Project Centaurus : A Framework for Indoor Mobile Services

Lalana Kagal, Vlad Korolev, Harry Chen, Anupam Joshi, Timothy Finin
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
email : {lkagal1,vkorol1,hchen4}@cs.umbc.edu
phone : 410-455-3971
fax : 410-455-3969

Abstract— **In an age where wirelessly networked appliances and devices are becoming commonplace, there is a necessity for connecting them to work together for a mobile user. The design outlined in this paper provides an infrastructure and communication protocol for providing services to these mobile devices. This flexible framework allows any medium to be used for communication between the system and the portable device, including infra-red, radio frequency and Bluetooth. Using Extensible Markup Language for information passing, gives the system a uniform and easily adaptable interface. We explain our trade-offs in implementation and through experiments we show that the design is feasible and that it indeed provides a flexible structure for providing services.**

I. Introduction

As the world moves towards greater automation in homes and offices, we enter the realm of 'SmartHomes' and 'SmartOffices' controlled by sensors and/or portable devices, where not only has mobility been incorporated, but where intelligence has become an inherent part of providing services. With Bluetooth [4] just upon us, wirelessly networked appliances and devices will soon be a reality of the future. Now-a-days, we see a lot of 'intelligent' services, that use some kind of logical reasoning to provide better and more relevant support to individual users. These devices and services will have to be integrated seamlessly into the environment that the user is familiar with and provide a uniform interface to any device that the user might want to use.

Our goal is to provide an infrastructure and communication protocol for providing wireless services, that minimizes the load on the portable device. While within a confined space, the Client can access the services provided by the nearest Centaurus System (CS) via some short-range communication. The CS is responsible for maintaining a list of services available, and executing them on behalf of any Client that requests them. This minimizes the resource consumption on the Client and also avoids having the services installed on each Client that wishes to use them, which is a blessing for most resource-poor mobile clients.

We also expect all Services to communicate via Extensible Markup Language (XML). We found this W3C Standard [3] to be very useful in defining ontologies and describing properties and interfaces of Services. As this is already being widely used, we think that it will help in integrating Centaurus with already existing systems.

To verify the feasibility of our infrastructure, we will use IR [10] for communication between the Client and the CS in our first stage of the development. One of the main drawbacks is the limitation of the infrared architecture. However, we believe that the simplicity and the affordability of the infrared devices can overcome these limitations. We would like to emphasize that any other medium could be used for communication including Bluetooth; all we provide is the framework.

This paper is organized as follows: Section II discusses other technologies, and compares Centaurus with one such technology, Section III gives a brief overview of our design. In Section IV, the design and modeling issues are covered. The actual implementation is detailed in Section V, with the protocol illustrated in Section VII. The results of the experiments are described in Section VIII, and Section IX concludes the paper with a summary of our results and a discussion of future research directions.

II. Related Work

In the last couple of years, a number of technologies have emerged that deal with 'Smart' Homes and Offices. Among them we were particularly interested in the UC Berkeley Ninja Project [1].

The Ninja project tries to link different services, through a range of devices devices ranging from PCs to cell phones and Personal Digital Assistants [1]. It has incorporated intelligence into the infrastructure and has the ability to adapt the content to a specific device.

Some differences lie in the implementation of the application, and the security infrastructure [2]. Currently we have not included security in our framework. But due to our flexible design, controlling access to services by means of Access Control Lists will be easy to integrate into the Service Manager. Ninja tends to concentrate on Web-based Services, whereas our system is able to support Services

based on any platform, as long they can communicate with either the Service Manager through sockets, or one of the Communication Managers through the native protocol and possess the ability to process CCML messages. We also do not distinguish between hardware and software Services, allowing the user to use either in the same way. Unlike the Ninja project, Centaurus infrastructure delegates the state management to the Services themselves with the Service Manager serving as the cache. The advantage of such approach is the decreased complexity of distributed state management and increased fault tolerance. Even in the event of Service Manager going down, the state information is still preserved, and it will be uploaded back to the Service Manager after it comes back up. This happens because the Services send regular status updates to the Service Manager. Since all of the communication between Services and Clients in the Centaurus project are done with the use of CCML, there is no need for complicated Operators and Paths used by the Ninja project to convert between different data representations.

Though both the Ninja project and Centaurus are aimed at providing a uniform infrastructure for a multitude of devices to use heterogeneous services, Centaurus is more applicable for 'SmartHomes' and 'SmartOffices' because of its independence of any kind of specific communication infrastructure; so it could be easily implemented in the wide range of environments. In addition, Centaurus architecture is less prone to the failures of its components because of the use of multiple communication managers and automatic state recovery in the event of the Service Manager failure.

III. Overview

The main design goal of Centaurus is to develop a framework for building portals, using various types of mobile devices, to the world of 'things' that users can communicate with and control. Centaurus provides a uniform infrastructure for heterogeneous services, both hardware and software services, to be made available to the users everywhere where they are needed.

The research and development of Centaurus falls in the categories of the service/resource discovery and intelligent room/space. Centaurus differs from many of the existing architectures like the Service Location Protocol (SLP) [8], Jini [6], E-Speak [5] and the 'universal interaction' architecture by UC Berkeley in the several ways. Some of the key features of Centaurus are the following:

- The Service interface and communication protocols allow users to use different types of mobile devices as the portals to the world of 'things' that they can communicate with and control. These devices may differ in the form of user interface, computation power, resource availability and size.
- Centaurus uses XML as the sole data exchange format between the service requester and service provider. The Centaurus infrastructure defines the Centaurus Capability Markup Language (CCML), which provides a flexible and simple content description that enables the creation of content UI (not limited to Graphical User Interface only) that scale across a wide range of devices, like laptop, PDA, cellular phones etc. The infrastructure also defines the Centaurus Interface Definition Markup Language (CIML), which provides a machine-understandable description that allows applications to validate and interpret service descriptions and capabilities. This information can be used in various inference processes.
- The Service Manager is designed to be decentralized. Services can dynamically join and leave the system in a robust and flexible fashion.
- A Service Requester, a user of a service, can either be an end-user or another service. This hybrid architecture allows one service to be a composition of many services.

IV. Design

A. Scenario

A 'SmartRoom' is equipped with a Centaurus Communication Manager, which continuously broadcasts ,through some medium, a client application. A person with a portable device who enters the room for the first time is given the option to install the software. Once the application is installed, it continuously reads the updated list of services. The person is able to choose a service, select a function, fill in the related options and execute the function. These services may be provided by Centaurus systems other than the one the portable device is connected to.

B. Centaurus protocol

Centaurus protocol is used to communicate with mobile clients and services. The Centaurus protocol consists of Centaurus Level1 protocol and Centaurus Level2 protocols. Centaurus Level1 Protocol is used as a glue between some existing communication architecture such as IrDA stack, Bluetooth, or TCP/IP and the generic Centaurus Level2 protocol. The Centaurus Level1 protocol handles connection and disconnection issues, identification and authentication of the clients and interaction with architecture specific protocols such as IrLAP and IrLMP or Bluetooth. The Centaurus Level2 protocol handles transmission of the XML messages, time synchronization, message fragmentation and re-assembly. The Centaurus Level2 protocol is designed to be insensitive to disconnections, handle multiple clients, provide minimal turnaround times and be easily portable. In fact in current implementation all communication managers and client communication modules use the exactly the same codebase.

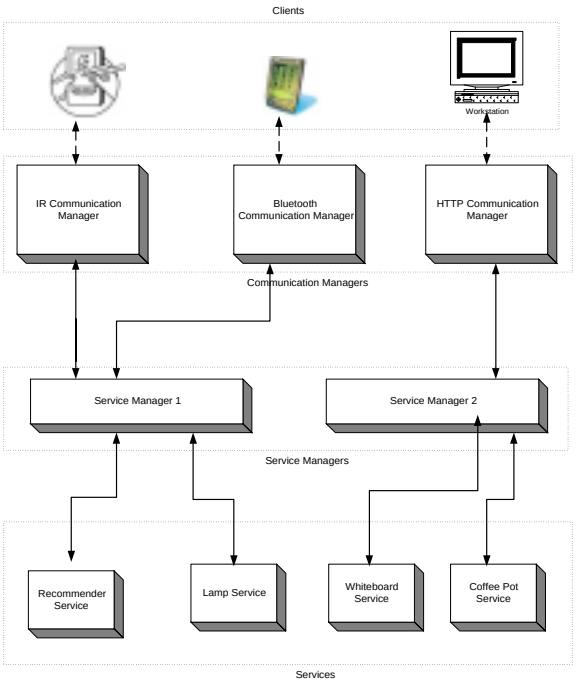

Fig. 1. Centaurus Components

C. Components

There are four main components in a Centaurus System; the Service Managers, the Services, the Communication Managers, and the Clients.

The Communication Managers handle all the communication with the Centaurus Client. There can be more than one Communication Manager, each implementing a different protocol, for example, one that handles IR, another that works with Bluetooth, one that works with HTTP to provide a web interface etc.

The Service Managers are the controllers of the system that co-ordinate the message passing protocol between Clients and Services. The Service Managers, arranged in a hierarchy, also interact with each other, so that the Services on each are available to all the Service Managers.

The Services are objects that offer certain services to the Centaurus Client. By querying the Service Manager on the top of the hierarchy, the Services are able to locate the closet Service Manager and register themselves with it. Once registered, the Services can be requested by any Client talking to any Communication Manager.

The Centaurus Client provides a user interface for accessing and executing Services.

Fig. 1 shows the different components and the relationships between them.

C.1 Service Manager

The Service Manager(SM) acts as a mediator between the Services and the Client. This is disregarding the fact that the Client sends the information to its Communication Manager that forwards it to the Service Manager. When a Service starts up, it has to register with the Service Manager, sending its CCML file. This file contains its name, id and the interfaces it implements. When a new Client comes along, the Service Manager sends it a ServiceList Object. This ServiceList object changes dynamically, according to the services registered with the Service Manager. So the Client always has the updated list of services. The Client can select a service, which causes the Service Manager to send the CCML file for the service. The Service Manager then updates its database to reflect that the specific Client is interested in the requested Service. Whenever the Service Manager gets a status update of the Service, it will send it to all interested Clients. The Client will continue to receive status reports from the Service, until it de-registers itself. The Client sends the new CCML file to the Service Manager, after invoking the interfaces of the Service. On receiving this CCML, the Service Manager validates the Client and the CCML. If the Service is still available, the Service Manager sends the CCML to it, otherwise it is queued for a certain amount of time. Once this timeout expires, an error is returned to the Client. The SM is also responsible for service discovery and leasing. It allows Services to register for a certain amount of time. If it does not receive any status update within that time, the registration is deleted. The SM implements an intelligent lookup for Services, enabling the Clients to search for Services that provide a certain kind of or related function.

In the current design, the main task of the Centaurus Service Manager can be described as the following:

1. Communicate with the Client through CCML
2. Inspect the incoming 'command' or 'update'
3. Dispatch the command to the appropriate services or the Service Management sub-components
4. Handle all status updates, and make sure all interested parties are informed of the updates
5. Interact with Services using CCML
6. Provide service registration services and discovery services

C.2 Communication Manager

This is responsible for the communication between the Client and the Centaurus system. As mentioned earlier, the system can have one Communication Manager for every type of communication it desires to implement. Each Communication Manager queries the highest Service Manager, and retrieves information about the Service Manager closest to it. Then it uses this Service Manager for client servicing, for a certain period of time. After which, it queries the topmost Service Manager again. Every time the Communication Manager receives information from a Client, it

sends this information directly to the Service Manager that it is currently attached to. When it receives data from a Service Manager, it validates the data and looks at the header to decide which Client to send it to.

C.3 Services

A Service performs a certain action on behalf of the Client. These Services could range from controlling a light switch or a coffee pot to controlling a printer or even a memo pad service, where Clients can leave messages for each other. Each Service has to register with at least one Service Manager, preferably the one closest to it. It sends its CCML file, along with its name, id and a brief description to each Service Manager it wants to register with and it only accepts requests from those Service Managers. Every time its status changes, it informs all the Service Managers that it is currently registered with.

One of the services provided by the Centaurus system is the Recommendation Service. Instead of returning a list of all possible services that are available to a Client, this service recommends a list of services that might be in the interest of the Client based on the existing environment context. For example, the system returns a coffee-maker control service during the morning to the user, and in the evening it returns a light control service to the user.

C.4 Client

A Client is a special kind of Service and is treated as a Service. It has to respond to commands and regularly send status updates. A Client talks to the appropriate Communication Manager and registers itself with a Service Manager. This registration is similar to the registration of Services. On registering, it receives the ServiceList Service, which contains the current list of Services. The ServiceList causes the Service Manager to send the list of Services, whenever a new Service registers, or a Service de-registers.

By choosing a Service, the Client expresses interest in it. It is then sent the CCML file describing the Service. The Client can invoke the specified functions on the Service, by choosing one of its interface. After changing values of certain variables, specified in the CCML for the particular Service, it sends the file to the Service Manager to perform that action. It will receive status updates from all Services that it is interested in through the Service Manager, until it specifically informs the Service Manager that it no longer wants to receive these messages. Every time it wants to perform a certain action on a Service, it retrieves the current CCML file from its list, changes the appropriate values and returns the changed CCML to the Service Manager, which forwards it to the selected Service.

D. Centaurus Capability Markup Language (CCML)

The CCML is divided into 'system' , 'data', 'addons', 'interfaces', and 'info', as shown in fig. 2.

The 'system' portion contains the header information, the id, timestamp, origin, etc. There are two opposing variables, 'update' or 'command'. An 'update' variable is used to inform other Centaurus components about status updates of Services and Clients, whereas the 'command' is only used by Clients to send a command to a certain Service. The system also contains the listening section for a Service or Client. It specifies all the Services that a Service or Client is interested in.

Using the 'addons' section, we can add a related Service to another Service, for example, add an Alarm Clock Service to a Lamp-Control Service. We are not currently using this section.

All information regarding the variables and their types are contained in the 'data' section.

The CCML for a Client always has one or more 'actions' in its data section that a Service Manager can invoke on it. This is used by the SM to change the state of the device.

Actions

- AddService : When this action is set, the Client adds the value of this variable to its InterestList; i.e. the list of services that it is interested in.
- RemoveService : This is set by the Service Manager, if the Service that the Client is interested in, is no longer available. It causes the Client to stop listening or using the Service and remove the Service from its InterestList.

The 'interface' section contains information about the interfaces that the object (Service/Client) implements.

Other details like the description, and icon for representation are in the 'info' section.

V. Implementation

The previous section outlines our overall design, but to facilitate the implementation, we had to make some assumptions and sacrifice some of the features and flexibility. These assumptions in no way compromise the design or results, they only helped in quicker implementation.

To verify the feasibility of our infrastructure, we use IR [10] for communication between the Client and the Communication Manager in the first stage of the development. We have a single Communication Manager that uses IR. The IR Communication Manager carries out the Client discovery. Once discovered, the Client is polled regularly for information. This polling completely eliminates the problem with collision, that occurs in a client push method, when more than one Client sends information at the same time. We also have a single Service Manager and two Services for testing. We assume that the client application is installed on the PDA before it enters the 'SmartRoom'. Communication between any two components in the Centaurus System is done via sockets. The Service Manager and the IR Base Manager have two dedicated sockets each,

```
<!--  Entities -->
<!ENTITY % name  "name CDATA #REQUIRED" > <!ENTITY
% value "value CDATA #REQUIRED" >
<!ENTITY % type  "type  CDATA #REQUIRED" >


<!-- Top level element -->
<!ELEMENT ccml
       (system , data?, addons?, interfaces?, info )  >
<!ATTLIST ccml  version CDATA #REQUIRED >

<!-- system declarations -->
<!ELEMENT system    (
         (full, (command|update), valid?, public?,
interactive?,
            id, manager, time, origin, location, paren
          |
          (diff, (command|update), valid?, public?,
interactive?,
              id, time, origin, location,parent?, list
        ) >

<!ELEMENT   command    EMPTY>
<!ELEMENT update EMPTY>
<!ELEMENT full EMPTY>
<!ELEMENT diff EMPTY>
<!ELEMENT valid EMPTY>
<!ELEMENT public EMPTY>
<!ELEMENT interactive EMPTY>
<!ELEMENT id EMPTY>
<!ELEMENT manager EMPTY>

<!ELEMENT time EMPTY>
<!ELEMENT origin EMPTY>
<!ELEMENT location EMPTY>
<!ELEMENT parent EMPTY>
<!ELEMENT listening (id)*>

                -1-
```

```
<!ATTLIST id        %name; >
<!ATTLIST manager   %name; >
<!ATTLIST time      %value;>
<!ATTLIST origin    %name; >
<!ATTLIST location  %name; >
<!ATTLIST parent    %name; >


<!--- data declaration -->
<!ELEMENT data (attrib)*>
<!ELEMENT addons  (addon)*>

<!ELEMENT addon  EMPTY>
<!ELEMENT attrib  EMPTY>

<!ELEMENT service (#PCDATA)>

<!ATTLIST addon   %name; >
<!ATTLIST attrib  %name; %type; %value;>

<!--- Interfaces declaration -->
<!ELEMENT interfaces (interface)*>
<!ELEMENT interface  EMPTY>

<!ATTLIST interface  %name; >


<!--- info declaration -->

<!ELEMENT info (description?,icon?) >
<!ELEMENT description (#PCDATA)>
     <!ELEMENT icon (#PCDATA)>

                -2-
```

Fig. 2. ccml.dtd

one for listening and one for sending information. As the Service Manager and the IR Communication Manager are at the heart of all communication, we wanted to speed up this process. By giving them a dedicated socket for each type of communication, we reduced the time spent in the creation of a new socket for each connection. Each Service also has a dedicated port for information from the Service Manager. The Service Manager listens to a certain socket for receiving CCML from all the Services. All these sockets are predefined in the Properties file for each component . The information flowing in the system is strictly in the form of CCML (Centaurus Capability Markup Language).

The Service Manager and the Services have been implemented in Java, whereas we chose C, for efficiency in resource management, for the IR Communication Manager and the Client. We found that most of the service discovery architectures are implemented in Java, like Jini and E-Speak. So, if we decide to move to another service discovery system, integration will be relatively easy as the Service Manager and Services are already in Java.

A. Implemented Components

A.1 Service Manager

As mentioned earlier, the Service Manager has to listen to two ports, one for incoming messages from Services and one for messages from the IR Communication Manager. When a Service registers itself, the Service Manager adds it to its list of Services by recording the CCML and the port number the Service listens to. Every new Client is added to the Clients list. All the Services it is interested in are added to the Service-Client list with both the Service and Client IDs. Whenever the Service Manager receives an update from a Service, it updates its Services list and reads its Service-Client list and sends the new CCML to every Client in that list. When a Client sends an update, the Service Manager changes the Clients list. It then reads the list of Services that the Client is now interested in, and appropriately modifies the Service-Client list.

A.2 Client

Our Client waits till it is discovered by the IR Communication Manager, then engages in our IR Protocol to send its CCML to the Manager. It has a list of Services that it is interested in, called InterestList. Whenever it receives a command from the Service Manager, its checks its action, AddService or RemoveService, and adds or removes Services accordingly from its InterestList. If it receives an update, it checks if the Service is in its InterestList. If it is not a Service that it is interested in, the Client discards the message. If it is, then it displays the CCML and waits for user input. If the user changes any variables, it modifies the Service's CCML and sends it to the IR Communication Manager, which in turns forwards it to the Service Manager. The Service Manager makes sure that the CCML is sent to the appropriate Service.

A.3 IR Communication Manager

We have one Communication Manager that handles infra-red communication. It has a port that it listens to, for updates from the Service Manager. The Communication Manager is currently a channel between the Client and the Service Manager, and implements a complicated IR protocol.

In our current implementation, the Communication Manager supports Centaurus Level1 and Centaurus Level2 protocols running on top of modified Linux IrDA stack. Modifications to IrDA stack were necessary for better handling of disconnections and discovery. The IR communication manger is the gateway between Service Manager and mobile Clients. On one side it maintains open TCP/IP connection with Service Manager and on the other side it communicates with mobile Clients. The IR Communication manager listens to both sides for incoming CCML messages and transmits them to appropriate destination.

A.4 Services

We have developed one hardware related service for controlling a lamp and one software service for playing MP3 files. There is another Service, ServiceList, that is an inherent part of the protocol, and is used for providing an updated list of services to the Client.

We have implemented a Service class and ServiceInterface class, that handle validation of the CCML, the reg-

istering of the Service with the Service Manager and the sending of the updates. All Services implemented in Java should, for conformity, extend the Service class, and implement the ServiceInterface class. The ServiceInterface class contains a commandHandler function that has to be implemented by every Service that implements the interface. This is the function that handles changes to the CCML file of the Service. A Java Service need only implement a constructor and this commandHandler to be integrated into a Centaurus system.

As mentioned earlier, the Centaurus system also handles non-Java Services as long they can use CCML and either communicate via sockets with the Service Manager or with a Communication Manager through some native protocol.

- ServiceList
  Each time, a Service registers or is no longer available, the ServiceList triggers the Service Manager to send the updated list of Services to all the Clients. This does not use the Service class or the ServiceInterface class. It is contained completely in the Service Manager. It is a special Service because it is handled in same way as other Services are, but within the Service Manager itself.
- Lamp-Control
  Using X10 [12] devices and FireCracker [11], we were able to control a lamp in the room. We can extend this to control any device because X10 is a power-line carrier protocol that allows compatible devices to communicate with each other via the existing 110V wiring. FireCracker is a Java class that allows a computer to communicate with the X10 device. The Service constructor makes sure that the X-10 device works. The commandHandler function looks for the value of the interfaces. If the 'Powered' interface has a value that is different from the status of the Power variable, then the commandHandler proceeds, otherwise the command is discarded. If the value is true, the lamp is set on, otherwise the lamp is set off. The CCML file is changed and an 'update' is sent to the Service Manager. As we see in fig. 3, the update propagates all the way back to all the Clients that are interested in the lamp service.
- MP3-Player
  Fig. 4 illustrates the working of the MP3-Player service included in our system. We are using a popular MP3 player for Unix, mpg123 [13], that has a Java wrapper around it to allow us to plug it into the rest of the system. The constructor for the Service, reads all the .mp3 files from a specified directory and creates its CCML files. It has a number of CCML interfaces, one for each song it can play. The commandHandler function checks the CCML interface and reads the songs selected. These songs are checked against the current list of songs. If they are valid, they are fed into mpg123 [13]. The new CCML file is created and sent to the Service Manager.

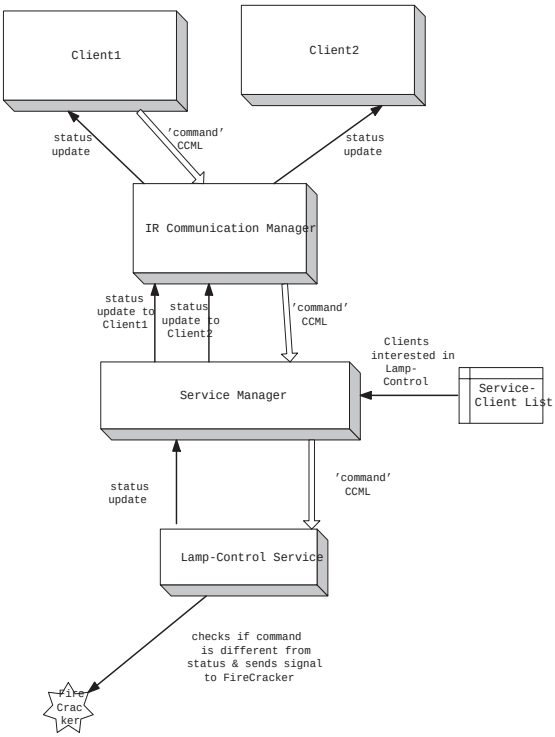

Fig. 3. Lamp-Control Service

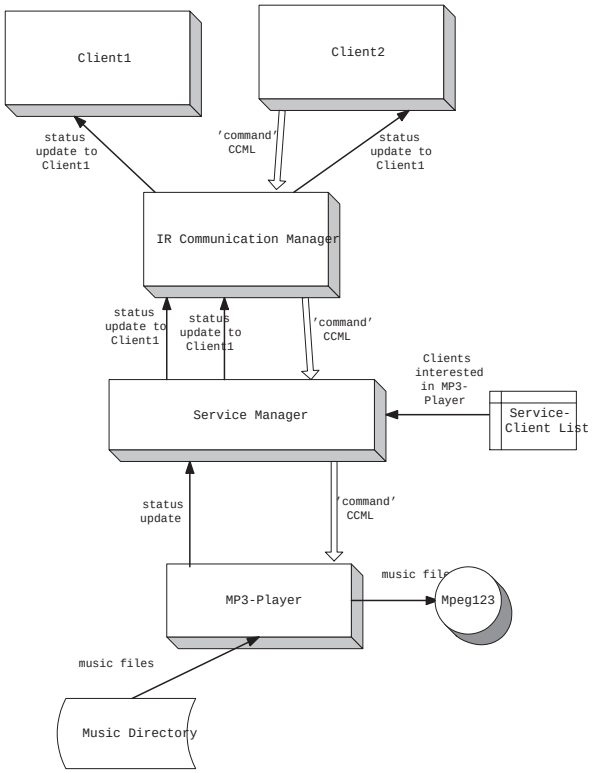

Fig. 4. MP3-Player Service

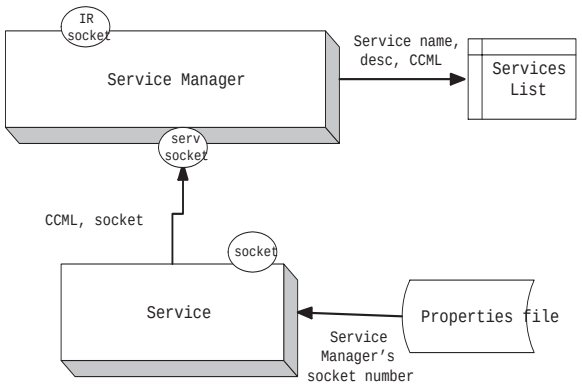

Fig. 5. Registration of Services

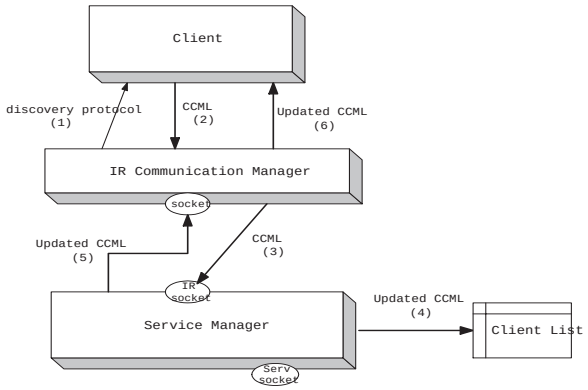

Fig. 6. Registration of Clients

B. Functions

B.1 Registration

Both Services and Clients have to register with the Centaurus system to be visible.

- Services
  As seen in fig.5, a Service on starting up, reads its properties file and retrieves the Service Manager's port number. After creating its CCML file, it sends its CCML and the port number that it is listening to, to the Service Manager's Service port. The Service Manager validates the CCML and adds the Service to its Services list.
- Client
  When a PDA enters the 'SmartRoom', we assume that it has the Client application installed on it, as mentioned in the beginning of this section. It is eventually discovered by the IR Communication Manager, and carries out the IR level protocol. Then it sends its pre-defined CCML file. The IR Communication Manager sends this to the Service Manager's port. The Service Manager after validating the CCML checks if the Client already exists in its Clients list. If it does, then the Service Manager updates its list, otherwise it adds the Client to its Clients list. The Service Manager, sets the ServiceList action and sends the CCML back to the Client.

B.2 Using the Service List

The flowchart in fig. 7 shows in great detail how the ServiceList is used to provide an updated list of Services to the Clients. When the Service Manager gets the CCML of a new Client, it sets the AddService action in data section of the CCML to ServiceList.

$< data >$
$< attribname = "AddService" type = "action" value = "ServiceList" / >$
$< /data >$
It also sets the 'command' variable in the header.

This new CCML is sent back to the Client. The Client realizes that it is a command and checks the actions. It adds the ServiceList to its currently empty list, InterestList; the list of Services that it is interested in.

$< listening >$
$< idname = "ServiceList" / >$
$< /listening >$
When the Client is polled next by the IR Communication Manager, it sends its updated CCML. The Service Manager reads the listening section, and finds the ServiceList. It updates its Service-Client list and sends the list of Services in CCML to the Client. Whenever the list of Services changes, the Service Manager goes through its Service-Client list and sends the new list to all the Clients that are interested in the ServiceList. In this way, the ServiceList works like any other Service, except that it is contained within the Service Manager.

If both our Services have registered then the data portion of the ServiceList would look like

$< data >$
$< attribname = "MP3Player1.0" type = "bool" value = "false" / >$
$< attribname = "Lamp - 001" type = "bool" value = "false" / >$
$< /data >$

B.3 Requesting a Service

When a Client receives the list of Services, it displays this list for the user. The user can select a Service to use. The Client then creates a command for the ServiceList. It changes the data portion of the ServiceList, with the value of the Service selected as 'true'. This is sent back to the Service Manager. As it is a command for the ServiceList, which is part of the Service Manager, the Service Manager handles it. From the system section, the Service Manager retrieves the name of the Client and checks the data section for the Services. It then retrieves the latest CCML for the Client from its Clients list and creates a command for the Client. It sets the AddService action to the Services

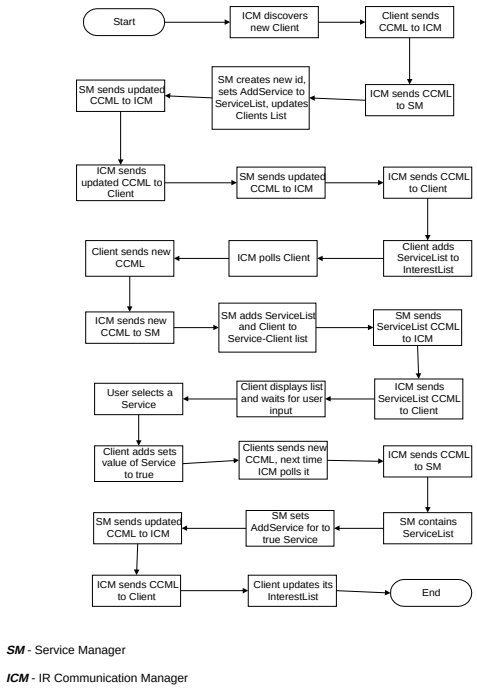

Fig. 7. ServiceList

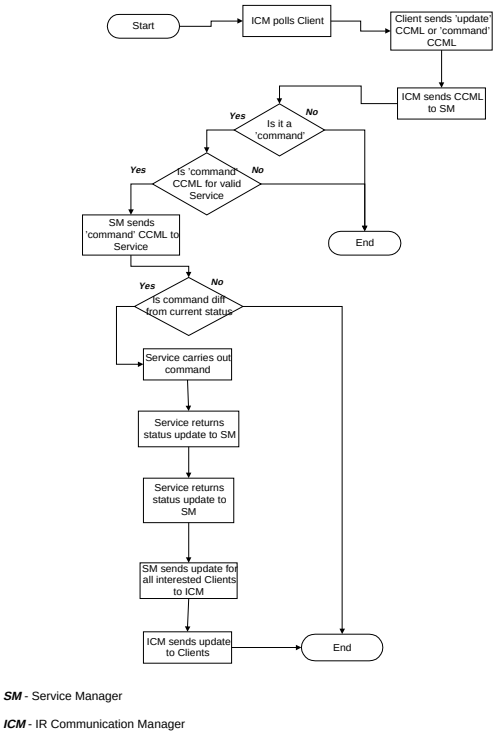

Fig. 8. Requesting a Service

selected and sends the CCML back to the Client. The Client processes this CCML as it would any AddService action by adding the Service to its listening section. It also adds the Service to its InterestList. When the Client is next polled it sends its updated CCML.

The Service Manager reads the list of Services that the Client is listening to, and picks out the new Services, ones that the Client was not previously listening to. It sends their CCML to the Client via the IR Communication Manager. It then, adds the new Service-Client pair to its Service-Client list.

Once the Client gets the CCML of the Service, it displays it for the user. The user can use the interfaces to perform actions. The Client modifies the Service's CCML to make a command, sets the new values and sends when polled.

The Service Manager realizes that it is a command and sends it to the appropriate Service. The Service carries out the command and sends the update to the Service Manager. Fig. 8 details the steps in the requesting of a Service.

B.4 Status Update

If a Service Manager receives an update from a Service, it checks its Service-Client list for all the Clients interested in this Service. It sends the updated CCML to these Clients.

When a Service Manager receives an update from a Client, it carries out certain functions on it. It checks the listening section and retrieves the list of Services that the Client is listening to. It picks out the new Services, ones that the Client was not previously listening to. It sends their CCML to the Client via the IR Communication Manager. Then for each new Service, it adds a new Service-Client pair to its Service-Client list.

Fig. 8 shows how a status update propagates through the Centaurus system.

VI. Protocol

A. *Overview*

- Every couple of minutes IR Communication Manager does a discovery via its infra-red transceiver, to locate the new PDAs in the room.
- Once a new PDA has been located, the Service Manager is contacted. The CCML file for the PDA is sent to the Service Manager through a pre-defined port number. The Service Manager creates a new ID and sets the action, AddService to ServiceList in the CCML file for the PDA and returns this CCML to the IR Communication Manager.
- Whenever the IR Communication Manager receives information on its reading port, it checks the header to find the Client id and then forwards it to the correct Client via infra-red.
- The IR Communication Manager polls all the PDAs in the room at regular intervals. A PDA responds by sending its own CCML as an 'update' or sending the CCML of a service it wants to use by issuing a 'command' .
- When the Service Manager receives any CCML from

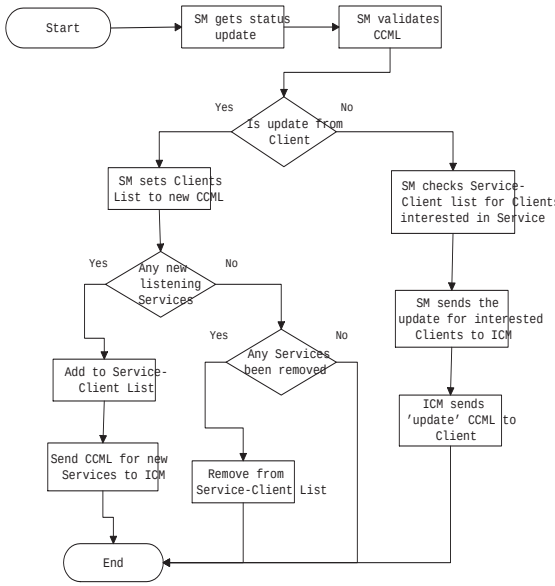

Fig. 9. Status Update

a Client, the CCML is first validated and the Service Manager makes sure that it came from an authorized Client. Then it checks the header to decide whether the CCML is an 'update' or a 'command'.

- If it is an 'update', it reads the list of Services that the Client is listening to, and picks out the new Services, ones that the Client was not previously listening to. It sends their CCML to the Client via the IR Communication Manager. It then, adds the Service-Client pair to its InterestList.
- On the other hand, if it is a 'command', it extracts the name of the Service, and sends it to the appropriate Service.
- If a Client receives an 'update', it is the status update of a Service that the Client is interested in.
- If a Client receives a 'command', if knows that the Service Manager has set an action on it. Following this, the Client validates the CCML, and if there is any AddService action, adds the value to the listening list in its CCML. It also sets the 'update' variable in its CCML header.
  If the action is RemoveService, it removes the Service from its list.
- After starting up, a Service has to register with the Service Manager, by sending its CCML file and its port number to the Service Manager. The 'update' variable is always set in a Service's CCML.
- When a Service receives a 'command' from a Service Manager, it tries to carry out the command. It then updates its CCML to reflect the new changes and sends it to the Service Manager.
- Whenever a Service Manager, receives an 'update' from a Service, it forwards this update to all Clients in the Service-Client list that are interested in this particular Service.

B. Centaurus Communication Protocol

The Centaurus Level1 protocol is running on top of IRDA's IrLAP and IrLMP protocols. [10] IrLAP is a low level protocol that provides device to device connection for reliable data transfer, device discover procedures and hidden node handling. IrLMP works on top of IrLAP and handles multiplexing of the IrLAP layer and multiple channels above an IrLAP connection as well as protocol and service discovery via the Information Access Service (IAS). The Centaurus Level1 protocol provides the glue between IRDA protocols (IRLAP and IRLMP) and Centaurs level 2 protocol.

Centaurus level 2 protocol is based on passing short command messages between the client and the server. These command messages are used to establish the session (`HELO`, `HELORSP`), synchronize clocks (`POLL`), find out which objects are available for transmission(`OBJ`,`POLL`, `NONE`), handle flow control (`PROCEED`, `DONE`, `ACK`) and do actual data transmission (`PK`).

The client and server use exactly the same code for both level 1 and level 2 protocols implementation, the only difference is the actions that are performed by the level 1 protocol to establish connection, and handling of `OBJ` and `NONE` messages by the level 2 protocol stack.

The level 1 protocol differences are the following. On the server side after connection is established and authentication is performed the level 1 protocol sends 'HELO' message and starts up the level 2 protocol. On the client side the level 1 protocol waits for the initial `HELO` message, after it receives this message it replies back with `HELORSP` message and starts up the level 2 protocol as well.

The state diagram for the level 2 protocol is presented in the fig. 10. Although most of the messages in the protocol are just plain text strings few messages deserve further explanation. The `POLL` message, if transmitted from the server, is followed by the current value of the server clock. The `OBJ` message is followed by the object name, its time stamp and its size. The `PK` message is followed by the actual data payload. The `PROCEED` message is followed by the received packets bitmap, when a packet is received by the receiver, it marks the bit corresponding to this packet number in the received packets bitmap, when all of the bits in a bitmap are marked the message is said to be received and receiver sends `ACK` message back to the sender. If for some reason disconnection occurs before the whole message is sent, during the next session the receiver sends the received packets bitmap to the sender and the sender either proceeds to send the remaining packets or discards the whole object if newer version is available.

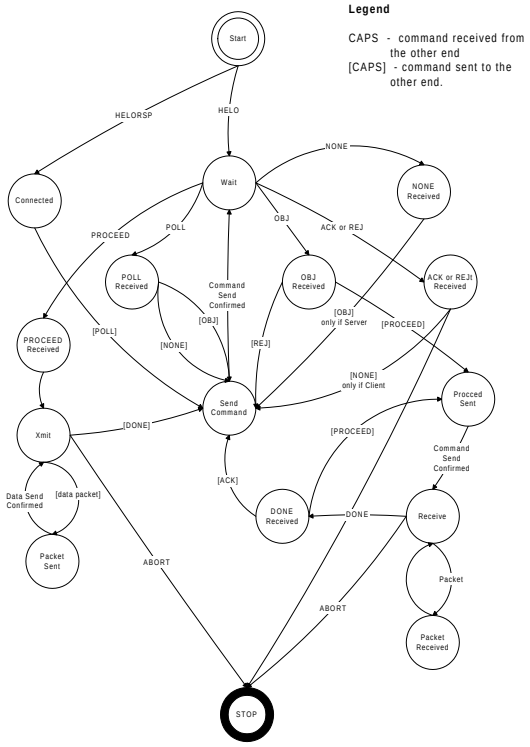

Fig. 10. Centaurus Level 2 Protocol

VII. Results

While testing, we had 3 PDAs in the room communicating with the Centaurus system. Each PDA requested one or two Services, and executed them. We even tried simultaneous execution of Services by the PDAs. These were automatically handled by the IR Communication Manager, which polled the PDAs at regular intervals, so only one request was sent to Service at a time.

We believe that the testing was successful as we were able to execute both the Services by any of the PDAs.

VIII. Conclusion

We have successfully developed the first version of Centaurus. We believe that by providing a uniform infrastructure using XML-encoded data exchange we have shown that it is appropriate and effective for deploying services in an indoor environment. The first stage development, including the Service Manager, IR Communication Manager, MP3 player services, Lamp services etc. has verified that our vision is definitely feasible.

Our future goals are to make Centaurus project available to the public and to provide more documentation for developing Centaurus services. Meanwhile, we are also developing the other components proposed in our design.

Although, our project is far from complete, we believe that now that the framework is in place, adding attractive interfaces for the portable devices, creating new services, and enabling more intelligent brokering of Services will follow easily. We believe that we have crossed all the major hurdles, and completing the remaining portion will be pretty straightforward.

References

[1] The Ninja Project **http://ninja.cs.berkeley.edu/**
[2] An Architecture for a Secure Service Discovery Service **Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz** Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99)
[3] W3C Consortium **http://www.w3.org/XML/**
[4] The Official Bluetooth Website **http://www.bluetooth.com/**
[5] Internet Business Solutions : E-Speak **http://www.e-speak.hp.com?qt=espeak/**
[6] Jini Connection Technology Executive Overview **http://www.sun.com/jini/**
[7] The Jini Community **http://jini.org**
[8] White Paper on Enterprise Service Discovery **http://playground.sun.com/srvloc/slp_white_paper.html**
[9] Service Location Protocol **http://www.svrloc.org/index.html**
[10] Infrared Data Association **http://www.irda.org**
[11] FireCracker **http://www.x10.com/welcome/firecracker/**
[12] X10 Devices **http://www.x10.com/**
[13] Mpg123, MP3 Player for Linux/Unix Systems **http://mpg123.org/**