7

Requirements Driven Service Agent Collaboration

Liwei Zheng, Jian Tang and Zhi Jin
Academy of Mathematics and Systems Science, CAS
China

1. Introduction

Composition of Web services has received many attentions. On the one side the business world has developed a number of XML-based standards to specify Web services and the flow-based composition of Web services, such as WSDL (Christensen, 2001) and BPEL4WS (BPEL, 2007). On the other side, the semantic web community focuses on reasoning about web resources by explicitly declaring the preconditions and effects of Web services with terms precisely defined in ontologies, e.g. the Resource Description Format (RDF), and OWL-S (OWL-S, 2004) etc. Based on these, many efforts have been done for retrieving, discovering, and composing Web services. However, most of efforts assume that the Web services are passive computation entities. They are published and recorded as entries in the service registration centres managed by Service Agency (Kreger, 2001). They just wait there for service requesters, normally a human at present, to discover and invoke them. Some work (Paolucci & Sycara, 2003; Roman & Lausen, 2005; Roman & Scicluna, 2005) on automatic discovery of Web services have also been done by using various matchmaking algorithms according to the similarity between inputs/outputs and even preconditions /effects of Web services (Paolucci et al., 2003).

However, what will go on if the service entities are active to become autonomous Web services, namely the service agents? In fact, there are already some works which have touched this topic. DAML-S (Ankoleka et al, 2002) gives the description that allows web service to connect and interact autonomously with little intervention from the programmers. Web service architecture (Ankolekar et al, 2002a; Ankolekar et al, 2002b) has also been proposed to take advantage of DAML-S description to support automatic discovery and interaction between web services. In our previous work (Zheng & Jin, 2007a; Zheng & Jin, 2007b), web services are considered as active entities distributed in Internet which can autonomously recognize the service requests and compete or cooperate with others for fulfilling the requirements. The principle behind these works is to assume that the web services are active rational agents.

An example of rational agents is BDI agents (Wooldridge, 2000). It is an agent which can take actions based on information and knowledge from its environment. The action a rational agent takes also depends on the estimated benefits and the chances of success of the actions. With this kind of rational agents as the carriers of Web services, the service computing world will become more active, autonomous and dynamical which can be figured out as follows. Service providers design service agents which normally have specific functionalities and deploy them onto Internet. These are available service agents. Service

Source: Multiagent Systems, Book edited by: Salman Ahmed and Mohd Noh Karsiti,
ISBN 978-3-902613-51-6, pp. 426, February 2009, I-Tech, Vienna, Austria

requesters submit their service requests, normally attached with payments, which are virtual and needed-to-be-fulfilled service agents with some desired functionalities. When there is a service request, service agents negotiate with the service request for deciding if they can make contribution and get benefits from the contribution. When a service request can be satisfied by a group of service agents, these service agents form a feasible coalition to fulfil the service request. That has been called on-demand collaboration of service agents (Zheng & Jin, 2007a; Zheng & Jin, 2007b).

This chapter presents the computing mechanism for on-demand collaboration of service agents. There are several functional parts in this computing mechanism. First of all, for allowing the negotiation among the service requests and the available service agents, function ontology has been designed to provide the sharable terminology and serve as the background knowledge for both the available service agents and the service requests. After the service requests and the available service agents make agreement on their contribution relationship, these service agents form candidate feasible coalitions based on the strategy of automated mechanism design (Sandholm, 2003) for fulfilling the service requests. Normally, several feasible coalitions can be formed for one service request. Then after a negotiation process between the service request and its candidate coalitions, one of them can be chosen out and a model of a multi-agent system consisting of the service agents in the chosen coalition can be derived as the specification of the service request.

The strategies adopted in our approach are reasonable and feasible. First of all, Ontology (Sycara & Paolucci, 2003) is often used for providing common knowledge in application domain. We use function ontology for allowing the service requests and the available service agents can understand each others' functionalities. So that they can match the required functions and the provided functions. In terms of the functional matchability between a service request and a set of service agents and the satisfiability of the service request, the set of service agents forms a coalition which can satisfy the service request. Secondly, automated mechanism design is a feasible technique for preventing agents from misreporting information and quitting an established coalition which would make the coalition unstable. That will guarantee those coalitions are feasible and stable. Thirdly, the service request needs to select a suitable coalition based on some criteria, such as the service quality. And on the other hand, the service agents need to decide what they are going to serve based on what they can earn for the contribution. A negotiation framework is designed for them to make the final decision and to build a solution for the service request.

The rest of this chapter is organized as follows. Section 2 presents the function ontology. In section 3, the coalition formation of service agents has been given. And the coalition stability has been analyzed, and how to generate coalition constraints based on automated mechanism design has been discussed. Section 4 formulates the solution selection as a negotiation process and gives a negotiation framework to generate a good enough solution. A case study is given in section 5 for illustrating the whole idea for the service agent coalition formation and the negotiation-based decision-making. Section 6 presents some related works and concludes the chapter.

2. Function ontology

2.1 Ontology concepts and associations

For enabling the requirements driven agent collaboration, service agents have to understand the service requests. ***Function Ontology*** is constructed for this purpose. This ontology gives

the terminology for describing the capability provided by service agents as well as the required capability submitted by service requesters. The Function Ontology is represented in a hierarchy of controllable ***resources*** together with the effects on them. For expressing the effects imposed on resources by service agents, concept *state* has been used. So the effect imposed on a resource by a service agent is characterized as the state change of the resource. These effects are caused by the *behaviours* of a service agent. Using the effect, the capability of service agents is grounded onto the state changes of the resources.

Resource. Generally speaking, anything that can be identified and can be operated by service agents is a kind of resource, e.g., information (value, date, etc.), physical entities (Hardware, people, etc.), and software systems and so on. All of these resources are described by a set of attributes, each of which features some aspect of one resource. The attributes of a resource are classified into static ones and dynamic ones. A static attribute is irrespective of time and keeps its value through its whole lifecycle. A dynamic attribute may change its value when some external behaviour happens.

Definition 2.1 (Resource) Resource is described as a 6-tuple:

$$Res:=\langle SAttr,DAttr,SARan,DARan,ValFuncofSAttr,ValFuncofDAttr\rangle$$

In which,

- $SAttr=\{sa_1,\ldots,sa_n\}$ is a finite set of static attributes of the resource;
- $DAttr=\{da_1,\ldots,da_n\}$ is a finite set of dynamic attributes of the resource;
- $SARan=\{sv_1,\ldots,sv_m\}$ is a finite set of static attribute values of the resource;
- $DARan=\{dv_1,\ldots,dv_m\}$ is a finite set of dynamic attribute values of the resource;
- $ValFuncofSAttr:SAttr\rightarrow SARan$ is a value function of static attributes; and
- $ValFuncofDAttr:DAttr\rightarrow P(DARan)$ is a value function of dynamic attributes. $P(DARan)$ is the power set of $DARan$.

For the convenience of description, we use '@' as a general separator which means 'of', e.g. '*sa*@*res*' means a static attribute *sa* of resource *res*, '*da*@*res*' means a dynamic attribute *da* of resource *res*;

State. A state of a resource is the attribute-value pair of its dynamic attribute. So we may also call the dynamic attributes the state attributes. The value of the dynamic attributes of a resource may change when some external event happens. That is the state change of a resource.

Definition 2.2 (State) Let *res* be a resource and $da_1,\ldots,da_n$ be dynamic attributes of *res*. The set of states of *res* is $state@res=(val_1,\ldots,val_n)$, in which, $val_i=ValFuncofDAttr(da_i),i=1,\ldots,n$.

Behaviour. Any possible state transition of a resource can be viewed as an effect imposed by an external behaviour. That is our essential idea by mapping behaviour to state changes.

Definition 2.3 (Behaviour) A behaviour is defined as a triple: $beh:=\langle res,s_0,s_1\rangle$, *res* is the resource operated by *beh*. $<s_0, s_1>$ is a state transition of *res*. That means that *beh* is a behaviour which makes *res* changing its state from s_0 to s_1.

If $beh_1=<res,s_1,s'_1>$ and $beh_2=<res,s_2,s'_2>$ and $s'_1= s_2$, then beh_1 and beh_2 are sequential, i.e. beh_1 is a direct precedent of beh_2, and beh_2 is a direct successor of beh_1

Function. Behaviours impose the finest-grain effects on resources. Functions could be fine- or course-grain effects. More importantly, functions can be used to claim patterns for decomposing a course-grain effect into a set of finer-grain effects (sub-functions). It tells how to decompose the function and which constraints should be followed when making the decomposition. Two function structures, i.e. the primitive function and the function

decomposition mode, have been used in our function ontology. These two are not exclusive. Using the primitive function structure means that any function can implemented by a set of behaviours, while using the function decomposition mode means some function can be decomposed into a set of sub-functions via the decomposition pattern. Function decomposition modes explicitly capture the hierarchy of function decomposition.

Definition 2.4 (Primitive Function) A primitive function is a 7-tuple:

$$Fun:=\langle Beh,Res,Cond,CondFuncofBeh,Prop,PropFuncofBeh,Dependency\rangle$$

In which,

- *Res* is a set of resources operated by the function;
- *Beh* is a set of behaviours which constitute the function;
- *Cond* is a set of the logic expressions on the resource states;
- $CondFuncofBeh:Beh\rightarrow P(Cond)$ maps behaviour into a set of conditions. It gives the invokable condition for each behaviour. *P(Cond)* is the power set of *Cond*;
- $Prop=\{p_1,p_2,...\}$ is a set of payoff distribution proportions;
- $PropFuncofBeh:Beh\rightarrow Prop$ is the proportion mapping function of behaviours so that $\sum_{i=1}^{|Beh|} p_i = 1$.
- Dependency:=⟨DirecBehDepen,CondBehDepen⟩ in which,
  - $DirecBehDepen:Beh\times Beh$ is a set of direct behaviour dependency relations. For any $beh_i,beh_j\in Beh$, $(beh_i,beh_j)\in DirecBehDepen$ iff beh_i is a successor of beh_i and $CondFuncofBeh(beh_j)=\varnothing$;
  - $CondBehDepen:Beh\times Beh$ is a set of conditional behaviour dependency relation. For any $beh_i,beh_j\in Beh$, $(beh_i,beh_j)\in CondBehDepen$ iff beh_j is a successor of beh_i and $CondFuncofBeh(beh_j)\neq\varnothing$.

Here, *Cond* is defined as a logic expression set of resource state. Suppose *r* is a resource, $state@res=(val_1,\ldots,val_n)$ is a logic expression of resource state. That means the current state of *r* is $(val_1,\ldots,val_n)$. Any composite logic expressions composed by logic connectors such as $\neg$, $\wedge$ and $\vee$ are also logic expressions of resource state.

Definition 2.5 (Function Decomposition Mode) A function decomposition mode of function *func* is described as a 6-tuple,

$$FuncDecMod(func):=\langle SubFuncs,Cond,CondFuncofSubFuncs,$$
$$Prop,PropFuncofSubFuncs,Dependency\rangle$$

In which,

- $SubFuncs=\{func_1,...,func_n\}$ is a set of functions, each $func_i(1\leq i\leq n)$ is a sub-function of *func*;
- *Cond* is a set of logic expressions on resource states of *func*;
- $CondFuncofSubFuncs:SubFuncs\rightarrow P(cond)$ maps a sub-function into a set of conditions. It gives the invokable conditions for each sub-functions;
- $Prop=\{p_1,p_2,...\}$ is a set of payoff distribution proportions;
- $PropFuncofSubFuncs:SubFuncs\rightarrow Prop$ is a proportion mapping function so that $\sum_{i=1}^{|SubFuncs|} p_i = 1$;
- $Dependency:=\langle DirecFuncDepen,CondFuncDepen\rangle$ in which,

- *DirecFuncDepen*:*SubFuncs*×*SubFuncs* is a set of direct function dependency relations. ($func_i$,$func_j$)∈*DirecFuncDepen* iff $func_j$ is a direct successor of $func_i$ and *CondFuncofSubFuncs*($func_j$)=∅;.
- *CondFuncDepen*:*SubFuncs*×*SubFuncs* is a set of conditional function dependency relations. ($func_i$,$func_j$)∈*CondFuncDepen* iff $func_j$ is the direct successor of $func_i$ and *CondFuncofSubFuncs*($func_j$)≠∅.

Table 1 summarizes the concepts and associations of the function ontology.

Concept class	Description	Super class
owl:thing	The root class.	without
Resource	The concept class of resource, including all the resource instances.	owl:thing
Attribute	The concept class of attribute.	owl:thing
Static attribute	The static attribute concept of resource.	Attribute
Dynamic attribute	The dynamic attribute concept of resource.	Attribute
State	A valid value of the dynamic attribute vector.	owl:thing
Behaviour	The concept for describing the state transition of resource.	owl:thing
Composite behaviour	The composition behaviour concept of two additive behaviours.	Behaviour
Function	The function concept, including all the function instances.	owl:thing
Sub-function	The sub-function concept.	Function
Function decomposition mode	The concept of function decomposition mode.	owl:thing
Execution condition	A group of logic expressions defined in resource states.	owl:thing
Execution condition of behaviour	The execution condition of behaviours in functions.	Execution condition
Execution condition of sub-function	The execution condition of sub-functions in function decomposition modes.	Execution condition
Payoff distribution proportion	The concept of payoff distribution proportion.	owl:thing
Payoff distribution proportion of behaviour	The Payoff distribution proportion concept of behaviours in functions.	Payoff distribution proportion
Payoff distribution proportion of sub-function	The Payoff distribution proportion concept of sub-functions in function decomposition modes.	Payoff distribution proportion

Table 1. Concept classes of function ontology

2.2 An example of function ontology in the domain of E-Learning

Function ontology for E-Learning domain introduced in CELF (CELF, 2005) has been developed as illustration. In this domain, an important resource is the question base. It

provides various question data in the learning process, and helps teachers or students finishing their teaching or learning tasks. The attributes of the question base include its name (a static attribute) and a state variable (a dynamic attribute). And its state can be opening, closing, reading or writing. The question base can be described as:

QuestionDatabase:={{databasename},{databasestate},{"QuestionData"},

{"open","read","write","close"},funSA,funDA}

That means that the name of *QuestionDatabase* is *QuestionData* and the value of the dynamic attribute could be *open, read, write,* or *close*, i.e. the state space of resource *QuestionDatabase* is {*Open,Read,Write,Close*}. When the value is *Open*, *QuestionDatabase* is in state "*open*".
Table 2 lists some resources of this domain and their state spaces.

Resource	State space
QuestionDatabase	{Open, Read, Write, Close}
QuestionBuffer	{NoQuestion, HasQuestions}
AnswerBuffer	{NoAnswer, HasAnswers}
StandardAnswerBuffer	{NoStandardAnswer, HasStandardAnswers}
CompareResultBuffer	{DataInvalid, ResultDataInitiated , HasResultData}
TestPointBuffer	{NoTestPointInfo, HasTestPointInfo}
TestPaper	{NoTestInfo, HasTestInfo, NotEvaluated, HasScore, Evaluated}

Table 2. Resources and their state space

Here we use behaviour *OpenDatabase* operated on resource *QuestionDatabase* as an example for showing the representation of the behaviour. This behaviour is to open a database, which will change the state of resource *QuestionDatabase* from *close* to *open*. According to the representation of behaviours, *OpenDatabase* can be described as:

OpenDatabase:=⟨QuestionDatabase,Close,Open⟩.

Table 3 gives some of the behaviours in the domain.

Behaviour	Resource	Starting state	Ending state
OpenDatabase	*QuestionDatabase*	*Close*	*Open*
ReadFromDatabase	*QuestionDatabase*	*Open*	*Read*
GetQuestions	*QestionBuffer*	NoQuestion	HasQuestions
CreateTestPaper	*TestPaper*	NoTestInfo	HasTestInfo
DisplayTestPaper	*Monitor*	*DisplaybuffEmpty*	*DisplaybuffNotEmpty*
GetStandardAnswer	*StandardAnswerBuffer*	NoStandardAnswer	HasStandardAnswers
GetAnswerFromUser	*KeyBoard*	*InputbuffEmpty*	*InputbuffNotEmpty*
RecordAnswer	*AnswerBuffer*	NoAnswer	HasAnswers
CompareAnswer	*CompareResultBuffer*	DataInvalid	HasResultData
CalculatePoint	*TestPointBuffer*	NoTestPointInfo	HasTestPointInfo
WritePointToPaper	*TestPaper*	*NotEvaluated*	*HasScore*
WriteCommentToPaper	*TestPaper*	*HasScore*	*Evaluated*
EvaluateTestPaper	*TestPaper*	*NotEvaluated*	*Evaluated*

Table 3. Behaviours and their state transitions

To illustrate the representation of the function, we use function *CreateTestPaper* as an example. Function *CreateTestPaper* gets question data from *QuestionDatabase*, and outputs question data in the form of test paper. It consists of two behaviours: *GetQuestion* and *GreateTestPaper*. In this function, each behaviour has a 50% payoff proportion. The invokable condition of *GetQuestion* is that *QuestionDatabase* is in state *Read*. The invokable condition of *CreateTestPaper* is that *QestionBuffer* is n state *HasQuestions*. According to the representation of function, *CreateTestPaper* can be described as follows.

$$CreateTestPaper:=\langle\{GetQuestions,CreateTestPaper\},\{QestionBuffer,TestPaper\},$$
$$\{Stateof(QuestionDatabase)=Read,Stateof(QestionBuffer)=HasQuestions\},$$
$$fun^{C},\{50\%,50\%\},fun^{P}\rangle.$$

Similar representation can be obtained for the function decomposition mode of function *GetQuestionPoint*. Function *GetQuestionPoint* obtains the point value of each question in a given test paper. It can be decomposed into two sub functions: *GetQuestionsFromPaper* and *GetDataFromDatabase.* The function decomposition mode of function *GetQuestionPoint* can be described as follows.

$$FuncDecMod(GetQuestionPoint):=\langle\{GetQuestionsFromPaper,GetDataFromDatabase\},$$
$$\{\},fun^{SC},\{40\%,60\%\},fun^{SF}\rangle.$$

3. Coalition formation

3.1 Coalition formation of service agents

A service requester submits a service request. To recognize the service request, the service agents have to understand the service request. Function Ontology provides the sharable terminology. A service requester should offer enough information about the request in the publication. These information includes the required functions, the payments which can be payed for the functions, and the assignment of the payments for the different functions. Formally, the structure of the service request can be given as follows.

Definition 3.1 (Service Request) A service request is described as a triple:

$$Req:=\langle Funcs,Payment,PayFuncofFuncs\rangle$$

in which,

- *Funcs* is a set of required functions;
- $Payment=\{p \mid p\in\mathbf{R}\}$ is a set of payments for the functions in *Funcs*; and
- $PayFuncofFuncs:Funcs\rightarrow Payment$ is the cost function of *Funcs*.

A service agent knows its provided functions and the prospective minimum payoff. And these information will be also needed by service requesters and other service agents. Accordingly, a service agent takes the following structure for self-description.

Definition 3.2 (Service Agent) Service Agent can be described as a triple:

$$SAgent:=\langle Beh,MinPay,MinPFuncofBeh\rangle$$

in which,

- *Beh* is a set of the behaviours which the service agent possesses;
- *MinPay* is a set of the minimum prospective payoffs of each behaviour in Beh; and

- $MinPayFuncofBeh:Beh \rightarrow MinPay$ is a minimum prospective payoff function of *Beh*.

As we have seen, the service requests and the service agents use the same terminology. That is the basis for service agents to understand the service requests. Service agents can match its behaviours with the functions' required by the service request.

Let *agt* be a service agent, *req* a service request and $func \in Funcs@req$ a required function of *req*. If $\exists beh \in Beh@func \cap Beh@agt$, then *agt* can contribute behaviour *beh* to *req*. If *agt* takes *beh* for *req*, it can get payment $Pay_{beh_k} = CostFuncofFuncs(func_j) \times PropFuncofBeh(beh_k)$. If there exists a set of service agents *C*, for $\forall beh \in Beh@func$, $\exists agt \in C$ so that $beh \in Beh@agt$, then we say that *func* can be satisfied by *C*, which is denoted by *cando*(*C*,*func*).

For a given service request, a **service agent coalition** will be established, that is driven by payoff which the service agent might obtain from the requester. We defined the coalition structure as follows.

Definition 3.3 (Feasible coalition for service request *req*)

Let *req* be a service request and *FC* a service agent coalition. If forall $func \in Funcs@req$, $\exists C \subseteq FC$ such that *cando*(*C*,*func*). Then *FC* is a feasible coalition for *req*.

To establish the feasible coalition is based on the prospective advantages of the service agents for the service request. With different function decomposition modes in function ontology, a service agent will receive different payoffs in different coalitions. Service agents need to decide in which coalition they can get more payoff so to decide which coalition they shoud take part in.

Let *req* be a service request and *FC* a feasible coalition of *req*. Let $agt \in FC$ be a service agent. The prospective payoff of *agt* in *FC* is

$$ProsPay_{agt} = \frac{1}{|FC|}\sum_{k=1}^{n} Pay_{beh_k}, beh_k \in FeasiBeh_{agt}$$

In which, $FeasiBeh_{agt}$ is a behaviour set for $\forall beh_j \in Beh@agt$, if $\exists func \in Funcs@req$ and $beh_j \in Beh@func$ then $beh_j \in FeasiBeh_{agt}$. $|FeasiBeh_{agt}| = n$.

As service agents can by themselves obtain the prospective payoffs they can get from different coalitions, they might "jump" from one coalition to another just for getting better payoff. For grarantteeing that the service request can be fulfilled, we need stable feasible coalitions in which all the agents have no "jumping" will.

Definition 3.4 (Stable feasible coalition) For an arbitrary feasible coalition *C* of service request *req*, *C* is a stable feasible coalition if:

1. $\forall agt \in C,\ Prospay_{agt} \geq \sum_{k=1}^{n} MinPFuncofBeh(beh_k), beh_k \in FeasiBeh_{agt}$;and
2. $\neg\exists C^*, C^*$ is a FC and if $agt \in C^*$, $Prospay_{agt}^{c^*} \geq Prospay_{agt}^{c}$

However, as the stability of coalition is just based on prospective payment, the final coalition could be unstable because there are not constraits for preventing service agents tell lies when forming the coalition which will leads to destroyable coalition. Automated Mechanism Design (Sandholm, 2003) gives solutions to this situation.

3.2 Automated mechanism design for feasible coalition

Mechanism design is the art of designing the mechanism (i.e., rules of the game) so that the service agents are motivated to tell their preferences truthfully and a desirable (according to

a given objective) outcome is chosen. In this paper, the objective is to select the solutions which are not destroyable for a given feasible coalition and guarantee the service agents telling truth.

Automated Mechanism Design (AMD) is an approach, where the mechanism is computationally created for the specific problem instance at hand. For conducting automated mechanism design for the coalition of service agents, we need:

- a finite set of outcomes *O*;
- a finite set of *N* service agents which are both in the same feasible coalition *FC* of a given service request *req*;
- for each service agent agt_i,
  - a finite set of types Θ_i, a probability distribution γ_i over Θ_i (in the case of correlated types, there is a single joint distribution Γ over $\Theta_1 \times \ldots \times \Theta_N$);
  - an utility function $u_i: \Theta_i \times O \rightarrow \mathbf{R}$;
  - an objective function whose expectation the designer wishes to maximize.

Further, we need to determine the outcome set and the type set of service agents.

Definition 3.5 (Type Set) Let FC be a coalition of service agents. The type set of a service agent in FC is a vector set whose elements are all the permutations of the functions the service agent can do for *FC*.

Definition 3.6 (Outcome Set) The outcome set is a vector set whose elements are vectors like $(FDM_1^1, \cdots, FDM_{i1}^1, FDM_1^2, \cdots, FDM_{i2}^2, \cdots, FDM_1^s, \cdots, FDM_{is}^s)$, $s \in \mathbf{N}$, satisfying the following two conditions:

- Each element of a vector is a function decomposition mode.
- Every Vector denotes one unique path to accomplish a function of *req*.

This AMD process generates deterministic mechanisms automatically. A deterministic mechanism consists of an outcome selection function $\mathbf{o}: \Theta_1 \times \ldots \times \Theta_N \rightarrow O$.

Two types of constraints are used in the AMD process, IR (individual rationality constraints) and IC (incentive compatibility constraints). IR constraints ensure that every service agent would gain its lower limit of payment at least. IC constraints are to ensure that the service agents will never misreport their type.

Ex interim IR is the IR constraints used in this paper. It means that the service agent would always participate if it knows only its own type, but not those of the others.

Definition 3.7 (Ex interim IR for a deterministic mechanism)

A deterministic mechanism is ex interim IR if for any service agent agt_i, and any type $\theta_i \in \Theta_1$, we have $E_{(\theta_1,\ldots,\theta_{i-1},\theta_{i+1},\ldots,\theta_n)|\theta_i}[u_i(\theta_i, o(\theta_1,\ldots,\theta_n)) - \pi_i(\theta_1,\ldots,\theta_n) - \delta] \geq 0$, in which $\delta = \sum MinPFuncofBeh(beh_k)$, $beh_k \in FeasiBeh_i$, $FeasiBeh_i$ is the behaviour set composed of all the effective behaviours of service agent agt_i in outcome $\mathbf{o}(\theta_1, \theta_2, \ldots, \theta_N)$.

The IC constraint used in this paper is based on Bayesian Nash equilibrium. A mechanism is said to implement its outcome and payment functions in Bayesian Nash equilibrium if truth telling is always optimal to any service agent when that service agent does not yet know anything about the other service agents' types, and the other service agents are telling the truth.

Definition 3.8 (IC based on Bayesian Nash equilibrium)

IC based on Bayesian Nash equilibrium is defined as: for any service agent agt_i, any type $\theta_i \in \Theta_1$, and any alternative type report $\hat{\theta}_i \in \Theta_1$, we have:

$$E_{(\theta_1,...,\theta_{i-1},\theta_{i+1},...,\theta_n)|\theta_i}[u_i(\theta_i,o(\theta_1,...,\theta_i,...,\theta_n))-\pi_i(\theta_1,...,\theta_i,...,\theta_n)]\geq$$
$$E_{(\theta_1,...,\theta_{i-1},\theta_{i+1},...,\theta_n)|\theta_i}[u_i(\theta_i,o(\theta_1,...,\hat{\theta}_i,...,\theta_n))-\pi_i(\theta_1,...,\hat{\theta}_i,...,\theta_n)]$$

This AMD process finds all the mechanisms which can satisfy all the constraints. That is an optimal problem which can be solved by existing optimizing algorithm packages e.g. CPLEX (CPLEX,2007). The complexity of such optimizing has been analyzed by Conitzer and Sandholm in 2002 (Conitzer & Sandholm, 2002).

4. Solution selection and negotiation

For service agents, different solutions mean different payoffs. But for service requester, different solutions mean different costs and service quality. Service agents always concentrate on the payoffs, while service requester pays more attention on service quality. A QoS ontology has been used for evaluating service quality, which includes quality evaluation items, such as responding time, throughput, latency, Load-Balancing, and etc (Maximilien & Singh, 2004). This Ontology can also be used to evaluate service agent. With this QoS Ontology, service requester is able to evaluate the candidate service agent in each quality item. We could define a quality evaluation function for deciding the quality of a service agent *agt* doing a particular function *Func*. Suppose that *n* quality items are considered in a unique evaluation interval [*0,M*], M∈**R**, the evalutaion function is as follows.

$$Q(agt,Func)=\frac{1}{n}\sum_{i=1}^{n}\omega_i\sigma_i$$

In which, $\sigma_i \in [0,M]$ is the evaluation value for the i_{th} quality item, ω_i is a preference value for the i_{th} quality item of the requester. Then the solution selection problem can be expressed as follows.

Definition 4.1 (Solution) Suppose there are n service agents $agt_1,agt_2,\ldots,agt_n$ in a given feasible coalition *C*, and *m* functions $Func_1,Func_2,\ldots,Func_m$ in a given service request *req*. p_i is the weight of $Func_i$ and $\sum_{i=1}^{m} p_i = 1$. Service agent agt_i has ability to fulfil $Func_j$, and the quality is $Q(agt_i,Func_j)\in[0,M]$. Suppose $W_i \subseteq F$, $F=\{Func_1,Func_2,\ldots,Func_m\}$ is the set of functions which agt_i has the ability to fulfil. A ***solution*** is a vector $(T_1,T_2,\ldots,T_n)$ requiring $\{T_1,T_2,\ldots,T_n\}$ to be a partition of *F*. $T_i \subseteq W_i$ is the set of functions fulfilled by agt_i in solution *S*.

Based on this, we can introduce more notations. Let *Q(S)* be the quality of solution *S*, then $Q(S)=\sum_{i=1}^{n}\sum_{j\in T_i} p_j Q(agt_i,Func_j)$. Let $P_i(S)$ be the payment of service agent agt_i in solution *S*, then $P_i(S)=\frac{\sum_{j\in T_i} p_j Q(agt_i,Func_j)}{Q(S)} P(S)$. Here, *P(S)* is the payment function that requester is willing to pay for solution *S*. *Q(C)* is the value that each function be assigned to the service agent who has the best quality value of doing it, and $Q(C)=\sum_{j=1}^{m} p_j \max_i Q(agt_i,Func_j)$. $P_i(C)$ is service agent agt_i's maximal payment that it is assigned all the functions in W_i, and $P_i(C)=\frac{\sum_{j\in W_i} P_j Q(agt_i,func_j)}{Q(S)} P(S)$.

Definition 4.2 (Solution Selection Problem**)** Given the reservation service quality request of the requester *MinQ* and the reservation payoff, $MinPay_i$, of each service agent agt_i, we want to find a solution S^* which satisfies:

- $\frac{Q(S^*)}{Q(C)}+\sum_{i=1}^{n}\frac{P_i(S^*)}{P_i(C)}=\max_{S}(\frac{Q(S)}{Q(C)}+\sum_{i=1}^{n}\frac{P_i(S)}{P_i(C)})$;
- Q(S*)≥MinQ, Pi(S*)≥MinPayi, i=1,2,…,n;

We call $\frac{Q(S^*)}{Q(C)}$ the satisfaction degree of the requester, and $\frac{P_i(S^*)}{P_i(C)}$ the satisfaction degree of service agent agt_i. The two constraints in Definition 4.2 means a solution which maximizes both the requester's satisfaction degree and the service agents' satisfaction degree under the condition of every minimal expectant benefit being satisfied.

Theorem 4.1. The solution selection problem is an optimization problem and the complexity is NP-complete.

Proof. Consider a special case of the problem. Let $n=2$; agt_1, agt_2 be identical and they can fulfil all the functions. For any solution S, $Q(S)=\sum_{j=1}^{m}p_jQ(Func_j)$ is a constant. Let $MinPay_1=MinPay_2=\frac{f(Q(S))}{2P_1(C)}=\frac{f(Q(S))}{2P_2(C)}$, then we should assign proper functions to each service agent such that $P_1(S)=P_2(S)$. That is the 2-partition problem, which is an NP-complete problem.

For solving this problem, we define a negotiation process among service agents and service requester. In this process, each service agent strives to get more payment, and the service requester tries to choose a solution which could get the service quality of as high as possible. Figure 1 shows the negotiation process which can be explained as follows:

step 1. The requester proposes a solution S_0 which he prefers the most. Let S_0 be the current candidate solution S_p.

step 2. Each agt_i, $i=1,2,\ldots,n$ accepts or refuses S_p according to their preference. If all the service agents accept S_p, the negotiation process stops; if not, the service agent who refuses S_p should modify the solution and propose the modified solution. The negotiation process stops if there are service agents who fail to propose a modified solution.

step 3. The requester chooses one solution S from the modified solutions proposed by the service agents. Let S be the current candidate solution S_p and go back to step 2 to enter a new round negotiation..

In the first step, the requester can propose the solution S_0 simply by assigning each function $Func_i$ to the service agent which has the max $Q(agt,Func_i)$. Moreover, there should be $Q(S_0)\geq MinQ$. The candidate solution S_p is referred to be the current considered solution. In the second step, agt_i modifies the candidate solution S_p according to its preference. For example, in solution S_p, agt_1 is assigned to fulfil $Func_1$, but agt_1 prefers to fulfil $Func_1$ and $Func_2$. In this case, agt_1 will modify solution S_p to let agt_1 fulfil $Func_1$ and $Func_2$, and keep the other part of S_p unchanged. Service agent agt_i accepts a solution S when $P_i(S)\geq MinPay_i$. The modified solution by a service agent should be acceptable; otherwise this modification fails. Additionally, the service agents never propose a solution that has been proposed by the requester before. In the third step, the requester chooses one among all the modified solutions. If the best solution S does not meet $Q(S)\geq MinQ$, the negotiation ends without any agreement on the final solution.

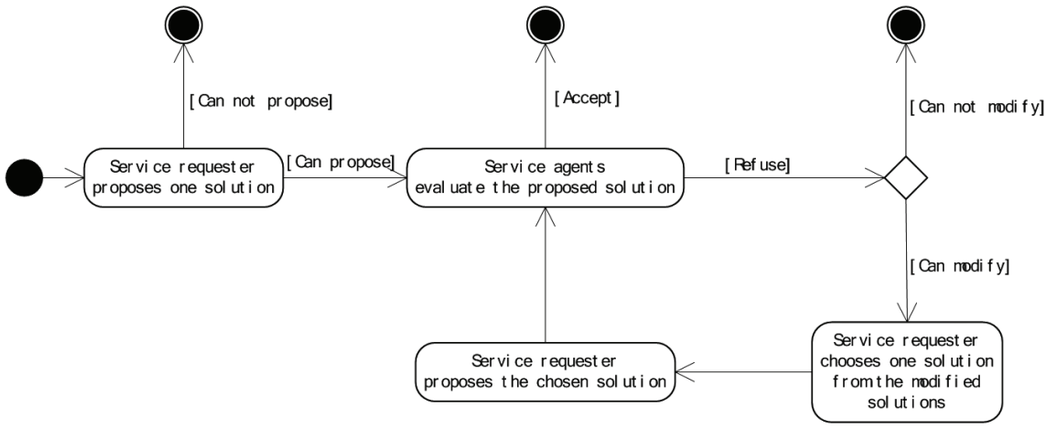


Fig. 1. Process of Negotiation

After the final solution has been chosen, we get a model of a multiple agent system which is able to implement the required service.

5. Case study

Based on the E-Learning domain function ontology in Section 2, a case study in on-demand e-learning domain is illistrated in this Section.
Firstly, the service agents and the service request should be given. For example,

$$DatabaseSearcher:=\langle\{OpenDatabase,ReadFromDatabase\},\{1,5\},fun^{BP}\rangle$$

Service agent *DatabaseSearcher* has the ability of searching certain data in database. It has two behaviours, *OpenDatabase* and *ReadFromDatabase.* Its expected minimum payoff for behaviour *OpenDatabase* is 1 and for behaviour *ReadFromDatabase* is 5.

$$OnlineTest:=\langle\{CreateTestPaperWithDatabase,Test,Evaluate\},\{40,30,30\},funFC\rangle.$$

That is a service request. It includes three functions: *CreateTestPaperWithDatabase*, *Test*, and *Evaluate*. The function *CreateTestPaperWithDatabase* creates a test paper based on the data from question database. Function *Test* manages the whole process of test and receives testees' answers from the test terminals. Function *Evaluate* grades the answers and gives comments about the test.
Service agents compute their expected payoff for this service reqest in the domain. For example, the service agent *DatabaseSearcher* has a behaviours *OpenDatabase* which is included in the behaviour set of request function *CreateTestPaperWithDatabase*. The payoff for fulfilling function *CreateTestPaperWithDatabase* is 40. The payoff propotion of behaviour *OpenDatabase* in function *CreateTestPaperWithDatabase* is 10%. So service agent *DatabaseSearcher* will gain 4 for contributing behaviour *OpenDatabase*. As *DatabaseSearcher*'s expected minium payoff for this behaviour is 1, it would participate the coalition for fulfilling the service request.
According to the expected payoff, a service agent coalition is established for the service request. In this case, a coalition includes four service agents is established for service request *OnlineTest*. Besides *DatabaseSearcher*, there are three service agents, *TestPaperCreater*, *InterfaceAgent*, and *Evaluater*. Table 4 gives the detail information of them.

Agent	Behaviours	*MinimumExpectedPayoff*
TestPaperCreater	*GetQuestions*	10
	CreateTestPaper	10
InterfaceAgent	*DisplayTestPaper*	5
	GetAnswerFromUser	5
	RecordAnswer	5
Evaluater	*EvaluateTestPaper*	10
	GetStandardAnswer	1
	CompareAnswer	5
	CalculatePoint	5
	WritePointToPaper	1
	WriteCommentToPaper	1

Table 4. Agent *TestPaperCreater, InterfaceAgent,* and *Evaluater*

Automated mechanism design is used to guarantee the stability of the coalition. The service agent type and the outcome set must be given firstly in AMD.

In service request *OnlineTest*, the appropriate functions of each service agent includes:

- *DatabaseSearcher* can participate to fulfil functions *GetDataFromDatabase*, *CreateTestPaperWithDatabase,* and *GetQuestionPoint*.
- *TestPaperCreater* can participate to fulfil functions *CreateTestPaper*, *GetQuestionsFromPaper*, and *CreateTestPaperWithDatabase*.
- *InterfaceAgent* can participate to fulfil only one function *Test.*
- *Evaluator* can participate *Evaluate, JudgeWongOrRight, CalculateFinalPoint,* and *GiveComment*.

The possible types of *DatabaseSearcher* include 6 different permutations of its appropriate functions. Different permutation means different preference of the service agent. A service agent prefers the functions which could bring it more payoff than others. The expected payoffs from the three appropriate functions of *DatabaseSearcher* are given as follows. Function *GetDataFromDatabase* is a sub-function of request function *CreateTestPaperWithDatabase*, and its payoff proportion is 50%. Function *CreateTestPaperWithDatabase* is one of the requested functions of service request, and its reward is 40. So the expected payoff of function *GetDataFromDatabase* is 20. Similarly, the expected payoff of function *CreateTestPaperWithDatabase* is 20. The expected payoff of function *GetQuestionPoint* is 1.8. Then the type of service agent *DatabaseSearcher* is θ_1=(*GetDataFromDatabase,CreateTestPaperWithDatabase,GetQuestionPoint*), or θ_2=(*CreateTestPaperWithDatabase, GetDataFromDatabase,GetQuestionPoint*).

According to definition 3.6, there are three outcomes for service request *OnlineTest.* They are o_1=(*FuncDecMod*(*GetQuestionPoint*), *FuncDecMod*(*Evaluate*)) for function *Evaluate*, o_2=(*FuncDecMod*(*CreatePaperwithDatabase*)) for function *CreatePaperwithDatabase*, and o_3= Φ for function *Test*. Utility function gives the payoffs of each type of service agent for all the outcomes. For example, table 5 gives the utility function of service agent *DatabaseSearcher*.

U_1	O_1	O_2	O_3
θ_1	1.8	20	0
θ_2	1	25	0

Table 5. Utility function u_1 of *DatabaseSearcher*

Similarly, we can give all the type sets of other service agents and their utility function. The type set of service agent *TestPaperCreater* includes permutation θ_3=(*CreateTestPaper, CreateTestPaperWithDatabase, GetQuestionsFromPaper*), and θ_4=(*CreateTestPaperWithDatabase, CreateTestPaper, GetQuestionsFromPaper*). The type set of service agent *InterfaceAgent* includes only one permutation θ_5=(*Test*). The type set of service agent *Evaluator* includes permutation θ_6=(*Evaluate*, *CalculateFinalPoint, JudgeWongOrRight*, *GiveComment*), and θ_7=(*Evaluate*, *CalculateFinalPoint, GiveComment, JudgeWongOrRight*). Table 6 lists the utility functions of the three service agents.

U	O_1	O_2	O_3
θ_3	1.2	25	0
θ_4	1	20	0
θ_5	0	0	30
θ_6	36	0	0
θ_7	36	0	0

Table 6. Utility values for different service agent types

According to the definition of mechanism, the mechanisms satisfying all the IR and IC constraints could be obtained by using some optimizing algorithms. In this case we use the optimal algorithm package provided by Lingo (Lingo, 2008). The computing result is as follows.

- $\{\theta_1,\theta_3,\theta_6\}\rightarrow o_1$
- $\{\theta_2,\theta_4,\theta_7\}\rightarrow o_1$
- $\{\theta_5\}\rightarrow o_2$
- $\{\theta_1,\theta_3\}\rightarrow o_1$
- $\{\theta_2,\theta_4\}\rightarrow o_1$

With the above mechanisms, the feasible coalitions for service request *OnlineTest* include:

- Coalition 1: *DatabaseSearche* and *TestPaperCreater* cooperate to fulfil function *CreateTestPaperWithDatabase*, *InterfaceAgent* fulfils function *Test*, and *Evaluator* fulfils function *Evaluate* by itself.
- Coalition 2: *DatabaseSearche* and *TestPaperCreater* cooperate to fulfil function *CreateTestPaperWithDatabase*, *InterfaceAgent* fulfils function *Test, DatabaseSearche* and *TestPaperCreater* cooperate to fulfil function *GetQuestionPoint*, and *Evaluater* fulfils function *JudgeWrongOrRight, CalculateFinalPoint*, and *GiveComment.*

The functions and function decomposition modes used in coalition generation are:

Evaluate:=⟨{*EvaluateTestPaper*},{*TestPaper*},{*Stateof*(*TestPaper*)=*NotEvaluated*},*funC*,{*100%*},*funP*⟩.

FuncDecMod(*Evaluate)*:=⟨{*GetQuestionPoint,JudgeWrongOrRight,CalculateFinalPoint,GiveComment*},{},fun^{SC},{*10%,20%,50%,20%*},fun^{SF}⟩.

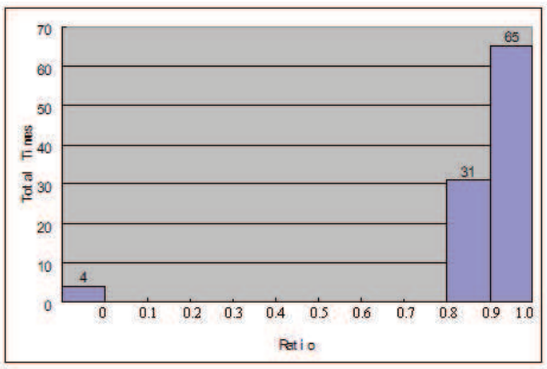
CreateTestPaperWithDatabase:=⟨{*OpenDatabase,ReadFromDatabase,GetQuestions,CreateTestPaper*},{*QuestionDatabase,QuestionBuffer,TestPaper*},{*Stateof(QuestionDatabase*)=*Read,Stateof(QestionBuffer)=HasQuestions*}, *funC*,{*10%,20%,30%,40%*},*funP*⟩.

FuncDecMod(CreateTestPaperWithDatabase):=⟨{*GetDataFromDatabase,CreateTestPaper*},{},*funSC*, {*50%,50%*},*funSF*⟩.

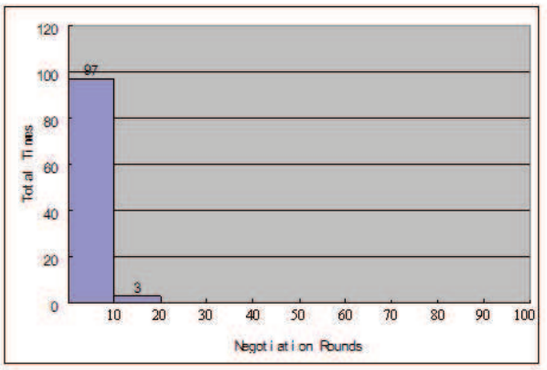
GetQuestionPoint:=⟨{*GetQuestions*,*OpenDatabase*,*ReadFromDatabase*},{*QestionBuffer*,*QuestionDatabase*},{*Stateof*(*TestPaper*)=*HasTestInfo*,*Stateof*(*QestionBuffer*)=*HasQuestions*}, *funC*,{50%,20%,30%},*funP*⟩.

FuncDecMod(*GetQuestionPoint*):=⟨{*GetQuestionsFromPaper*,*GetDataFromDatabase*},{},fun^{SC},{40%,60%},fun^{SF}⟩.

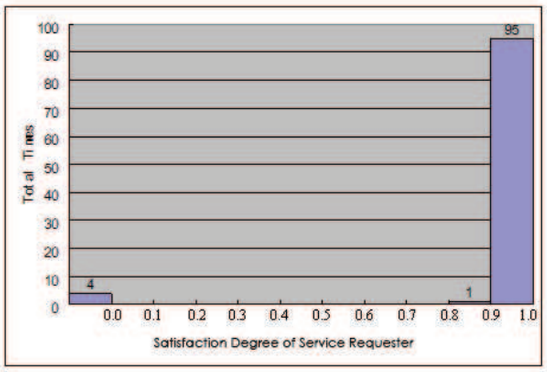
We simulate the negotiation process to test the negotiation. In the numerical experiment, there are 4 groups: 10 service agents and 20 functions; 3 service agents and 21 functions; 4 service agents and 20 functions; 5 service agents and 20 functions. Each group is simulated for 100 times and each time we randomly generate the information of service agents and service requester, i.e. the quality of service of service agents, reservation payoff of service agents, and reservation quality requirement of service request. Five statistic data is concerned: the successful rate of negotiation, the result of negotiation comparing with optimal result in theory, the negotiation rounds when the negotiation process is terminated, the satisfaction degree of service requester, and the satisfaction degree of service agents.
Take the group with 10 service agents and 20 functions for example, the statistic data is explained as following.



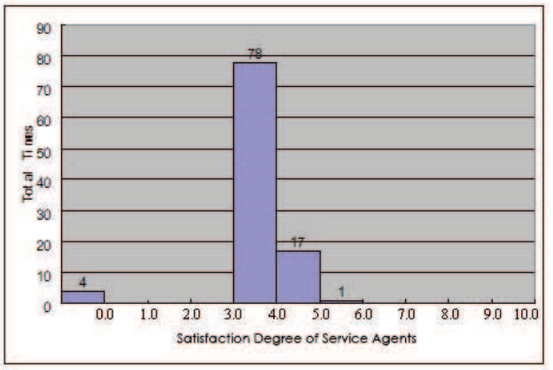
(a) Ratio of Negotiation to Optimal Result



(b) Rounds of Negotiation



(c) Satisfaction Degree of Service Requester



(d) Satisfaction Degrees of Service Agents

Fig. 2. Statistic Data of 10&20

First, within 100 simulations, they all have solution with optimal total satisfaction degree. By negotiating, 4 of them fail. That is to say, there is 4 times that service requester and service agents could not get consensus but there is a solution which could satisfy every one. So the successful rate of the negotiation is (100-4)/100=96%.

Second, the result of negotiation comparing with optimal result in theory is concerned. Denote the total satisfaction degree of negotiation σ^* and optimal total satisfaction degree σ. Figure 2(a) shows the statistic data about $\tau = \sigma^* / \sigma$. Except for that 4 times negotiation fails, τ is no less than 0.8. Further, most of them (65 in 96) are more than 0.9.

Third, we want to know when the negotiation terminates, i.e. how many rounds of the negotiation. Every time service requester chooses one solution and proposes it(first time service requester proposes one) in the modified solutions proposed by service agents, we say that the negotiation passes one round. In our numerical experiment, we tolerate the negotiation round no more than 100 or we stop it without waiting for its terminating. The numerical experiment data of negotiation round is given in Figure 2(b). In the Figure, there are 3 times those negotiation rounds are more than 10, but 97 times that are less than 10. So no matter the negotiation is successful or not, the negotiation is easy to be convergent.

Next is the satisfaction degree of service requester. Figure 2(c) shows the numerical experiment data of satisfaction degree of service requester. There are 96 successful negotiations. All of them have the service requester satisfaction degree higher than 0.8. Further, most of these negotiations (95 in 96) have the value higher than 0.9. According to the data given in Figure 2(c), if the negotiation is successful, the service requester could get high satisfaction degree.

How about the service agents' satisfaction degree? In our numerical experiment setting, each agent can do averagely 6 functions. On the other hand, there are 10 service agents and 20 functions; so the average number of functions that each agent could be assigned is that 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 $0.33 \times 10 = 3.3$.

We record the total satisfaction degree of 10 service agents and show them in the Figure 2(d). Within 96 negotiation that are successful, 20 of them have the total satisfaction degree of service agents lower than 3.3, and there are 76 negotiations that have the total satisfaction degree of service agents higher than 3.3. According to the analysis given in the above paragraph, such a result indicates that in the negotiation, service agents could frequently get higher total satisfaction degree than expect average value.

Analysis of other three groups is similar with the group with 10 service agents and 20 service requesters. For comparison, the concrete statistic data is given in Table 7. From the table, we see that in all the groups, the successful rate is more than 90%. All the properties discussed in the 10&20 group remain.

According to the statistic data above, we can conclude that our negotiation framework has a high successful rate and the negotiation result is close to the optimal. Additionally, negotiation process is easy to be convergent. Moreover, service requester could expect high satisfaction degree (higher than 0.8) and mostly, service agents could get higher total satisfaction degree than expect average value. These all confirm that our negotiation framework generates good result.

Agent& Function	Successful Rate	Result Ratio		Negotiation round		Requester Satisfaction		Agents Satisfaction	
10&20	96%	0	4	0	0	0	4	0	4
		0.0-0.1	0	0-10	97	0.0-0.1	0	0.0-1.0	0
		0.1-0.2	0	11-20	3	0.1-0.2	0	1.0-2.0	0
		0.2-0.3	0	21-30	0	0.2-0.3	0	2.0-3.0	0
		0.3-0.4	0	31-40	0	0.3-0.4	0	3.0-4.0	78
		0.4-0.5	0	41-50	0	0.4-0.5	0	4.0-5.0	17
		0.5-0.6	0	51-60	0	0.5-0.6	0	5.0-6.0	1
		0.6-0.7	0	61-70	0	0.6-0.7	0	6.0-7.0	0
		0.7-0.8	0	71-80	0	0.7-0.8	0	7.0-8.0	0
		0.8-0.9	31	81-90	0	0.8-0.9	1	8.0-9.0	0
		0.9-1.0	65	91-100	0	0.9-1.0	95	9.0-10.0	0
7&21	91%	0	9	0	0	0	9	0	9
		0.0-0.1	0	0-10	89	0.0-0.1	0	0.0-1.0	0
		0.1-0.2	0	11-20	4	0.1-0.2	0	1.0-2.0	0
		0.2-0.3	0	21-30	2	0.2-0.3	0	2.0-3.0	61
		0.3-0.4	0	31-40	0	0.3-0.4	0	3.0-4.0	30
		0.4-0.5	0	41-50	0	0.4-0.5	0	4.0-5.0	0
		0.5-0.6	0	51-60	2	0.5-0.6	0	5.0-6.0	0
		0.6-0.7	0	61-70	1	0.6-0.7	0	6.0-7.0	0
		0.7-0.8	1	71-80	1	0.7-0.8	0	7.0-8.0	0
		0.8-0.9	4	81-90	0	0.8-0.9	3	8.0-9.0	0
		0.9-1.0	86	91-100	1	0.9-1.0	88	9.0-10.0	0
5&20	92%	0	8	0	0	0	8	0	8
		0.0-0.1	0	0-10	87	0.0-0.1	0	0.0-1.0	0
		0.1-0.2	0	11-20	2	0.1-0.2	0	1.0-2.0	0
		0.2-0.3	0	21-30	2	0.2-0.3	0	2.0-3.0	90
		0.3-0.4	0	31-40	2	0.3-0.4	0	3.0-4.0	2
		0.4-0.5	0	41-50	1	0.4-0.5	0	4.0-5.0	0
		0.5-0.6	0	51-60	0	0.5-0.6	0	5.0-6.0	0
		0.6-0.7	0	61-70	2	0.6-0.7	0	6.0-7.0	0
		0.7-0.8	0	71-80	0	0.7-0.8	0	7.0-8.0	0
		0.8-0.9	1	81-90	1	0.8-0.9	2	8.0-9.0	0
		0.9-1.0	91	91-100	3	0.9-1.0	90	9.0-10.0	0
4&20	94%	0	6	0	0	0	6	0	6
		0.0-0.1	0	0-10	92	0.0-0.1	0	0.0-1.0	0
		0.1-0.2	0	11-20	2	0.1-0.2	0	1.0-2.0	0
		0.2-0.3	0	21-30	3	0.2-0.3	0	2.0-3.0	93
		0.3-0.4	0	31-40	0	0.3-0.4	0	3.0-4.0	1
		0.4-0.5	0	41-50	1	0.4-0.5	0	4.0-5.0	0
		0.5-0.6	0	51-60	1	0.5-0.6	0	5.0-6.0	0
		0.6-0.7	0	61-70	0	0.6-0.7	0	6.0-7.0	0
		0.7-0.8	0	71-80	0	0.7-0.8	0	7.0-8.0	0
		0.8-0.9	2	81-90	0	0.8-0.9	2	8.0-9.0	0
		0.9-1.0	92	91-100	1	0.9-1.0	92	9.0-10.0	0

Table 7. Statistic Data of Negotiation

6. Conclusions

This paper proposes a feasible strategy for agent-based service computing. The main contributions of this paper may fall into two points.

Firstly, we figure out the vision of agent-based service computing. In which, both the service requests and the available services are agents. The service requests are virtual

agents which need the available service agents or their composition to implement the required functionalities and give out the payoffs as rewards. And the available services are service agents which offer their capabilities to implement required functionalities and gain necessary payments. With some constraints, the agent society can form stable coalitions.

Secondly, we design a process for enabling the agent-based service computing. A function ontology has been introduced to allow the understanding between the required service requests and the available service requests so that the required service requests can know which available service agents they need and the available service agents can know which required service requests they can make contributions to. Then those available service agents who can fulfil one required service agent when they form a group form a stable coalition by using automated mechanism design. Finally, the required service agents negotiate with that candidate stable coalition to select the final solution via a negotiation process on the criteria of service quality as well as the payoff.

One of the future directions is extending this work by integrating the widely-used Web service description standards. For example, generating a BPEL-based specification of the multiple agent system for fulfilling the required service agent from the coalition of service agents will allow the agent-based composite service to be executable. That will help to combine this approach with the current efforts in industry.

7. References

ABNF. (2005). Augmented BNF for Syntax Specifications: ABNF, http://tools.ietf.org /html/rfc4234, 2005.

CPLEX. (2007). ILOG Mathematical Programming Optimizers, http://www.ilog.com/ products/cplex/, 2007.

BPEL. (2007). Ws-BPEL2.0(OASIS), http://docs.oasis-open.org/wsbpel/2.0/, 2007.

SOAP. (2004). SOAP version 1.2. http://www.w3.org/TR/soap/,2004.

OWL-S. (2004). OWL-S: Semantic markup for Web services. http://www.daml.org/ services/owl-s/1.0/, 2004.

Jaeger, M.; Engel, L.; Geihs, K. (2005). A Methodology for Developing OWL-S Descriptions. *Proceedings of First International Conference on Interoperability of Enterprise Software and Applications Workshop on Web Services and Interoperability*, pp. 13–31. ISBN1846281512, Geneva, Switzerland, Feb, 2005, Springer Berlin, Heidelberg.

Roman, D.; Lausen, H.; Keller, U. (2005). The Web Service Modeling Ontology WSMO. *WSMO Working Draft D2, final version 1.2.* www.wsmo.org/2004/d2/, April 2005;

Roman, D.; Scicluna, J.; Feier, C. (2005). Ontology-based Choreography and Orchestration of WSMO Services. *WSMO Working Draft D14.* www.wsmo.org/2004/d14/, March 2005.

Conitzer, V. ; Sandholm,T. (2002). Complexity of mechanism design, *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI02)*, pp. 103-110,ISBN978-1558608979,Edmonton,Canada,Aug.2002,Morgan Kaufmann, San Fransisco.

Christensen, E.; Curbera, F. (2001). Web services description language (WSDL)1.1,http://www.w3.org/tr/wsdl.

Maximilien, E. M.; Singh, M. P. (2004). A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, Vol 8,No. 5, Sept.-Oct. 2004, 84–93, ISSN1089-7801.

Paolucci, M. ; Sycara, K.; Kawamura, T. (2003). Delivering semantic web services. Technical Report CMU-RI-TR-02-32, Robotics Institute, Carnegie Mellon University.

Sandholm, T. (2003). Automated mechanism design: A new application area for search algorithms. *Proceedings of the International Conference on Principles and Practice of Constraint Programming(CP03) LNCS2833*, pp.19-36,ISBN3540202021, Kinsale, Ireland, Spet.2003,Springer,Heidelberg.

Ankolekar, A.; Burstein, M. et al. (2002a). DAML-S:Web Service Description for the Semantic Web, *Proceedings of The First International Semantic Web Conference (ISWC) LNCS2342*, pp. 348-363,ISBN978-3540437604, Sardinia, Italy, June, 2002, Springer, Heidelberg.

Ankolekar, A.; Huch, F.; Sycara, K. (2002b). Concurrent Semantics for the Web Services Specification Language Daml-S. *Coordination Models and Languages LNCS2315,* pp. 567-577, ISBN9783540434108, Jan. 2002, Springer, Heidelberg.

Wooldridge, M. (2000). *Reasoning About Rational Agents*. TheMIT Press,ISBN978-0262232135,Cambridge, MA.

Zheng, L.; Jin, Z. (2007a). Requirement driven agent collaboration. Proceedings of the 2007 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2007, pp. 446–448,ISBN978-8190426275, Honolulu, Hawaii, May 2007,ACM New York, USA .

Zheng, L.; Jin, Z. (2007b). Requirement driven agent collaboration based on functional ontology and AMD. *Proceedings of 11th IEEE International Workshop on Future Trends of Distributed Computing Systems*, pp. 189–198. ISBN0769528104, Sedona, AZ, April, 2007, IEEE Computer Society, New York, USA .

Kreger, H. (2001). Web services conceptual architecture (WSCA 1.0). IBM Software Group Note.http://www-306.ibm.com/software/solutions/webservices-pdf/WSCA.pdf, 2001.

CELF. (2005). Computing Research Association. Cyberinfrastructure for education and learning for the future: A vision and research agenda. http://www.cra.org/reports/cyberinfrastructure.pdf, 2005.

Sycara, K.; Paolucci, M. (2003). Automated discovery, interaction and composition of semantic Web services. *Journal of Web Semantics*. Vol 1,No.1,Dec. 2003, 27–46, ISSN 1570-8268.

Sycara, K.; Paolucci,M. (2004). Ontologies in agent architectures. *Handbook on Ontologies in Information Systems*. Springer-Verlag, 2004, 343-364, ISBN3540408347, Berlin.

Paolucci, M.; Sycara, K. (2003). Autonomous Semantic Web services. *IEEE Internet Computing*. Vol 7, No.5, Sept.-Oct. 2003, 34 – 41, ISSN1089-7801.

Lingo. (2008). An Overview of LINGO. www.lindo.com, 2008.

Casati, F.; Ilnicki, S ; Jin, L. (2000). Adaptive and dynamic service composition in eflow. *Proceedings of 12th International Conference on Advanced Information System Engineering LNCS1789*, pp. 13–31. ISBN9783540676300, Stockholm, Sweden, June, 2000, Springer Berlin, Heidelberg.

Multiagent Systems

Edited by Salman Ahmed and Mohd Noh Karsiti

ISBN 978-3-902613-51-6
Hard cover, 426 pages
Publisher I-Tech Education and Publishing
Published online 01, January, 2009
Published in print edition January, 2009

Multi agent systems involve a team of agents working together socially to accomplish a task. An agent can be social in many ways. One is when an agent helps others in solving complex problems. The field of multi agent systems investigates the process underlying distributed problem solving and designs some protocols and mechanisms involved in this process. This book presents an overview of some of the research issues in the field of multi agents. It is a presentation of a combination of different research issues which are pursued by researchers in the domain of multi agent systems as they are one of the best ways to understand and model human societies and behaviours. In fact, such systems are the systems of the future.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Liwei Zheng, Jian Tang and Zhi Jin (2009). Requirements Driven Service Agent Collaboration, Multiagent Systems, Salman Ahmed and Mohd Noh Karsiti (Ed.), ISBN: 978-3-902613-51-6, InTech, Available from: http://www.intechopen.com/books/multiagent_systems/requirements_driven_service_agent_collaboration

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen