Assignment Problem in Requirements Driven Agent Collaboration and its Implementation

Jian Tang
Academy of Mathematics and Systems Science
Beijing 100190, China
jtang@amss.ac.cn

Zhi Jin
Key Laboratory of High Confidence Software Technologies (MoE)
Peking University
Beijing 100871, China
zhijin@sei.pku.edu.cn

ABSTRACT

Requirements Driven Agent Collaboration (RDAC) is a mechanism where the self-interested service agents actively and autonomously search for the required services submitted by the request agents and compete to offer the services. This mechanism would be more suitable for large number of autonomous service providers on internet compared with the current service-oriented computing framework. Collaboration is an important issue in this mechanism.

This paper focuses on the collaboration issue in RDAC. First, we define the assignment problem in RDAC and show that it is NP-complete. Then, we model it as a Kripke structure with normative systems on it. This makes it possible to bridge the assignment problem and the existing efforts in normative systems, games, mechanisms, etc. Thirdly, a negotiation-based approach is given to solve the problem and the performance of the negotiation is evaluated by simulation. Finally, a tool for RDAC has been implemented to put it into practice and for further testing and evaluation.

Categories and Subject Descriptors

I.2.11 [**ARTIFICIAL INTELLIGENCE**]: Distributed Artificial Intelligence -Intelligent agents, Multiagent system.

General Terms

Design; Theory

Keywords

Task Allocation, Normative Systems, Negotiation

1. INTRODUCTION

With the introduction of service-oriented computing [13], Web-based applications gain more and more attentions. A typical framework for Web-based applications is Service Oriented Architecture (SOA)[5]. It contains three primary parties. The service provider publishes service descriptions and provides the implementations for the services. The service consumer finds the service descriptions in a service registry and bind and invoke the services. The service broker provides and maintains the service registries. Here, service consumer is responsible to find and compose suitable services and bind and invoke them in a right order.

However, in many situation, it is quit difficult for service consumer to do the job for using and composing services distributed over Internet as the service consumer often has no intent, interest or knowledge on how its needs can be fulfilled. For example, a service consumer wants to have a PC. It may not be expected that any consumer wants to know that a PC consists of CPU, mainboard, hard disks and so on as well as how to assemble the components. On the other hand, in SOA, the provided services registered in service brokers can only wait for being discovered and invoked. The service providers have no way to take part in the service discovery and composition and express whether they are willing to supply the services based on for example the payoff. In one word, the autonomy of the two parties has been ignored.

In order to achieve the full power of service oriented computing, both the service providers and the service consumers should be understood as autonomous active computational entities. They can autonomously search for registered services and choose the roles that they will play in the collaboration for invoking and/or delivering a service so that the interactions between them can be established dynamically, and connected flexibly. Following this principle, automated discovery, dynamic coordination, and autonomously collaboration of Web services become very hot issues[15].

Requirement Driven Agent Collaboration (RDAC) [17] has been proposed to be an approach for dealing with the autonomy issue. In RDAC, both service providers and service consumers are treated as agents. The service consumer needs not to know how to fulfill its service requests (i.e. the request agents) but publishes them in somewhere on the Internet. The provided services (i.e. the service agents) will actively discover those service requests to which they can contribute. If there are two or more service agents who can provide the same service, they compete to be selected. If a service request can not be satisfied by one service agent, several service agents will form a coalition to supply the service.

RDAC includes three parts: the knowledge part, the aggregation part, and the collaboration part. The knowledge part (mainly an function ontology FO) provides the shared knowledge among all service agents and request agents. In

Cite as: Assignment Problem in Requirement Driven Agent Collaboration and its Implementation, J. Tang and Z. Jin, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 839-846

the aggregation part, based on the knowledge in FO, service agents search for the request agents to which they can contribute even partially. In this way, service agents aggregate around the request agents with a 'contribute-to' relation and a 'expecting-repay' relation. Finally, in the collaboration part, these service agents and request agents collaborate with each other to work out a solution that satisfies all agents of both sides, i.e. the request agents consume the expected services and the service agents obtain the expected repay.

The main issue in the collaboration part is in fact a task allocation problem[14]. But in RDAC setting, this is quite challenge as both the request agents and the service agents are autonomous and self-interested. We can not get the solution by only maximizing the assigner's benefit but should by maximizing the agents' total welfare under the condition that everyone is satisfied. [12] proposed a multiagent approach for finding a solution to the task selection problem. This approach is based on an architecture and programming model in which agents represent applications and services. However, the agents in this work are considered as passively waiting for being chosen, which lacks of the interactions between agents such as competing, negotiating, requesting and so on.

Also, there are many other related works. However, most of them ignore the privacy of agents and are based on centralized settings. For example, [9] develops a protocol that enables agents to negotiate and form coalitions. It assumes that the agent knows the capabilities of all others. And the proposed protocol is centralized where one manager is responsible for allocating the tasks. [10] provides the possibilities of achieving efficient allocations in both cooperative and non-cooperative settings. They propose an algorithm to find the optimal solution, but it is also centralized.

This paper aims to give a negotiation-based solution for the task allocation problem among self-interested agents on the decentralized setting. First of all, we give a definition of the assignement problem in mathematical form and show that it is NP-complete. Then we define its model based on a Kripke structure with normative systems. That bridges the problem with the current efforts on normative systems, games, mechanisms, etc. After that, a negotiation-based approach is given. Experiments shows that the performance of the negotiation is quite satisfiable.

This paper is organized as follows. In section 2, we give the concepts and process in RDAC, and then figure out the assignment problem. Section 3 models the problem as a Kripke structure with normative systems defined on it. Section 4 defines a negotiation process for the task assignment problem. The simulation results are also included in this section. Section 5 designs an execution tool for RDAC. Finally, we discuss the related work in section 6 and conclude the paper in section 7.

2. ASSIGNMENT PROBLEM IN RDAC

With the decentralized settings in RDAC, both service agents and the request agents need to understand each other for obtaining a collaboration. An ontology, i.e. the Function Ontology (FO), has been built for this purpose. By using the terminology in FO, service agents can understand the required functions of the request agents and request agents can know the capabilities of the service agents.

FO not only makes agents understand each other, but also provides knowledge on to achieve a composed function by using a set of elementary functions by using a function decomposition tree. When a service consumer submits a service request, FO produces a function decomposition tree and asks service agents to make contribution. Any service agent then examines the function decomposition tree and finds the elementary functions that it is capable to do. In this way, around a function decomposition tree, candidate service agents aggregate together to form a coalition. Normally, such a coalition is not unique and one stable coalition needs to be chosen among them to be the realization body for satisfying the needs of the request agent. Some kind of mechanisms are needed to ensure a stable coalition.

After obtaining a stable coalition, for implementing the required function, each elementary function need to be assigned to a competent service agent in the coalition. As agents are self-interested, every agent wants to get more functions for gaining more payoff. But, there is not an agent which can satisfy the request agent by its own. The agents in a coalition have to collaborate. We assume that every service agent has a minimum expected payoff, known as reservation payoff in Economics. A service agent will quit the coalition if its reservation payoff can not be satisfied.

Then, the question is how to assign the elementary functions to the service agents of a coalition so that the assignment can satisfy all the service agents' reservation payoff and in it each service agent can get more than in other coalitions? Here, we consider "better" to be the service agent's total welfare, i.e. the satisfaction degree that will be defined below.

On the other hand, like the service agents, any request agent has also its reservation payoff that can be defined as the minimum expectant quality of provided services. Inspired by Maximilien's OoS Ontology [11] which includes a set of quality evaluation items, such as the responding time, the throughput, the latency, the load balance and so on, we introduce a quality evaluation function to compute the quality of agent A fulfilling function F. If there are n quality items which are on an unique evaluation interval $[0, M], M \in \mathbb{R}$, the function is

$$Q(A, F) = \sum_{i=1}^{n} \omega_i \sigma_i$$

In which, $\sigma_i \in [0, M]$ is the evaluation value for the i^{th} quality item, ω_i ($\sum_{i=1}^{n} \omega_i = 1$) is the weight for the i^{th} quality item of the request agent.

Definition 1. (Assignment Problem) Let $C = \{A_1, A_2, \ldots, A_n\}$ be a stable coalition and $F = \{F_1, F_2, \ldots, F_m\}$ be a set of elementary functions that the coalition needs to implement. p_i ($\sum_{i=1}^{m} p_i = 1,\ 1 \leq i \leq m$) is the weight of F_i. A_i is capable of implementing F_j and the quality is $Q(A_i, F_j)$. Suppose $W_i \subset F$ is the set of elementary functions that A_i is capable of implementing and $T_i \subset W_i$ is the set of functions that are assigned to A_i. An assignment S is an n-dimensional vector $(T_1, T_2, \ldots, T_n)$ requiring $\{T_1, T_2, \ldots, T_n\}$ to be a set-partitioning of F.

Given $\overline{Q}$, the reservation payoff of the request agent, and $\overline{P_i}$ $(1 \leq i \leq n)$, the reservation payoff of each service agents, it is desired to find an assignment S^* which satisfies:

1. $\frac{Q(S^*)}{Q(C)} + \sum_{i=1}^{n} \frac{P_i(S^*)}{P_i(C)} = \max_S(\frac{Q(S)}{Q(C)} + \sum_{i=1}^{n} \frac{P_i(S)}{P_i(C)})$; and
2. $Q(S^*) \geqslant \overline{Q}, P_i(S^*) \geqslant \overline{P_i}, i = 1, 2, \ldots, n$

In which,

- $Q(S) = \sum_{i=1}^{n} \sum_{j \in T_i} p_j Q(A_i, F_j) \in [0, M]$ is the quality of assignment S;
- $P_i(S) = \frac{\sum_{j \in T_i} p_j Q(A_i, F_j)}{Q(S)} P(S)$ is the payment of A_i in assignment S;
- $P(S) = f(Q(S))$ is the payment function that the request agent is willing to pay for S;
- $Q(C) = \sum_{j=1}^{m} p_j \max_i Q(A_i, F_j)$ is the request agent's ideal assignment;
- $P_i(C) = \frac{\sum_{j \in W_i} p_j Q(A_i, F_j)}{Q(S)} P(S)$ is A_i's maximum payment in S.

Let $SDR(C, S) = \frac{Q(S)}{Q(C)}$ be the satisfaction degree of the request agent and $SDA_i(C, S) = \frac{P_i(S)}{P_i(C)}$ the satisfaction degree of service agent A_i, the assignment problem in RDAC is to find out an assignment which maximizes both the service agents' and the request agent's total satisfaction degree and satisfies their respective reservation payoff.

THEOREM 1. *The complexity of the assignment problem in RDAC is NP-complete.*

PROOF. Consider a special case, i.e. let $n = 2$,

$$W_1 = \{F_1, F_3, F_4, \ldots, F_m\},$$

$$W_2 = \{F_2, F_3, F_4, \ldots, F_m\},$$

$$Q(A_1, F_j) = Q(A_2, F_j), (j = 3, 4, \ldots, m).$$

For any S,

$$Q(S) = p_1 Q(A_1, F_1) + p_2 Q(A_2, F_2) + \sum_{j=3}^{m} p_j Q(F_j)$$

is constant. Let

$$\overline{P_1} = (p_1 Q(A_1, F_1) + \frac{1}{2} \sum_{j=3}^{m} p_j Q(F_j)) \frac{f(Q(S))}{Q(S)}$$

$$\overline{P_2} = (p_1 Q(A_2, F_2) + \frac{1}{2} \sum_{j=3}^{m} p_j Q(F_j)) \frac{f(Q(S))}{Q(S)}$$

We are finding an assignment S^* such that $P_i(S^*) \geq \overline{P_i}$. Notice that, $P_1(S^*) + P_2(S^*) = \overline{P_1} + \overline{P_2} = f(Q(S))$. This is equivalent to the partition problem which is an NP-complete problem. □

Transforming it into a 0-1 integer programming problem, this problem can be solved with some optimization tools such as Lingo etc. However, some obstacles frustrate to use these tools directly. First, its time complexity is very high. Second, some necessary information about the agents may not be available because of the privacy. For example, the agents may not want to disclose their reservation payoff as they may lose some possible benefits if others know the information. They are likely willing to keep their reservation payoffs private. Finally, in practice, it is difficult to determine some variable's values. Again, we take reservation payoff for example. A service agent can tell whether to accept or refuse a payoff but not the exact value of its reservation payoff. In fact, any service agent is always trying to make its payoff as much as possible.

3. NORMATIVE SYSTEMS

Since we treat both the service provider and the service requester as agents, we take the advantage of technologies in multiagent systems. Normative systems has been considered to be a highly influential approach to coordination in the area of multiagent systems[3]. And some other technologies are helpful to modeling the assignment problem, such as the Kripke Structures [6] and the Computation Tree Logic (CTL)[6].

For defining the Kripke structure of the assignment problem in RDAC, we first define the state in it. A state s of the assignment problem in RDAC is an m-dimensional vector $(\rho_1, \rho_2, \ldots, \rho_m)$, where $\rho_i \in \{1, 2, \ldots, n\}$ indicates that function F_i is assigned to agent A_{ρ_i}. Then the Kripke structure of the assignment problem in RDAC is defined as a 6-tuple:

$$< S, S^0, R, A, \alpha, V >$$

where,

- S is a finite, non-empty state set;
- $S^0 \subseteq S$ is an initial state set;
- $R \subseteq S \times S$ is a total binary relation on S, which is called the transition relation;
- $A = \{1, 2, \ldots, n\}$ is a set of agents;
- $\alpha : R \to A$ labels each transition in R with an agent;
- $V : S \to 2^{\Phi}$ labels each state with the set of propositional variables that are true in this state.

Based on the Kripke structure, the semantics of the assignment problem in RDAC and some restrictions can be given.

$S = \{(\rho_1, \rho_2, \ldots, \rho_m) | F_i \in W_{\rho_i}, i = 1, 2, \ldots, m\}$, i.e. any state in S is an assignment.

$S^0 = \{(\rho_1, \rho_2, \ldots, \rho_m) | Q(A_{\rho_i}, F_i) = \max_j Q(A_j, F_i)\}$, i.e. any initial state in S^0 is the state that every elementary function is assigned to the service agent which has the best implementation quality.

Let $((\rho_1, \rho_2, \ldots, \rho_m), (\rho_1^{'}, \rho_2^{'}, \ldots, \rho_m^{'})) \in R$. If $\rho_i \neq \rho_i^{'}, (1 \leq i \leq m)$, agent ρ_i and agent $\rho_i^{'}$ compete for the i^{th} function. R contains only the transitions from which only one agent can compete for a function. For example, transition $((1, 2, 3), (2, 2, 3))$ means the state transits from "function 1 is assigned to agent 1" to "function 1 is assigned to agent 2"; transition $((1, 2, 3), (2, 3, 3))$ means that agent 1 and 2 are competing for function 1 meanwhile agent 3 is competing with agent 2 for function 2. The former is included in R, and the latter is not. That is just for simplification by allowing

a single agent to execute a single action in one state (interleaved concurrency). Going on with the example, α will map $((1,2,3),(2,2,3))$ to agent 2, and $((2,2,3),(1,2,3))$ to agent 1. So we conclude that a transition in R indicates an agent is requesting for carrying out more functions.

Furthermore, we use Computation Tree Logic (CTL) to express the objectives of the normative systems:

- $(E \bigcirc \varphi)$ on some path, φ is true next;
- $E(\varphi \mathcal{U} \psi)$ on some path, φ until ψ;
- $E(\diamond\varphi)$ on some path, eventually φ;
- $E(\Box\varphi)$ on some path, always φ;
- $A(\bigcirc\varphi)$ on all paths, φ is true next;
- $A(\varphi \mathcal{U} \psi)$ on all paths, φ until ψ;
- $A(\diamond\varphi)$ on all paths, eventually φ;
- $A(\Box\varphi)$ on all paths, always φ;

With these notations, for any $1 \leq i \leq n$, agent i's objective γ_i can be expressed as $A(\diamond\varphi)$, where φ is in the form of $s_1^i \vee s_2^i \vee \ldots$, here s_j^i $j = 1, 2 \ldots$ is the state. That means that the system will always get to one of the states s_1^i, s_2^i,... eventually. Under these states, i will get the payment that is more than its reservation payoff. Then the objective of our Kripke structure is $\gamma_1 \wedge \gamma_2 \wedge \ldots \wedge \gamma_n$. Fulfilling this objective needs constraints on agents' behaviors, i.e. on the transition relation. That can be modeled as a normative system. Formally, a normative system $\eta \subseteq R$ is defined in the context of a Kripke structure such that $R \backslash \eta$ is a total relation. That is, $(s, s^{'}) \in \eta$ means transition $(s, s^{'})$ is forbidden.

By providing a suitable normative system, the assignment problem in RDAC can be solved. This builds the bridge between RDAC and the available works on normative systems, games, mechanisms, etc. With these works, some interesting issues like robustness [4] or applying power indices [2] can be introduced into RDAC. Next section will provide a normative system for the assignment problem in RDAC in a negotiation-based manner.

4. NEGOTIATION

This section presents a negotiation framework in which the service agents and the request agent negotiate to change the state of the system. We focus on the state transition in the Kripke structure, i.e. a state transition is viewed as a proposal by an agent. In the process of negotiation, one agent transits the state to another; another agent can accept or refuse it. If the proposal is accepted, the state transition happens, otherwise, the first agent proposes a new proposal for transiting the state to a new one. This process is showed as an UML sequence diagram in Figure 1.

In Figure 1, the request agent first choose a state to make proposal, meaning he wants the assignment of that state. It is the initial state of the Kripke structure, i.e. every elementary function is assigned to the service agent which has the best quality to implement the function. This is reasonable as any request agent wants to get the best service. Then the service agents evaluate the state to see if it is satisfiable. If a service agent accepts the state, it will do nothing; otherwise, it will transit the state. The decision on being "accepted"

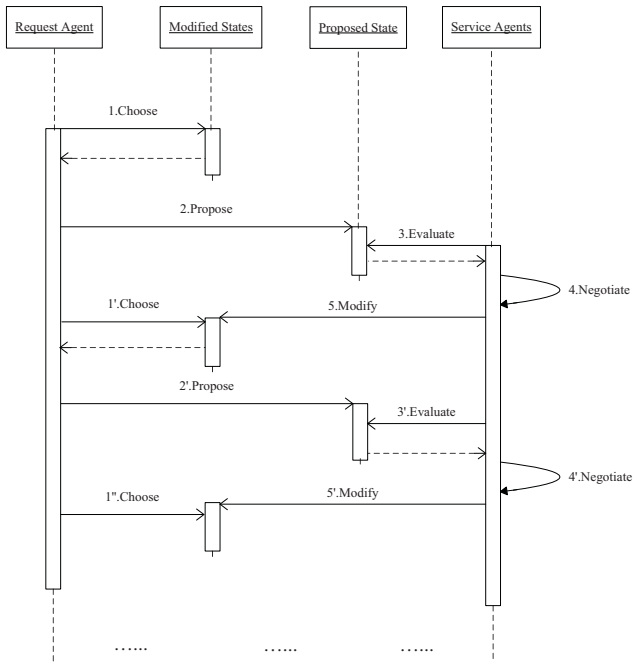

Figure 1: Process of Negotiation

or being "refused" is made based on the reservation payoff. Those states that satisfy the service agent's reservation payoff will be accepted but others will be refused.

After that, the service agents which refuse the proposed state should negotiate with others and then modify the states. They trigger transitions that are included in R. Any agent i, will not negotiate with those agents which are assigned functions that are not included in its capable set W_i. And they may work out several states that have already satisfied some of the agents but not all. Then, the request agent chooses one state that satisfies itself best (with the greatest total quality) and after that the request agent proposes it to the service agents. This proposed state starts a new round of negotiation.

Three possible situations will terminate the negotiation.

- if all service agents accept the request agent's proposal in some round, the negotiation ends with success.
- if the service agents could not make a valid transition when it negotiates with others, the negotiation ends with failure
- if the request agent can not make a choice in the modified states, i.e. all modified states worked out by the service agents can not satisfy the request agent's reservation payoff, the negotiation ends with failure.

We simulate the negotiation process to evaluate the negotiation. We set a numerical experiment that includes 4 groups of different service agents and elementary functions. Randomly generating the information of the service agents and the request agent (i.e. their reservation payoff), each group is simulated for 100 times. Take the group with 10 service agents and 20 functions for example, the statistic data is given as follows.

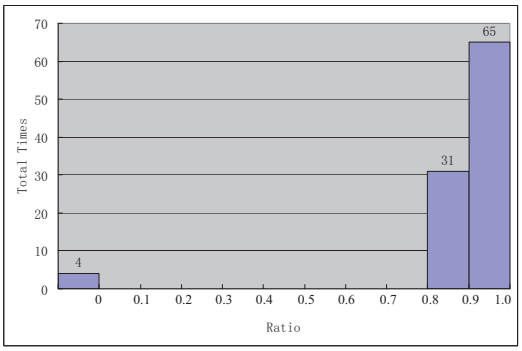

Figure 2: Negotiation to Optimal Result

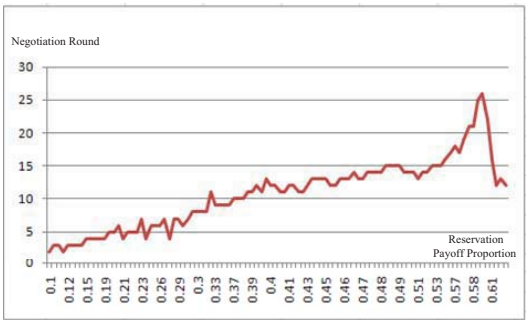

Figure 3: Relation between Proportion and Negotiation Round

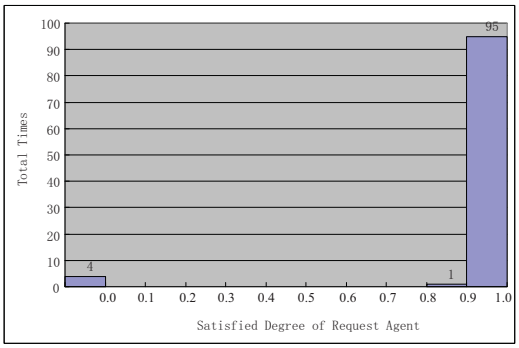

Figure 4: SD of Request Agent

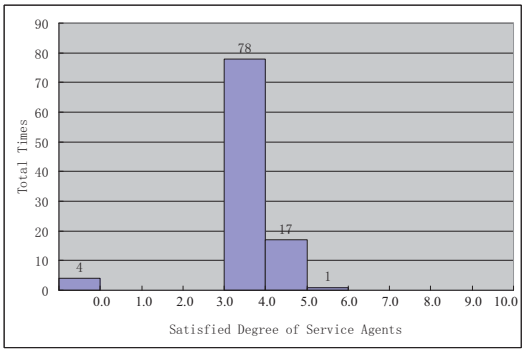

Figure 5: SD of Service Agents

First, in each simulations, optimal assignment exists. But four of them terminate without an assignment. Thus, the successful rate of negotiation is $(100-4)/100 = 96\%$.

Second, we compare the assignment got by negotiation with optimal one. Denote σ^* as the total satisfaction degree of assignment got by negotiation, and σ as the total satisfaction degree of optimal assignment. The ratio $\tau = \sigma^*/\sigma$ describes how close the negotiation assignment is to the optimal assignment. Figure 2 shows the statistic data of τ, in which x axis is the value of τ, and y axis is the number of negotiations corresponding to η. We can see that, in 96 successful negotiations, 65 of them have the value larger than 0.9.

Third, we want to know when the negotiation terminates, i.e. how many rounds does the negotiation take when it terminates. This mainly relates to average proportion of the agent's reservation payoff in its fully assigned total payoff. For example, if an agent's reservation payoff is 50 and it will get 100 if it is assigned all its capable functions, the proportion is 50%. Figure 3 illustrates the direct proportion relation between the proportion and negotiate round. In fact, agent with greater proportion is more likely to unsatisfied in an assignment, so more negotiation round is needed. This is well proved in the figure. The number of negotiation round rises as the proportion rises, and it becomes steeper when the proportion is higher. But after a steeper rise, it falls down very sharply. The reason is that when proportion is too large, agents require more payment so they can not be satisfied easily. So the agents will quickly understand that they could not get a deal, and negotiation breaks down, i.e. the number of negotiation round becomes smaller.

Next, Figure 4 shows the satisfaction degree of the request agent. There are 96 successful negotiations. In all of them, the request agent's satisfaction degree is higher than 0.8. Further, most of them (95 in 96) have the value higher than 0.9.

The service agent's satisfaction is a little complicated. Here we consider an "average" concept. Let's look at the setting of the simulation. In the setting, there are 10 service agents and 20 elementary functions and each agent can do averagely 6 functions. So the average number of functions that are assigned to an agent is approximately $20/10 = 2$. Then the average expect satisfaction degree could be calculated by $2/6 = 0.33$. So the total satisfaction degree of 10 service agents is $0.33 \times 10 = 3.3$. Figure 5 records the total satisfaction degree of 10 service agents.

According to the statistic data, among 96 negotiations that are successful, 20 of them have the total satisfaction degree of service agents lower than 3.3; 76 of them are higher than 3.3. This indicates the that service agents could frequently get higher total satisfaction degree than the expect average value.

Analysis of other three group is similar with the group with 10 service agents and 20 request agents. For comparison, the concrete statistics data of the 3 groups are given in appendix. We see that in all the group, the successful rate is more than 90%. All the properties discussed in the 10&20 group remain.

5. THE IMPLEMENTATION

In order to put the RDAC into practice, we design a tool

for it. It is implemented in Java language and is designed on basis of two frameworks: Protege [8] and JADE [7]. Protege, a free, open source ontology editor and knowledge-base framework, mainly helps us model, edit and manage the Function Ontology. An ontology agent is designed. It is in charge of ontology management by the help of protege. The request agents and the service agents communicate with the ontology agent to get information on the function decomposition. Then the published service functions can be understood by the service agents. Protege ontologies can be exported into a variety of formats including RDF(S), OWL, and XML schema. Here in the tool, OWL is selected.

JADE (Java Agent DEvelopment Framework) is a powerful software framework that simplifies the implementation of multiagent systems through a middle-ware that claims to comply with the FIPA specifications and through a set of tools that supports the debugging and deployment phase. The agent platform in JADE can be distributed across machines even with different OS, and this provides convenience for service providers distributed on the Internet and makes it possible to realize communications among different kinds of providers. We choose JADE as the platform of RDAC tool, because it shares advantages that better suit our need.

Agent class in JADE fully complies with FIPA specification. It represents a common base class for user defined agents where service provided by an agent can be implemented as one or more behaviors. A scheduler, which is fulfilled by JADE, automatically manages the scheduling of behaviors. Therefore, we design service agent to extend the base Agent class in JADE. This implies the inheritance of features to accomplish basic interactions with the agent platform and a basic set of methods that can be called to implement the custom behavior of the agent (e.g. send/receive message, use standard interaction protocols, etc.). It is also very convenient for us to introduce our designed agent by figure 6 and-8, which are derived from Agent life-cycle as defined by FIPA. A agent can be in one of several states, according to Agent Life Cycle in FIPA specification where:

- INITIATED: the agent is created with all needed information but hasn't registered itself yet. In this stage, the agent can not do any behavior and even has neither a name nor an address.
- ACTIVE: the agent has registered itself and has all features that the agent should have in JADE. This is the main stage of the life cycle.
- Transit: in this stage, the agent might migrate to a new location. But it continues sending and receiving messages at the new location.
- WAITING: the agent is blocked and will be waked up when a certain condition is met. Typically, the agent is often waked up when a message sent by another agent arrives.
- SUSPENDED: the internal thread of the agent is suspended and no behavior is being executed in this stage.

There are three kind of agents: Service Agent (SA), Request Agent (RA) and Ontology Agent (OA). Figure 6 is the life cycle of service agent containing detailed state transitions in each stage. First, the service agent is initiated. In this stage, the service agent is initialed with its capable functions, reservation payoff, and strategy for negotiating. Then

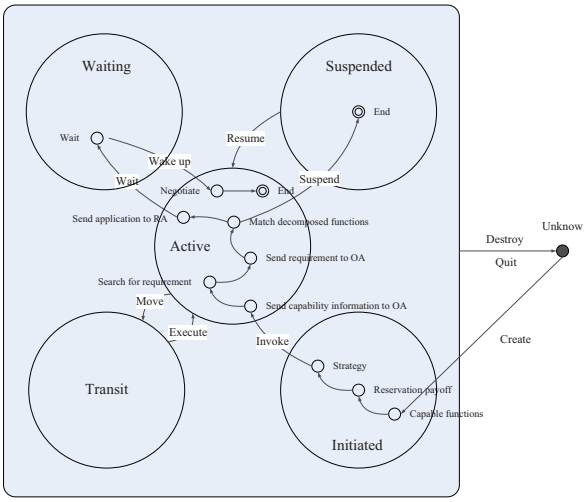

Figure 6: Service Agent Life Cycle

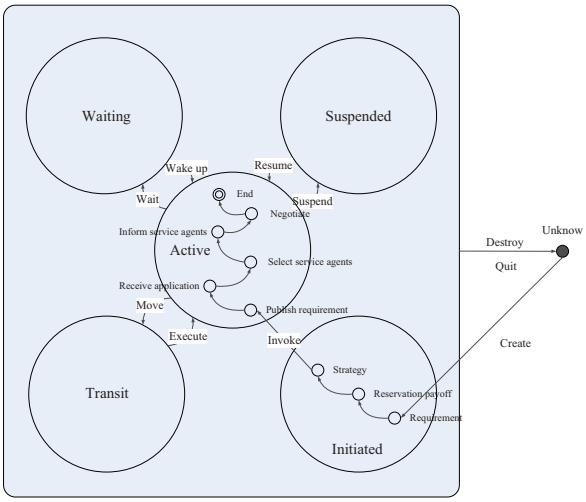

Figure 7: Request Agent Life Cycle

It is invoked to be active. In this stage, the service agent first sends its capability information to the ontology agent in order to get ontology information on its providing services. Then the service agent searches for the service request on the internet. After that, it sends the newly obtained request to the ontology agent, and then gets the function decomposition of request. Next, it matches the elementary functions in this decomposition with its capable functions' description to see whether it can provide services for the request. If no, it is then suspended; if yes, it sends the application to the request agent and goes to waiting stage to wait for being waked up by the request agent. When backing to active stage, the service agent negotiates with the request agent. After negotiation terminating, the service agent ends and is destroyed.

Similar to the service agent, the life cycle of the request agent is designed as showed in Figure 7. The request agent is initialed with its request, reservation payoff and strategy for negotiating. When invoked to active stage, it publishes its request and then will receive applications from the service agents. Then the request agent selects preferable ser-

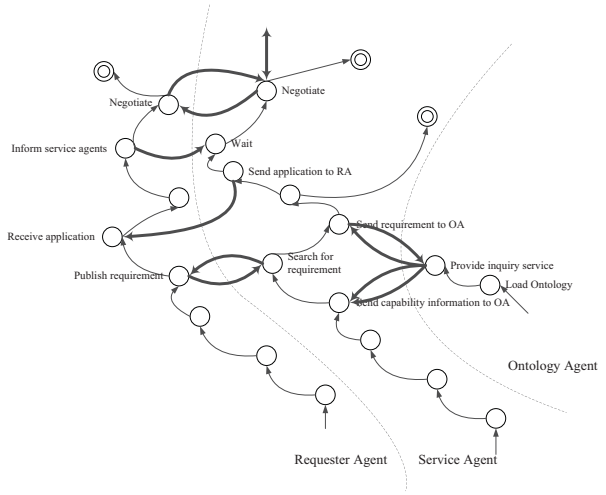

Figure 8: Overview of agents and communications

vice agents and informs them. Finally, the request agent negotiates with the service agents to assign the elementary functions.

The ontology agent is designed for providing knowledge for the service agents and the request agents. Function Ontology [17][16] is used. The ontology agent includes only two states. First, the ontology agent is initialed with the function decomposition patterns. Second, the ontology agent provides the inquiry services to the service agents and the request agents.

Figure 8 shows the overview of the communication among these three kinds of participants. The service agents, the request agents and the ontology agent are connected. They communicate with each other by sending messages. In the figure, we use dotted line to separate three kinds of agents. The communications (messages) between them are drew as bold arrows. There are many service agents in the system, and we draw one of them only. The double-head bold arrow on the top of the figure indicates the communication between two agents.

In order to demonstrate the process, we first run the application with an example of 10 service agents and a request submitted by a request agent *OnlineTest*. The main interface is shown in Figure 9 which includes two parts. The first is the platform that displays all the agents (at the left side) and records all information on agents' communication. The second is Sniffer Agent who is designed for watching the communication in an easy way.

Second, we use Dummy Agent, which is designed for monitoring the process (usually used as a debugging tool when programming), to start the RDAC process by letting Dummy Agent send ACL message "OnlineTest" to service agents (Figure 10).

Finally, Figure 11 shows the communications among all the agents in a whole process of RDAC. By double clicking corresponding line, detail information on message can be viewed.

6. RELATED WORK

Task allocation in multiagent systems has been investigated in recent years with different assumptions and em-

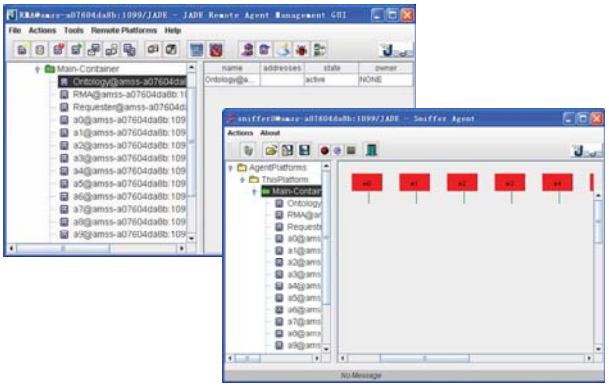

Figure 9: Main Interface

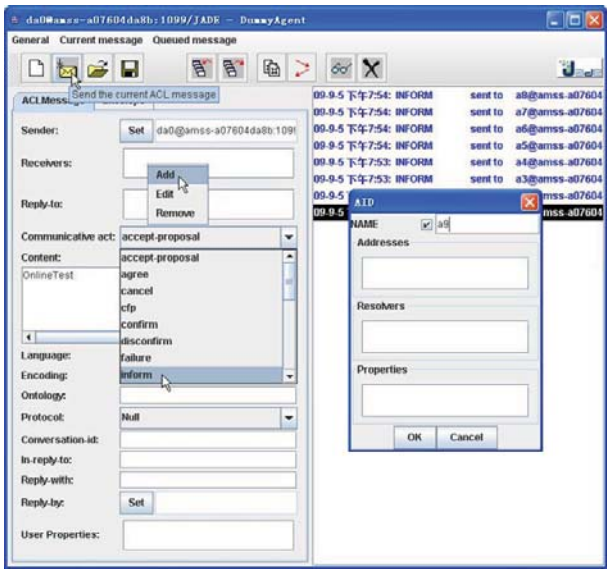

Figure 10: Start the RDAC

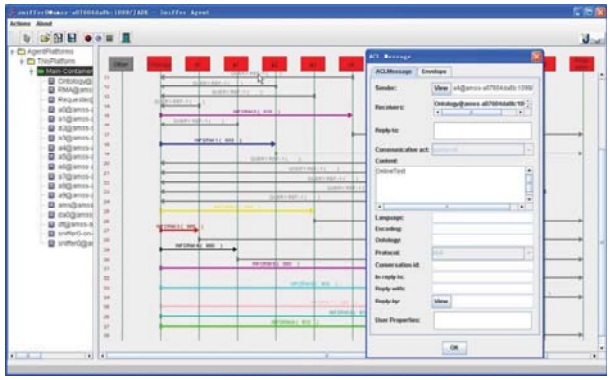

Figure 11: Communications

phases. However, most of the them ignore the privacy of agents, and study the problem in a centralized setting. For example, [9] develop a protocol that enables agents to negotiate and form coalitions. It assumes that the agent has the capability information of all others. Also the proposed protocol is centralized where one manager is responsible for allocating the tasks. [10] provides the possibilities of achieving efficient allocations in both cooperative and non-cooperative settings. They propose an algorithm to find the optimal solution, but it is centralized. There are some works relates the problem with web services. [1] gives mediators who receive the task and have connections to other agents. They break up the task into subtasks, and negotiate with other agents to obtain commitments to execute these subtasks. Different with this work, we have need no mediator so we should focus on modeling the whole agent system and negotiation process but not just the decision process of just a single mediator; also the agents gain more flexibility in our setting. A typical work is [12]. The agents in this work are selected according to the trust values of them and the selection is centralized.

Normative systems is a hot topic in multiagent systems area recently. It is a useful tool to coordination in multiagent systems. [3] gives the basic concepts and assumptions of it. Some more results are given such as robustness, applying power indices, etc. These results will help us for the future work of strategy of agents.

7. CONCLUSIONS

In RDAC, the service agents actively search for service requests and then match themselves with the elementary functions in the requests. In this mechanism, on one hand, service providers no longer has to understand the request. On the other hand, it need not an service agency in SOC to give a center control over or manage provided services.

The assignment problem of RDAC is defined as finding out an assignment that can satisfy all the agents' reservation payoffs. This is an NP-complete problem. We model it as a Kripke structure with a normative system on it. This builds the bridge between RDAC and the available works on normative systems, games, mechanisms, etc. With these works, some interesting issues like robustness [4] or applying power indices [2] can be introduced into RDAC.

We give a negotiation framework to allow service agents to negotiate with each other. The negotiation is proved to have good properties by simulation. The negotiation also helps us to exert special more effective normative systems on the Kripke structure. That is within our future work.

8. ACKNOWLEDGMENTS

This work was supported financially by the National Natural Science Fund for Distinguished Young Scholars of China (Grant No.60625204) the National Basic Research and Development 973 Program (Grant No. 2009CB320701), and the Key Projects of National Natural Science Foundation of China (Grant Nos. 90818026, 60736015). Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

9. REFERENCES

[1] S. Abdallah and V. Lesser. Modeling task allocation using a decision theoretic model. In *Proceedings of the fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2005)*, pages 719–726, 2005.

[2] T. Agotnes, W. van der Hoek, M. Tennenholtz, and M. Wooldridge. Power in normative systems. In *Proceedings of the eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2009)*, 2009.

[3] T. Agotnes, W. van der Hoek, and M. Wooldridge. Normative system games. In *Proceedings of the sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2007)*, 2007.

[4] T. Agotnes, W. van der Hoek, and M. Wooldridge. Robust normative systems. In *Proceedings of the seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2008)*, 2008.

[5] A. Arsanjani. Service-oriented modelling and architecture: How to identify, specify and realize services for your soa. *IBM developerWorks, IBM Corporation*, http://www.ibm.com/developerworks/library/ws-soa-design1/, 2004.

[6] E. A. Emerson. *Temporal and modal logic*. MIT Press, Cambridge, MA, USA, 1991.

[7] http://jade.tilab.com/.

[8] http://protege.stanford.edu/.

[9] S. Kraus, O. Shehory, and G. Taase. Coalition formation with uncertain heterogeneous information. In *Proceedings of the second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, pages 1–8, 2003.

[10] E. Manisterski, E. David, S. Kraus, and N. R. Jennings. Forming efficient agent groups for completing complex tasks. In *Proceedings of the fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2006)*, pages 834–841, 2006.

[11] E. Maximilien and M. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8,Issues 5,:84–93, Sept-Oct 2004.

[12] E. M. Maximilien and M. P. Singh. Toward autonomic web services trust and selection. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 212–221, 2004.

[13] M. P. Papazoglou. Service-oriented computating: Concepts, characteristics and directions. In *4th International Conference on Web Information Systems Engineering*, pages 3–12, December 3-12 2003.

[14] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101,Issues 1-2,:165–200, May 2005.

[15] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1(1):27–46, September 2003.

[16] J. Tang, L. Zheng, and Z. Jin. Web services composing by multiagent composing. *Journal of Systems Science and Complexity*, 21(4):597–608, December 2008.

[17] L. Zheng and Z. Jin. Requirement driven agent collaboration. In *Proceedings of the 2007 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2007*, pages 446–448, May 2007.