Int J Software Informatics, Vol.4, No.4, December 2010, pp. 401–418 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
 Tel: +86-10-62661040

An Agent Based Framework for Internetware Computing

Liwei Zheng[1], Jian Tang[1], and Zhi Jin[2]

[1] (Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China)

[2] (Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

Abstract Internetware intends to be a paradigm of Web-based software development. At present, researches on Internetware have gained daily expanding attentions and interests. This paper proposes an agent based framework for Internetware computing. Four principles are presented that are followed by this framework. They are the autonomy principle, the abstract principle, the explicitness principle and the competence principle. Three types of agents with different responsibilities are designed and specified. They are the capability providing agents, the capability planning agents and the capability consuming agents. In this sense, capability decomposition and satisfaction turns to be a key issue for this framework and becomes a communication protocol among these distributed and heterogenous agents. A capability conceptualization is proposed and based on the conceptualization, an agent coalition formation mechanism has been developed. This mechanism features that (1) all the participants make their one decisions on whether or not joining the coalition based on the capability realization pattern generated by the capability planning agents as well as the benefits they can obtain; and (2) the coalition selection is conducted by a negotiation process for satisfying the expectations of all the participants as the complexity of this problem has been proven to be NP-complete.

Key words: internetware computing; multi agent system; coalition formation

Zheng LW, Tang J, Jin Z. An agent based framework for internetware computing. *Int J Software Informatics*, **2010, 4(4): 401–418. http://www.ijsi.org/1673-7288/4/i67.htm**

1 Introduction

As the Web becomes pervasive and software development on the Web is getting popular, the Web will become not only the unified platform on which software systems are deployed and executed and but also a virtual software development platform that enables business and technology personnel to jointly build applications. As such, this new revolutionary platform will eventually replace the traditional platform[25].

* This work is sponsored by the the National Natural Science Fund for Distinguished Young Scholars of China under Grant No.60625204, the National Basic Research and Development 973 Program under Grant No.2009CB320701, and the Key Projects of National Natural Science Foundation of China under Grant Nos. 90818026 and 60736015.
Corresponding author: Zhi Jin, Email: zhijin@sei.pku.edu.cn
Received 2010-08-20; revised 2010-11-30; accepted 2010-12-25.

Compared with the traditional software development and execution platform, this new platform has different features such as the openness, dynamism and uncertainty. So the software systems should be more autonomous, more sensitive to context changes, more reactive, and more evolvable. Researchers in China named such a software development and execution on this platform the Internetware computing[26] 8 years ago and are conducting a project on this issue with the sponsorship of the National Basic Research Program in China. The mission of this project includes studying, establishing and using systematic, disciplined, quantifiable approaches for software development, execution and maintenance on World Wide Software Web.

At present, technical framework for Internetware has been researched in many aspects. References [13,15] propose an architecture centric technical framework for the definition, incarnation and engineering of Internetware. It gives a software model, designed a middleware and proposes an engineering methodology for Internetware. Self-adaptation is one of the basic features of Internetware. Focusing on this issue, Ref.[15] proposes software architecture centric approach for Internetware's self-adaptation. The knowledge for self-adaptation has been captured, organized and reasoned so that the automatic analysis and decision-making can be achieved.

Reference [10] proposes an agent-based approach to the open coordination software model for Internetware. It concentrates on resource sharing and service integrating in the open environment. As results, a programming model of agent-based coordination has been devised, an agent-based middleware for multi-mode coordination and a dynamic software architecture centric software coordination mechanism are provided. Furthermore, Ref.[11] focusses on the environment-driven model and corresponding enabling techniques. A software structuring model for environment-driven systems is presented for addressing the issue of how to deal with the openness, dynamism and uncertainty of the environment. Another work[5] on agent based Internetware model presents an architecture called EBDI (electronic business document exchange) to describe the components which can autonomously plan themselves at runtime to handle variable environments, and uses dynamic binding relationship to illuminate the self-adaptive and evolutionary components.

Our previous work[22,24] propose to use agent to model the Internetware entities and use the agent coalition and collaboration to model the computing style of Internetware. We call this agent-based computing style for Internetware computing the Agent-based Internetware Computing style (ABIC, for short). In Ref.[22], we propose the requirements driven Internetware entities collaboration in ABIC. In which, all of the available Interenetware entities are autonomous being represented as software agents. When there are requirements for building an Internetware application, those available software agents relevant to the requirements will collaborate with each others and form the coalition and fulfill the requirements. That is for allowing Internetware entities to identify the application development requirements and interact with each other autonomously for satisfying the requirements. Furthermore, in Ref.[24], we model the Interentware entity collaboration as a task allocation problem and give a negotiation-based solution for the task allocation problem among self-interest Internetware entities on decentralized setting.

Along this line, this paper systematically tackles the issues concerned in ABIC. The main contributions of this paper include: (1) summarizing and refining the gen-

eral principles of ABIC; (2) designing its architecture and structuring and specifying the three main parties in ABIC; (3) developing a capability conceptualization as the communication protocol among the three parties; and finally (4) figuring out the collaboration mechanism by capability reasoning.

This paper is organized as follows. In section 2, we present the general principles and give the architecture. Section 3 presents a capability conceptualization. That will serve as the vocabularies for the agents' reasoning on the capability competency. Section 4 introduces the computing mechanism of ABIC. The coalition formation and collaboration based on capability reasoning are the main issues among others. Section 5 discusses some related work. Finally, section 6 concludes the paper.

2 Principles and Architecture

2.1 General principles

Four general principles are followed when the framework for ABIC is developed. They are the *autonomy principle*, the *abstract principle*, the *explicitness principle* and the *competence principle*.

The *autonomy principle* states that all the Internetware entities are autonomous, active and persistent and they can control their own resources and their own behaviors and can even show the social ability through collaboration with each other through dynamic discovery and negotiation. They can autonomously search for each others and choose the roles that they will play in the collaboration for satisfying a desired requirements so that the interactions between them can be established dynamically and connected flexibly. In this sense, the autonomous Internetware entities can be treated as *agents*.

The *abstraction principle* states that any complicated capability has its own abstract realization patterns. Each abstract realization pattern defines a way for decomposing a complicated capability into a set of simpler capabilities. It also assigns the simpler capabilities to a set of roles and allows them to collaboratively realize the complicated capability. These roles can taken by the competent autonomous Internetware entities.

The *explicitness principle* states that all aspects of the autonomous Internetware entities must be explicitly specified covering both the syntax and semantic. This enables the semantic correctness of the Internetware entity interactions and the capability reasoning to be assessable.

The *competence principle* states that for an autonomous Internetware entity to be able to dynamically join in an abstract realization pattern to collaborate with each others meaningfully and correctly, it must be competent to understand the capability required and to follow the interaction protocols instructed by the roles' specifications, which according to the explicitness principle, is accessible.

2.2 Architecture

Following the general principles, first abstraction we made is figuring out three kinds of parties in the computing style. We use three types of agents to represent the different parties. They are the capability providing agents, the capability planning agents and the capability consuming agents. Within the framework, different parties

have different responsibilities:

- The capability consuming agents are the initiators. When a software consumer wants to develop an application system to realize a capability, he/she creates and deploys a capability consuming agent to ask for the realization of a required capability.
- The capability providing agents are the realization bodies of the required capabilities. When identifying the capability requests, they firstly determine whether or not they are compete with the required capabilities. If yes and they are willing to be the capability realizer, they make a bid (by joining some coalition if the calling for capability realizers is from a capability planning agent) to these capabilities.
- The capability planning agents are planners to produce realization plans for the required capabilities. Each plan contains a decomposition of a particular capability into a set of ordered part capabilities and a role model to realize the capability which consists of a set of collaborative roles. This plan is also responsible to assign the part capabilities to the roles to assure these roles can collaborate with each other to realize the capability. When observing a required capability that is what they are competent, they will compete with each others to make a plan for the required capability.

  From the viewpoint of and capability decomposition and the role model for capability realization, a capability realization plan is an abstract capability realization coalition of the required capability. It in fact is a pre-defined pattern for realizing a particular capability. When all of the roles in an abstract coalition have been taken by available capability providing agents, the concrete coalitions consisting of the available capability providing agents will be formed to be the candidate realization bodies of the required capability. Then the capability consuming agent will negotiate with the capability providing agents in the coalitions for making the role allocation among the capability providing agents.

Figure 1 shows the architecture of this framework for ABIC. There are three agent pools for containing three types of agents. The capability providing agents (CPrA) are provided by Internetware entity providers. Each CPrA can realize certain capabilities. The capability consuming agents (CCoA) are initiated by Internetware application consumers. Each CCoA expresses a required capability that needs to be satisfied by the to-be developed Internetware application. The capability planning agents (CPlA) are provided by domain experts. Each CPlA is a capability realization pattern.

Among the three types of agents, both the capability providing agents and the capability planning agents are pre-defined and relative stable during the computing process. But, the capability consuming agents are dynamically initiated and undergo a five-state lifecycle. This lifecycle can, in some sense, represent the ABIC mechanism:

- Initiated: The capability consuming agent is initiated by an Internetware application consumer.
- Planned: The capability realization patterns for the required application capability have been proposed by capability planning agents.

- Coalition-formed: the coalitions of the capability realization patterns have been formed. Each coalition consists of available capability providing agents.

- Role-allocated: a stable feasible coalition has been selected as the realization body of the capability consuming agent.

- Destroyed: the Internetware application has been developed and the required capability has been realized.

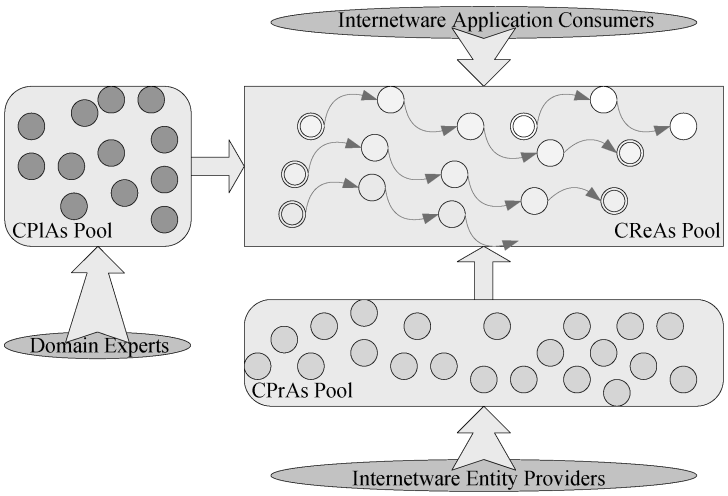

Figure 1. Architecture of ABIC

3 Capability Conceptualization and Capability Satisfaction

3.1 Capability ontology

As shown in Fig.1, this paper assumes that each capability planning agent represents a kind of realization plan for a capability and is responsible to propose the capability realization pattern. Behind this, we actually assume a domain capability ontology that is shared by both parties in ABIC. Each capability planning agent contains only an integral fragment of the knowledge in this ontology and the whole set of the capability planning agents constitute a distributed representation of the ontology.

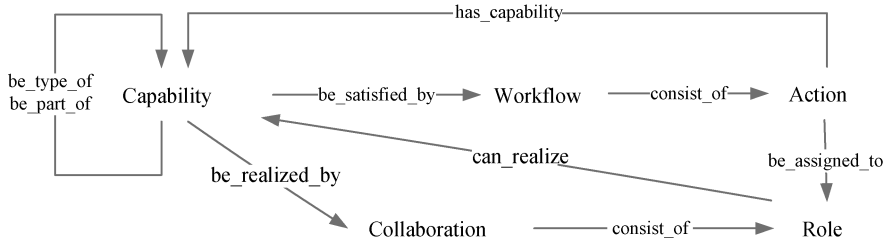

Figure 2. Capability upper ontology

Figure 2 shows the capability upper ontology. It explicitly indicates the associations between the five concept categories: the capability, the workflow, the action, the collaboration, and the role. They are:

- A capability can
  - be a refinable capability that can be refined into a set of sub-type capabilities
  - be a composite capability that can be decomposed into a set of part capabilities
  - be an atomic capability if it has no any part capabilities and has no any sub-type capability which has part capabilities
- A capability can be satisfied by at least one workflow
- A workflow consists of a set of actions
- An action has a certain capability
- An action is assigned to a role
- A capability can be realized by at least one collaboration
- A collaboration consists of a set of roles
- A role can realize at least one capability

With this capability upper ontology, the domain capability ontology can be represented accordingly. Let $\mathcal{C} = \{c_1, \cdots, c_{|\mathcal{C}|}\}$ be a set of domain capabilities. For any composite capability $c \in \mathcal{C}$, c has at least a capability realization pattern $pattern(c)$ that represents a way for realizing c. Any capability realization pattern consists of three parts. The first two are the workflow of its part capabilities and the role model for realizing these part capabilities. It also contains an assignment of the part capabilities to its roles, i.e.

$$pattern(c) =< WFlow(c), RModel(c), Assignment(c) >$$

is a capability realization pattern of c.

$$WFlow(c) = (Cap, COrd, CapWeights, \tau)$$

is a workflow for satisfying c. It is a directed acyclic graph with the part capabilities as the nodes and the control flow as the edges. $Cap \subseteq \mathcal{C}$ is the set of part capabilities of c. $COrd$ expresses the control flow among the capabilities in Cap. Apart from the capability decomposition, the workflow also assigns weights to the part capabilities. Each weight attached with a particular part capability represents the importance degree of this part capability in this workflow. $CapWeights = \{\omega_1, \cdots, \omega_{|Cap|}\}$ is a set of weights where $\sum \omega_i = 1$ $(1 \leqslant i \leqslant |Cap|)$. $\tau : Cap \leftrightarrow CapWeights$ is a bijective function from Cap to $CapWeights$. That $\tau(c_i) = \omega_j$ $(c_i \in Cap$ and $1 \leqslant i, j \leqslant |Cap|)$ means the weight of c_i in this workflow is ω_j.

$$RModel(c) = (Roles, Prots)$$

is a role model for realizing c. It consists of a set of roles and a set of interaction protocols. Like the setting in Gaia[21], in ABIC, roles are abstract constructs in capability realization bodies. All roles are atomic constructs and cannot be defined in terms of other roles. *Roles* names all of the roles that will take part in the realization of c. *Prots* specifies the interactions among the roles during the realization. It details the interaction flow as well as the messages that will be exchanged in interactions.

$$\begin{aligned} Assignment(c) &= assign(WFlow(c), RModel(c)) \\ &= \{assign(c_i, r_j) | c_i \in WFlow.Cap, r_j \in RModel.Roles\} \end{aligned}$$

is a set of the assignment of an action with a part capability in $WFlow$ to a role in $RModel$. All of the actions in $WFlow$ need to be assigned to one role to implement.

Figure 3 examples a fragrant domain capability ontology[9]. It talks about the capability of land-based shipping. The two main parts of this ontology fragrant are a weighted workflow and a role model. It also contains the assignment of the part capabilities to the roles. Table 3.1 gives the schema representation of this domain capability ontology.

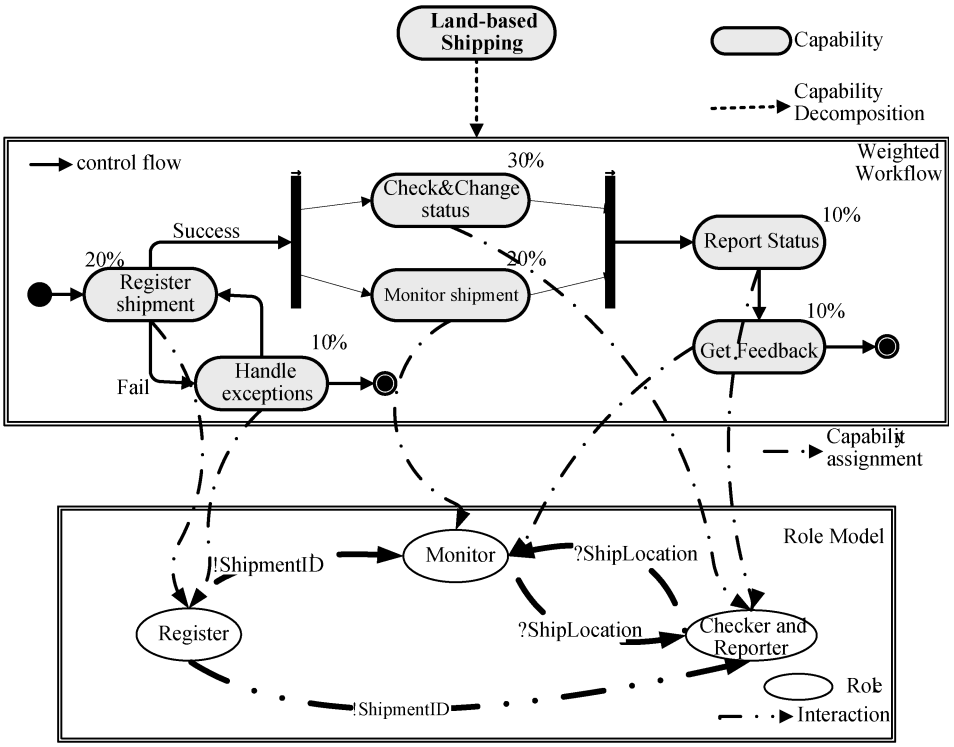

Figure 3. Shipping capability ontology

4.1 Capability satisfaction

With the capability conceptualization, some lightweight capability reasoning can be supported. Let CAP contain all the domain capabilities. Let $cap \in CAP$ a capability, and $capS \subset CAP$ a set of capabilities. $capS$ satisfies cap if

Table 1 Domain capability ontology of land-based shipping

Capability: Land-Based Shipping		
Sub-Capability	**Capability**	**Importance Degree**
	register shipment	0.2
	handle exceptions	0.1
	check and change status	0.3
	monitor shipment	0.2
	report status	0.1
	get feedback	0.1
Workflow	register shipment(*Flag*) *if Flag=success*, *fork*(check and change status, monitor shipment) report status get feedback *if Flag=fail*, handle exceptions	
Role Model	**Role**	**Responsibility**
	Register	register shipment handle exceptions
	Monitor	monitor shipment get feedback
	Checker and Changer	check and change status report status
Interaction	(Register, !shipmentID, Monitor) (Register, !shipmentID, Checker and Reporter) (Monitor, !location, Checker and Reporter) (Checker and Reporter, ?location, Monitor)	

- Cap is atomic in CAP and $cap \in capS$
- Cap is refinable, $CapS = refine(cap)$ is the set of the sub-type capabilities of cap, and there exists a capability $c' \in CapS$ such that $capS$ satisfies c'
- Cap is decomposable, $CapS = decompose(cap)$ is the set of the part capabilities of cap, and for all the $c' \in CapS$ such that $capS$ satisfies c'

4 ABIC Computing Mechanism

The previous sections give the main principles and the architecture. This section focuses on the mechanism of ABIC.

4.1 The settings of the mechanism

As mentioned above, there are three types of agents in ABIC. They are the capability providing agent, the capability consuming agent and the capability planning agent. This section details the constituents of these agent types.

4.1.1 Capability providing agent

The capability providing agents are basic capability realization bodies. They are

responsible for realizing the required capabilities. For simulating the characteristics of the agents' benefit-pursuing mechanism, each capability providing agent has some other properties besides the capabilities, i.e. each capability providing agent has a set of minimum prospective payoffs. Each prospective payoff is the payment that the agent wants to obtain when offering one of its capabilities. Thus, the capability providing agent is described as

$$CPrA :=< CapPros, ExpPays, ExpCapPay >$$

in which

- $CapPros = \{cap_1, \cdots, cap_n\}$ is a set of capabilities that can be realized by this agent. Each cap_i $(1 \leqslant i \leqslant n)$ represents a capability;
- $ExpPays = \{\rho_1, \cdots, \rho_n\}$ is a set of minimum prospective payoffs; and
- $ExpCapPay : CapPros \leftrightarrow ExpPays$ is a bijective function from $CapPros$ to $ExpPays$. That $ExpCapPay(cap_i) = \rho_j$ $(1 \leqslant i, j \leqslant n)$ means that the agent wants get a payment $\rho \geqslant \rho_j$ when it relizes capability cap_i

4.1.2 Capability consuming agent

The capability consuming agents are delegations of software capability consumers. Any software capability consumer can initiates a capability consuming agent when he/she wants to consume a software capability. Each capability consuming agent needs to declare clearly the capability that is required to be realized and the payment that can be offered if the capability is realized. A capability consuming agent is described as

$$CCoA :=< Cap, Payment >$$

in which

- Cap is a required capability;
- $Payment = \phi$ is a maximum payment that the agent can offer. This agent will offer $\phi' \leqslant \phi$ as the payment if the required capability can be realized.

Generally speaking, any capability consuming agent can ask for more than one capability and make different payment for each required capability. That we limit only one required capability here is just for simplifying the description.

4.1.3 Capability planning agent

The capability planning agent is responsible to produce the abstract capability realization pattern for a required capability submitted by a capability consuming agent. Each abstract capability realization plan defines a set of its sub-capabilities as well as the relationship (i.e. the control flow) among these sub-capabilities. It also designs a role model (with interaction protocol) and assigns each of the sub-capabilities to a role. The selected abstract capability realization plan will become the proposal for allowing the capability providing agents to join in and then to form a concrete capability realization coalition.

The capability planning agent can be represented as

$$CPlA =< Cap, CapS, Workf, RoleS, Coll, CapToRole >$$

in which

- Cap is the capability that the agent is going to make plan;
- $CapS = \{< cap_1, \omega_1 >, \cdots, < cap_n, \omega_n >\}$ is a capability-weight set. Each element is a part capability of Cap with its weight ω_i $(1 \leqslant i \leqslant n)$. Here, $\Sigma_{i=1}^{n} \omega_i = 1$.
- $Workf : CapS \times CapS$ is a workflow of the part capabilities. It defines the order of the capability realizations, i.e. the *precede* relation between capabilities;
- $RoleS = \{rol_1, \cdots, rol_l\}$ is a set of participants that will be involved for realizing the capabilities in $capS$;
- $Coll : RoleS \times RoleS$ defines the interactions between participants during the realization of Cap, i.e. the *interaction* relation between two roles with the message transmitted.
- $CapToRole : CapS \rightarrow roleS$ is a surjective function *assign* which assigns a part capability in $capS$ to a role in $Roles$.

4.2 The process of the computing mechanism

Section 2 presents a five-state lifecycle for the capability consuming agents which in fact is the computing process of ABIC mechanism. Among the five stages of the process, the *initiation*, the *plan making* and the *capability realization* are straightforward in terms of the explanation of the previous sections. In following, we only discuss the *coalition formation* and the *role allocation*.

4.2.1 Stable and feasible coalition

When an application capability request is issued by a capability consumer, a capability consuming agent is initiated. If the required application capability is atomic, there is no need to form a coalition. This required application capability can be directly allocated to a capability providing agent. Otherwise, it needs a capability planning agent to propose a capability realization pattern. With this proposed capability realization pattern, the capability providing agents can ask for joining the pattern to form coalitions that are competent to realize the application capability.

A coalition is an alliance among individuals during which they will cooperate in joint action, although each in their own self-interest, joining forces together for a common cause. How can the capability providing agents form temporarily such a coalition around the proposed pattern for a required application capability?

More formally, given Cap is the required capability of capability consuming agent $CCoA$ and

$$CPlA =< Cap, CapS, Workf, RoleS, Coll, CapToRole >$$

is a capability realization pattern for Cap generated by capability planning agent $CPlA$. In order to form a coalition upon $CPlA$, the capability providing agents need firstly to know whether or not they are effective capability providing agents of this pattern. Without loss of generality, let agt be a capability providing agent that is willing to take part in the coalition and $r \in RoleS$ a role in $CPlA$. With the capability-role assignments in $CapToRole$, agt can judge whether or not it can be competent to one of the role in the pattern, i.e. it is competent to a role if it has all the capabilities that has been assigned to this role. Those capability providing agents that are competent to roles in $CPlA$ are effective agents of $CPlA$.

Then, the capability planning agent $CPlA$ needs to decide if the coalitions can be formed after the capability providing agents make their bids. A set of capability providing agents $CPrAS = \{capProAgt_1, \cdots, capProAgt_m\}$ forms a coalition for realizing Cap upon $CPlA$ if there exists an one-to-many mapping from $CPrAS$ to $RoleS$ in $CPlA$, i.e.

- Any role $r \in RoleS$ has been bided by one and only one effective agent $agt \in CPrAS$; and
- Each of the agents $agt \in CapProAgtS$ has bided at least one role $r \in RoleS$ that it is competent to.

We call $CPrAS$ a coalition for Cap upon $CPlA$.

It is crucial for the agent-based Internetware computing to choose a socially desirable outcome. Mechanism design is the art of designing the mechanism (i.e. rules of the game) so that the agents are motivated to report their preference truthfully and a desirable outcome is choose according to a given objective. Traditionally, mechanism design has been a manual endeavor where the designer uses his experience and intuition to hypothesize that a certain rule set is desirable in some way and then tries to prove that this is the case. Automated mechanism design was introduced by Conitzer and Sandholm where the mechanism is automatically created for the setting and objective. It models the mechanism design as a computational optimization problem[17].

In multiagent settings of the Internetware computing, the three types of agents have conflicting objectives and preferences. We extend the automated mechanism design into the requirements driven mechanism design[22] by introducing instantiations to the automated mechanism design setting. They are:

- A meaningful outcome is a realization pattern for the required capability of a capability consuming agent.
- The type of the capability providing agent is a vector of the domain capabilities that this agent can realize. The type of the capability consuming agent is a vector of the domain capabilities that this agent wants to realize. We call the vector of the domain capabilities the agent type.
- The objective function is defined to be benevolent, i.e. to pursue the happiness of two sides, i.e. both the capability consuming agents and the capability providing agents.

As such, the requirements driven automated mechanism design setting can be defined. First, we are given

- Let (δ) be the type of the capability consuming agent $CCoA$, in which δ is a domain capability.
- A finite set of outcomes O produced by capability planning agents, each of them is a capability realization pattern of δ
- A finite set $CPrAS$ of N capability providing agents for realizing the capability;
- For each capability providing agent $capProAgt_i \in CPrAS$,
  - a finite set of capability types Θ_i
  - a probability distribution γ over Θ_i
  - a utility function $u_i : \Theta_i \times O \to \mathbb{R}$
- An objective function which maximizes both the satisfaction degree of the capability consuming agent $CCoA$ and all the capability providing agents in $CPrAS$.

The requirements driven mechanism is a deterministic mechanism with payments. Different from the general deterministic mechanism with payments, the requirements driven mechanism concerns two kinds of agents and thus needs to consider two kinds of payments. Concretely, with a fixed payment that can be offered by the capability consuming agent, this agent cares about the quality that the capability realization, while the capability providing agents care about the payments they can obtain when realizing the capabilities. Thus, the requirements driven mechanism consists of:

- An outcome selection function

$$o : \Theta_1 \times \Theta_2 \times \cdots \times \Theta_N \to O$$

- For each capability providing agent $capProAgt_i$, a payment selection function

$$\pi_i : \Theta_1 \times \Theta_2 \times \cdots \times \Theta_N \to \mathbb{R}$$

  where $\pi_i(\theta_1, \cdots, \theta_N)$ gives the payment obtained by $capProAgt_i$ when the reported types are $\theta_1, \cdots, \theta_N$; and

- For capability consuming agent $CapConAgt$, a quality selection function

$$\phi : \Theta_1 \times \Theta_2 \times \cdots \times \Theta_N \to \mathbb{R}$$

  where $\phi(\theta_1, \cdots, \theta_N)$ gives the quality of realizing δ when the reported types are $\theta_1, \cdots, \theta_N$.

With this setting, we can define the computational problem of the requirements driven mechanism design as also an optimization problem that can be stated as follows. We are given a requirements driven automated mechanism design setting, an IR

notion and an IC notion, and we are asked whether there exists a deterministic mechanism that satisfies the IR and IC notions and maximizes the benevolent objective function.

In Ref.[23], we sorted out such a benevolent objective function. Two criteria have been defined. Let $Coalition =< CPrAS, CCoA >$ be a coalition, for each $capProAgt \in CPrAS$, the payment that it can be offered when it joins in the coalition to realize some of the part capabilities is

$$P(capProAgt, CCoA) = \sum_{i=1}^{n} \omega_i \phi$$

in which (1) ϕ is the maximum payment that $CCoA$ can offer; and (2) ω_i is the weight of a part capability that will be assigned to $capProAgt$ if it joins in the coalition[1].

The capability consuming agent $CCoA$ cares about the quality of the capability realization. We borrowed Maximilien's QoS Ontology[12] to define the quality of the coalition. Assume that we take n quality items into account and use an unique quality interval $[0, M], M \in R$ for the n quality items. Let cap is a part capability of Cap. Then, the quality of the realization of cap by $capProAgt$ is

$$Q(capProAgt, cap) = \sum_{i=1}^{n} \varrho_i \kappa_i,$$

in which, $\kappa_i \in [0, M]$ is the evaluation value for the ith quality item, ϱ_i ($\sum_{i=1}^{n} \varrho_i = 1$) is the weight of preference for the ith quality item of the capability consuming agent. To evaluate the quality of coalition, the capability consuming agent computes the weighted sum of qualities that every part capabilities.

Then we can define the feasible coalition[23] as follows. Let Cap be the capability consumed by $CCoA$, $Coalition =< CPrAS, CCoA >$ be a coalition for Cap, $CapS = \{cap_1, cap_2, \cdots, cap_m\}$ be the set of part capabilities of Cap upon a capability realization pattern. Assume that the weight of cap_i in $CapS$ is η_i ($\sum_{i=1}^{m} \eta_i = 1$). Let $capProAgt_i$ be capable of realizing cap_j and $Q(capProAgt_i, cap_j)$ denote the quality of this realization. Let

$$Cap(capProAgt_i) = \{cap_j | cap_j \in capS \text{ and } capProAgt_i \text{ is capable of realizing } cap_j\}$$

$Coalition$ is feasible, if and only if:

1. $\bigcup_{i=1}^{n} Cap(capProAgt_i) = CapS$ and $\bigcap_{i=1}^{n} W_i = \emptyset$;
2. $\forall j \Big(\bigcup_{i=1}^{n} Cap(capProAgt_i) \Big) \backslash Cap(capProAgt_j) \subset Caps$;
3. $\sum_{j=1}^{m} \eta_j \max_i Q(capProAgt_i, cap_j)$ has the largest value of all coalitions.

[1] The set of part capabilities that will be assigned to a capability providing agent contains all the part capabilities of all the roles that the agent takes.

4.3 Role allocation and negotiation-based solution

After obtaining the potential capability realization coalitions, next step is finding the optimal coalition of the best quality and all the capability providing agents can get best payments. In Ref.[24], we modeled the coalition selection as an assignment problem and has proved that the coalition selection in ABIC settings is NP-complete.

Also, in Ref.[24], we proposed a negotiation based solution for finding a feasible coalition. Recalling and summarizing the ABIC settings. Let Cap be the capability required by $CCoA$, $Coalition =< CPrAS, CCoA >$ a coalition for Cap upon a capability realization pattern proposed by $CPlA$, $CPrAS = \{capProAgt_1, \ldots, capProAgt_n\}$ the set of the capability providing agents, $Roles = \{r_1, \cdots, r_m\}$ the set of roles for the capability providing agents in $SAgts$ to take in $CPlA$, $CapS = \{cap_1, \cdots, cap_k\}$ the set of part capabilities of Cap in $CPlA$.

Suppose ρ_i ($\sum_{i=1}^{k} \rho_i = 1$) is the weight of cap_i ($1 \leqslant i \leqslant k$), $CapS(r_i) \subset CapS$ is the set of part capabilities that role r_i needs to realize, $Cap(capProAgt_i) \in CapS$ is the capabilities that $capPrvAgt_i$ is capable of realizing, and $Q(capProAgt_i, cap_j)$ is the realization quality of cap_j by $capProAgt_i$.

We define a Kripke structure[6,18] for this setting. A state s of the role assignment problem in ABIC setting is an m-dimensional vector $(\rho_1, \ldots, \rho_m)$, where $\rho_i \in \{1, \ldots, n\}$ indicates that role r_i is assigned to $capProAgt_{\rho_i}$. Then the Kripke structure of the assignment problem in ABIC setting is defined as:

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

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;
- $CPrAS = \{1, \ldots, n\}$ is the set of capability providing agents;
- $\alpha : R \to CPrAS$ labels each transition in R with a capability providing agent;
- $V : S \to 2^{\Phi}$ labels each state with the set of propositional variables that are true in this state.

Then the semantics of the role assignment problem in ABIC setting and some restrictions can be given.

$S = \{(\rho_1, \ldots, \rho_m) | cap_i \in Cap(capProAgt_{\rho_i}), i = 1, \ldots, m\}$, i.e. any state in S is an assignment.

$S^0 = \{(\rho_1, \ldots, \rho_m) | \max_j \sum_{r_j \in R_i} \sum_{cap_k \in Cap(r_j)} p_k Q(capProAgt_i, cap_k)\}$, i.e. any initial state in S^0 is the state that every role is assigned to the capability providing agent which has the best realization quality.

Let $((\rho_1, \rho_2, \ldots, \rho_m), (\rho_1^{'}, \rho_2^{'}, \ldots, \rho_m^{'})) \in R$. If $\rho_i \neq \rho_i^{'}, (1 \leqslant i \leqslant m)$, ρ_i and $\rho_i^{'}$ compete for the i^{th} role. R contains only the transitions from which only one capability providing agent can compete for a role. Furthermore, we use Computation

Tree Logic (CTL)[6] to express the objectives of the normative systems. That is, for any $1 \leqslant i \leqslant n$, capability providing 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 the capability providing agents' behaviors, i.e. on the transition relation. That can be modeled as a normative system[2]. 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.

The negotiation framework focuses 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.

The negotiation process is:

1. In according to the stable and feasible coalition, the capability consuming agent chooses 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 role is assigned to the capability providing agent which has the best realization quality.

2. The capability providing agents evaluate the state to see if it is satisfiable. The capability providing agent who is willing to accept the state will do nothing; otherwise, it will transit the state. The decision on being "accepted" or being "refused" is made based on the reservation payoff. Those states that satisfy the capability providing agent's reservation payoff will be accepted but others will be refused.

3. The capability providing agents which refuse the proposed state will negotiate with others and then modify the states. They trigger the transitions in R. Any capability providing agent i will not negotiate with those capability providing agents which are assigned the role r that contains the capabilities that it is not capable of realizing, i.e. $Cap(r) \not\subset Cap_i$. And they may work out several states that have already satisfied some of the agents but not all.

4. The capability consuming agent chooses one state that satisfies itself best (with the greatest total quality) and after that it proposes the chosen state to the capability providing agents. This proposed state starts a new round of negotiation.

The negotiation will terminate in three possible situations.

- If all capability providing agents accept the capability consuming agent's proposal in some round, the negotiation ends with success.

- If the capability providing agents can not make a valid transition when it negotiates with others, the negotiation ends with failure.

- If the capability consuming agent can not make a choice in the modified states, i.e. all modified states worked out by the capability providing agents can not achieve its reservation quality, the negotiation ends with failure.

The negotiation is proved to have good properties by simulation[24].

5 Related Work

The Internetware entities are intended to be independent, active, adaptive, and have the capability to apperceive and influence the environment, the active Internetware entities are more profitable for realizing the Internetware entity autonomous aggregation and collaboration. Agents which are autonomous, reactive and sociable can be one of the most appropriate software modality for Internetware. That is main consideration when we develop the agent based Internetware computing framework.

Within the agent-based framework for Internetware computing, coalition formation and task allocation are two key issues among others. In the area of multiagent systems, these issues have been investigated in recent years with different assumptions and emphases. However, most of them ignore the privacy of agents, and study the problem in a centralized setting. For example, Ref.[8] develops 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. Ref.[13] 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.

There are also some other works on Internetware model, such as the trust based Internetware model[19]. While, these works are mainly based on the architecture-based model for Internetware.

There are some related works relates coming from the area of service oriented computing, especially the work on the service composition. There are mainly two research directions. One is based on standardized description. Many XML-based languages in industrial community are used to describe services, service composition process, and the execution process such as RDF, OWL-WS, and BPEL4WS, etc. have been developed. With the standard specification, the composition solutions is mainly reuse-based. However, the composition solution is normally given manually and the services passively wait for being invoked. The other is the composition based on Semantic Web. Ontology is used in this kind of research to describe the service capability and execution effects precisely[7]. Base on the precise descriptions supported by the domain ontology, the dependency relations between resources could be obtained by reasoning. Using the descriptions and dependencies, service composition is modeled. However, though this appraoch is adaptive in the dynamic environment, the computing time cost is always too high to be accepted.

There are also some works which have similar setting with ABIC. For example, Ref.[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. Another work is Ref.[12], in which the agents are selected according to their trust values but the selection is also centralized. Different with these 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 a single mediator; also the agents gain more flexibility in our setting.

6 Conclusion

This paper gives an agent based framework for internetware computing. Four principles are presented that are followed when developing this framework. They are the autonomy principle, the abstract principle, the explicitness principle and the competence principle. Three types of agents are identified and precisely specified. They are the capability providing agents, the capability planning agents and the capability consuming agents. Different types of agents have different responsibilities in the computing mechanism.

For the purpose to supporting the agents' communicating and reasoning, a capability conceptualization has been developed which offers the sharable vocabulary for the heterogenous and distributed parties of the computing mechanism.

Based on these, we design the coalition formation for the self-interested agents on the decentralized setting, and give a negotiation-based solution for the task allocation in coalition selection.

In this paper, the task assignment problem is solved by providing a suitable normative system[24]. This builds the bridge between ABIC with the available works on normative systems, games, mechanisms, etc. With these works, some interesting issues like robustness[3] or applying power indices[4] can be introduced into ABIC. That will make ABIC more powerful and robust. Including different negotiation strategies for different agents might be also an interesting issue for making ABIC more practical.

References

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

[2] Agotnes T, van der Hoek W, Wooldridge M. Normative system games. Proc. of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2007). 2007.

[3] Agotnes T, van der Hoek W, Wooldridge M. Robust normative systems. Proc. of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2008). 2008.

[4] Agotnes T, van der Hoek W, Tennenholtz M, et al. Power in normative systems. Proc. of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2009). 2009.

[5] Chang Z, Mao X, Qi Z. Component model and its implementation of internetware based on agent. Journal of Software, 2008, 19(5): 1113–1124 (in Chinese).

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

[7] Hou L, Jin Z, Wu B. Modeling and verifying Web services driven by requirements: an ontology-based approach. Science in China (Series F: Information Sciences), 2006, 49(6): 792–820.

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

[9] Liu Z. Requirement elicitation and modeling for service-oriented requirement engineering [MS Thesis]. Arizona State University.

[10] Lu J, Tao X, Ma X, et al. On agent-based software model for internetware. Science in China

(Series F: Information Sciences), 2005, 35(12): 1233–1253 (in Chinese).

[11] Lu J, Ma X, Tao X, et al. On environment-driven software model for internetware. Science in China (Series F: Information Sciences), 2008, 51(6): 683–721.

[12] Maximilien EM, Singh M P. Toward autonomic web services trust and selection. Proc. of the 2nd International Conference on Service Oriented Computing (ICSOC-2004). 2004. 212–221.

[13] Manisterski E, David E, Kraus S, et al. Forming efficient agent groups for completing complex tasks. Proc. of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2006). 2006. 834–841.

[14] Mei H, Huang G, Zhao H, et al. A software architecture centric engineering approach for Internetware. Science in China (Series F: Information Sciences), 2006, 49(6): 702–730.

[15] Mei H, Huang G, Han L, et al. A software architecture centric self-adaptation approach for Internetware. Science in China (Series F), Springer, 2008, 51(6): 722–742.

[16] Shehory O, Kraus S. Methods for task allocation via agent coalition formation. Artificial Intelligence, 2005, 101(1-2): 165–200.

[17] Sandholm T. Automated mechanism design: a new application area for search algorithms. Proc. of the International Conference on Principles and Practice of Constraint Programming (CP). 2003.

[18] Shrot T, Aumann Y, Kraus S. Easy and hard coalition formation problems - parameterized complexity analysis Proc. of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2009). 2009. 443–450.

[19] Wang Y, Lu J, Xu F, et al. A trust measurement and evolution model for internetware. Journal of Software, 2006, 17(4): 682–690 (in Chinese).

[20] Wooldridge M. Agent-based software engineering. IEE Proc. on Software Engineering, 1997, 144: 26–37.

[21] Wooldridge M, Jennings NR, Kinny D. The gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems, 2000, 3(3): 285–312.

[22] Zheng L, Jin Z. Requirements driven agent collaboration. Proc. of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2007), 2007.

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

[24] Tang J, Jin Z. Assignment problem in requirements driven agent collaboration and its implementation. Proc. of the 9th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010), 2010: 839–846.

[25] Tsai W, Jin Z, Bai X. Internetware computing: issues and perspective. International Journal of Software and Informatics, 2009, 3(4): 415–438.

[26] Yang F, Mei H, Lu J, Jin Z. Some discussion on the development of software technology. ACTA ELECTRONICA SINICA, 2002, 30(z1).

[27] Yang F, Lu J, Mei H. Technical framework for internetware: an architecture centric approach. Science in China (Series F: Information Sciences), 2002, 51(6): 610–622.