Middleware for Mobile Information Access

Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Timothy Finin, Yelena Yesha
{dchakr1,fperic1,joshi,finin,yeyesha}@cs.umbc.edu

Abstract

Mobile information access involves retrieving information from wired service providers. Often there are situations where the information is not available from a single service provider but can be obtained by combining information from multiple service providers. It is inefficient and some times impossible for a resource poor mobile device connected over a low bandwidth wireless link to coordinate such activity. In this paper, we describe a middleware to support this mode of access from resource poor mobile devices that take into consideration mobility, resource constraints and service heterogeneity.

1. Introduction

Relentless progress in a wide range of mobile technologies, from networking protocols to smart devices has ushered in a new age of information access. We are seeing the deployment of heterogeneous mobile devices like iPaqs, PalmPilots, HP jornadas each of which differ from the other in terms of platform specifications, processing power, memory availability etc. In the past, these devices were mostly used as *standalone* personal digital assistants. However, with the advance of wireless networking technologies most of these devices have some type of connectivity to the Internet. Thus, a change is being observed in the usage pattern of these devices. Consumers of these devices have started using them not only as standalone organizers, but also as *clients to connect to servers or services in the wired Internet* to access information or carry out transactions. The Internet has always been a huge repository of information and services. Internet-based wired services are capable of providing general as well as customized information to their clients. However the key difference of accessing these services from mobile units rather than accessing them from wired clients lies in the last mile, the disconnection prone and bandwidth limited wireless channel, and the end client. Information access from mobile devices has to take into consideration, limited resources of the mobile devices such as display capability, processing power. Wireless link related problems like disconnection, bandwidth, jitter need to be considered when carrying out end-to-end transactions from wireless clients. *Client-proxy-server* type of architectures for mobile information access [4, 15, 3, 10] was developed in the academia for this purpose, where the proxy handles the above-mentioned problems and standard solutions (e.g. Wireless Application Protocol) are being used in the industry.

The assumption behind the *client-proxy-server* model is that the information that the mobile device needs is available from a single service provider on the wired side. This is not always the case. Often the information a mobile device needs is not available from one single service provider but can be generated by combining multiple different services [14, 12] in the fixed wired infrastructure. Examples of such requests may be that of a business person requesting a change in the business appointments due to a sudden change of plans, which might include notifying other people about a change in meeting time or canceling a hotel reservation etc.

This gives rise to a *client-proxy-multiple-servers* type of scenario. Multiple services coordinate with each other and are executed in a certain manner to provide information to the end-user. It is essential for some entity to act as the *coordinator* for such requests. Clearly, it is very difficult for mobile devices to coordinate such activity not only because it *might not have the required resources*, but also because some of the transactions might need a *disconnection-free high bandwidth channel*. Multiple heterogeneous services also need to have an uniform way of interacting with each other.

There might be instances where *some part of the computation needs to be done at the end-client* because of the interaction level required with the user. This might be a problem in mobile devices. Standard proxy-based solutions to mobile information access do not handle such cases.

In this paper we describe an agent-based middleware architecture to address the above-mentioned problems that arise when a mobile unit asks for information that require coordination of multiple services. Central to our design is the concept of a broker agent that resides on the wired infrastructure and handles queries from mobile devices. The

middleware has been developed over the Ronin Agent Development framework [6]. The Ronin Agent Development framework provides an uniform communication infrastructure for multiple agent-services to collaborate with each other. It also has the notion of agent-deputy that is capable of handling disconnections in the wireless channel. Services are modeled as Ronin Agents by wrapping up the actual services with the corresponding wrapper. Agent-services are discovered in the system using DReggie: a semantic service discovery system that uses DAML+OIL [13] to describe and reason about capabilities of services. The middleware broker receives a profile of the resource limitations of the mobile device along with the request and schedules or organizes the computations accordingly. It also carries out optimizations to reduce bandwidth utilized while carrying out the execution of multiple services and hence reduce the cost of executing a request.

2. Middleware Design

In this section, we present the layered design architecture of the middleware. We discuss the various layers and the functionalities enclosed within each layer.

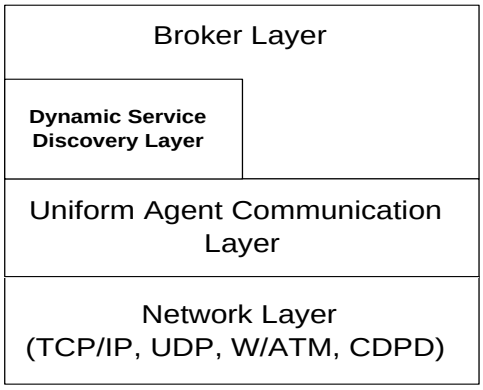

Figure 1. Layered Architecture

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. The Network Layer encapsulates the whole underlying transport module needed to transfer data from one system to another. The Ronin Agent development framework allows the use of different protocols for transferring its messages. The Network Layer provides the upper layer with the required abstraction to switch between different networks as and when required.

Uniform Agent Communication Layer: This layer manages the delivery of information between agents. The Ronin Agent Framework [6] provides a network-layer independent uniform mechanism of communication between different agent-services. Each agent consists of two parts: *Ronin Master Agent* and *Ronin Agent Deputy*. Unlike a traditional Jini service that only provides services through the predefined service methods, a Ronin Master Agent provides services through agent communication as well. The Agent Deputy mediates agent communication with its master agent and acts as the local lightweight representative for its owner master agent in the network. The communication is carried out in a language and network independent manner. An Agent Deputy communicates with master agents using "Envelope" objects that contain the recipient, sender, agent language being followed and deputy locator that provides an object to locate the agent deputy of the sender agent. This makes the communication independent of any agent language. The framework is open and hence implementation independent.

The Agent Deputy part of the Ronin Agent Framework is responsible for handling messages in this layer in our middleware. When a mobile host represents a Ronin Agent, its 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. The design of the Agent Deputy enables us to change the network transport mechanism dynamically when a mobile host moves across networks and hence achieves network independence.

Broker Layer: The Broker Agent is the core engine, which finds the individual services required in computing required information and manages the execution of services. This layer has the knowledge base of what services should interact to produce the required information to the end-user. Broker Agents are domain-specific and capable of executing complex queries for a particular domain. In a nutshell, each Broker performs the following tasks:

- *Task Decomposition:* Once a Broker receives a particular task to be executed, first it tries to figure out whether the task can be broken down into multiple subcomponents. The problem of splitting a task into sub-tasks is complex and goes into the domain of planning [8] in AI, which is outside the scope of our present work. Current implementation of our architecture assumes the Broker has the adequate static knowledge to decompose a query into its subcomponents. It is a straightforward exercise to plug in an external planning system into the design that will provide the system with a process model of execution for a composite service.

- *Service Discovery and Hierarchical Service Screening:* The Broker Agent uses the DReggie [5] system to discover Ronin Agents capable of providing the individual services. Using the DReggie system the Broker also obtains a description of the current resources available on particular agent platforms. Multiple instances of same agent-services are screened in a hierarchical manner based on certain parameters such as estimated execution time, estimated wait-time, etc. The best available agent-service is chosen amongst the ones discovered.

- *Execution of the components and run-time optimizations:* Once the various services have been discovered, the Broker tries to determine an optimal way of executing the query. Optimization criteria used may be, amount of data transfer over the network, amount of bandwidth utilized, amount of CPU resources used or average response time. Information about the service platforms is obtained while discovering the services using the DReggie system. The Broker Agent gathers this information and determines an execution path to be followed to execute the query. The Broker then manages the invocation of the different agent-services and manages the execution following the path it has determined. Based on the resource capability of the mobile device (e.g. high-end laptops), the Broker Agents might decide to execute some component at the client also.

Fixed networked hosts (that are resource-rich) are capable of providing general-purpose execution platforms to execute components/services remotely. The Broker utilizes these execution platforms when some computation cannot be done at the mobile client. These platforms are also utilized when due to system overloading or some other reasons, a service platform (hosting a particular agent-service) might be unwilling to serve a certain request locally. Rather it might be willing to give a serialized code out that could be executed in the execution platform.

Dynamic Service Discovery Layer: This layer represents the mechanisms through which agent-services are discovered and the mechanisms through which agent-services express them. Traditional approach to service discovery uses either interface-based or attribute-based [1, 11] or unique-id based service matching [2] to discover matching services. However, such forms of service discovery are unable to discover services based on their functionality descriptions or by reasoning about the capabilities of services in a semantically meaningful way.

We developed the DReggie system: a Jini-based service discovery framework where service discovery is done using semantic descriptions of services using DAML+OIL. Service requests are also encoded in DAML+OIL and those requests are matched with service descriptions. We use the class-subclass hierarchy of DAML+OIL and the rules provided within it to reason with service descriptions and discover "nearly" matching services (e.g. discover a color printer when the request was to discover a black-and-white printer). This increases the flexibility in discovering a broad range of heterogeneous services.

In this middleware, Ronin Agent-services can be discovered using agent attributes (Ronin *Common Agent* attributes and *Domain Agent* attributes) [7]. However, in addition to that, we have also provided a DAML+OIL description of the agent and the service it encapsulates. We have represented an agent-service in terms of service name, service capabilities, service requirements (platform type, memory needed), service inputs, service outputs and service platform characteristics (resource details of the platform on which the agent-service is residing now). The ontology we developed for this purpose can be obtained from *http://daml.umbc.edu/ontologies/dreggie- ont.daml*. Thus Broker Agents can specify different features that it wants an agent-service to satisfy to discover services. This adds great flexibility to the process of discovering agent-services.

3. Implementation

As an example implementation of the above middleware architecture, we implemented a set of agents and services to handle complex stock related queries from resource poor mobile clients. These queries need the cooperation of multiple information providers and computation platforms to provide an answer back to the client. We implemented a broker that can handle complex stock-related queries. The essential entities in our system are listed below:

3.1. End User

The End User (client) of our system is essentially a person with a mobile device (Laptop or iPaq). An End User discovers broker agents capable of performing complex tasks that require coordination amongst multiple agent-services. The client obtains a list of the functions that a Broker Agent can perform. It invokes the required task accordingly.

3.2. Broker Agent

We have implemented vertical brokers capable of handling requests from mobile clients. As an example, one broker handles complex stock related requests. All brokers are Ronin Agents. When a Broker receives a request, it determines the different subcomponents required to execute that request. We have implemented this section by keeping

a plan of the different agent-services required to execute a certain job and the partial order in which they should be executed.

The Broker uses the DReggie system to discover and screen appropriate services. It also obtains a description of the resource availability on the service platforms and the mobile device. It discovers computation platforms that can execute serialized code of services. It then performs the process of execution of the individual components. The Broker tries to optimize some essential parameters like bandwidth, or execution time while performing the component executions. Components/services could be executed either at the mobile device (based on requirement), or at the service platform or at another execution platform (provided the component is serializable) . The Broker schedules computations that cannot be done at the mobile client to the different execution platforms. It also handles transferring the execution of agent-services to execution platforms if the local platform is heavily loaded with requests.

Some executable service components are sent to the mobile client if it is capable of performing certain computations.

3.3. Agent Services

We have implemented various types of agent-services to handle various stock-related queries. All the agent-services are resident on the wired infrastructure and the end client is mobile. These agent-services are essentially wrappers over the actual web services that provide the information. All agents in the system use Ronin's uniform communication infrastructure to communicate with each other. Some of the agent-services are enlisted below:

- *Single Stock Quote Provider Agent:* This Agent is capable of providing stock quotes of a company on that day. It connects to a stock provider agency (yahoo.com in our implementation) and retrieves real-time stock information from it.

- *Execution Platform Agent:* This agent provides an execution platform for computations that cannot be performed on the mobile client or computations that current service platforms are unwilling to perform (due to load on the local system). The main concept behind creating the service is that due to system overloading or some other reasons, a wired service might be unwilling to serve a certain request locally. Rather it might be willing to give a serialized code out to the client to be executed locally. However, a mobile client might not have resources to execute the code locally. These types of agent-services provide an execution platform for execution of those types of components.

- *Average-Stock-Calculator Agent* This Agent accepts a range of stock values and computes the average of those stock values. This is a simple service. We implemented this service to show the dynamic interaction of different such services in our system. In the future, there could and will be complex services doing specific tasks like computing the standard deviation of some values. The main aim is to show that such services can easily be injected into our system.

3.4. Communication Deputies

We have implemented two different types of concrete agent deputies in order to facilitate efficient agent communication in different environments. An Agent Deputy may be implemented to offer various different proxy type services to the owner agent. Each implementation of an Agent Deputy holds a transport object that actually performs the task of transmitting the data over a particular network. Whenever a deputy is asked to deliver a message to the corresponding owner agent, it delegates this responsibility to the "Transport" object it holds. Thus the transport object can be dynamically changed as the network of a mobile device changes. We have implemented two different implementations of Agent Deputies; a *Simple Reliable Agent Deputy* and *Store-and-Forward Agent Deputy*. The A Simple Reliable Agent Deputy encapsulates TCP sockets in its transport object and provides a reliable connection-oriented service while the Store-and-Forward Agent Deputy handles disconnections with the help of *store* and *forward* control messages with the master agent.

We used the DReggie system to do agent discovery in this environment. Agents can be discovered using agent attributes as well as by reasoning over the DAML+OIL description of its capabilities, platform dependencies etc. We have explained the discovery process in section 2.

3.5. Summary of Experiments

We performed several experiments to test our system and checked the adaptive ness of our broker with respect to different constraints of the agent-services and different constraints on the resource availability of the platforms having those services. We have several agent-services running on multiple platforms (Solaris with SunOS 5.7, Intel with Linux 2.2.16), interconnected through a 100MBPS LAN. We used several laptops having 802.11b Wireless LAN connectivity as mobile devices.

In our implementation of the Broker, its main objective is to find an optimal solution path of executing multiple service-agents by trying to optimize the amount of data transfer over the network by taking into consideration the available bandwidth. It also tries to optimize the average

response time required to serve a query. We have introduced a notion of "monetary cost" associated with executing each service. In the future, it is quite likely for web services to charge certain amount of money for providing information [9]. The broker, while doing hierarchical service screening takes into consideration the cost of executing a service. It considers the mobility of the components at a later stage of service screening. The Broker calculates an optimal path by balancing several factors like cost, mobility, amount of data transfer required over the network, bandwidth available etc. In our implementation, the Broker gives highest priority towards optimizing bandwidth usage over the network.

In our experiments, we have observed that the overall nature of the system is flexible and adaptive to the resource changes in the environment. For a weak mobile client (a client having low processing power and memory), all the computations are performed at the agent-service platform or the execution platforms. If the service-agent platform is heavily loaded, then the execution is moved to an execution platform that can execute the serialized code of the service-agent (provided the agent-service is serializable). For moderately capable clients, some of the components are sent to the mobile host again depending on the amount of data transfer required. The Agent Deputies handle certain other limitations like disconnection. Due to space limitations, we are unable to provide a detailed description of the experiments.

4. Conclusions

To realize the vision of ubiquitous information access from mobile devices, we need to consider instances where the information is not readily available from information sources but needs to be generated by combining multiple such heterogeneous service providers. In this paper, we have presented an agent-based middleware architecture to address this problem. We have used the Ronin Agent Framework and the DReggie semantic service discovery system to address the problem of heterogeneity of information sources, network resources and facilitate flexible service-agent discovery. The middleware broker architecture computes a cost-effective way of executing multiple services by taking into consideration resource limitations of the various platforms and bandwidth for information transfer.

References

[1] K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler, and J. Waldo. *The Jini specification*. Addison-Wesley, Reading, MA, USA, 1999.

[2] S. Avancha, A. Joshi, and T. Finin. Enhancing the Bluetooth Service Discovery Protocol. Technical report, University of Maryland Baltimore County, August 2001. TR-CS-01-08.

[3] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakyl. An active transcoding proxy to support mobile web access. In *Proc. IEEE Symposium on Reliable Distributed Systems*, October 1998.

[4] C. Brooks, M. S. Mazer, S. Meeks, and J. Miller. Application-specific proxy servers as http stream transducers. In *Proc. WWW-4, Boston*, May 1996.

[5] D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. Dreggie: A smart service discovery technique for e-commerce applications. In *20th Symposium on Reliable Distributed Systems*, october 2001.

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

[7] H. Chen, A. Joshi, and T. Finin. Dynamic service discovery for mobile computing: Intelligent agents meet jini in the aether. *Baltzer Science Journal on Cluster Computing, Special Issue on Advances in Distributed and Mobile Systems and Communications*, 2001.

[8] K. Erol, J. Hendler, and D. Nau. HTN planning: Complexity and expressivity. In *Proc. AAAI.*, 1994.

[9] P. Fishburn, A. Odlyzko, and R. Siders. Fishburn, p.c., a.m. odlyzko and r.c. siders. In *Hurley, D., Kahin, B., Varian, H. (eds.) Internet publishing and beyond: The economics of digital information and intellectual property. Cambridge, MA: MIT Press*, 1997.

[10] A. Fox, I. Goldberg, S. Gribble, D. Lee, A. Polito, and E. Brewer. Experience with top gun wingman: A proxy-based graphical web browser for the USR palmpilot. In *Proc. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, 1998.

[11] E. Guttman, C. Perkins, and J. Veizades. RFC 2165: Service location protocol, 1997.

[12] R. Katz, E. A. Brewer, and Z. Mao. Fault-tolerant, scalable, wide-area internet service composition. Technical Report. UCB/CSD-1-1129. CS Division. EECS Department. UC. Berkeley, January 2001.

[13] D. A. M. Language and O. I. Layer. `http://www.daml.org/2001/03/daml+oil.daml`.

[14] D. Mennie and B. Pagurek. An architecture to support dynamic composition of service components. Systems and Computer Engineering. Carleton University, Canada.

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