Technical Report TR-CS-00-05

A Component Based Architecture For Mobile Information Access

Chaitanya Pullela, Liang Xu, Dipanjan Chakraborty, Anupam Joshi

Department of Computing Science and Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250

31 March 2000

A component based architecture for mobile information access

Chaitanya Pullela, Liang Xu, Dipanjan Chakraborty, Anupam Joshi
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD 21250
{cpulle1, lxu2, dchakr1, ajoshi}@cs.umbc.edu

Abstract

Implicit in today's mobile information access scenarios is the assumption that the information required by the mobile host is readily available on the network and it's location is "known" in a static fashion. There is a significant ongoing research on ways to overcome the problems (such as low-bandwidth of mobile networks) existing in delivering available information to mobile systems. We investigate the situations where the information required is not readily available on the network and it needs to be obtained by dynamically locating the required data and then possibly initiating a series of computations to obtain the information. This paper presents a layered architecture for addressing this problem generally and also our initial implementation of this architecture.

1 Introduction

To realize the dream of truly ubiquitious access to the information superhighway, it is necessary that the information be made available to mobile platforms. Typically, these mobile computers are resource poor (limited compute power, disks , battery life etc.) and are connected over low-bandwidth wireless networks. Prior research work has focussed on overcoming these problems in delivering information to mobile platforms. Therefore, the effort has been on transcoding[8, 9], managing disconnection[10] and network related issues[20, 21]. These works are based on the assumption that the information to be delivered is readily available on the network.

However, it is possible that the information required by a mobile user may not be available per se but needs to be computed dynamically using various resources (software, data or hardware services) distributed over the network. Recent research advances in area of software components and the availability of dynamic service location platforms have provided the necessary groundwork to acheive this purpose. The componentization of software allows for independently manufactured software units to interact easily with each other. Also, thanks to inexpensive disk storage, there is an increasing amount of raw data available on network. For example, on-line trading services provide a wealth of stock information for customers to research. We see a paradigm where independent software and data resources could "plug and play", with computations performed on the fly, to obtain required information. Realizing this paradigm in a mobile information access environment poses a problem - The mobile clients may not have adequate computational power on board to perform these computations and will need to borrow compute power from networked resources. There are two issues to be considered here- 1)Efficiency demands that some of the computation be performed on the mobile host. This is especially true for components that need a lot of user interaction. Executing these on the network will generate a lot of traffic.2)The wide variety of mobile platforms available in the market, argue against statically defining the location where a particular component should be executed, because some clients may not be able to handle even the slightest computation demands.

The Web environment does not offer a solution to this problem. Traditionally, Web has been an information delivery vehicle. Distribution of computation has been limited to sending executable content in form of java byte codes or other scripting codes. It does not consider cases where the client may not have enough power to run the executable content. Recently, some solutions which essentially provide a light weight script suitable for mobile clients have been proposed. However, this is focussed on phone type devices and ignores the cases where mobile clients posses more computational power, thus resources being under utilized. Considering above issues, we clearly see a need for a model where the distribution of computation is based on availability of resources at the mobile client.

To find the appropriate components needed to obtain information, it is necessary to have robust dynamic service location mechanisms. Recent commercial systems such as Jini[12] and e-speak[14] are works in that direction. But these come with limited syntax based matching techniques which limit the expressive power of services.

This paper presents the middleware that we created to address the above mentioned problems at various levels. This paper has 5 sections. In section 2, we present the background for our research. In particular, we describe the Jini architecture and the Ronin agent framework. We then, in section 3, describe the limitations of Jini framework and present our solution (XReggie). In section 4, we present the general architecture of the system and our implementation of it. Section 5 finally summarizes our work.

2 Background

2.1 Service location mechanisms

Recent commercial systems such as Jini[12], e-Speak[14], salutation[15] and UPnP[13] provide the mechanisms through which service location is possible. These systems are different from the more traditional CORBA type approaches in that the traditional approaches tend to be static and are more concentrated towards achieving platform, language independence than supporting features such as dynamic service discovery. We build our system based on the Jini architecture for it provides a robust infrastructure for distributed components to locate and interact with each other.

Jini technology is a simple infrastructure for providing services in a network, and for creating spontaneous interactions between programs that use these services. Services can join or leave the network in a robust fashion, and clients can rely upon the availability of visible services, or at least upon clear failure conditions. The Jini discovery infrastructure supports both unicast and multicast service discovery protocol. This infrastructure allows services to be found in a uniform way, either on a local or a remote network.

There are many advantages to employ Jini for service discovery in a dynamic mobile environment. First, in a dynamic mobile environment, components are often heterogenous. Embedded hardware devices need to cooperate with mobile software applications. For example, a user wants to print out an email from his/her palmtop using the printer that is available in the airport. In such scenario two different types of hardware devices need to communicate with each other. The Jini service-interface advertisement mechanism provides a uniform and high-level abstraction to both hardware and software components.

In a mobile environment, resources (computation power, memory, storage space, communication channel etc.) are very critical. For example, it is very common for mobile devices to become disconnected from the network, either intentionally or unintentionally. Upon the disconnection of a mobile device, the resource that was held by the device should be freed. The Jini leasing model provides a robust leasing infrastructure that is beneficial in the mobile environment, where partial failure can cause holders of resources to fail or become disconnected from the resources before they explicitly free them.

2.2 Ronin Agent Framework

The Ronin Agent Framework[11] is a Jini-based agent development framework that is designed to aid in the development of next generation smart distributed mobile systems. The Ronin Agent Framework introduces a hybrid architecture, a composition of agent-oriented and service-oriented architectures, for deploying dynamic distributed systems. Ronin contains a number of features that distinguish it from comparable frameworks, including an Agent Communication Language (ACL) and network protocol independent communication infrastructure. It also includes an agent description facility that allows agents to discover and lookup each other, an agent proxy architecture that provides agent mobility behavior and customizable agent communication scheme.

Ronin defines an open framework that specifies the infrastructure requirement and the interface guidelines for the interaction and communication between agent-oriented Jini components. Ronin framework models Jini services as agents. Central to Ronin architecture is the notion of Ronin agent and it's corresponding agent deputy. An agent deputy acts as a front-end interface for the other agents in the system to communicate with the Ronin agent it represents. Ronin does not define how an agent deputy should communicate with a Ronin agent.

3 XReggie

3.1 Enhanced Jini Lookup Service

The Jini discovery infrastructure provides a good base foundation for developing a system with components distributed in the network that need to discover each other. Unfortunately, Jini is still not adequate because it essentially operates at a syntactic level. In other words, a client needs to specify the exact function/interface needed in order to discover a service. We aim to build a dynamic service discovery mechanism where the lookup process can operate at a semantic level. Namely, the client should be able to provide a service description at a higher level of abstraction than the interface description. We also want to be able to specify features(such as the cost of a service) that Jini does not envision handling. We want to be able to specify constraints on these features(e.g. cost $<$ max) – recall that Jini can only handle equality relationships. Finally, we want to be able to allow "similarity" matches and return services that are similar to, but not the same as, the requested service. For example, the system returns a service providing a 3 month stock chart when the client asks for a service providing 6 month stock chart. This is useful when exact matches cannot be found for the service requested by the user.

Our solution to this problem is XReggie system, which specifies how Jini (and similar systems) can be taken beyond their simple syntax based service matching approaches. The idea is to modify the Jini lookup service to match services using XML[17] attributes instead of interfaces and java attribute objects. Each component(service) provides an 'XML' entry describing its functionality and requirements when it registers into the Jini Lookup Service. When a client wants to use a service, it creates a XML DOM object describing the service it needs along with constraints. It then finds the appropriate service using the XML match function. The XML match handles constraints such as requirements, cost, mobility etc. For instance, the client can ask that the service cost less than some amount, or that it be mobile. In order to use XML in the match, the entities in the system need an agreement on a 'ontology'[18]. In our system, we assume that both services and clients have previously agreed on an ontology and knows what each entry in the XML description means.

3.2 XReggie modifications

In order to make the XML based match work, we have made modifications to Reggie, the Sun's implementation of Jini Lookup Service. Reggie provides all the functions and methods defined in the Jini Core package, besides some extra functionality provided by Sun. We modify it as follows:

- A Xmlentry class is added in the net.jini.lookup.entry package, which takes a XML DOM object as argument in its constructor.
- An XmlMatch interface is added in the com.sun.jini.reggie package, which defines the various functions needed in the XML match function
- An XmlMatchImpl class is pluged in the com.sun.jini.reggie package, which implements the functions defined in the XmlMatch interface. It uses the Java Xml parser to convert the XML description to a Java DOM object and does the comparison with the two XML DOM objects from client and server side.
- The RegistrarImpl class in the com.sun.jini.reggie package is modified, which lets the enhanced Jini Lookup Service(XReggie) know when and how to do the XML attribute match instead of doing the ordinary Name/Type syntax-level match.
- The CreateLookup class in the com.sun.jini.reggie package is changed, which lets the enhanced Lookup Service(XReggie) know where to find the imformation of XML parser package
- The command line to start Jini Lookup Service has also been adjusted to contain some extra options.

4 System Architecture and Implementation

4.1 General Architecture

We now present the general architecture of our system . The architecture comprises of 3 layers which form the middleware through which mobile clients and individual services interact. (Figure 1).

- Dynamic service location

  This layer represents the mechanisms through which services are discovered and the mechanisms through which services express themselves. Thus this layer enables the core function of dynamic service location. It is important that the platform support dynamic interaction between services and the clients of the services, with no or little prior knowledge of each other.

- Brokering

  This layer has the knowledge base of what services should interact to produce the required information to the end-user. It also handles the interactions between the services and manages the distribution of computation depending on the resources available to mobile user and constraints (such as cost) if any. This layer utilizes the services of the dynamic service location layer. It is comprised of some helper services which assist in performing its duties. For example, the broker would have at its disposal an execution platform service which provides compute power on the wired side.

- Delivery

  This layer manages the delivery of information to the user. It could take into account any limitations of the user and try to deliver the result. For example, for mobile clients connected to low bandwidth network, the delivery layer could perform some filtering and compression to reduce the data needed to be transmitted or even change the data to different format (say, as a WAP proxy).

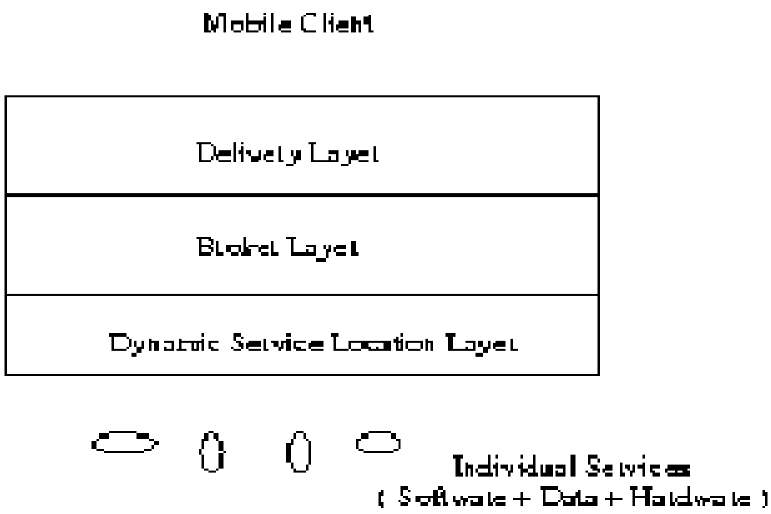

Figure 1: General Architecture

4.2 System overview

The system is targeted to run on mobile devices that are capable of executing Jini enabled applications. Currently, high end mobile devices such as laptops can support fully featured Java Virtual Machine(JVM) and are capable of executing Jini applications. A light-weight JVM for handheld devices such as palmtop known as KVM[19] is currently being developed. Thus our assumption of a Java/Jini system does not cause a loss of generality. Our system involves the following components:

1. End User - The user of hand-held device who requires some information.
2. A Palmtop - End-user device.
3. Broker - The service which helps the user in finding required information.
4. Agent Deputies -
   - Agent for mobile client which sits on wired side. This deputy handles the information to be delivered to mobile client.
   - Agent for broker which sits on the palm device. This deputy handles the information to be delivered to broker. Note that since the mobile client actively participates in the computation, there is some information travelling from mobile client to broker.
5. Execution platform - The broker has at its disposal, this helper service which is ready to provide compute power on the wired side i.e run components on behalf of mobile clients that do not have adequate compute power.

6. Individual services - component services available on the network.

4.3 System Behaviour

We explain the overall behaviour of our system with the help of some simple examples.

- A busy investor learns from the latest news-feed on his pager that the value of Japanese yen is down compared to American dollar. Since he has investments in Japan, he wants to find out how this would affect the net-value (in American Dollars)of his investment, adjusted for exchange rate change. He instructs his palmtop to find any financial information services offered over the network. His palmtop discovers a JINI lookup service on it's local network by multicast service discovery protocols. Using normal JINI lookup mechanisms, it then looks-up for a financial services broker which is ready to find the required information for the user. The user-interface for the broker is automatically downloaded by the JINI lookup process.

  Upon making sure through the user-interface that the service he wants is offered by the broker, the user selects the option to find the information he wants. The user-interface prompts the user to enter required information (Ticker, number of shares, value at purchase and the duration of purchase). The constraints of user's palmtop, which are stored as XML attributes are also sent to the broker.

  The broker has a mapping for each service it offers to individual services needed to be used to compute the information. In this case, the broker needs the following services

  - Gives the current exchange rate for a particular foreign currency to another, for any particular day.
  - Gives the current stock price given the ticker .
  - Software component that computes net loss/gain after adjustment in exchange rate,given required data.

  The broker attempts to find required services using dynamic service location mechanisms(XReggie). Once the services are found the broker decides on an execution sequence to obtain the required information. Broker decides on the execution sequence taking into account the XML attributes specified by the user and the services. For our example, the broker first obtains the data and since the computation to be performed is not very processor intensive, the component and the data are sent to the palmtop for execution. Upon completion of execution, required information is displyed on palmtop.

- Noticing the fact that the stock of a company he invested in, is widely fluctuating, an investor wishes to find the average value of a share over one year. When he requests the information to the broker as explained in the previous scenario, the broker sees that it primarily needs a service which provides the data for last 1 year. Since the amount of data required is significant, the broker figures out that it is unwise to transfer it over the network. Also the computation is fairly processor intensive in this case. So the broker tries to locate a stock data provider which is ready to provide some compute power locally, so that the software components could be sent to data site. Since data services provide information about local processing using XML attributes, the 'XReggie' system would perform this screening and return any such services to the broker. The broker would send the software components to compute the average to the data site and simply return the result to the user. Note that if a data service which is ready to provide local processing is not found, the broker would send the data and software components to the execution platform service, obtain the results from execution platform and return the result to the user. In any case, the broker would avoid transferring raw data and computation to mobile host.

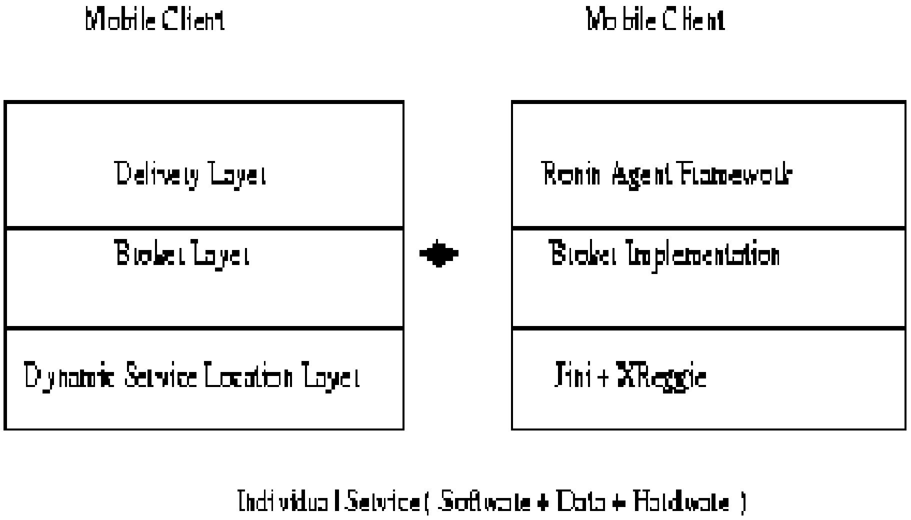

Figure 2: Implementation

- The system could also handle cases where the required information is readily available. For example, when a mobile user requests for a stock quote, the broker finds a single service that simply returns the quote given a ticker.

4.4 Dynamic service location layer

We implement this layer using 'XReggie'. (Figure 2).Individual services are registered onto the network with capabilities and constraints expressed as XML attributes. The 'capabilities' attributes describe the actual function that service provides. The 'constraints' attributes represent any additional properties of the service.

Examples of the 'constraints' attributes we have used are:

For Software services:

- mobility : Describes if service is implemented as remote service (non-mobile) or local(mobile code) service.
- RAMrequired: processing power that service requires.
- size : size of software.

For Data services :

- Size : size of raw data.
- LocalProcessing : Describes if there is some compute power available at data site.(So that a component may be sent to data site instead of transferring lot of raw data over network)

4.5 Broker

The broker is the core engine which finds the individual services required in computing required information and manages the distribution of computation. In the first phase, the broker finds all the required individual services required for solving the particular problem. The broker uses the XML match functionality of the Jini lookup service to screen some services at the lower layer. For example, if broker decides that the mobile client is too weak to perform any computation, it looks up for services that are implemented as remote. Recall that 'mobility' is one of the attributes which is specified by software services. In the second phase, the broker decides where each component should be executed depending on the resource requirements and constraints of the services and resource availability, constraints of the mobile client.

The broker that we implemented solves a set of problems within financial services domain (vertical solution). In our implementation, the broker has a mapping of each service that it provides with the individual services required in solving the particular problem. We have implemented this mapping statically as a proof of concept. It is not difficult to envision brokers utilizing complex AI features to find services required in a dynamic fashion. Further, we envision networks being populated with brokers, each being capable of solving problems within a particular domain.

The broker may decide to execute some components on the mobile host itself. To execute a particular component on mobile host, the broker sends the following information to the mobile host. 1) Unique Jini serviceID of the component. 2) The method name to be invoked on service object 3) Arguments to be used for the method. The mobile host then fetches the service object from the lookup service, performs the necessary computation and returns the results to the broker. Currently, we are improving on this implementation by making the broker to fetch the service object itself and send it to the mobile host thus relieving the burden of fetching from the mobile host. There are two points worth mentioning here - 1)The mobile host need not have any knowledge (like the service interface) about the service it is using. It simply uses the Java reflection mechanism to invoke a method on the service object. The broker acts as a middleware between mobile host and the Jini framework. 2) The broker would assign the mobile host to perform this computation only after it decides (from XML attributes specified by mobile host) that the mobile host has the resources to do so. Any information (data or software component) that is to be delivered to the mobile client is done so via the delivery layer.

4.6 Delivery

We have used the Ronin agent framework to implement this layer. In the Ronin framework, each agent consists of two parts: Agent Deputy and Ronin Agent. The functionality of an Agent Deputy is similar to the functionality of a Proxy object in the Jini programming paradigm. An Agent Deputy acts as a front-end interface for the other agents in the system to communicate with the Ronin Agent it represents. For example, when a mobile host represents a Ronin agent, it's agent deputy sits on the wired network. Ronin does not define how an Agent Deputy should communicate with a Ronin Agent. However, it does define the interface for the communication between the Ronin Agent and Agent Deputy.

Ronin is particularly suitable for our purposes because it is designed in such a way that the mode of transportation could be easily changed at run time. Thus, When a mobile host moves between various type of networks, the transport module appropriate for that particular network could be chosen dynamically. Also, the ability to use different transport modules would definitely help in mobile environments where wide variety of mobile clients are available. If the broker notices from XML attributes of the mobile client that the client lacks a particular resource, the deputy would provide appropriate transcoding or filtering solution.

We have implemented a concrete agent deputy to handle disconnection's of mobile clients. This deputy communicates with Ronin agent using network socket model and uses store-forward message delivery scheme. Just before a mobile client is about to be disconnected, it sends a 'store' message to it's deputy on wired side, which instructs the deputy to Queue any messages. Later, on re-connection to network, the mobile client sends a 'forward' message to it's deputy there by receiving all the collected messages. We are currently implementing a deputy which automatically recognizes disconnections and stores the messages.

5 Conclusions

To realize the paradigm of pervasive computing, in addition to ubiquitious delivery of already available information, it is important to consider cases where the information required by mobile unit is not readily available, but needs to be computed dynamically. In this paper, we presented the initial steps we have taken in addressing this problem. A general architecture which may be employed in these scenarios is presented. Our implementation of this architecture, using extensions to Jini and Ronin agent frameworks is presented. It is worth pointing that the implementation of broker could be improved by incorporating complex AI techniques so that more general problems are addressed. Also, knowledge recommender agent systems such as PYTHIA[16] could be used to intelligently manage the distribution of computation between mobile platform and the stationary machines. These systems use previously seen computations as a basis for predicting the execution profile of a computation about to be scheduled.

References

[1] T.Drashansky,S.Weerawarna,A.Joshi,R.Weerasinghe, and E.Houstis. Software architecture of ubiquitious scientific computing environments. *ACM-Baltze Journal on Mobile Networks and Applications,1,1997.*

[2] T.Drashansky, A.Joshi, S.Weerawarna, and E.Houstis. Sciencepad:Scientific computing in a ubiquitious environment.*Intl.J.Microcomputer Applications*,15(3),1996.

[3] B.R.Badrinath, A.Acharya, and T.Imielinski. Impact of mobility on distributed computations. *Operating Systems Review,27:15-20,1993*

[4] T.Imielinski and B.R.Badrinath. Mobile Wireless Computing:Challenges in Data Management. *Comm. ACM*,37(10):18-28,1994.

[5] A.Joshi, N.Ramakrishnan, and E.Houstis. Multiagent systems to support networked scientific computing.Technical Report CSD-TR-97-021, CS Department, Purdue University, 1997.

[6] A.D.Joseph,A.F.deLespinasse, J.A.Tauber, D.K.Gifford, and M.F.Kassshoek. Rover:A Toolkit for Mobile Information Access. In *Proc. 15th Symposium on Operating Systems Principles.*ACM,December 1995.

[7] A.K.Elmagarmid, J.Chen, W.Du, O.Bukhres, and R.Pezzoli. Interbase: an Execution Environment for Global Applications over Distributed, Autonomous and Heterogenous Software Systems. Technical Report TR-92-016,Dept. Comp. Sci., Purdue University,1992.

[8] A.Fox and E.A.Brewer. Reducing www latency and bandwidth requirements by real-time distillations. In *Proc. Fifth International World Wide Web Conference*, May 1996.

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

[10] A.Joshi,S.Weerawarna, and E.N.Houstis. Disconnected browsing of distributed information. In *Proc. Seventh IEEE Intl. Workshop on Research Issues in Data Engineering*, pages 101-108.IEEE,April 1997.

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

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

[13] http://www.microsoft.com/HWDEV/UPnP/default.htm

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

[15] Rekesh John. UPnP,Jini and Salutation - A look at some popular coordination frameworks for future network devices.Technical Report, California Software Labs,1999. Available online from http://www.cawl.com/whitepaper/tech/upnp.html

[16] S.Weerawarna, E.N.Houstis, J.R.Rice, A.Joshi, and Houstis C.E. PYTHIA: A knowledge based system to select scientific algorithms. *ACM Trans. Math. Software*, 22:447-468,1997.

[17] eXtensible Markup Language(XML) - www.w3.org/xml

[18] ontology.org: Frequently asked questions. Available online from http://www.ontology.org/main/papers/faq.html

[19] KVM home page: http://java.sun.com/products/kvm/

[20] R.O.LaMarite, A.Krishna, and H.Ahmadi. Analysis of a Wireless MAC Protocol with client-server Traffic and Capture. *IEEE Journal on Selected Areas in Communication*12(8):1299-1313,Oct 1994.

[21] C.Perkins. IP Mobility Support - RFC 2002 ,october 1996.