An Approach to Dynamic Service Management in Pervasive Computing Systems[*]

Lalana Kagal
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
lkagal1@cs.umbc.edu

Sasikanth Avancha
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
savanc1@cs.umbc.edu

Vladimir Korolev
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
vkorol1@cs.umbc.edu

Anupam Joshi
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
joshi@cs.umbc.edu

Tim Finin
Computer Science & Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21244
finin@cs.umbc.edu

Abstract

In the near future, we will see dramatic changes in computing and networking hardware. A large number of devices (e.g., phones, PDAs, even small household appliances) will become computationally enabled. Micro/nano sensors will be widely embedded in most engineered artifacts, from the clothes we wear to the roads we drive on. All of these devices will be (wirelessly) networked using Bluetooth, IEEE 802.15 or IEEE 802.11 for short range connectivity creating pervasive environments. In this age, where a large number of wirelessly networked appliances and devices are becoming commonplace, there is a necessity for providing a standard interface to them that is easily accessible by any user. This paper outlines the design of Centaurus, an infrastructure for presenting services to heterogeneous mobile clients in a physical space via some short range wireless links. The infrastructure is communication medium independent; we have implemented the system over Bluetooth, CDPD and Infrared, three well-known wireless technologies. All the components in our model use a language based on Extensible Markup Language (XML) for communication, giving the system a uniform and easily adaptable interface. Centaurus defines a uniform infrastructure for heterogeneous services, both hardware and software, to be made available to diverse mobile users within a confined space.

[*]This work was supported by NSF Awards IIS 9875433 and CCR 0070802, and the Defense Advanced Research Projects Agency under contract F30602-00-2-0 591 AO K528.

1 Introduction

In the ubiquitous computing paradigm, information and services are accessible virtually anywhere and at any time via any device - phones, PDAs, laptops or even watches [14, 10]. The "SmartHome" and "SmartOffice" scenarios present a step towards realizing this vision. Smart homes and offices consist of intelligent services that are accessible to users via handheld devices connected over short range wireless links. These SmartSpaces will use sensors to gather information about the user and environment, and allow the user to use more interactive forms of input like voice, eye movements etc. The intelligent services themselves will be more receptive to the user's requirements and use logical reasoning to provide better and more relevant support to individual users. The services will be integrated seamlessly into the environment that the user is familiar with, enabling easy and automatic usage. This is the vision that guides our research on the Centaurus system. We define a SmartSpace as a dynamic environment that provides an infrastructure for providing services to mobile users via some short range wireless communication link.

Our system is called Centaurus after the constellation which honors the *Centaur Chiron*, who was known as a wise teacher, healer and prophet. The goal is to design an infrastructure and communication protocol for providing services to heterogeneous mobile clients in the SmartSpaces scenario. This framework is a part of our larger research program aimed at realizing ubiquitous computing systems that are composed of highly intelligent, articulate and social components. These components automatically become aware of each other and can exchange information to cooperatively provide services to the users. In particular, the idea of ad-hoc sets of entities that are dynamically formed to pursue individual and collective goals can be used to create the software infrastructure needed by the next generation of mobile applications. This infrastructure requires rethinking the neatly layered approach that separates networking, data management and user interface considerations, as our system design illustrates.

Centaurus consists of Communication Managers, Service Managers, Services and Clients. Communication Managers handle communication between various entities in the system. Service Managers are responsible for client and service management. Within a confined space, the client can access services provided by the nearest Centaurus System via some short-range wireless technology. Centaurus acts as an active proxy by executing services 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.

All clients and services communicate via Centaurus Communication Markup Language (CCML) (described in section 3.6) which is based on Extensible Markup Language[1] (XML). We found that this W3C Standard is very useful in describing ontologies, and defining properties and interfaces of services. It will also help in integrating Centaurus with emerging semantic languages like DARPA Agent Markup Language (DAML+OIL) [5]. The Communication Manager is flexible and allows any medium to be used for communication, but for implementation purposes, we have used Infrared[1], CDPD[12] and Bluetooth[3].

This paper is organized as follows: Section 2 discusses other work in the area of "smart environments", and compares Centaurus with them. The design of Centaurus is outlined in Section 3 we conclude with a summary in Section 4.

2 Related Work

In the last couple of years, a number of technologies that deal with 'SmartHomes' and 'SmartOffices' have emerged. Among them are the Berkeley Ninja Project [6], the Portolano project[13] from the University of Washington, Stanford's Interactive Workspaces Project [11], and Active Spaces [9, 7] from University of Illinois at Urbana-Champaign.

The Ninja project tries to link different services, through a range of devices ranging from PCs to cell phones and Personal Digital Assistants [6]. It has incorporated intelligence into the infrastructure and has the ability to adapt the content to a specific device.

Centaurus differs from Ninja in its service leasing abilities and state management. 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, it can recover easily because although it does not store any state information, the Services send it regular status to maintain their lease. 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 Centaurus Capability Markup Language(CCML) messages. We also do not distinguish between hardware and software Services, allowing the user to use either in the same way. Since all of the communication between Services and Clients in the Centaurus project are done with the use of XML, there is no need for complicated Operators and Paths used by the Ninja project to convert between different data representations.

[1]http://www.w3.org/XML

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 modules and automatic state recovery in the event of the Service Manager failure.

University of Washington's Portolano project is in the early stages and is mainly involved in 'invisible computing' a term invented by Donald Norman [8] to describe ubiquitous computing, where devices supporting distributed services blend into the user's environment and become practically 'invisible'. The user would invoke these services not just by input but also through augmenting forms of interfacing like user movement, proximity of devices, identification tags, etc. However they are still in the preliminary phases, so we are unsure of how the system will actually be implemented or perform.

Stanford's Interactive Workspaces Project [4] which endeavors to provide a system for interconnecting and integrating heterogeneous COTS legacy devices and software components. In addition, to provide interoperability, their endpoints communicate through a mediating infrastructure that transforms data so it will be compatible from one device type to another. The concept of data translation differs significantly from the Centaurus approach where XML is used as the sole format for data exchange.

An Active space [9, 7] is a physical space including its different physical and virtual components, managed by an operating system, Gaia OS, which acts as a layer of abstraction over the particular properties of an Active Space. The Gaia OS manages the resources of an active space. Gaia does not define high-level policies regarding the behavior of the entities in the space. It concentrates on providing an infrastructure for the physical space and projecting a unified interface. This model, by insisting that the services be implemented as CORBA services, restricts the application developers. In Centaurus services in any language will be seamlessly integrated into the system, as long as they use CCML to communicate. The Active Spaces project does not seem to be easily extended to support mobile users or different modes of communication i.e., Bluetooth, IR, CDPD and Ethernet, whereas Centaurus has been specifically designed to allow flexibility in communication protocols.

3 System Design

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 where ever they are needed.

Centaurus consists of several components: The CentaurusComm Transport Protocol, Communication Managers, Service Managers, Services, and Clients. CentaurusComm Transport Protocol is an efficient, message-object based transport protocol which abstracts out the medium specific information. Communication Managers handle all the communi-

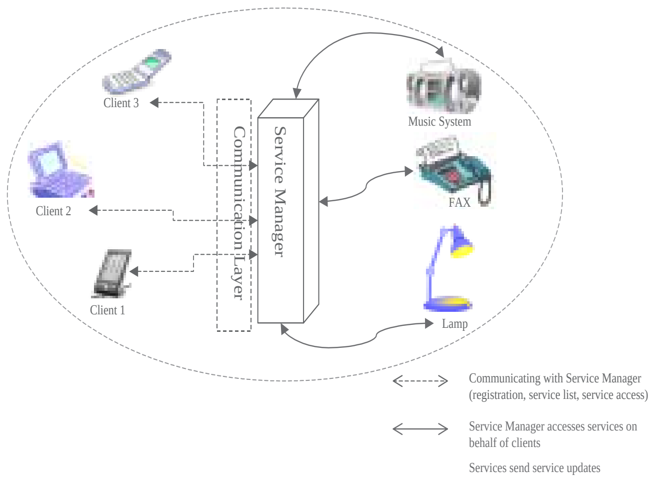
Client 3
Client 2
Client 1
Communication Layer
Service Manager
Music System
FAX
Lamp
Communicating with Service Manager (registration, service list, service access)
Service Manager accesses services on behalf of clients
Services send service updates

Figure 1: The different components in a Centaurus system

cation with the Centaurus client using different modules of CentaurusComm Protocol. The Communication Manager is capable of communicating over varied media such as Ethernet, Infrared, CDPD and Bluetooth. Service Managers control access to the services and act as gateways between the services and clients. Services are objects that offer certain functionality to Centaurus clients. Services contain information to enable them to locate the closest Service Manager and register themselves with it. Once registered, the services can be requested by any client communicating with the Communication Manager. The client provides an interface to the user for interaction with the services provided in the SmartSpace. Figure 1 shows the different components and the relationships between them.

3.1 The Transport Protocol in Centaurus : CentaurusComm

CentaurusComm consists of one or more lower level protocol modules (designated as Level I), one higher level module (designated as Level II) and an application program interface. Level I modules are communication medium dependent; the Level II module is medium independent. The API is responsible for accepting the objects from the application layer for transmission and notifying it when messages are received.

The protocol is implemented as a collection of data structures and state machines. The principal data structures include the *transmit* queue and the *receive* queue. As the protocol is designed to run on a wide range of low power systems such as PDAs and low power embedded computers, it does not depend on any advanced operating system features such as signals and multithreading, that are typically not part of such systems. This is in contrast to TCP, which requires substantial support from the OS for signaling. CentaurusComm is designed such that the the transmission, reception and recovery procedures are divided into many small sub-procedures that last for a very short time. With the exception of domain name resolution, which occurs very infrequently, the protocol never blocks. This design gives an impression of concurrent execution of the user program and the protocol modules.

3.2 Communication Manager

This component is responsible for the communication between the client and other components of Centaurus. The Communication Manager uses a specific TCP port to communicate with the Service Manager. This is so that Communication Managers and Service Managers need not be on the same system. When the Communication Manager receives information from a client, it sends this information to the Service Manager. 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. Currently we have communication modules for Bluetooth, IR and CDPD allowing Centaurus to communicate with clients using those communication media.

3.3 Service Manager

The Service Manager(SM) acts as a mediator between the services and the client. When a service starts up, it has to register with the Service Manager, sending its name, identification, description, interfaces it implements and leasing period in CCML. The SM is also responsible for service leasing. It allows services to register for a certain amount of time. If it does not receive any status update or a 'ping' from the service within that time, the registration is deleted. When a new client comes along, it has to register itself with the Service Manager as well. The Service Manager sends it a *ServiceList* object. This *ServiceList* object is updated dynamically, according to the services registered with the Service Manager, so the client always has the updated list of services. The user can select a service from this list which the client sends back to the Service Manager. The Service Manager replies with the CCML description for that service and updates its database to reflect that the client is *interested* in the service that it just requested. The client displays the CCML of the service and waits for the user's input. The user can invoke any interface of the service causing the client to generate the appropriate CCML command and send it back to the Service Manager. 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 and a timeout is set. Once this timeout expires, an error is returned to the client. 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.

Though the Service Manager does manage some state, most of the state is distributed among the clients and services. In case a Service Manager fails, the services keep pinging it at increasing intervals until it comes up again. Then they re-register themselves with it. Similarly the Service Manager stores a list of services that a client is interested in, so it can be sent status updates from those services. However, every time the client notifies the Service Manager of a service it is interested in, the Service Manager forces the client to update its own CCML description to reflect the new service. As soon as the failed Service Manager is up again or if a replacement Service Manager is used, the clients register themselves. Their CCML description informs the Service Manager what services they are interested in. By forcing the services to be responsible for maintaining their own state, by de-registering any timed out services, and making clients store the list of services they are interested in, the Service Manager is capable of automatic state recovery leading to greater fault tolerance.

3.4 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 printing a document or even a memo pad service where clients can leave messages for each other. Each service registers with a Service Manager by sending the CCML description of its name, identification, location, a brief description of its functionality and its leasing period. Every time its status changes, it informs the Service Manager. If its status has not changed during the leasing period and it wants to renew its lease, it has to send a short renew message to the Service Manager. It accepts requests only from the Service Manager that it is registered with.

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 registering 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.

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 though it is handled in same way as other Services are, it is contained entirely within the Service Manager itself.

- Lamp-Control

  Using X10[2] devices and FireCracker[3], the Lamp Control Service is 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. If the 'Powered' interface has a value that is different from the status of the Power variable, then the command is processed 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, which forwards it to all the Clients that are interested in the lamp service.

- MP3-Player

  We are using a popular MP3 player for Unix, mpg123 [2], 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 file. It has a number of CCML interfaces, one for each song it can play. The Service checks the CCML command received and reads the songs selected. These songs are checked against the current list of songs. If they are valid, they are fed into mpg123. A new CCML status file is generated and sent to the Service Manager.

[2]http://www.x10.com/

[3]http://www.x10.com/welcome/firecracker/

3.5 Client

A client is a special kind of service in that it has to respond to commands and regularly send status updates. A client communicates with the Communication Manager and registers itself with a Service Manager. This registration is similar to the registration of services. On registration, it receives the *ServiceList*, which contains the current list of services. The *ServiceList* is a service itself, and causes the Service Manager to send the new list of services, every time its status changes, that is, each time a new service registers, or an existing service de-registers.

By choosing a Service, the client *expresses interest* in it. The Service Manager sends the client the CCML description of the Service. The client displays the CCML file for the user, who can invoke the specified functions on the Service, by choosing one of its interfaces. After the user changes values of certain variables, specified in the CCML for the particular service, the client sends the CCML back to the Service Manager in the form of a *command.* The client will receive status updates from all services that it expresses interest in through the Service Manager, until it specifically informs the Service Manager that it no longer wants to receive these messages.

3.6 Centaurus Capability Markup Language (CCML)

The CCML is divided into *system* , *data*, *addons*, *interfaces*, and *info.*

The *system* portion contains the header information, the *id*, timestamp, origin, etc. There are two variables, *update* and *command.* A CCML file can only have one or the other. An *update* variable is used to inform other Centaurus components about status updates of services and clients, whereas the *command* is used 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. All information regarding the variables and their types are contained in the *data* section. Using the *addons* section, one 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.

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. We have defined two actions for clients namely AddService and RemoveService.

- AddService : When this action is set, the client adds the value of this variable to 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 the list of services that it is interested in.

The *interface* section contains information about the interfaces that the object (Service/Client) implements. This section generally causes the variables in the *data* section to change their values.

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

4 Summary

We have successfully developed the first version of Centaurus. We believe that our infrastructure is appropriate and effective for deploying services in an indoor environment. Such environments are typified by handheld clients connecting to services on a fixed infrastructure using wireless ad-hoc networks such as those based on Bluetooth. The first stage of development, including the Service Manager, Communication Manager, MP3 player services, Lamp services etc. have verified that our vision of dynamic service discovery and management over ad-hoc wireless networks is feasible.

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. Although, we have a long way to go, we believe that Centaurus has brought us a couple of steps closer to realizing our vision of intelligent, adaptable and highly perceptive pervasive environments.

References

[1] Infrared Data Association (website). http://www.irda.org.

[2] Mpg123, mp3 player for linux/unix systems. http://mpg123.org/.

[3] The Official Bluetooth Website. http://www.bluetooth.com.

[4] George Candea and Armando Fox. Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment. In *Second International Symposium on Handheld and Ubiquitous Computing*, pages 248–254, 2000.

[5] I. Horrocks et al. DAML+OIL Language Specifications. http://www.daml.org/ 2000/12/daml+oil-index, 2001.

[6] Steven D. Gribble et al. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks*, pages 473–497, March 2001.

[7] Fabio Kon, Christopher K. Hess, Manuel Roman, Roy H. Campbell, and M. Dennis Mickuna. A Flexible, Interoperable Framework for Active Spaces. In *OOPSLA'2000 Workshop on Pervasive Computing*, October 2000.

[8] Donald Norman. The Invisible Computer. *MIT Press*, 1998.

[9] Manuel Roman and Roy H. Campbell. Gaia: Enabling Active Spaces. In *9th ACM SIGOPS European Workshop*, September 2000.

[10] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Communications*, 2001.

[11] Stanford Interactive Workspaces Project. http://graphics.stanford.edu/projects/iwork/.

[12] M. Taylor, W. Waung, and M. Banan. Internetwork Mobility : The CDPD Approach. Prentice Hall Professional Technical Reference, 1999.

[13] Dept of Computer Science University of Washington and Engineering. Portolano: An Expedition into Invisible Computing. http://portolano.cs.washington.edu/.

[14] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, September 1991.