assume that the network formed supports multicasting so that service advertisement requests sent out will reach all nodes in the multi-hop network. These service advertisements contain time-to-live (TTL) information and help in self-healing of the systems. The clients can cache this service information to use it later. This caching can be based on the user preferences in the form of filtering techniques or the device capabilities like memory. Each cached service entry requires minimal amount of memory as it stores only the name, TTL and URL of the service.

Prospective users may also need to locate services and can use a distributed pull method to retrieve desired services. This helps them to get the real time services available in the network. With the increase in the number and variety of services, locating a particular service (such as chess), or services of particular types (such as games), becomes increasingly difficult. Depending upon the network size and other factors, the information obtained about available services - either via actively discovering them or by passively caching the advertisements - can be in large quantities. Also, the kind of m-commerce oriented services being targeted requires a high level of user interaction. This makes it necessary to have a registry that stores service information in an extremely manageable and user friendly fashion. Another advantage of this feature would be the support for possible human interaction during advertising and discovery.

To handle the above requirement, Konark presents a service registry based on a tree-structure. We use a basic tree skeleton with service classification levels that are generic at top and become more specific as we move down the tree levels. The nodes in the tree are not services themselves but act as placeholders for services in that particular category. We use XML to internally represent this basic tree, making it scalable so that new class of services can be easily added. We present the same tree to the users as an interface to help them easily manage and use the services interactively. We strongly believe that using this basic tree structure, one will be able to represent most of the services needed by a mobile user. Standardization of this tree will help to realize this goal.

The tree structure described above is not only used for maintaining services and information about available services, but also for advertising and discovery. Discovery can be done at any level of the tree – the broader the range of services desired, the higher the level of the tree- using the tree-paths. Keywords can be used for discovering specific services. Advertising can be done at various levels of the tree – from generic advertising at the higher levels of the tree (such as all games) to very specific at the leaf node (e.g. chess). The use of the tree-paths during advertising and discovery helps to solve the service-matching problem considerably. Since the internal storage of the services is mapped to the user interface, the quantity of information in the tree of the device can be controlled.

Yet another important design issue is the actual delivery of services. As more and more manufacturers push out handheld devices into the market, the heterogeneity of ad-hoc networks vis-à-vis platforms increases. To support this trend, it is critical to support cross-platform service delivery. Konark uses the widely accepted standards such as HTTP and SOAP to handle service delivery. It provides for a micro HTTP server on each device that can handle service requests from clients. The service requests and responses are based on SOAP.

Having discussed the various research issues, we now proceed with the design of Konark. It has two main components -

- Service Discovery: This deals with the addressing, service registry structure, and advertisement and discovery of services.
- Service Description and Delivery: This deals with describing services and using discovered services.

In the subsequent sections, we will cover each of these in detail.

III. RELATED WORK

As mentioned earlier, some work has been done in the field of service discovery protocols in both industry and academia. One of the popular protocols is Jini, developed by Sun Microsystems. One of the major drawbacks of Jini is its centralized nature. The architecture entails a central Jini Lookup Service that maintains information about available services and stores the proxy service objects. As discussed in the previous section, having any kind of significant centralized component is not suitable for ad-hoc networks. Another drawback for Jini is its dependence on the Java platform. Both service description and delivery are completely Java based, a fact that goes against the importance of cross platform support.

Microsoft backed UPnP is another popular service discovery protocol. A significant feature of UPnP is its completely distributed nature, and its leverage of open standards such as HTTP and SOAP. However, UPnP is geared more towards home networks and focuses on services offered by devices rather than device independent services. For a service to be UPnP compliant, its service description has to go through a standardization process and be approved by a UPnP committee. This fact does not augur well for the type of services envisioned by Konark. It would not be possible for each type of service to have its own service description template. What is needed is a generic service description format that can support a wide variety of services.

There are other protocols like Salutation and IBM-DEAPspace. Salutation has a complex addressing scheme and current version doesn’t support service delivery on handheld devices. IBM-DEAPspace is geared towards home-networks
and device capabilities as services. It also doesn’t define the service delivery component.

While Jini and UPnP are industry led initiatives, there has been some work in the academia too. The Ronin Agent Framework [CHE00] was service discovery protocols developed in academia. Both these are based on Jini and thus inherit the drawbacks of being Java dependent along with they weren’t targeted towards ad-hoc networks.

IV. DESIGN

Konark can be considered to follow two different but related sequences of events. First – a server advertises a service, receives service description requests from a client, replies with the service description, receives service delivery requests, and furnishes the delivery requests. Second - a client sends out a search message, receives a response from a server, sends out a service description request, receives the service description, sends out a service deliver request, and receives the service.

A distinguishing feature of Konark is that it enables each device to act as a server and a client simultaneously. We use the term “client” for any device that is interested in using a service being offered by some peer in the network. We use the term “server” for any device whose service is being used by some peer in the network. Konark defines components that allow devices to assume such a dual role. Each device includes a Client application that facilitates human interaction to initiate and control advertising, discovery and service usage. It also includes Service Managers and Registry that together maintain service objects and information about services offered by peers in the network. A micro-HTTP server is present to handle service delivery requests.

A. Service Discovery

Each device has a Konark SDP Manager that is central to the service discovery mechanism. This acts like a broker to discover the required services, and also registers and advertises device’s own services. Figure-1 shows, the Konark service discovery protocol stack. Konark SDP Manager interacts with the messaging layer to send and receive the discovery and advertisement messages. The messaging layer is built above the TCP/IP or UDP/IP. Service Discovery component of Konark deals with device-addressing and networking, service registry structure, and service discovery/advertisement mechanism.

![](image)

**Figure 1. Konark Service Discovery Stack**

1) Networking and Addressing Assumptions

The devices with local wireless connectivity form an ad-hoc network when they come in each other’s vicinity. We assume an IP level connectivity between these devices over any wireless link layers like 802.11 or Bluetooth. In the absence of the administrative services, these devices obtain an IP address by zero-configuration mechanisms like Auto IP techniques used in this protocol. Most of the current operating systems like Windows-CE implement this technique. These devices can be in each other radio range to form a picone network. If the devices are not in each other range, we assume that each node has a routing capability to form a wider ad-hoc network. All these devices join a locally scope multicast group - 239.255.255.251. This enables peer-to-peer networking among these devices.

2) Service Registry

Service registry is a structure that enables devices to store their local services. It also allows them to maintain information about services that they might have discovered or received via advertisements. Existing service discovery protocols do not provide a registry structure for this latter client functionality. While designing a registry, it is vital to consider the limited memory resources of some devices. Each device should be able to delete and avoid unwanted services. There is also a requirement for a watch-mechanism that deletes services from local registries when those services are no longer available in the network. Konark uses a distributed approach to service registry. The Konark SDP manager maintains a tree-based structure as registry. Thus we use the terms registry and service tree interchangeably. The tree structure simplifies management of the services as it provides a hierarchy to access services. It also enables direct correlation between the storage of services and their advertisement and discovery.

![](image)

**Figure 2. Example Service Tree**
The tree has a number of levels that represent the service classification. As we move down the tree from root to the leaves, services become more specific. An example service tree is shown in the figure-2. The oval shaped nodes represent generic service types that form a basic service tree. Such a tree will be uniform and present across all devices implementing Konark architecture. We argue that it is possible to come up with a basic generic service tree that can accommodate most of the services needed by a mobile user. Such a tree would be stored as an XML file to be easily scalable.

Services shown in rectangles are actual services and can be added under any generic service node based on their classification type. These actual services can either be offered by the device itself or available in the environment. The former are called Registered Services and have to be registered by the user to be present in the tree. The example tree shows the service “Chess” registered under the path “RootService:Services:Entertainment:Games:Chess”. Services available in the environment are termed as Available Services and have to be either discovered or should be advertised by the device on which they were registered. The example tree shows the service “Fax” stored under the path “RootService:Services:Information:Personal:Fax.” The path from the root to the node uniquely identifies each service. If there is more than one service, either available or registered, that subscribe to the same path, then all of them will be stored in the same location. These will be differentiated on the basis of their respective URLs. From a user perspective, these services could also be different on the basis of their Service Names and their available service description, both of which have been explained in a subsequent section.

Services can be further classified as “all”, “generic” or “specific” based on the tree-level. This classification can either be for registered services or available services. Services identified at the root node (Path=RootService:Services) will signify “all” services in the tree. Service at any of the basic generic service node will signify “generic” services. These include all the actual services present below that node up-to the last level. Actual services (either registered or available) are called “specific” services and constitute leaf nodes of the tree.

The service tree is useful to devices when acting as client as well as server. This tree is used by a server to register local services that it wishes to offer, to advertise the registered services at any level - “all”, “generic” or “specific”- and to respond to the client discovery requests that is based on either path or keyword. Clients use the service tree to discover the services - “all”, “generic” and “specific”- based on path or keyword and manage these services. Use of options like inclusive or exclusive filtering at different levels of tree can help the user to check the number of services entry into this tree. Device capabilities like memory size can also be used as a parameter before adding any discovered or advertised service into the tree.

Each service is associated with a lease time, that is, the time for which the service will be available in the network. This time is specified as time-to-live when the service is registered on its host device. When the service is advertised or discovered, the time information is transferred to the client as well. At the client side, a lease thread uses this time information to check all available services periodically and remove services that are no longer available. This helps in building a self-healing system. Once a service’s time expires, the server has to update its the time-to-live if it wants to keep that service registered.

The Konark service tree can be considered as a Christmas Tree. The reason behind usage of these semantics was a vision that when a user would walk into an ad-hoc network, his device would discover services which would show up on his tree in different colors and forms. These lights would bring an ambience to the service tree the way lights decorate a Christmas Tree. These lights would turn off when the services would not be available.

### 3) Service Discovery and Advertising

To discover services in the network, clients use a discovery process known as active pull mechanism. Servers use an advertisement process to periodically announce their registered services. This mechanism is termed as passive push. Konark supports both push and pull mechanisms for variety of reasons. Firstly, it provides flexibility to both clients and servers to discover and advertise services on a need basis. This is especially useful for environments rich with m-commerce oriented opportunities. Secondly, clients don’t have to keep a global-view of services at all times as they can use a pull mechanism. This reduces the memory storage requirements at the client side. The servers also do not have to broadcast too often to provide an updated service view. This reduces the number of messages sent by the server.

Discovery process has two steps. In the first step, a client sends out a discovery message on a fixed multicast group. In the second step, all the servers that have the service being sought would respond. To accomplish the first step, the client creates a discovery message that contains either path from the service tree or keyword. The path is used when the client desires “all” services in the network or services defined by some “generic” service type. For the discovery of a particular service (“specific”), keyword will be used. The message also has the port number where the client listens for the server reply by unicast. The figure-3 shows the discovery message.

![Figure 3. Service Discovery Message](image)

The client sends out this message on a fixed multicast group address so that each node receives this message. On receiving a discovery message from the client, the server performs the service matching. If it is a path-based discovery, the server matches the path with its registry tree and gets all the registered specific services under that node or its children nodes. If the discovery is based on a keyword, the server matches this word with the defined keywords for each of its own registered services. If the server finds a match, it creates a service advertisement message for each match found. This message contains the actual service name, the path of the service, the type of the service, the URL where the service description will be available and the time-to-live of the service.
specified in minutes. Figure-4 shows the service advertisement messages.

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Path</th>
<th>Type</th>
<th>URL</th>
<th>TTL</th>
</tr>
</thead>
</table>

Figure 4. Service Advertisement Message

Similar to the discovery process, advertisement can also be based on “all”, “generic” or “specific” services. The server can advertise “all” its registered services, “generic” services identified by some path in the service tree, or a specific service defined by a leaf node in the tree. Depending on the number of services that match the selected advertisement criteria, the server creates an advertisement message for each of those services and sends it to the multicast group. On receiving a service advertisement messages, the clients pick up the path and match it with their registry tree. While doing so if they find that there is an exclusive filter on any of the node of the path, the message is discarded. If not, the service is added under the specified path. This service will be associated with a lease thread that will delete this service once it is expired. If the path has inclusive filter set, it notifies the user about this service.

B. Service Delivery

In this section we describe how Konark achieves service delivery. As mentioned in previous sections, delivery is again a two-step process – service description where the client learns about the properties and capabilities of the service, and service usage where the client avails the capabilities. Before we explain any of these steps, it is imperative to understand the service description language as both the description and usage depend upon it.

1) Service Description Language

Under Konark architecture, each service is a bundle of two components – a service description file that describes a service, and a service object. The registry of any server device contains these two components for each service being offered by that device. While the service object can be in the form of a class file or a DLL, the description file is a plain text file containing complete information about the characteristics and functions of the service. Konark defines an XML based service description language to enable services to explain their characteristics. Figure-5 defines the various tags defined in the aforementioned language.

The root of the document is the Service tag. It represents the start of the service definition. The first child is ServiceName. It is used to define a user-friendly name for the service. This is followed by ServiceType, which defines the type of the service, e.g. printer, music. The service type along with the relevant path in the tree maybe used to advertise the service. The third child is Keywords. It includes words that could be possible search types used during discovery phase. These words may describe the service type or some characteristics of the service, e.g. for a service offering opportunities to print documents, while the ServiceType could be “print”, keywords could include “laser printers”, “color printers”, or “printing”.

While the first four children of the Service tag are primarily used in service discovery and advertisement, Properties and Functions are used in service description and delivery. Properties describe the characteristics of the service. Each service can have any number of properties. Each property is a combination of a name, a description, and a value. The Name of the property is used for communication between server and client. Using this name, the client application may be allowed to subscribe to events related to this property, e.g., the client may be interested in being informed when the property changes to a particular value. The Description is a user-friendly explanation of the property. Value gives the current value of the property.

```xml
<Service>
  <ServiceName>Service Name</ServiceName>
  <ServiceType>Service Type</ServiceType>
  <Keywords>Keywords</Keywords>
  <Properties>
    <Property>
      <Name>Name</Name>
      <Description>Description</Description>
      <Value>Value</Value>
    </Property>
  </Properties>
  <Functions>
    <Function>
      <Name>Name of the function in the service object</Name>
      <Description>Description of the function</Description>
      <Parameters>
        <Parameter>
          <Name>Name</Name>
          <Type>Type</Type>
          <Description>Description</Description>
        </Parameter>
      </Parameters>
    </Function>
  </Functions>
</Service>
```

Figure 5. Service Description Language
Functions are related to the actual usage of the service. A service can have any number of functions. Each function is a combination of a name, a description, some parameters, and a return parameter. The Name of the function is the name given to the actual method in the service object. This is used for communication between server and client. Description is an explanation to the user as to what action the function provides. Parameter represents the arguments needed to invoke a function. Similar to properties, a parameter also contains a name, a description and a type. While Name is for communication between server and client, Description is to guide the user as to what information is required to invoke the function. Type specifies the data type for the argument, e.g. type could be string, integer or file. The client application uses Type information to enforce the validity of user input, e.g. if the type is defined as 'int', client application can ensure that user does not enter a 'string.'

Each function also includes a ReturnParameter tag. This is similar in structure to Parameter tags. It represents the information obtained on invoking the function.

The language described above is loosely based on WSDL (Web Services Description Language) [W3C01], the emerging standard for describing web services. However, WSDL, in its current form is used by applications specific to a service. E.g. a client side application that is used for displaying weather data to a user will be based on a WSDL document describing a weather service. The same client application cannot be used for providing information about printers. Also, if the WSDL file is changed at any time, all applications using that service will have to change. In our language, we tried to come up with a template that can describe any type of services. It may not be as deep and powerful as WSDL, but it has its own merits of simplicity and a wide range of services that can be described.

2) Architecture

Service delivery primarily involves communication between a Client Application and the micro-HTTP server of the service provider.

The client device invokes the URL obtained during the discovery phase to get more information about the services. A typical URL would be similar to http://169.254.30.42/music.xml. This implies that a device with IP address 169.254.30.42 is offering some service whose description can be found in the file music.xml. The HTTP server returns this file to the client device that retrieves the necessary information from the file.

```
POST 169.64.32.10 HTTP /1.1

SOAPACTION: Music for you!

<SOAP-ENV:Body>
<GetSongs>
<Parameters>
  <Parameter>clip.wma</Parameter>
</Parameters>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The client device invokes the URL obtained during the discovery phase to get more information about the services. A typical URL would be similar to http://169.254.30.42/music.xml. This implies that a device with IP address 169.254.30.42 is offering some service whose description can be found in the file music.xml. The HTTP server returns this file to the client device that retrieves the necessary information from the file.

![Figure 6. Service Delivery Components](image)

Once a service has been discovered by a client device, the client has very limited knowledge about the service. It only knows the user-friendly name of the service, service type, IP address of the device offering the service, and the duration of availability of the service. Based on this information, if the user of the client device is interested in the service, he attempts to get more information of the service. This stage is known as service description, and is the first step in service delivery.

```
<SOAP-ENV:Envelope soap:encodingStyle="urn:schemas-xmlsoap-encoding" soap:namespace="urn:konark">
  <SOAP-ENV:Body>
    <GetSongs>
      <Parameters>
        <Parameter>clip.wma</Parameter>
      </Parameters>
    </GetSongs>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 7. Service Delivery Request

The service description file contains complete information about the properties of the service and the methods provided by the service. The user can interact with the service by invoking any of the available functions with proper parameters. The user function invocation is packaged as a SOAP [W3C00] request and sent to the HTTP-server. The figure-7 shows an example of a method invocation request.

The micro-HTTP server parses the SOAP request to retrieve service name, and method name and arguments. The HTTP server retrieves the service object from the registry using the service name. It then passes the method name and the arguments to the service object for method invocation. The HTTP server then passes the result of the above operation back to the client device. The result could be some error message if the server incurs any errors while invoking the function and can also contain more information about the contents it is returning to the client device.

V. IMPLEMENTATION

This section covers the implementation of Konark. We divided the implementation into two sections, one for service discovery and another for service delivery.

A. Implementation Platforms

Our protocol is operating system and programming language independent. However, it assumes IP networking among the devices. We have implemented our protocol in two versions of Java, namely: Personal Java 1.2 [SUN97] and J2ME CLDC/MIDP [SUN01].

Personal Java 1.2 is JDK1.1.8 compliant. For this implementation, we used Pocket-PC 3.0 based iPAQs from Compaq and the Jeode VM from Insignia Inc [INS02]. These
iPAQs have SA-1110 processor from Intel. We used Lucent’s wireless card for the 802.11 wireless interface to form an ad-hoc network among iPAQs. Our second implementation was for J2ME CLDC/MIDP platforms on devices like Motorola iDEN Phones.

B. Service Discovery

The Konark-SDP Manager, along with the messaging layer, helps in the discovery process. The figure-9 shows the Konark service discovery stack.

1) General Architecture

The stack has TCP/IP and UDP/IP over wireless link layers like 802.11 or Bluetooth. Service Manager along with Registry and Services pool form a Konark SDP Manager. The stack has two components: Konark SDP Manager and Messaging Layer.

Konark SDP Manager: The Service Manager along with Registry and Services Pool form a Konark SDP Manager. This provides an application user interface to build Konark server and client applications. These APIs are broadly classified into two categories. One set of the APIs for service discovery and advertisement while another set is for service management. Service Manger forms the heart of the Konark SDP Manager and it is the one that provides the APIs. It interacts with Registry and Lower Messaging Layer to perform all the operations. Registry is the data structure that holds the services in a tree structure. Service Manager interacts with this registry to manage the services. Each node in the tree represents a service. Both the client and the server share a basic generic service tree. In addition to this, the client can have the services available in that environment. In case of the server, it will have its registered services to be offered.

Messaging Layer: This is the lowest layer of Konark Service Discovery Stack that takes care of sending and receiving unicast and broadcast messages. These are the discovery and advertisement messages.

2) Service Discovery APIs

Registry provides the interface to the Service Manager for the service management and other operations. Service Manager performs the following tasks:

- Provide Application level APIs to manage the services in the registry,
- Provides Application level APIs for discovering the services to the Client,
- Provides Application level APIs for advertisement of the services to the Server,
- Provides the interface to the lower level-messaging layer to handle the advertised server services in the client and to handle the client discovery requests in the server,
- Delegates the task of sending and receiving the messages to the messaging layer.

The lower messaging layer has client component to perform client discovery and handling server advertisement messages. Its server component performs service advertisement and handling client discovery requests. The table-1 lists all the packages that implement Konark Service discovery and delivery protocol.

C. Service Delivery

As discussed in section 3, the three major components used in service delivery include an HTTP server, a Konark client application, and a registry. To support the functionality of these components, there are some utilities, namely, “deliver-utils”, and “description-utils”. While the HTTP server is used exclusively for service delivery, the client application can provide a user interface and the registry holds the service objects.

1) Micro-HTTP Server

The HTTP Server can be considered as the main component of the service delivery architecture. It has two main functions – responding to service description requests from clients, and responding to function invocation requests. The micro-HTTP server provides the traditional HTTP server functionalities. It can handle multiple user requests. It uses HTTPRequest and HTTPResponse objects to communicate between the client device and HTTP-server. It also handles the SOAP requests during the service delivery phase. It interacts with the registry through Service Manager to retrieve proper service objects and interact with them.

2) Utilities

There are two main categories of utilities – description utilities, and delivery utilities. The description utilities help the client application during the service description phase. The delivery utilities assist during the function invocation phase. The description utilities provide the following functionalities at the client side:

- Parse the service description file received from the HTTP-server of the service provider,
- Create a service object from the parsed information and put it into the registry,
- Use the service object to present the service properties and available functions to the user in a human-understandable format.
The delivery utilities provide the following functionalities during the service delivery phase:

- Package the service-function invocation as a SOAP request and send it to the micro-HTTP server of the service provider,
- Present the output of the above operation from HTTP server to the user, including error handling when required.

3) Creating a Service

Writing services for Konark architecture is a two-fold process – writing the description XML file using our service description language, and then implementing the service. The required format of the service description has been discussed at length in section 3.2.1. It includes name of the service, service type, a list of keywords, a list of properties, and a list of functions available. While the type, the keywords, and the properties of the service are related to, and thus influenced by, the business perspective of the service offering, the functions are the critical part for the application developer. The developer has to decide what all parameters would be required for invoking a function, the types of each of those parameters, and the return parameter.

### TABLE I. KONARK PACKAGES

<table>
<thead>
<tr>
<th>Packages</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>edu.ufl.asdp.serverservicemanager</td>
<td>Server related operations of the Service Manager.</td>
</tr>
<tr>
<td>edu.ufl.asdp.clientservicemanager</td>
<td>Client Related operations of the Service Manager.</td>
</tr>
<tr>
<td>edu.ufl.asdp.registry</td>
<td>Registry Data Structure and Basic Service Tree parsing (XML File).</td>
</tr>
<tr>
<td>edu.ufl.asdp.services</td>
<td>All the types of services like generic service type and specific services.</td>
</tr>
<tr>
<td>edu.ufl.asdp.util</td>
<td>Utility classes and Message Related classes.</td>
</tr>
<tr>
<td>edu.ufl.asdp.lease</td>
<td>Leasing related classes.</td>
</tr>
<tr>
<td>edu.ufl.asdp.serverservicelayer</td>
<td>Server related operations of the messaging layer.</td>
</tr>
<tr>
<td>edu.ufl.asdp.clientservicelayer</td>
<td>Client related operations of the messaging layer.</td>
</tr>
<tr>
<td>edu.ufl.asdp.exception</td>
<td>Konark exception for simple error messages.</td>
</tr>
<tr>
<td>edu.ufl.asdp.HTTP</td>
<td>MicroHTTP server related classes.</td>
</tr>
<tr>
<td>edu.ufl.asdp.descutils</td>
<td>Description Utilities.</td>
</tr>
<tr>
<td>edu.ufl.asdp.deliverutils</td>
<td>Delivery related classes.</td>
</tr>
<tr>
<td>edu.ufl.asdp.ui</td>
<td>User Interface related classes.</td>
</tr>
</tbody>
</table>

All services must include a function declared as public byte [] handleSOAPRequest (String methodName, Vector inParams) throws ServerException defined in the AppService Interface. This method can be considered as the interface between the HTTP server and the service object. This method achieves two goals:

- Abstraction of the service method implementation from the method invoker and the HTTP server,

- Output from HTTP server to the client device always in the same format of byte[] with more information about how to extract it.

VI. USER INTERFACE

We provide a simple application programmer interface (APIs) so that the actual services can be built easily and can be advertised, discovered and delivered. We have developed a simple, intuitive user interface to demonstrate the Konark functionalities that map to the APIs. Figure-9 (a) shows registered service “Song for you!” under “Music” and available service “Print Your Files” under “Personal”. The client device is going to discover service under “RootService:Information:Food” as shown in Figure 9(a). Services and Tree Options together provide all the functionalities of discovery, advertisement, service registration and exclusive/inclusive filter setting. Figure 9(b) shown invocation of action on a “Song for you” service by another iPAQ that is using the service of iPAQ in figure 9(a). This comes under service delivery phase. The user can enter the proper parameters for the function a dialog box shown and can invoke a function to get a music file from the iPAQ of figure 10(a). We provide all the functionalities of Service Delivery through this simple user interface.

![Figure 9. User Interface](image)

VII. PERFORMANCE CHARACTERIZATION

This section covers the performance characterization of our protocol. Currently, there is no technology independent benchmark for services and scenarios [MAT01]. Also there are no proper tools to measure the performance in such environment of devices and services. We simulated the environment using two iPAQs and a laptop.

The performance characterization is done at two levels - service discovery, and service delivery. For discovery, from a client’s perspective we measured the latency time to discover the service, and from a server’s perspective we measured the access time for the registry. In case of service delivery, we measured the performance of the HTTP server. The main aim of this performance evaluation was to find the metrics and quantitatively measure them.
A. Service Discovery Performance Characterization

We have done preliminary work in characterizing the performance at the service discovery level. This has been done in the form of latency time and registry access time.

Latency time is the time taken to discover a service in the network. The graph in figure-10 shows the affect on the latency with increase in the number of services in the network.

We considered three cases to measure the latency:

- To get “all” the services in the network,
- To get the specific service by “path”,
- To get the specific service by “keyword”.

The latency time to get “all” the services from the network increases as the number of services increase. This is as per expectations. In case of search by “path”, the latency time remains constant. This is because the time taken to map the desired path to the service tree remains the same. In case of search by keyword, as the number of services increase, the time taken increases. This is because search by keyword is achieved by using a linear search mechanism. This factor is more dependent upon the number of services per device and the time that device takes to search its services for a particular keyword.

We have also measured the registry access time when the server gets a discovery request from the client. The access time has been measured for the requests of the following cases:

- To search and get “all” the services in the registry,
- To search and get specific service by “path”,
- To search and get specific service by “keyword”.

As shown in figure-11, the access time increases for searching “all” the services with an increase in the number of services increase. The access time for search by “path” remains almost the same. In case of search by “keyword”, it makes a linear search, searching each service for the keyword thereby increasing the access time as the number of services increase per device.

B. Service Delivery Performance Characterization

The primary component of our service delivery architecture is the HTTP server. Thus, to characterize Konark’s performance in service delivery, we focus primarily on HTTP server.

As mentioned in the design, a participating node in the architecture has a client and a server component. Both these components act independent of each other. We evaluated performance of the server in the following two cases:

1. When the device is only in the server mode,
2. When the device is in the client and server mode simultaneously.

The performance of the device as a client is not very significant as it depends on the network traffic, the characteristics of the underlying network, and more importantly on the performance of the other servers.

1) Actions Analyzed

The server has two basic functions, namely, handling service description requests, and handling service usage requests. For the service description requests, the server’s only job is to read the requested file and return it to the asking client. Thus the performance depends solely on the server. However, when the server receives a service usage request, the performance depends to a lot extent on the time taken by the service object, e.g., if the action requested is to book a ticket for some movie, then the overall time taken would depend on the database being used by the service. However, if the action invoked were as simple as getting some static “hard-coded” data, then the response time would appear much less. Considering this factor, we discuss the performance of the server while handling service description requests.

2) Measurement Approach

Benchmarking requires two main tools – a load generator, and workload. For the load generator, we used a laptop running a simple Java program that simulates the actions of a real participant in the network, that is, makes service description requests, and make service usage requests. For each of the two tasks, the program takes in a numeric argument and spawns that many number of threads, each of which makes simultaneous requests to the same server. To vary the workload, different values of the numeric argument were used.

![Figure 10. Latency Time Graph](image)

![Figure 11. Registry Access Time Graph](image)
We used an iPAQ as a server with a sample service description file of 10Kb. 

3) Performance in handling service description requests

As can be seen in the above graph in figure-12, as the number of simultaneous hits increases, the average response time increases. However, if we assume an average load of 5, that is, an iPAQ is offering some service in which 5 people are interested at the same instance, we see that the response time is around 2.5 seconds.

Figure 12. Response Time Graph

VIII. CONCLUSION AND FUTURE WORK

We were able to design a framework for service discovery and delivery for ad-hoc networks for the device independent services. This can be implemented on any operating system and in any programming language. We implemented the protocol for two Java versions namely Personal Java 1.2 and J2ME CLDC/MIDP. We demonstrated the protocol using a simple user interface. We have done preliminary work in performance analysis.

We did not consider security while designing the protocol. Security has to be considered at all the levels of protocol. Standardization of the basic generic service tree is also an important aspect to accommodate most of the services needed by a mobile user. Standardization of the Konark service description language is also necessary along with extending it to describe wide variety of services. Finally the human-device interaction in the form of user interface has to be improved to give the feel of actual Christmas tree. Some of the possible improvements and extensions suggested here are underway as the second phase of this project at Harris Mobile Computing Lab at the University of Florida.

References


[MAT01] Olivier Mathieu, Doug Montgomery and Scott Rose, Empirical Measurements of Service Discovery Technologies, NIST Information Technology Laboratory, Advanced Networking Technologies Division, Gaithersburg, April 30 to May 4, 2001, PC 2001 Workshop.


