Technical Report TR-CS-00-02

Dynamic Service Discovery for Mobile Computing: Intelligent Agents Meet Jini in the Aether

Harry Chen, Anupam Joshi, Timothy Finin

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

20 Feburary 2000

Dynamic Service Discovery for Mobile Computing: Intelligent Agents Meet Jini in the Aether *

Abstract

The emergence of ad-hoc pervasive connectivity for devices based on Bluetooth-like systems provides a new way to create applications for mobile systems. We propose to realize ubiquitous computing systems based on the cooperation of autonomous, dynamic and adaptive components (hardware as well as software) which are located in vicinity of one another. In this paper we present this vision. We also describe a prototype system we have developed that implements parts of this vision – in particular dynamic service discovery in mobile environments. We point out why existing systems such as Jini are not suited for this task, and how our system improves on them.

1 Introduction

This paper describes initial steps in our research program aimed at realizing ubiquitous computing systems based on the cooperation of autonomous ("active"), self-describing ("articulate"), highly interactive ("social"), and adaptive ("intelligent") components which are located in "vicinity" of one another. Such hardware and software components will automatically become aware of each other, establish basic (wireless) communication, exchange information about their basic capabilities and requirements, discover and exchange APIs, and learn to cooperate effectively to accomplish their individual and collective goals. This represents a paradigm shift in the way we build complex information infrastructure/applications in general, and the way we approach the design of mobile/pervasive systems in particular.

In particular, the idea of "ad-hoc" teams 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. These will use the emerging 3^{rd} and 4^{th} generation broadband wireless systems, as well as short range narrowband systems such as Bluetooth. Heretofore, the software and applications component of mobile computing has lagged behind its hardware (communication, computing and networking) aspects. Much of the research in the applications area is often limited to allowing applications built for the wired world (web, databases etc) to run in the wireless domain using proxy based approaches. Our research seeks to move beyond this and provide a framework which uses the power of mobility and ad-hoc wireless connectivity to enable novel applications. The results will point us toward a physical environment in which devices all around us and even on our body are in constant communication and organize themselves to cooperate to do our bidding.

Sojourns into the future

We envision our protagonist (Jane) driving down I95. A palmtop and cellular phone are in her purse, and the car has its own onboard computer. All these devices are networked using Bluetooth. The car's onboard computer connects with computers in nearby cars using Bluetooth, and exchanges information necessary to coordinate their driving in a Distributed Traffic Management Environment. Sometimes when the driver is

*We wish to thank Vladimir Korolev, our Palm development expert, who made the Palm user interface possible for the Agents2Go project.

not sure of things (where the nearest gas station with a car wash is, for example), the computer can ask the other cars around it. Every 5 miles or so down the highway, while the car is within range of an *electronic-sign* or a rest area, it is connected to broadband high speed wireless LAN. While driving, Jane receives a page on the palmtop alerting her that her office is sending her a fax. Her palmtop dials out on the phone to retrieve the fax and requests that it be converted to an audio stream that can be played on her car's sound system. The server cannot honor this request, so the palm activates a proxy agent when the car passes the next e-sign. This agent locates and negotiates with fax-to-audio conversion services in the vicinity to effect the conversion, and migrates to the next e-sign the car will pass. When the car is in range, the proxy agent streams the audio to the car. After listening, Jane figures that this is something she will need to read carefully. It is getting close to 7 in the evening, so Jane decides to call it a day. She tells her palmtop to book her a room at the next Chariott hotel and have the fax print there. She wants to have dinner at a Mexican restaurant before going to the hotel. At the next e-sign, the Palm device contacts the proxy agents for the local hotel/motel and restaurant associations. It negotiates to get Jane the best possible deal on the room she wants (non smoking, King). It also finds out that the hotel and the Mexican restaurant are at some distance, but that there is a Tex-Mex restaurant close to the hotel which serves Jane's favorite dish. The Tex-Mex restaurant also agrees to print her fax at no extra charge, and the agent reserves her a table, downloads directions into her car's computer and routes the fax to the restaurant's printer.

This paper has 6 sections. In Section 2 we first provide some background and rationale on our research work. In particular, we discuss the reasons that made us believe that service discovery plays a very important role in the mobile computing environment. Also we discuss how the Jini discovery mechanism can help services to find each other, why the existing Jini discovery mechanism is not sufficient for handling service discovery in a mobile environment, and how can we make improvements to the Jini discovery infrastructure. We then, in Section 3, give an overview of the Jini technology and describe the advantages of employing the Jini discovery mechanism in a dynamic environment. In Section 4 we evaluate the Jini service discovery mechanism in a mobile computing environment. The Agents2Go system, a restaurant recommendation system that we have developed, is described and discussed in Section 5. Finally, our research findings are summarized in Section 6.

2 Background and Rationale

The basic assumption behind our work is that at the level of computing and networking hardware, we will see dramatic changes in the next few years. More specifically, we will assume the emergence of (i) palmtop and wearable/embeddable computers, (ii) Bluetooth-like systems which will provide short range, moderate bandwidth connections at extremely low costs, and (iii) widely deployed, easily accessible wireless LANs and satellite WANs. We assume that these will be a mixture of traditional low bandwidth systems and the next generation high speed ones. These developments will lead to wireless networks that will scale all the way from ad hoc body area networks to satellite WANs, and link together supercomputers, "walkstations" and embedded controllers. There is ongoing research in creating the hardware and low level networking protocols that is needed to realize this vision.

The industry's present vision of the use of Bluetooth-like devices is fairly narrow – essentially as point to point communicators to replace cables, similar to IrDA for computers and printers. However, both industry and academia have of late realized that these devices can also be used in creating ad hoc networks – e.g. when police and emergency personnel respond to a call, their computers would automatically get networked when they come in each others' vicinity. There is significant ongoing work in solving various network layer problems associated with mobile ad hoc systems. The point of departure for us is the creation of a software infrastructure that can actually exploit ubiquity arising from ad hoc networking, and enable a new class of applications.

An important enabling component of this vision is the work in distributed object-oriented systems in general, and Java/Jini in particular, which our agent-oriented abstraction will build upon. In order for an entity to cooperate with others in its vicinity, it first needs to discover other entities as it moves into a new location. This problem of "service discovery"[1] has recently been explored elsewhere as well in the context of distributed systems. State of the art systems such as Jini[2], Salutation[10], UPnP[10], as well as IETF's draft Service Location Protocol[8] provide for networked entities to advertise their functionality. Within these newly emerged distributed systems, Jini provides a flexible and robust infrastructure for distributed components to find each others.

The Jini discovery infrastructure supports both unicast and multicast service discovery protocol [2]. This infrastructure allows services to be found in a uniform way, either on a local or a remote network. The service advertisement takes the form of interface descriptions. This simple form of the advertisement mechanism can be easily employed to provide high-level of abstractions for both software and hardware entities in the network. This is how a Jini-enabled printer can talked to a Jini-enabled digital camera.

It is clear that 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. However, the Jini discovery process is tightly bound with the simple interface description advertisement. This leads to a loss of expressive power in the component description. For example, the Jini discovery and lookup protocols are sufficient for service clients to find a printer service. However, they are not sufficient for clients to find a printer service that has the shortest print queue or that is geographically the closest. Clearly, this can create problems in a mobile environment. Furthermore, the simplicity of the Jini architecture also leads to the cross-domain service interoperatablity problem.

One way to enhance the Jini discovery mechanism to work well in a mobile environment is to re-implement or expand the existing Jini lookup process, by adding XML support as an alternative service discovery/lookup mechanism [9]. Another approach, we will describe in detail in this paper, to enhance the Jini discovery mechanism for mobile computing is to apply the agent-oriented design model to the Jini service designs.

3 Discovering Services Using the Jini Technology

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 [2]. Its discovery/lookup protocols allow services that are distributed in the network to be discovered in the location independent fashion.

There are many advantages to employ the Jini discovery mechanism for the 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, possibly through the help some software infrastructure like Jini. The Jini service-interface advertisement mechanism provides a uniform and high-level abstraction to both hardware and software components.

Second, in a mobile environment "things" are often moving from one location to another in a short period of time. For example, as a person driving down a highway, the cellular phone that he/she is using might enter and leave a number of different wireless radio cells. The Jini discovery and join protocols provide a flexible and robust infrastructure for dynamic components to join and leave the network seamlessly.

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.

[1] We use the term service here in a broad sense

4 Evaluating the Jini Service Discovery in a Mobile Environment

Imagine we are going to create a restaurant recommendation system that allows people to find the restaurants of their choices using mobile devices like PDAs and cellular phones. In such system, each restaurant has an agent that can provide information about the restaurant it represents. These agents are distributed in the network, and can be discovered by the restaurant recommendation services. Some of the recommendation services might provide recommendations based on the user profiles. Others might provide services free of charges, or might be capable of making reservations with the restaurant agents on the behalf of the users.

A typical usage scenario that can be realized by such recommendation system can be described as the following:

You are on a business trip from a foreign country visiting Baltimore. During the lunch hour, as you are walking down the street, you are looking for a restaurant. Since you are not so familiar with the Baltimore area and do not have any traveler's guide to local restaurants, you turn on your PDA and ask it to help you to find a restaurant. Given that your PDA is connected with the local network wirelessly, your Jini-enabled mobile application finds one or more restaurant recommendation services, and interacts with these services. Later the application returns back to you with a list of recommended restaurants information.

One might think that creating a such system is trivial since we already have a number of these restaurant recommendation systems all over the Internet. These traditional Yellow Page like services often have centralized databases that contain all of the knowledge that they need to make recommendations. Unfortunately, this centralized approach cannot be applied to a mobile system like the one we have described in above.

In a mobile environment, dynamic attributes and unknown factors often have dramatic impacts on the decision making processes of the system. Let's take our restaurant recommendation system as an example. Dynamic attributes like the driving distances between the restaurants and the traveler, the current traffic conditions between the restaurants and the location of the traveler, the current waiting times of the restaurants etc., could have dramatic impact on the recommendation process. Therefore, in a mobile environment efficient service discovery becomes important.

Due to the increase in the number of dynamic attributes and unknown factors that are needed to be considered in the decision making processes, the complexity of the system also increases as a whole. Further, the interactions between the end users and the service locator system becomes more complicated. For example, a recommendation service would need more inputs from a user who is in a mobile environment than a user who is in a home or an office.

Mobile users are situated in an environment where they only have access to a limited amount of resources, such as the amount of computation power, the types of input devices, the bandwidth of the communication channel etc. Given the increased complexity of the discovery process, thus it raises the question of how to discover desired services effectively with a restricted amount of resources.

It is clear that there is a need for a more expressive and flexible discovery mechanism here. Instead of re-implementing the existing Jini discovery mechanism, we propose a hybrid architecture, a composition of service-oriented and agent-oriented architecture, for implementing services that are designed to work in a mobile environment. The agent-oriented approach to the design of a Jini service does not only provide a self-contained high-level abstraction, but it also increases the expressive power of the communication between services and service clients.

Expressive communication is the key to create a better discovery mechanism. Distributed architectures like Jini, UPnP and E-Speak [7], have defined the protocol specifications for deploying, finding, administrating services. However, they often lack specifications for how services and clients should interoperate. Taking Jini as an example, when a service client makes a service request to the service, it invokes one or more service *interface* methods that are pre-defined in the system ontology [2]. Unfortunately, such an interoperation scheme lacks expressive power since the information is encapsulated at the object level, which is not suitable for modeling high-level interactions like negotiations and contracts. High-level descriptive Agent

[2] An ontology provides a set of well-founded constructs that can be leveraged to build meaningful higher level knowledge. The terms in an ontology are selected with great care, ensuring that the most basic (abstract) foundational concepts and distinctions are defined and specified. See [1]

Communication Language (ACL) like KQML [13], FIPA [5] etc. can be used to provide expressive power for modeling high-level interactions [4]. Using ACLs also offers a uniformed communication infrastructure between the services and the clients, increasing the interoperatablity between these entities.

5 Agents2Go: A Smart Restaurant Recommendation System

To demonstrate the feasibility of dynamic service discovery for mobile computing using our hybrid architecture, we have developed a smart restaurant recommendation system, *Agents2Go*. The Agents2Go project has realized a scenario that is similar to the restaurant recommendation system that is discussed in Section 4. The current implementation allows an end user with a Palm device to dynamically download a restaurant recommendation form from the network when a restaurant recommendation request is desired. The current system also provides support to process the recommendation request even when the end user is disconnected from the network.

5.1 Ronin Architecture Overview

The Ronin Agent Framework [3] 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.

The framework encourages the reuse of non-Java AI applications by promoting the development of Jini-enabled services for these AI applications. For instance, as a part of the Jini-enabled Agent Development Toolkit, a Jini service, Jini Prolog Engine Service (JPES) [6], is developed to provide knowledge sharing and inference capabilities for Ronin agents across the network.

The design goal of Ronin is to define an open framework that specifies the infrastructure requirement and the interface guideline for the interaction and communication between agent-oriented Jini components. Ronin framework models Jini services as agents. 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. Ronin Agent is a specialized Jini service, which can be realized in the form of software or hardware.

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. Specifically, each Agent Deputy must implement a method, `deliverMessage(Envelope)`. The semantic meaning of the method is that after invoking this method, Agent Deputy will try its best to deliver an `Envelope` object, which contains the actual message, to the Ronin Agent it represents. A developer is free to implement any transport protocol and delivery scheme between the Ronin Agent and Agent Deputy. For example, *DeputyArthur*, a concrete Agent Deputy implementation, is implemented to communicate with the owner Ronin Agent through the Distributed Event model [12], which is based on the Remote Method Invocation infrastructure [11]. Another concrete Agent Deputy implementation, *DeputyArjun*, is implemented to communicate with the owner Ronin Agent through the generic network socket model, and is designed to handle the customized *store-and-forward* [2] message delivery scheme. Agent Deputies can be easily designed and implemented to handle other complex delivery scheme such as *notification mailbox* etc. [2].

The messages that are interchanged between Ronin Agents are embedded within `Envelope` objects during the delivery process. This meta-level approach allows Ronin Agents to interchange messages with arbitrary content message types under a uniform communication infrastructure [14]. Within each `Envelope` object, the type of content message and the ontology identifier of the content message are also stored.

There is a set of attributes associated with each Ronin Agent. These attributes can be further divided into two disjoint subsets. The first set of attributes, Agent Attributes, defines the generic functionality of an agent in domain independent fashion. Ronin framework defines the types and the semantic meanings of Agent Attributes. The second set of attributes, Agent Domain Attributes, defines the domain specific functionality of an agent. The Ronin framework does not define the Domain Attribute types and their semantic meanings. Both Agent Attributes and Agent Domain Attributes are are registered with a Jini Lookup Service as Jini `Entry` objects along with the Agent Deputy that represents the agent.

5.2 Agents2Go Architecture Overview

The Agents2Go system is targeted to work with users who have handheld devices that are capable of executing Jini-enable Java applications. However, due to the computation resource limitation of the existing handheld devices like Palm(III, V, VII), the current mobile application that executes on the Palm device relies on a Jini-enabled proxy service, that lives in a network computer, for handling heavy-weighted computation. The sole task of the Palm application is to take user input and display the recommendation results.

The current Agents2Go implementation is based on the Ronin Agent Framework. The system architecture involves the following components (see Figure 1):

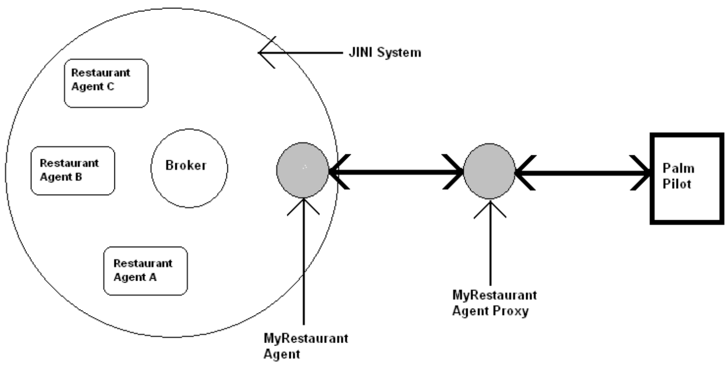

Figure 1: Agents2Go architecture

1. **An End User:** a user who is looking for restaurants.
2. **A Palm Device:** the end user input device.
3. **A MyRestaurant Agent:** this agent helps the user to find restaurants. Like other Ronin agents, this agent has an Agent Deputy who is responsible for deliver incoming messages to the owner Ronin Agent. This agent is designed to live on the Palm device. Due to the current computation constraints on the Palm device, the implementation of this agent is divided into three sub-components:
   - **Palm Application:** *Netform* is a Palm application that interacts with the end user and the MyRestaurant proxy agent.
   - **MyRestaurant Proxy Agent:** this component mediates information between the MyRestaurant agent and the Palm Application. This proxy agent lives on a network computer that is reachable by the Palm device through a TCP/IP connection. The TCP/IP connection can be realized through either a wireline or wireless network.
   - **MyRestaurant Agent:** this component lives in the Jini system and interacts with other agents in the system.

4. **Restaurant Recommendation Broker:** the broker is responsible for handling recommendation requests.

Netform is a Palm application that runs on the end user's Palm device. When an end user comes to the Agents2Go network and starts the Netform application for the first time, the Netform application makes a Point-to-Point Protocol (PPP) connection with the MyRestaurant Proxy Agent that is executing on a local area network computer. If the authentication between the Netform application and the MyRestaurant Proxy Agent succeed, then Netform dynamically downloads a restaurant recommendation input form from the MyRestaurant Proxy Agent. This downloaded form is a GUI-rendered description, that tells the Netform how should it display a GUI input interface on the Palm device. (See Figure 2) The end user enters the recommendation constraints through this form.

After the end user has filled out the recommendation form, the constraints are sent to the MyRestaurant Proxy Agent, who is responsible for mediating information between the Netform and the MyRestaurant Agent. Upon receiving the recommendation constraints data, the MyRestaurant Proxy Agent translates the data that is described in Palm document format to an ACL request message, and then this agent submits the request message to the MyRestaurant Agent.

Figure 2: Netform: the palm application takes user recommendation requests.

The MyRestaurant Agent is a Ronin agent that is responsible for interacting with other Ronin agents in the system and handling the end user disconnection from the network. After the MyRestaurant Proxy Agent had submitted a request message to this agent, this agent finds a Jini Lookup Service using the standard Jini Lookup protocol. It then submits a lookup request to the Lookup Service to find a Restaurant Recommendation Broker Agent. The Lookup Service returns an Agent Deputy for the broker to the MyRestaurant Agent. After the broker's Agent Deputy has been found, the MyRestaurant Agent submits the recommendation request to the Agent Deputy of the broker along with the end user personal profile that might help the broker to make a more intelligent recommendation.

One of the key features of the MyRestaurant Agent is the ability to handle the end user disconnection. An end user disconnection state occurs when the MyRestaurant Agent is isolated from the network either intentionally or unintentionally. To simulate the end user disconnection state, we force the MyRestaurant Agent to disconnect and reconnect from the network after a certain amount of time. Just before the MyRestaurant Agent disconnects from the network, it deploys its Agent Deputy to be available to other Ronin agents, and it sends a *STORE* command to its Agent Deputy, instructing the deputy to collect all the incoming messages on its behalf while it is isolated from the network. When the MyRestaurant Agent reconnects back to the network, it sends a *NEW_LOCATION* command and a *FORWARD* command to its

Agent Deputy, instructing the Agent Deputy to forward all of the collected messages to the new location of the MyRestaurant Agent.

The Restaurant Recommendation Broker is a Ronin agent that is responsible for making intelligent restaurant recommendations. In the current system, the broker is capable of making recommendation inference using Prolog queries. After the broker has received a query request, and if it does not currently have any appropriate recommendation that can be inferred from its knowledge base, then it tries to discover all restaurant agent deputies in the local area through the Jini Lookup Services. The broker agent negotiates with the deputies using an expressive Agent Communication Language like KQML. After the broker had collected enough information, it restarts the inference process. The inferred recommendation reply message is sent back to the MyRestaurant Agent through the Agent Deputy of the MyRestaurant Agent.

After the MyRestaurant Agent has received the reply from the broker, it passes the messages to the MyRestaurant Proxy Agent, whose tasks are translating the reply message to a Palm document format and delivering the translated message to the Palm device.

After the end user has received recommendation, the user could ask for more detailed information about a particular restaurant, such as the daily special of the restaurant and the driving directions to the restaurant.. It is also possible for the user to make a reservation with the restaurant of his/her choice.

6 Conclusion

Computing is no longer a discrete activity bound to a desktop; mobile computing is fast becoming a part of everyday life. If we are going to rely on these mobile devices and applications more often to accomplish our daily tasks, these components need to be aware of others components seamlessly, and these components need to become "smart" to enable us to be more efficient in our work and to give us more leisure time.

Deploying service-oriented systems using distributed architectures like the Jini architecture is not a difficult task, but deploying systems that can perform well in a dynamic mobile environment is not a trivial task. Regardless of the types of the services, software-based or hardware-based, it is important for these services to be able to discover and communicate effectively.

The Ronin Agent Framework introduces a hybrid architecture, a composition of service-oriented and agent-oriented architectures, that provides a simple and uniform scheme for deploying highly dynamic distributed "intelligent" components. These Ronin agents can be realized in the form of software or hardware. Embedding these agents in the mobile applications and devices enables these mobile components to easily discover and communicate with others effectively in a dynamic heterogenous environment. In addition, the success of deploying the Agents2Go system confirms that the vision we have all shared is a feasible approach to developing dynamic mobile systems.

References

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

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

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

[4] T. Finin et. al. KQML as an Agent Communication Language. In *Proc. III Intl.Conf. on Information and Knowledge Management*. ACM, ACM Press, 1994.

[5] FIPA. FIPA 97 Specification Part 2: Agent Communication Language. Technical report, FIPA - Foundation for Intelligent Physical Agents, October 1997.

[6] Harry Chen. Jini Prolog Engine Service (JPES). Available online from `http://gentoo.cs.umbc.edu/jpes/`.

[7] Hewlett-Packard. *E-Speak Architectural Speciification*, version beta 2.2 edition, December 1999.

[8] IETF. Mobility Support for IP v.4 Protocol Draft, v.12. `ftp://ds.internic.net/internet-drafts/draft-ietf-mobileip-protocol-12.t%xt`.

[9] Liang Xu. Using Jini and XML to build a component based distributed system. Technical report, University of Maryland Baltimore County, 2000.

[10] Rekesh John. UPnP, Jini and Salutaion - A look at some popular coordination framework for future network devices. Technical report, California Software Labs, 1999. Available online from `http://www.cswl.com/whitepaper/tech/upnp.html`.

[11] Sun Microsystems, 901 San Antonio Road, Palo Alto, CA 94303, USA. *Java Remote Method Invocation Specification*, revision 1.5, jdk 1.2 edition, October 1998.

[12] Sun Microsystems, 901 San Antonio Road, Palo Alto, CA 94303, USA. *Jini Distributed Event Specification*, revision 1.0 edition, January 1999.

[13] Tim Finin and Yannis Labrou and James Mayfield. KQML as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, 1997.

[14] H. Chen X. Luan. A meta protocol for agent communication. Technical report, University of Maryland Baltimore County, 1999.