A Reflective Heterogeneous Service Coordination Middleware
Based On Mobile Agents in Wireless Environments

Haiyang Hu\textsuperscript{1,2}, Hua Hu\textsuperscript{1}, Yi Zhuang\textsuperscript{1}, Lianghuai Yang\textsuperscript{3}
\textsuperscript{1}College of Computer and Information Engineering, Zhejiang Gongshang University
\textsuperscript{2}State Key Laboratory for Novel Software Technology, Nanjing University
\textsuperscript{3}College of Information Engineering, Zhejiang University of Technology
hh@mail.zjgsu.edu.cn

Abstract

Software services distributed in open wireless environments are implemented using different middleware types and advertised using different discovery protocols. Client application needs to adopt a flexible approach to discover and bind these services. Compared with other related work, this paper presents a new reflective coordination middleware based on mobile agent. It dynamically adapts both its binding and discovery protocol to allow interoperation with heterogeneous services by uploading different functional components. By dynamically creating subagents at run time, it can interoperate with multiple services in parallel. As a result, the client burden is released, and the efficiency of the coordination process is enhanced.

1. Introduction

Today, the continuous development and improvement of wireless networks and mobile computing devices, together with their challenging limitations, provide an open and dynamic environment for software services running in it becoming more autonomous and heterogeneous \cite{1}. To interact with these services, application platforms must have the capability of both deployment-time configuring and runtime reconfiguring. Also for the services developed upon a range of middleware types and advertised using different service protocols unknown to the developer, some efficient service coordination mechanisms are needed to discover these services, bind and interact with them.

At present there are four important service discovery protocols in the commercial realm: Jini \cite{2}, Service Location Protocol \cite{3}, Universal Plug and Play \cite{4}, and Salutation \cite{5}, these protocols are used to publish application services, and provide service discovery and coordination mechanisms. Object-oriented middleware such as CORBA \cite{6}, DCOM \cite{7} and Java/RMI \cite{8} also provide similar service discovery and coordination function. However, all these systems can’t support interaction between heterogeneous services developed upon different types of protocols. As an example, a service is published using Jini protocol. The applications calling it must also be developed upon Jini discovery protocol. Though the aim of CORBA is to support interaction between applications developed in different program languages, the client and server have to communicate with each other by ORB. As a result, in open dynamic environments, an client application using just one single discovery or band protocol can’t interact with the services developed upon other protocols.

To support the interoperation with heterogeneous services, client platform must have the capability of dynamically reconfiguring its structure and behavior so as to find the required
services irrespective of their service discovery protocol and to interact with these services. Until now, two kinds of reflective middleware have been proposed to cope with this.

The design of Universally Interoperable Core (UIC) [9] is based upon the reflective middleware DynamicTAO [14]. Its goal is to support interactions with multiple service platforms in ubiquitous environments. UIC provides the capability to interact with a service implemented by CORBA, and also with the same service type implemented by Java RMI or SOAP. However, this platform provides a skeleton structure to only object-oriented request brokers; it offers only one single service discovery mechanism and provides no solution to the different heterogeneous service discovery.

ReMMoC [10] provides functions that support the adaptation of the service discovery protocol used by middleware, based on the service discovery infrastructure available in its current environment. It can interoperate with a range of middleware platforms and can discover services advertised using different service discovery protocol. It depends on a set of OpenORB components including service discovery components and service banding components. However, it can’t contain multiple service banding components operating in parallel, thus, it can’t support interactions with multiple heterogeneous services in parallel. During service discovery process, it uploads different discovery components by turns to monitor the environment continuously. As a result, this “Cycle and See” approach consumes too much resource of client application.

In this paper, we present a new reflective service coordination middleware based on mobile agents [11][13] to solve heterogeneous services coordination. This middleware uses mobile agents to discover and band the services distributed in open environments, thus, the client application avoids reconfiguring its structure and behavior frequently. By dynamically creating multiple subagents at run time, the middleware can support client applications interacting with multiple heterogeneous services in parallel. As a result the client burden is released and the efficiency of the coordination process is enhanced.

The paper is organized as following. Section 2 overviews the system framework. Section3 discusses the reconfiguration mechanism of mobile agents. Performance study is given in section 4. Finally, in section 5, we compare this middleware with other related works and present the conclusion.

2. System Overview

In this middleware (in Fig.1.), client application uses mobile agents to coordinate heterogeneous services in environments, so that to deduce its burden. By uploading different function components, mobile agents can discover and band the heterogeneous services in environments. When the mobile agent gets the results, it sends them back to client application. We call this type of mobile agents as coordination agents, as its main use is to interact with the services in distributed environments for client application. Function components are a set of well encapsulation entities, and each entity is a certain discovery protocol or banding protocol with only its interfaces exposed. Coordination agent reconfigures its behavior and structures by uploading different function components. The structure of coordination agent is not static at runtime. It can split itself into several subagents. On one side, with each subagent uploading a certain type of service discovery component, these agents can run in parallel to monitor the environments so as to search for the different types of heterogeneous service required by application as quick as possible to save the time and resource; On the other side, with each subagent uploading a certain type of
service banding component, the coordination agent can interact with multiple heterogeneous services in parallel.

A coordination agent consists of a function components list (FCL), a subagents list (AL), service management (SM) and the interfaces to interact with client application and subagents created by itself. The function components stored in coordination agent identify the agent's capability of discovering and interacting with the types of services in environments. Coordination agent can dynamically create several subagents at run time and record them in SL. The record item includes identifier of the subagent instance, the function components that the subagent carries, and its current physical location. The role of SM is to manage the instances of subagents, such as sending, receiving, and canceling. And these functions are fulfilled with the help of the Agent Server at local physical nodes. Other functions of SM include selecting the best result from the subagents, interacting with the client terminal and transmitting function components to the subagents.

```java
public class CoordinationAgent implements Serializable, AgentInterface {
    private Vector Result; // store the results from the subagents
    private Vector Assignment; // the tasks it will fulfill
    private String Identity;
    private String canMelt; // This token shows if coordination agent wants to be melted
    private String canSplit; // This token shows if this agent can dynamically create subagents
    private Hashtable FCL; // The list of function coordination components
    private Vector SL; // The list of subagents that coordination agent has dynamically created.
    private Vector NodeAddress; // the physical node which the agent will migrate to
    private AgentServer AS; // the name of the agent server
    private int splitNum; // the number of subagents it will create;
}
```

Figure 1. The framework of middleware based on mobile

Figure 2. The structure of coordination
In open environments, there may be multiple services providing the similar service methods needed by client. Coordination agent can call these services to select the result which fits client’s requirements best. Here, coordination agent has two running schemes. One scheme is to call these services sequentially one by one. In this scheme, coordination agent migrates from one physical node to another for calling the services distributed on these nodes, and then selects the best result. The average cost of this scheme is as follows,

\[ \sum_{i=0}^{n-1} (T_{M}(L_{i}, L_{i+1}) + T_{RC}(Comp_{i+1}) + T_{S}(S_{i+1})) + T_{SEL}(SR) \] (1)

Here, the cost of coordination agent migrating from node \( L_{i-1} \) to node \( L_{i} \) is denoted as \( T_{M}(L_{i-1}, L_{i}) \); the cost of interacting with a service is denoted as \( T_{S}(S_{i}) \); \( T_{RC}(Comp_{i}) \) is the cost of coordination agent reconfiguring its internal structure by uploading a function component naming \( Comp_{i} \); \( T_{SEL}(SR) \) is the cost of agent selecting the best one from a set of results.

The other scheme is to create several subagents at runtime, thus, each subagent discovers and interacts with a certain type of service by uploading the type of service discovery protocol component and service banding protocol components. When the subagents come back, coordination agent selects from the results they carry back. The average of this scheme is as follows,

\[ T_{C}(n) + \max (T_{M}(L_{i}, L_{i}) + T_{S}(S_{i}) + T_{M}(L_{i}, L_{i})) + T_{SEL}(SR) \] (2)

Here, the cost of dynamically creating \( n \) subagents is denoted as \( T_{C}(n) \).

3. Reconfiguration mechanism based on mobile Agents

Computational reflection is the activity performed by a computational system when doing computation about its own computation [12]. A reflective system is a computing entity which reasons about itself in a causally connected way. The concept of causally connected is defined as, a system is said to be causally connected to its domain if the internal structures and the domain they represent are linked in such a way that if one of them changes, this leads to a corresponding effect upon the other. Reflective architecture gives the system the ability to dynamically reconfigure its structure and behavior at run time. Coordination agent fetches a function component from the component repository and dynamically uploads it into internal structure, it creates a component adapter for the component. This component adapter interacts with other ones substituting for the directly connection among function components, forming a connection graph of the function components.
Each component adapter has output interfaces to which other adapters can be attached. The output interfaces indicate the component methods this function component can provide. Service adapter translates the data produced by the function service into standard formats that can be used by other services. The component adapter also has input interface being attached to other adapters. The input interfaces indicate the service methods this component requires from other components. Component adapters are also responsible for transferring events across the inter-dependent components. Such common events are the failure of a component, internal reconfiguration or replacement of a function component.

For a component adaptor A (in fig.5), if it can use the service interfaces provided by another component adaptor B, then the two adaptors must be interface compatibility and behavior compatibility.

```java
public interface ComponentAdaptor {
    public void destroyComponentAdaptor();
    // specify that another component depends on this component
    public void addComponent(String componentName, ComponentAdaptor ca);
    // break the dependence
    public void deleteComponent(String componentName);
    // specify this component depends on other components.
    public void addToComponent(String componentName, ComponentAdaptor ca);
    // break the dependence.
    public void deleteToComponent(String componentName);
    // specify another component has sent an event here
    public void eventFromComponent(ComponentAdaptor ca, ComponentEvent e);
    // return a string containing the identifier of the component
    public String name();
    // returns a string containing a description of the component
    public String selfInfo();
    // return a list of the components that depend on this component
    public Vector listComponentAdaptors();
}
```

Figure 5. The interface of component adaptor

After giving a formal definition of the adaptor, we discuss interface compatibility and behavior compatibility in details.
Definition 1. A component with expansion of behavioral protocol can be defined as \( C = \langle I_C^p, I_C^r, A_C, L_C, P_C \rangle \), where:

- \( I_C^p \) is a set of service interface provided by \( C \). For any \( Ite_i \in I_C^p \), \( Ite_i \) is a service interface of \( C \);
- \( I_C^r \) is a set of request interface required by \( C \). For any \( Ite_i \in I_C^r \), \( Ite_i \) is a request interface of \( C \);
- \( A_C \) is a finite set of actions operated by \( C \), including the request, providing and internal actions sets, \( A_C^r, A_C^p \) and \( A_C^i \), respectively. And they are disjoint.
- \( L_C \) is a set of \( C \)'s object reference links, which express the connections between \( C \) and other components in the system. For any \( l_i \in L_C \), \( l_i = < Rlte, Plte, Ins, PL > \), where \( Rlte \in I_C^r \) is a request interface required by \( C \) linked to the others, \( Plte \) is a service interface provided by \( C \) to the component linked. \( PL \) is the location of \( C \)'s reference instance.
- \( P_C \) is the behavioral protocol of \( C \), which is formal defined by process algebra.

Here, \( P_C = (S_C, \Gamma_C) \), where \( S_C \) is a finite set of states of components interaction. And \( s_{\text{init}}, s_{\text{final}} \) are the initial and final states of \( P_C \), respectively. \( \Gamma_C \) is a finite set of state transitions and \( \Gamma_C \subseteq S_C \times A_C \times S_C \).

Definition 2. For interface compatibility, B provides at least the service interfaces methods required by A. And \( \forall \text{ipa}_1 \): input parameter of request interface of A, \( \text{ipa}_2 \): the corresponding parameter of service interface of B, then \( \text{ipa}_1 \) has the same type of \( \text{ipa}_2 \), or \( \text{ipa}_1 \) is a subtype of \( \text{ipa}_2 \). The results of A, B are interface compatibility if the result type in A is the same type of the result type in B, or a supertype of that in B.

Definition 3. For behavior compatibility, suppose two assembled components \( C_i = \langle I_{C_i}^p, I_{C_i}^r, A_{C_i}, L_{C_i}, P_{C_i} \rangle \), \( C_j = \langle I_{C_j}^p, I_{C_j}^r, A_{C_j}, L_{C_j}, P_{C_j} \rangle \) interact with each other through an interface \( Ite \), then \( C_i \) and \( C_j \) are behavioral compatibility on the interface \( Ite \) if and only if, for the interaction \( P_{C_i} \parallel_{Ite} P_{C_j} \), the initial states \((s_{\text{init}}^i, s_{\text{init}}^j)\) can reach the finial states \((s_{\text{final}}^i, s_{\text{final}}^j)\) through a set of actions \( \sigma \), such as \((s_{\text{init}}^i, s_{\text{init}}^j) \stackrel{\sigma}{\Rightarrow} (s_{\text{final}}^i, s_{\text{final}}^j)\), where \( \sigma \in (A_{C_i} \cup A_{C_j})^* \) and \( \sigma \uparrow_{Ite} x < > \).

4. Performance Study

We use IBM NetVistas to run Client Application, and use a set of Dell PowerEdge 1400SC servers to run as agent Servers. The heterogeneous services are also distributed on these servers with the connection of 100MB/S Ethernet network.

Figure 6 shows the cost of dynamically creating subagents. The creation includes dynamically creating an instance of subagent, setting the correlated parameters and sending them out. Figure 7 shows the cost of dynamically uploading function components. The process includes translating the byte codes into class file, creating the instances of the component and the corresponding component adaptor, setting the entry method and initialized parameters, and reconfiguring the internal structure of the coordination agent.
Figure 8 compares the service coordination in sequence style with the parallel style. As seen from the figure, when the number of service interacted with is below 3, parallel style has no advantage. That is because in this style, when the number of services coordinated is not large, the main cost is consumed to create the subagents and send them out, on the contrary the cost of interacting with services occupies only a little part of the overall. However, with the number of services coordinated increasing, the parallel scheme shows its high efficiency. While under the sequential scheme, the agent has to move from one node to another to visit the services one by one, and this migration process consumes much.

5. Conclusion

Recently, several reflective middleware have been proposed with the ability to reconfigure itself structure and behavior to support heterogeneous coordination in open
dynamic environments. We compare them with our mechanism based on mobile agents below:

<table>
<thead>
<tr>
<th></th>
<th>Granularity of reflection</th>
<th>Entity of reconfiguration</th>
<th>Supporting multiple Services coordination in parallel</th>
<th>Heterogeneous service discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReMMoC component</td>
<td>Client application</td>
<td>No</td>
<td>“cycle and see”</td>
<td></td>
</tr>
<tr>
<td>UIC component</td>
<td>Client application</td>
<td>No</td>
<td>Not support</td>
<td></td>
</tr>
<tr>
<td>Coordination Based on mobile agents</td>
<td>Coordination agents</td>
<td>Based on multiple subagents running concurrently</td>
<td>Based on multiple subagents running concurrently</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 Comparing with the related works

Seen from the table, heterogeneous services coordination based on mobile agents help client application avoid reconfiguring the structure and behavior of itself frequently, and as a result release the burden of client application. Also it has the ability of discovering and banding multiple heterogeneous services distributed in environments in parallel by dynamically creating subagents and uploading different types of function components, thus, adapts to the open environments well and shows its efficiency.

References

[10] Paul Grace, Gordon S. Blair and Sam Samuel. “ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability”. In Proceedings of International Symposium on Distributed Objects and applications (DOA), Catania, Sicily, Italy, November 2003


**Authors**

**Haiyang Hu** born in October, 1977, male, Ph.D., senior membership of China Computer Federation. He received BS, MS and Ph.D. in computer science and technology from Nanjing University in 2000, 2003 and 2006 respectively. His current research interests include mobile agents, software middleware, mobile computing.

**Hua Hu** born in November, 1964, male, Ph.D., Professor. His current research interests include autonomous computing, and pervasive computing.

**Yi Zhuang** is a recipient of IBM Ph.D Fellowship 2007-2008. He is currently an associate professor at the College of Computer & Information Engineering in Zhejiang Gongshang University. He obtained his PhD degree in computer science from Zhejiang University in Mar. 2008. Dr. Zhuang is currently a member of ACM, a member of ACM SIGMOD, a member of IEEE and member of CCF. He has served as PC chair and PC members in several international workshops, also served as external reviewers for some top technical journals and leading international conferences. His research interests include database systems, index techniques, parallel computing and multimedia retrieval and indexing, etc.

**Lianghuai Yang**, PhD., associate professor at Zhejiang University of Technology, his research interests include database system, data integration and data mining.