In Proceedings of 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000), Manchester, UK, April 2000, pp. 393-396.

Dynamic Application Integration Using Agent-Based Operational Administration

Junwei Cao, Darren J. Kerbyson, and Graham R. Nudd
Department of Computer Science, University of Warwick, U.K.
{junwei, djke, grn}@dcs.warwick.ac.uk
http://www.dcs.warwick.ac.uk/~hpsg/

Abstract

Operational Administration Agents (OAAs) can be used as the kernel of many practical application systems to implement integration functions [Cao99]. The dynamic application integration discussed in this work is essential for the OAAs to support openness, scalability and maintenance of the application system.

Dynamic application integration can be achieved by reconfiguration of multi-agent system. Traditional coordination mechanism, multicast communication, has obvious shortcomings if used in loosely coupled C/S system. In this work a new mechanism of using rule-based reasoning in a three-tier capability model is presented to solve the problems and improve the performance of agent coordination.

1 Introduction

Many practical application systems are basically distributed multi-server-multi-client systems. The coordination of different kinds of C/S computing system was summarized in [Adler95], which shows that an agent-based design can be used to coordinate client interactions with multiple services. An agent-based system is one in which the key abstraction used is that of an agent [Jennings98]. Operational Administration Agents (OAAs) are such an agent-based system designed to serve as matchmakers between the clients and servers [Cao99].

There are also some agent systems that support automatic mechanism for agents to find each other. In DPSS system [Brooks97], when an agent is started on the new host, it will inform all other agents about the new server on the host. Though multicast communication is very simple and easy to implement, it is not the most efficient way especially in loosely coupled C/S systems whose servers are only needed by a small group of the system end-users.

In this paper, a three-tier capability model used in the OAAs aims to maximize the efficiency of the system running normally. A simple rule-based reasoning is also added to each agent to support the special dynamic functions. The new mechanism can be exploited to support the dynamic reconfiguration and application integration much more efficiently than multicast in loosely coupled C/S system.

2 Operational Administration Agents

The OAAs architecture is illustrated in Figure 1. Each host has only one agent which is responsible to manage the local clients and servers of the applications. Agents have exactly the same structure, which consist of communication, control, and allocation layers. The agent receives the requests from both local clients and remote agents, and takes them as the same. The broker is the kernel of the OAAs system that is responsible to manage the global information. An application is divided into a client and a server. The server can be installed in one of the host and client in all of the hosts licensed. In fact, the server of one application may

require the information from the server of the other application and, thus, can be taken as a client of that application.

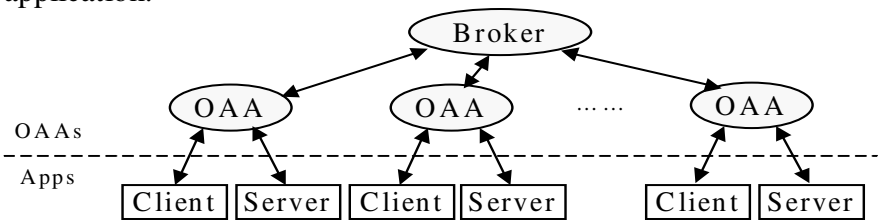

Figure 1. Architecture of OAAs

The most essential problem is how an agent coordinating with the other agents can find the target server for a request from a local client or a remote agent in the most efficient way (which will be discussed in Section 3), and in this way if the dynamic application integration can be supported (which will discussed in Section 4).

3 Three-Tier Capability Model

In order to coordinate the OAAs to find the servers, three kinds of Agent Capability Tables (ACTs) can be used in the system to record the details of the servers and their addresses, which include local ACT (L_ACT), global ACT (G_ACT), and local global ACT (LG_ACT).

- *L_ACT* is maintained by the local agent. Each agent uses one L_ACT to record the local server information. These servers can be taken as the capabilities of the agent. When a server is added to or removed from a host, the data file of the L_ACT will be modified to reflect the changes at once. This means that the L_ACT can record the real information about local servers all the time.
- *G_ACT* is maintained by the broker. It is actually a sum of L_ACTs. When one modification is made in one of the L_ACTs, the new copy of the L_ACT will be sent to the G_ACT at the same time and the same modification is made in the data file of the G_ACT as well. So the G_ACT can record the real information about all of the servers all the time. The contents of the G_ACT should be always consistent with those in the local agents.
- *LG_ACT* is maintained by the local agent dynamically. When an agent is started on a host, it will ask the broker for a copy of current G_ACT and store it in the memory. So the contents of a LG_ACT only reflect the real situation of the servers on the time the agent starts, and may or may not be true after that. When the agent is shut down, the LG_ACT is lost.

In *three-tier* model three kinds of the ACTs are all used. The LG_ACT can serve as a G_ACT locally, so no connections to the broker are needed when running normally. However, the dynamic integration of applications can not be supported because the coordination of agents in the three-tier model fully depends on the LG_ACTs, which can not be changed simultaneously when servers are added to or remove from the system and, thus, may become unreliable. This will be discussed in the next section.

4 Dynamic Application Integration

Simple rule-based reasoning can be added into the control layer of each agent to support dynamic application integration. When an agent can not find the server information in either L_ACT or LG_ACT, it detects that a change has occurred. It can ask the broker for help, and modify the LG_ACT. The idea will be described in a more formal way below.

4.1 Formal Representation

The formal representation is the basis for system behavior modeling using rule-based

reasoning, which is summarized in Table 1.

Components	H_i $(i=1, \cdots\cdots, n)$, *ith* host in a n-host system; A_i $(i=1, \cdots\cdots, n)$, the agent on H_i B, the broker of the agent system; C_i, one of the clients on H_i S_i, one of the servers on H_i; s, a given service request
Processes	$C_i(s)$, C_i sends the request s; $A_i(s)$, A_i processes the request s $B_i(s)$, B looks up the request s required by A_i
Evaluations	$l(s) \in \{T, F\}$, evaluation result of s in L_ACT T, within the local capabilities; F, beyond the local capabilities $lg(s) \in \{A_k(k=1, \cdots\cdots, n), null\}$, evaluation result of s in LG_ACT $g(s) \in \{A_k(k=1, \cdots\cdots, n), null\}$, evaluation result of s in G_ACT A_k means s is within the capabilities of H_k; *null* means no service s is available

Table 1. Formal Representation

4.2 Rule-Based Reasoning

We represent the process for a client to require a service in a logical way. The reasoning results according to the rules record the route for a request from a client to reach the target server when the server is reconfigured dynamically. Ten rules are used.

Rule 1. $C_i(s) \Rightarrow C_i \rightarrow A_i(s)$

One process always begins from a request sent from a client. This rule means that the client only sends the request to the local agent and waits the results. The symbol $\rightarrow$ is used to describe the route from one component to another.

Rule 2. $A_i(s) \Rightarrow A_i \rightarrow (l(s), lg(s))_i$

This rule means that when an agent receives a request, it will look up its own capabilities in L_ACT and LG_ACT.

Rule 3. $B_i(s) \Rightarrow B \rightarrow (g(s))_i$

The broker can receive requests from agents and check the server in G_ACT. Note that the $\rightarrow$ behind the B does not mean the broker will connect to the next agent, but that the A_i will connect to the next agent after asking for help from the broker.

Rule 4. $(T, *)_i \Rightarrow S_i$

This represents the rule that if the L_ACT shows that the service is within the local capabilities, one of the local servers is sure to be able to provide the service and the LG_ACT needs not be checked. The S_i is normally the end of a process.

Rule 5. $(F, A_k)_i \Rightarrow error \quad (if\ k = i)$

If L_ACT evaluation results show that the service can not be provided locally, the states of the LG_ACT must be checked. The rule shows if the LG_ACT records that the service is within the local capabilities, there must be a system error. As mentioned above, the LG_ACT should have been informed if a local server was changed.

Rule 6. $(F, A_k)_i \Rightarrow A_k(s) \quad (if\ k \neq i)$

If the LG_ACT shows that the service is on another host, the agent will transfer the request to that one no matter whether the information is true or fault.

Rule 7. $(F, null)_i \Rightarrow B_i(s)$

If the LG_ACT can not find the service information either, the agent will ask the broker for help.

Rule 8. $(A_k)_i \Rightarrow error \quad (if\ k = i)$

If G_ACT evaluation results show that the service can be provided locally by the source agent, there must be a system error, because the information in L_ACT and G_ACT, which should be always coherent, is now contradictory.

Rule 9. $(A_k)_i \Rightarrow A_k(s) \,/\, (F, null)_i \Rightarrow (F, A_k)_i \quad (if\ k \neq i)$

If the G_ACT shows that the service is on another host, the agent will transfer the request

to that one and add this information to LG_ACT.

Rule 10. $(null)_i \Rightarrow null$

If the G_ACT can not find the service information either, there must be no the server at all on the platform.

These rules can be organised together to give the route for a request from a client to the correspondent server through logical reasoning and represent all of the situations of the dynamic integration. In the next section, these are illustrated through a concrete case study.

4.3 A Case Study

The case study in this section shows how the dynamic integration is modeled exactly using simple reasoning. The typical situation studied is that C_1 on H_1 want to require service s from the server, which was just removed from the H_2 and added onto the H_3.

The process is shown below for the first time that C_1 asks for the service s after the server location was changed. For each step, those under the line show the evaluation results of all of the ACTs to the service s, which replace the correspondent parts, $(l(s), lg(s))_i$ and $(g(s))_i$, in the process automatically. The number on top of each $\Rightarrow$ indicates the rule used for the transformation.

$$\frac{C_1(s)}{(F,A_2\rangle_1,(F,null)_2,(T,*)_3,(A_3)_i} \overset{(1)}{\Rightarrow} \frac{C_1\rightarrow A_1(s)}{(F,A_2\rangle_1,(F,null)_2,(T,*)_3,(A_3)_i} \overset{(2)}{\Rightarrow} \frac{C_1\rightarrow A_1\rightarrow(F,A_2)_1}{(F,A_2\rangle_1,(F,null)_2,(T,*)_3,(A_3)_i} \overset{(6)}{\Rightarrow} \frac{C_1\rightarrow A_1\rightarrow A_2(s)}{(F,A_2\rangle_1,(F,null)_2,(T,*)_3,(A_3)_i} \overset{(2)}{\Rightarrow}$$

$$\frac{C_1\rightarrow A_1\rightarrow A_2\rightarrow(F,null)_2}{(F,A_2\rangle_1,(F,null)_2,(T,*)_3,(A_3)_i} \overset{(7)}{\Rightarrow} \frac{C_1\rightarrow A_1\rightarrow A_2\rightarrow B_2(s)}{(F,A_2\rangle_1,(F,null)_2,(T,*)_3,(A_3)_i} \overset{(3)}{\Rightarrow} \frac{C_1\rightarrow A_1\rightarrow A_2\rightarrow B\rightarrow(A_3)_2}{(F,A_2\rangle_1,(F,null)_2,(T,*)_3,(A_3)_i} \overset{(9)}{\Rightarrow} \frac{C_1\rightarrow A_1\rightarrow A_2\rightarrow B\rightarrow A_3(s)}{(F,A_2\rangle_1,(F,A_3\rangle_2,(T,*)_3,(A_3)_i} \overset{(2)}{\Rightarrow}$$

$$\frac{C_1\rightarrow A_1\rightarrow A_2\rightarrow B\rightarrow A_3\rightarrow(T,*)_3}{(F,A_2\rangle_1,(F,A_3\rangle_2,(T,*)_3,(A_3)_i} \overset{(4)}{\Rightarrow} \frac{C_1\rightarrow A_1\rightarrow A_2\rightarrow B\rightarrow A_3\rightarrow S_3}{(F,A_2\rangle_1,(F,A_3\rangle_2,(T,*)_3,(A_3)_i}$$

The final result $C_1 \rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow A_3 \rightarrow S_3$ shows that five connections are needed for the whole process. Though the server has been removed from the A_2, A_1 does not know and still transfer the request to A_2 according to its LG_ACT. A_2 has to ask for help from the broker and modify its LG_ACT. However, when C_1(in fact it can be any other clients) asks the service s for the second time, the broker is not needed and A_2 can transfer the request directly to A_3. The final result will become $C_1 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow S_3$, in which only four connections are used. Also after A_1 was restarted and its LG_ACT is now the same as the latest G_ACT. The request will not be transferred through A_2. Only three connections are needed for a remote service requirement, $C_1 \rightarrow A_1 \rightarrow A_3 \rightarrow S_3$.

5 Conclusions

In this work, a new agent coordination mechanism is presented to support dynamic application integration. The performance of multicast is mainly relative to the system scale n. The performance of the new mechanism is mainly relative to the server request frequency l and the agent life cycle T. So for the reconfiguration of loosely coupled systems, where $n >> lT$, the new mechanism should be more efficient.

References

[Adler95] R. M. Adler. Distributed Coordination Models for Client/Server Computing. IEEE Computer, 28(4), 1995, pp. 14-22.

[Brooks97] C. Brooks, B. Tierney, and W. Johnston. JAVA Agents for Distributed System Management. LBNL Report, 1997.

[Cao99] J. Cao, Y. Fan, and C. Wu. Research and Design of Operational Administration Agents. Computer Integrated Manufacturing Systems - CIMS, 5(3), 1999, pp. 39-43.

[Jennings98] N. R. Jennings and M. J. Wooldridge (eds). Agent Technology: Foundations, Applications, and Markets. Springer-Verlag, 1998.