An Agent Discovery Architecture using Ronin and DReggie

Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, Anupam Joshi
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD 21250
{dchakr1, fperic1, savanc1, joshi}@cs.umbc.edu

Abstract

Agent-based systems are rapidly gaining prominence in the industry. These agent-systems are all heterogeneous in nature and usually are not inter-operable with each other. The agents follow different languages and offer customized services. We foresee that in the future, there would be a need to discover agents depending on the type of work they do and the type of service they offer. We would also need an uniform network independent communication infrastructure to make the agents collaborate with each other. In this paper, we describe an architecture which aids in the efficient discovery of agents using semantic knowledge. The agents are developed using the Ronin Agent Development framework which enables agents to communicate with each other in a network independent manner.

1 Introduction

In the recent past, we have observed a growth in the development of agent-based systems. There are various advantages associated in modeling a system in an agent-oriented manner.We are moving away from the traditional discipline of developing a single, overarching design for a complex system in which all of the parts are carefully engineered to fit together. In the future we envisage agent-based systems where each complex system would be composed of multiple agents, each skilled in performing its own set of tasks. System development is taking a new approach in which the individual atomic components/agents would be designed to be autonomous ("active"), self-describing ("articulate"), highly interactive ("social"), and adaptive ("intelligent"). We observe that the different multi-agent systems are incapable of interacting with each other due to various problems related to agent language, communication protocol etc. These agents usually are capable of providing certain customized or general services to agents in their own system and following the same language.

In a networked environment, one might want to take advantage of the services offered by these agent systems by being able to utilize them as service providers when needed. Hence it is very essential to come up with a robust architecture for discovering agents and being able to know what services they offer. Along with that, we also need an efficient agent development platform which solves the potential problems that crop up when heterogeneous agents want to inter-operate with each other.

In this paper, we describe an agent-based discovery architecture which enables flexible agent discovery based on the type of services these agents offer. We use DAML [2] as a language to represent agent characteristics and the services that these agents might offer. We have developed our system using the Ronin Agent Framework [4]. In section 2, we describe some limitations of the present discovery techniques for discovering agents in a system. In section 3, we describe the Ronin Agent Development framework. In section 4 we describe the DReggie system: an enhanced Jini-based service discovery system. In section 5, we describe the architecture which enables intelligent agent discovery in a distributed environment. We conclude in section 6.

2 Background

Research in the field of dynamic agent discovery essentially coincides with the research in the field of service discovery. Quite a few industry standards have been implemented in the recent past in the field of service discovery. Examples of such prototype systems include the Service Location Protocol [5], Upnp, Salutation and Jini [6, 1]. We are unable to provide a description of these service discovery protocols due to space constraint. These systems typically base their discovery protocol on attribute or interface matching. Such trivial matching limits the capability to discover complex agents which implement complex services. These discovery mechanisms also dont provide the client with reasoning capabilities based on the characteristics of the agent. Thus even though these systems provide a reasonably good foundation for discovering simple services, we argue that they are not sufficient for building complex distributed agent-based systems. We highlight a few limitation of the traditional service discovery systems.

- **Lack of Rich Representations:** Agents in an agent-based system has certain characteristics and certain properties. It is capable of providing certain services. These services are defined in terms of their functionalities and capabilities. The existing service discovery infrastructures lack expressive languages,representations and tools that are good at representing a broad range of service descriptions and are good for reasoning about the functionalities and the capabilities of the services [4].

- **Lack of Constraint Specification and Inexact Matching:** The existing service discovery protocols lack the ability to do inexact matching. Service capability matchings are processed in the object-level and syntax-level only.For instance, the generic Jini Lookup and Discovery Protocol allows a client to find a printing service that supports color printing, but the protocols are not powerful enough to find a geographically closest printing service that has the shortest print queue. The protocols do exact semantic matching while finding out a service. Thus they lack the power to give a close match even if it was available.

- **Lack of Ontology Support:** In a heterogeneous environment, agent descriptions and the services they provide need to be understood and agreed among various parties.In other words, well-defined common ontology must be present before any effective service discovery process can take place. We found that common ontology infrastructures are often either missing from or not well represented in the existing service discovery architectures.

3 Ronin: The Agent Development Framework

The Ronin Agent Framework is a Jini-based agent development framework [4] 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 distinguishes it from other comparable frameworks. These include an Agent Communication Language (ACL) and network protocol independent communication infrastructure. It also includes an agent description facility that allows agents to discover and find each other, an agent proxy architecture that provides agent mobility behavior and customizable agent communication scheme.

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:

- Ronin Agent
- Agent Deputy

3.1 Ronin Agent

A Ronin Agent is a specialized version of a Jini service that is designed with the agent-oriented abstraction in mind. Unlike a traditional Jini service that only provides services through the predefined service methods, a Ronin agent provides services through agent communication as well. A Ronin Agent can be realized either by hardware or by software. Ronin Agents may differ in their actual implementation, however, each Ronin Agent must have a set of Common Agent Attributes, that can be used to describe the generic agent properties and functionalities across different domains. Ronin agents communicate with other agents through message passing using ACLs.

3.1.1 Ronin Agent Attributes

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 a 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 can be registered with a Jini Lookup Service as Jini Entry objects along with the Agent Deputy that represents the agent.

3.2 Ronin Agent Deputy

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. Each Ronin Agent may have zero or more agent deputies in the network. Conceptually an agent deputy can be thought of as a light-weight copy of the owner agent in the network. The main duty of the Agent Deputy are

- **Mediate agent communication:** The Agent Deputy accepts information on behalf of the owner agent and delivers the messages to their owners. They may mediate on the content of the information also. The agent deputies are considered as trusted representatives of the owner agent.
- **Provide local agent representation:** The Agent Deputy acts as a local representative for the owner Ronin Agent in a different system. This local proxy thus can answer some questions on behalf of the owner Ronin Agent.

3.3 Ronin Communication Infrastructure

Ronin does not define how an Agent Deputy should communicate with its owner 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 owner Ronin Agent and its Agent Deputy. For example, DeputyArthur, a concrete Agent Deputy implementation, has been implemented to communicate with the owner Ronin Agent through the Distributed Event model, which is based on the Remote Method Invocation infrastructure. Another concrete Agent Deputy implementation has been implemented to communicate with the owner Ronin Agent through the generic network socket model, and is designed to handle the customized store-and-forward message delivery scheme. Agent Deputies can be easily designed and implemented to handle other complex delivery scheme such as notification mailbox etc.

3.4 Envelope Objects

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 metaprotocol. Each envelope contains the following components:

- **Recipient:** The intended reciever of the content message
- **Sender:** The message sender
- **Language:**The ACL identifier of the content message
- **Ontology:** The ontology identifier of the content message
- **Deputy Locator:** The locator is used to locate the Agent Deputy of the message sender, making the two-way communication possible between the receiver and the sender. The framework does not define how a Deputy Locator should locate an Agent Deputy in the system. This makes the framework open and implementation independent.

4 DReggie Overview

The project DReggie [3] is an attempt to take Jini and similar service discovery systems beyond their simple syntax-based service matching techniques and enable semantic matching capabilities by adding highly expressive service description facilities. Semantic service matching widens the scope of a certain service discovery request by being able to locate services which 'approximately' match the given request. Service matching can be carried out based on the functional description of the service. Thus even though a service exactly matching a given request may not be available in the environment, semantic discovery has the ability to find out services which have been advertised in a differenct way but carries out the same functionality. DReggie enables such discovery by adding the facility of rich service description using DAML and introducing an intelligent reasoners to be able to carry out the semantic matching process. At the heart of DReggie is an enhanced Jini Lookup Service (JLS) that enables smart discovery of Jini-enabled services. DReggie also helps clients discover services in a manner unchanged from the existing Jini Lookup and Discovery infrastructure. In addition, it allows services to advertise their capabilities, constraints and system features in a well-structured, descriptive format using DAML. The clients requesting services have the option to utilize the intelligent prolog matching module to find out a wide array of services 'nearly' matching the given service request.

Any service using the DReggie infrastructure is described using DAML in terms of its capabilities, requirements and service attributes. When a service registers with the enhanced Jini Lookup Server (JLS), it registers a DAML description entry as well. This DOM object is stored in the lookup server and updated whenever the service registers again. When a client wants to use a service, the client creates a DAML DOM object that describes the desired service along with its constraints. The DOM object provided by the client contains "approximate" information about the type of service the client is attempting to discover. The DReggie lookup server contains two different modules to carry out the matching process of the service request with descriptions of the advertized services: a simple java-based matching module and an advanced prolog-based reasoning module. In our initial implementation of the infrastructure, the simple java-based reasoner has been integrated with the JLS. The java-based module compares the two DOM objects and finds exact or approximate service matches. The other advanced reasoning module, the prolog engine service is capable of more complex matching based on DAML service profiles. In our future work, we aim to integrate the Prolog engine as a module with the Jini Lookup Server to discover m-services more efficiently and successfully. Figure 1 summarizes our vision.

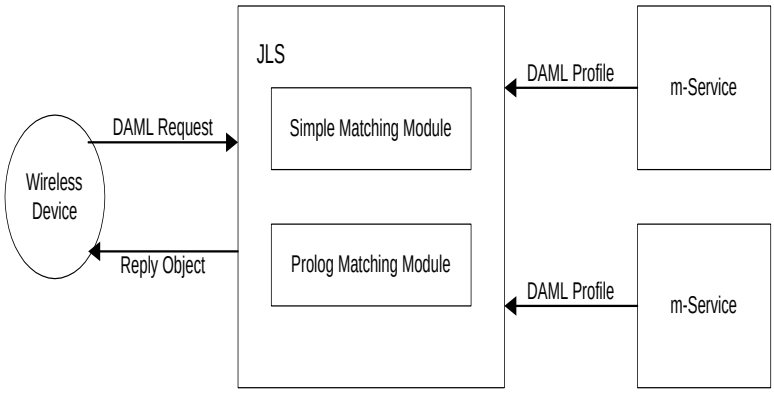

Figure 1: DReggie Components

It is assumed that the request that a client issues uses the same ontology that a service uses to describe itself. This is a very important assumption. It does restrict service matching to one particular ontology, but enables more knowledge based discovery of services. In the future, it is quite likely that there would be translator services that would enable translation of one ontology to another and hence enable cross-ontology service discovery. The DAML match in JLS handles constraints such as client requirements, cost, mobility etc. It ensures that a client discovers only those services that it is capable of executing in terms of hardware or software requirements.

In order to compare DAML profiles in the Jini Lookup Server, we have made modifications to Reggie - Sun's implementation of Jini Lookup Service. Reggie provides all the functions and methods defined in the Jini Core package, besides some extra functionality. We modified it as follows:

- A new entry class is added in the net.jini.lookup.entry package, which takes an well-formed DOM object as an argument in its constructor.
- An DAMLMatchImpl class is plugged in the com.sun.jini.reggie package, which implements the functions required to compare two DAML instances. It uses the Java Xml parser to convert the DAML description to a Java DOM object and does the comparison with the two 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(DReggie) know when and how to do the DAML-based 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(DReggie) know where to find the imformation of XML parser package

5 Architecture Overview

The architecture is divided into three layers, viz. the network layer, the dynamic agent discovery layer and the uniform agent communication layer. Figure 2 explains the layered architecture.

5.1 Network Layer

The Network layer is the lower most layer in our architecture. This layer represents the underlying communication infrastructure being used to communicate between different Ronin Agents in our system. Some distinctive features of the network layer are:

- The network layer encapsulates the whole underlying transport module needed to transfer data from one system to another. It is not analogous to the standard network layer in OSI or TCP/IP protocol suite. Rather our network layer comprehensively encapsulates the various different network transport protocols like TCP/IP, UDP, W/ATM, Bluetooth or CDPD.

- The Ronin Agent development framework allows the use of different protocols for transferring its messages. This is particularly useful since our network layer can encapsulate any type of communication mechanism. The network layer provides the upper layer with the required abstraction to switch between different networks as an when required. The network layer thus helps in seamless integration of different network protocols in our system.

- Instead of using standard network protocols like TCP/IP, the network layer also allows higer level message transport mechanisms. Thus the transport modules could also be written based on the Distributed Event and Notification system, or any other communication protocol.

5.2 Dynamic Agent Discovery Layer

This layer represents the mechanisms through which Ronin agents are discovered and the mechanisms through which the agents express their functionalities. This layer enables the architecture to support dynamic interaction between agents with no or little prior knowledge of each other.We implement this layer using the 'DReggie' system.
The DReggie system has been discussed in detail in section 4. Each Ronin Agent in the system registers itself with the DReggie Lookup Server. Each agent registers its essential common and domain agent attributes to enable simple attribute-based search for the agent. Along with that, the agent also registers a DAML description entry as well. Thus, a description of the capabilities and constraints of all agents existing in the local network is kept inside the DReggie lookup server. An agent description is cached and purged if the agent fails to re-register within a certain amount of time. The DReggie Lookup Server forms the core of the agent discovery layer. We however follow a common ontology to specify services and requirements in the system. The reason for doing this has been explained in section 4. Thus each agent can discover the other services in the system by little or almost no information about the others.
We have created an ontology to describe a Ronin Agent in DAML. The ontology can be found at http://www.daml.umbc.ed
ont.daml. We mention some of the characteristics we have considered, to articulately describe an agent.

- Agent Name

- Agent capability : Contains a description of what the agent can do

- Agent Requirements: Agent requirements include the resource requirements of a particular agent. We considered the CPU type, CPU speed, Memory required and other softwares required for the agent.

- Agent Mobility: whether the service can provide a serialized code to be executed else where in any other system

- Agent Inputs: The input types and arguments required to invoke the service that the agent offers

- Output Types: The format in which the agent returns a result

- Agent Machine Characteristics: It includes a description of the different attributes of the system on which the agent is currently running.

Whenever an agent wants to find out another agent which provides a particular service, it specifies some requirements in an DAML file following the same ontology. This helps us achieve a more semantic level agent discovery in our system. However, the client agent might not have all information required to properly

recognise the service it wants. The client then only partially fills up the DAML file with the information it knows. The rest of the fields are left as 'UNSPECIFIED'. The modified lookup server uses the DAML Dom objects produced from the XML descriptions to appropriately find out matching services. Please refer to section 4 for details.

5.3 Uniform Agent Communication Layer

This layer manages the delivery of information to the agent. Ronin Agents implement a common way of handling and delivering messages. Thus, even if the underlying transport mechanism used to actually send the messages are different, the agents can send and receive messages in a uniform way. The Agent Deputy part of the Ronin Agent Framework is responsible for handling messages in this layer. An Agent Deputy imitates 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. As discussed in section 3, the remote sender Agent uses a standard interface to communicate with any Agent Deputy. The communication mechanism between the Agent Deputy and the owner agent is however not specified. It is implementation dependent. This proxy-based uniform agent communication mechanism has a few distinctive advantages in an agent-oriented infrastructure.

- When a mobile host represents a Ronin Agent, it's Agent Deputy sits on the wired network. Similarly, the Agent Deputy of a Ronin Agent sitting on the wired network could reside in the mobile host. Ronin does not define how an Agent Deputy should communicate with it owner Ronin Agent. However, it does define the interface for the communication between the sender Ronin Agent and Agent Deputy of the recieving agent. Thus different heterogenenous agents can use the same agent communication primitive to communicate with each other.

- 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.

- The Ronin Agent Deputy inherits many unique features from the original design of the remote proxy object. In addition to that, there are a number of positive design side-effects of using the Ronin Deputy-based infrastructure as the agent communication layer.

    1. The design of the Agent Deputy incorporates mobility features to the owneer Ronin Agent. Each Owner Agent may have more than one Agent Deputy. Once these deputies have been deployed in the network, the Owner Agent may move off to another location. It is the task of the Agent Deputy to establish contact with its owner mobile agent.
    2. The design allows the Agent Deputy to act like a *store-and-forward agent* [1], capable of delivering messages based on various policies for reliability. For example, the Agent Deputy could try to deliver a new message once and if that event fails, not try again. Or the Deputy might implement a acknowledge-based reliable delivery scheme. Either way, its the duty of the agent deputy to ensure that the messages reach its Owner Agent. The Deputy could also act as a message mailbox where the owner agent could store messages for a certain amount of time when it is doing some higher priority work.
    3. The Agent Deputy could act as a message filter or a transcoding proxy for the Owner Agent[7]. Eg. If the Owner Agent moves into a low bandwidth network, the Agent Deputy could transcode the content sent to the Owner Agent. It could also implement security by intercepting messages that failed to pass the security check policy.

4. The Agent Deputy could act as a message multiplexer or demultiplexer. Upon recieving a new message, the Agent Deputy could multiplex the message by delivering the message to the deputies of other agents. The deputy could also combine several messages and send a digested version of the message to its owner agent.
5. The Agent Deputy could also act as a translator of messages coming from agents following other ACLs. A Ronin Agent may be capable of processing a restricted set of ACLs and ontologies. Thus different *translator* agents could be deployed in the network to translate the message to an understandable format.

6 Conclusions

The utility of discovering agents capable of delivering certain services will become increasingly important as we move into the future. With more and more systems being modeled with an agent paradigm, the need for discovering useful agents and reusing their capabilities will increase in the world of distributed systems. In this paper, we have described an architecture, which helps in finding out agents capable of providing certain services in an intelligent manner. The system also helps heterogeneous agents to inter-operate using the Ronin Agent development framework. The DReggie system provides the basis for discovering customized agents in a distributed system.

References

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

[2] Tim Berners-Lee, James Handler, and Ora Lassila. The Semantic Web. In *Scientific American*, may 2001.

[3] D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. DReggie: A Smart Service Discovery Technique for E-Commerce Applications. In *Workshop held in conjunction with 20th Symposium on Reliable Distributed Systems (SRDS). New Orleans.*, october 2001.

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

[5] E. Guttman, C. Perkins, J. Veizades, and M. Day. "rfc2068: Service location protocol, version 2". ftp://ftp.isi.edu/in-notes/rfc2608.txt, 1999.

[6] 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.

[7] B. Zenel. *A Proxy Based Filtering Mechanism for The Mobile Environment.* PhD thesis, Department of Computer Science, Columbia University, N/A.