Strategic Capability Modelling of Services

Lin Liu, Chi-hung Chi
School of Software, Tsinghua University, Beijing, China 100084,

Zhi Jin
Academy of Mathematics and Management Science, CAS, Beijing, China, 100080,

Eric Yu
Faculty of Information Studies, University of Toronto, Toronto, Canada, M5S 3G6

{linliu,chichihung}@tsinghua.edu.cn, zhijin@math.ac.cn, yu@fis.utoronto.ca

Abstract

This paper introduces a services modeling ontology that describes Services requirements in terms of strategic capabilities of an actor. We argue that the modeling language together with heuristic rules-based reasoning mechanism offer a potentially more substantive approach to understand the nature of service systems in a variety of social contexts. Furthermore, understanding the underlying assumptions and constructs through the use of the services capability modeling framework will not only inform researchers of a better design for service-oriented systems, but also assist in the understanding of intricate relationships between different factors that services are situated in. We present a few illustrative services situations as proof-of-concept examples to illustrate the proposed approach.

1. Introduction

The concept of services, having achieved prominence in the context of Service-Oriented Architecture and Web Services, is now gaining even broader scope. IBM's recent call upon Service Science, Engineering, and Management [19] has pointed out a promising paradigm for next generation business and computing. It aims to investigate emerging issues in the transformation of IT infrastructure and software industry towards on-demand e-business and real-time services. It calls for theories and practical techniques from business, management, and information studies, to most (if not all) major areas in computer science and engineering. While there are different approaches to services, one of the fundamental characteristics of the service concept is its close tie with requirements – requirements of service user, service provider together with the constraints in the social-technical environments. From this viewpoint, the eventual success of services as a new business and computational paradigm is determined by how well requirements could be understood and addressed.

Why is RE for service so important, and difficult? In essence, the requirements engineering process for services are conducted in parallel and separately by the multiple participants: Service Provider, Service Requestor, and different Service Intermediaries. Before binding, each of these actors only have knowledge about themselves, knowledge about other actors are partial and hypothetical. A service binding will take place only if an agreement forms among the involved parties. As a consequence, the more accurately an actor understands others' requirements in advance, the better chance it has to form agreements with other actors. In other words, in comparison to software product development, service development needs to deal with more uncertainties when mapping the solution space into the problem space, since the latter's scope and

context are not as predetermined. It requires a components-in-advance mode, but how do we know what components to build if we don't know the actual requests? For instance, conventional off-the-shelf software is usually general-purpose software that does not take the variations of user requirements into full consideration. The service concept implies inherently customer orientation. Customization has to be done in order to handle variations on functional and non-functional requirements, on the composition of the product, and on the operational environments. As a consequence, this leads to the problem of what to provide, how to provide, how much quality to provide, and how to demonstrate the function and quality that can be provided. Remember that planning and performing of all these has costs. Requirements engineering for service is therefore difficult.

In order to handle the uncertainties in services engineering, we propose to start by modelling the strategic capabilities and needs of a service organization. In this paper, we aim towards building an automated reasoning framework for supporting open multi-agent societies. In such societies, strategic actors with services capabilities and requirements form social networks in order to fulfill their needs, and to serve others with service capabilities. Based on concepts from the strategic actors modeling framework in *i**, we build a service modeling ontology to represent and reason about alternative service strategies an actor (representing an organization) can adopt. The existing modelling constructs from the *i** language are mapped into elements of executable service-oriented systems. The entire SOA process workflow (from service publishing, request, discovery, selection to binding) is captured by the framework to allow automatic QoS-based service composition.

2. A Service Requirements Ontology Based on Strategic Capability

In this section, we first define the terms in our service requirements ontology, so that w can then provide precise rules for reasoning. Basic concepts of the service ontology include actor, actor's service capability, actor's service requirements, actor's knowledge on service composition and on other actor's capability and requirements, actor's actual behaviour of performing a service, delegating a service to another actor, and informing other actors of its requirements, capability and knowledge. To move towards automated support for services manipulation, we build a formalism based on these concepts to set up the services and analysis mechanism. We will not present the graphical notation in this paper, in the interest of space.

Definition 1. $A = \{a_1, ..., a_n\}$ *is a set of Actors*. $S = \{s_1, ..., s_n\}$ is a set of *Services*. $Q = \{q_1, ..., q_n\}$ is a set of *Quality attributes*. These are primitive concepts of a service setting.

Definition 2. $ME = S \rightarrow S$, is a set of *means-ends* relationships. Textually, we write $s' \rightarrow\!\!\!\!* \, s$, where s' represents the *end,* and s represent the *Means*. $DC = S \rightarrow \!\!\!\!\bullet \, S$, is a set of *decomposition* relationships. Textually, we use $s \rightarrow\!\!\!\!\bullet \{s_0, ..., s_n\}$.

Definition 3. $f = A \times Q \times S \rightarrow Int$ is a set of Quality of Service functions. $f(a_i, q_j, s_k)$, in which $a_i \in A$, $q_j \in Q$, $s_k \in S$, describes the value of a quality attribute q_j of a service s_k provided or required by actor a_i.

Definition 4. For each actor $a \in A$, there is an $FR \subseteq A \times S$ representing the set of *Functional Services Required* by *a.* There is also $NFR \subseteq A \times Q \times S \times Int$, representing the set of *Non-Functional Services Required* by *a.* Textually, we write ***Requires*** $_a$ s

Definition 5. For each actor $a \in A$, there is a $FC \subseteq A \times S$ representing the set of Functional services that a is capable of. There is also $NFC \subseteq A \times Q \times S \times Int$,

representing the set of *Non-Functional Services Can be provided* by *a*. We use $\textbf{\textit{Can}}_a\ s$.

Definition 6. For each actor $a \in A$, there is a $K \subseteq A \times FC \cup NFC \cup FR \cup NFR$ representing the set of *Knowledge* about services capabilities and requirements. Textually, we use $\textbf{\textit{Know}}_a\ x$

Definition 7. O = $\{o_1, ..., o_n\}$ is a set of applicable *Operations* to a service context $sc = <A, R, C, K>$. There are following basic types of operations: ***delegate***, ***tell***, and ***perform***.

1. ***delegate*** (a, s, b), represents that there is an inter-actor *delegation*, where $a, b \in A, s \in S$.
2. ***tell*** (a, y, b), represents an inter-actor *communication*, where a, $b \in A$, $y \in FR \cup NFR \cup FC \cup NFC$.
3. $\textbf{\textit{perform}}_a\ s$, represents a *service delivery,* where $a \in A, s \in S$.

For each delegation operation, we call the delegating actor the *delegator*, and the actor who is delegated upon the *delegatee*. By delegating a service to another actor, an actor (the delegator) is able to be served that it was not able, or not as easily or as well otherwise. At the same time, the delegator becomes vulnerable. If the delegatee fails to deliver the service, the delegator would be adversely affected in its ability to achieve its goals.

A world of services is an open environment, in which each of the above sets can be updated dynamically. In other words, actors will come into and get out from the environment. New request will be initiated or removed by actors; new capabilities will be added into or removed by the actors. In such a highly dynamic and distributed environment, automated service discovery, service agreement formation, and service selection has to be manipulated by certain machine process-able rules and policies. Below, we define some of the rules that can be applied under a service context: $sc_i = <A, R, C, K>$.

Rule 2.1: Service Delivery Rule

If an actor *a* is capable of providing a service *s*, and it also has the requirements of performing the service, it can perform the service. The requirement could be direct requirements of his own, or indirect requirements from other service requestors, depends on how the social rule of the service community are defined [20]. The operation "$\Rightarrow$" below is used as a production operation, which means that if the condition on the left holds, then action on the right hand can be triggered. The operation is not mandatory, but is optional according to the actor's preference.

$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}(s) \wedge \textbf{\textit{Can}}_a\ s \wedge \textbf{\textit{Requires}}_a\ s \Rightarrow \textbf{\textit{perform}}_a\ s$.

Rule 2.2: Service Composition/Transformation Rule

If an actor *a* is capable of providing a set of services $\{s_1 ..., s_n\}$, and it also has knowledge on how to compose or transform it into other more complex service s_0, then it is capable of providing the transformed or composite service s_0.

(1) ***OR composition through Means-ends link***

$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}\ (s_0) \wedge ... \wedge \textbf{\textit{Service}}\ (s_n) \wedge \textbf{\textit{Know}}_a \{ s_1 \rightarrow s_0, ..., s_n \rightarrow s_0 \} \wedge \textbf{\textit{Can}}_a\ s_{j\ (1 \le j \le n)} \Rightarrow \textbf{\textit{Can}}_a\ s_0$.

(2) ***AND composition through Decomposes link***

$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}\ (s_0) \wedge ... \wedge \textbf{\textit{Service}}\ (s_n) \wedge \textbf{\textit{Know}}_a\ (s_0 \rightarrow \{s_1 ..., s_n\}) \wedge \textbf{\textit{Can}}_a\ s_1 \wedge ... \wedge \textbf{\textit{Can}}_a\ s_n \Rightarrow \textbf{\textit{Can}}_a\ s_0$.

Rule 2.3: Request Decomposition/Transformation Rule

If an actor *a* requires a services *s*, and it also has knowledge on how to decompose or transform it into other more concrete services $\{s_1 ..., s_n\}$, then it can request for those the transformed or component services instead.

(1) ***AND decomposition***

$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}\ (s_0) \wedge ... \wedge \textbf{\textit{Service}}\ (s_n) \wedge \textbf{\textit{Know}}_a\ (s_0 \rightarrow \{s_1 ..., s_n\}) \wedge \textbf{\textit{Requires}}_a\ s_0 \Rightarrow \textbf{\textit{Requires}}_a\ s_1 \wedge ... \wedge$

Requires $_a$ s_n.

(2) ***OR decomposition***

Actor *(a)* $\wedge$ ***Service*** (s_0) $\wedge$ ...$\wedge$***Service*** $(s_n)\wedge$ ***Know*** $_a$ $\{ s_1 \rightarrow s_0, \ldots, s_n \rightarrow s_0 \}\wedge$ ***Requires*** $_a$ s_0 $\Rightarrow$ ***Requires*** $_a$ $s_1 \vee \ldots \vee$***Requires*** $_a$ s_n.

Rule 2.4: Publication Rule

An actor *a* may inform other actors about its request, capability about a service. The rules given below shows a possible strategy an actor may take during decision making related to service publication. It is a rather simplified example to show how the proposed procedure works.

(1) **Publish Request to Known Provider**

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Service*** *(s)* $\wedge$ ***Requires*** $_a$ s $\wedge$ ***Know*** $_a$ ***(Can*** $_b$ *s)* $\Rightarrow$ ***tell*** *(a,* ***Requires*** $_a$ *s, b)* $\wedge$ ***Know*** $_a$ ***(Know*** $_b$ ***(Requires*** $_a$ *s)).*

An actor *a* may publish a request to a known provider with the intention of building a service agreement. A direct effect of this publication action is that the publisher knows that the receiver of the message will know about his requirement on this service. This rule only considers the knowledge update from the publisher's side, knowledge update on the receiver's side is addressed by Rule 2.5.

(2) **Publish Request to an Expert on Service Transformation/Composition/Decomposition**

Actor *(a)*$\wedge$***Actor*** *(b)*$\wedge$***Service*** *(s)* $\wedge$ ***Service*** *(s')* $\wedge$ ***Requires*** $_a$ $s\wedge$ ***Know*** $_a$ ***(*** ***Know*** $_b$ *(s'*$\rightarrow$*s))* $\Rightarrow$ ***tell*** *(a,* ***Requires*** $_a$ *s, b)* $\wedge$ ***Know*** $_a$ ***(Know*** $_b$ ***(Requires*** $_a$ *s)).*

An actor *a* may publish a request to a known expert, who has knowledge on service composition, decomposition, or transformation, with the intention of knowing relevant steps of fulfilling a service. A direct effect of this publication action is that the publisher knows that the receiver of the message will know about his requirement on this service.

(3) **Publish Request to Service Registry (or Other Information Intermediary)**

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Actor*** *(x)* $\wedge$ ***Service*** *(s)* $\wedge$ ***Service*** *(s')* $\wedge$ ***Requires*** $_a$ s $\wedge$ ***Know*** $_a$ ***(Know*** $_b$ ***((Requires*** $_x$ *s)* $\vee$ ***(Can*** $_x$ *s)* $\vee$ ***(Know*** $_x$ *s'*$\rightarrow$*s))))* $\Rightarrow$ ***tell*** *(a,* ***Requires*** $_a$ *s, b)* $\wedge$ ***Know*** $_a$ ***(Know*** $_b$ ***(Requires*** $_a$ *s)).*

An actor *a* may publish a request to a known information center, who might be a web services registry, or simply another actor, who has knowledge on capabilities, requests, knowledge on other unknown actors, with the intention of knowing relevant information of fulfilling a service. A direct effect of this publication action is that the publisher knows that the receiver of the message will know about his requirement on this service.

(4) **Request Broadcasting**

Actor *(a)* $\wedge$ ***Service*** *(s)* $\wedge$ ***Requires*** $_a$ s $\Rightarrow$ ***tell*** *(a,* ***Requires*** $_a$ *s,* ***all)*** $\wedge$ ***Know*** $_a$ ***(Know*** $_{all}$ ***(Requires*** $_a$ *s)).*

An actor *a* may broadcast a request with the intention of obtaining relevant information of fulfilling a service. A direct effect of this publication action is that the publisher knows that the receiver of the message will know about his requirement on this service.

(5) **Publish Service to Known Requestor**

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Service*** *(s)* $\wedge$ ***Can*** $_a$ s $\wedge$ ***Know*** $_a$ ***(*** ***Requires*** $_b$ *s)* $\Rightarrow$ ***tell*** *(a,* ***Can*** $_a$ *s, b)* $\wedge$ ***Know*** $_a$ ***(Know*** $_b$ ***(Requires*** $_a$ *s)).*

An actor *a* may publish a service to a known requestor, with the intention of building service agreement. A direct effect of this publication action is that the publisher knows that the receiver of the message will know about his capability on this service.

(6) **Publish Service to Known Expert on Service Composition/Transformation**

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Service*** *(s)* $\wedge$ ***Service*** *(s')*$\wedge$ ***Can*** $_a$ s $\wedge$ ***Know*** $_a$ ***(*** ***Know*** $_b$ *s*$\rightarrow$*s')* $\Rightarrow$ ***tell*** *(a,* ***Can*** $_a$ *s, b)* $\wedge$ ***Know***

$_a$ ***(Know*** $_b$ ***(Can*** $_a$ *s)).*

An actor *a* may publish a service to a known expert, who has knowledge on service composition, decomposition, or transformation, with the intention of knowing relevant steps of building a new service based on existing ones. A direct effect of this publication action is that the publisher knows that the receiver of the message will know about his capability on this service.

(7) **Publish Service to Information Intermediary**

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Actor*** *(x)* $\wedge$ ***Service*** *(s)* $\wedge$ ***Service*** *(s')* $\wedge$ ***Can*** $_a$ *s* $\wedge$ ***Know*** $_a$ ***(Know*** $_b$ ***((Requires*** $_x$ *s)* $\vee$ ***(Can*** $_x$ *s)* $\vee$ ***(Know*** $_x$ *s'*$\rightarrow$*s))))* $\Rightarrow$ ***tell*** *(a,* ***Can*** $_a$ *s, b)* $\wedge$ ***Know*** $_a$ ***(Know*** $_b$ ***(Can*** $_a$ *s)).*

An actor *a* may publish a service to a known information center, who might be a web services registry, or simply another actor, who has knowledge on capabilities, requests, knowledge on other unknown actors, with the intention of knowing relevant information of promoting a service. A direct effect of this publication action is that the publisher knows that the receiver of the message will know about his capability on this service.

(8) **Service Advertising**

Actor *(a)* $\wedge$ ***Service*** *(s)* $\wedge$ ***Can*** $_a$ *s* $\Rightarrow$ ***tell*** *(a,* ***Can*** $_a$ *s,* ***all)*** $\wedge$ ***Know*** $_a$ ***(Know*** $_{all}$ ***(Can*** $_a$ *s)).*

An actor *a* may broadcast an advertisement of a service with the intention of obtaining relevant information of promoting a service. A direct effect of this publication action is that the publisher knows that the receiver of the message will know about his capability on this service.

Rule 2.5: Knowledge Update Rule

$\exists x \in R \cup C \cup K \cup B$, ***Actor*** *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***tell*** *(a, x, b)* $\Rightarrow$ ***Know*** $_b$ *x.*

An actor will update his Knowledge when receive a message about a requirement, a capability, a piece of knowledge. A direct effect of this action is that the receiver of the message will know about the relevant information.

Rule 2.6 Knowledge Contradiction Resolution Rule

Actor's knowledge from different sources may be contradicting to each other, for more effective decision making based on these knowledge, we need to resolve these contradictions first.

(1) No Contradiction: ***Actor*** *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Know*** $_b$ ***(Know*** $_a$ *x)* $\wedge$ *no* ***Know*** $_b$ *not x* $\Rightarrow$ ***Know*** $_b$ *x.*

If an actor has indirect knowledge about x, and it does not have contradicting knowledge about x, then this knowledge can turn to direct knowledge.

(2) **Ignore:** ***Actor*** *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Know*** $_b$ ***(Know*** $_a$ *x)* $\wedge$ ***Know*** $_b$ *not x* $\Rightarrow$ *no* ***Know*** $_b$ *x* $\wedge$ *no* ***Know*** $_b$ *not x.*

If an actor has indirect knowledge about x, and it does have contradicting knowledge about x, then both pieces of knowledge will be removed from the knowledge base.

(3) Ask public opinion about a contradicting knowledge

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Know*** $_b$ ***(Know*** $_a$ *x)* $\wedge$ ***Know*** $_b$ *not x* $\Rightarrow$ ***tell*** *(b, not x, all).*

If an actor has indirect knowledge about x, and it does have contradicting knowledge about x, then it will broadcast its knowledge about x, to cause a conflict in other actor's knowledge base for a consensus.

(4) Confirm with the sender about a contradicting knowledge:

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Know*** $_b$ ***(Know*** $_a$ *x)* $\wedge$ ***Know*** $_b$ *not x* $\Rightarrow$ ***tell*** *(b, not x, a).*

If an actor has indirect knowledge about x, and it does have contradicting knowledge about x, then it will send its knowledge about x back to the knowledge source, to cause a conflict in other actor's knowledge base for a debate.

(5) Accept the sender's knowledge although contradicting:

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Know*** $_b$ ***(Know*** $_a$ *x)* $\wedge$ ***Know*** $_b$ *not* $x \Rightarrow$ ***Know*** $_b$ *x.*

If an actor has indirect knowledge about x, and it does have contradicting knowledge about x, but if it considers the new indirect information has higher certainty, then it will accept it any ways.

The five rules in Rule 2.6 are alternatives for an actor to resolve knowledge conflict. They are applied according to the preferences and contexts of decision an actor encounters.

Rule 2.7: Service Agreement / Delegation Rule

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Service*** *(s)* $\wedge$ ***Requires*** $_a$ *s* $\wedge$ ***Know*** $_a$ ***(Can*** $_b$ *s)* $\wedge$ ***tell*** *(b, s, a)* $\wedge$ ***satisficing(****a, **f****(b, q,s))*$\Rightarrow$ ***delegate*** *(a, s, b).*

A service agreement is established when an actor *a* has a requirement, and he knows that another actor *b* could provide the service, and also receives a message from *b* about his capability regarding the service. A direct effect of a service agreement is a delegation action.

Rule 2.8: Reciprocal Dependency Rule

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Service*** *(s)* $\wedge$ ***delegate*** *(a, s, b)* $\wedge$ ***delegate*** *(b, s', a)* $\Rightarrow$ ***Requires*** $_b$ *s.*

A delegation will take effect to the delegatee only if he believes that it is reciprocal. That is, he also needs exchange-services from the requestor. In real world case, general exchange for services could be payment, social benefits, etc.

Rule 2.9: Capability Propagation Through Delegation

Actor *(a)* $\wedge$ ***Actor*** *(b)* $\wedge$ ***Service*** *(s)* $\wedge$ ***delegate*** *(a, s, b)* $\wedge$ ***Perform*** $_b$ *s* $\Rightarrow$ ***Can*** $_a$ *s* .

A delegation will take effect to the delegator, only if the delegatee performs the service provisioning procedure. That is, if a delegatee does not deliver the expected services, the fulfillment of the delegator's service request is problematic.

The reasoning procedure to be applied to a service situation $SC = <A, R, C, K>$ is to find a sequence made up of links (k) applied to SC such that for each ***Requires***$_a$ *s* $\rightarrow$ ***Can***$_a$ *s.*

The rules listed above build the basic reasoning structure for the proposed formalism. By pursuing further about usage of quality attributes, other service composition/decomposition rules, formal models with richer expressiveness can be built, and analyzed. For instance, by explicitly representing quality requirements, we will be able to reason about how quality requirements can be used in service selection. By considering scenarios that actor tells false capabilities, knowledge, and beliefs out of malicious intent, we will be able to model trust issues in the service world.

3. Modelling Generic Service Patterns

3.1 A world of one party: the service transformation model

To start, we may think of a strategic capability model with only one actor. An example setting could be the experience of IKEA. The organizational actor has requirements to be fulfilled by itself, e.g., "IKEA makes profit." In the mean time, it possessed some abilities, such as Furniture Design, Manufacture, and Marketing etc. If IKEA is situated in conventional closed enterprise mode, the organization has no one else to rely on in fulfilling its required services. Thus, it has to satisfy life requirements by itself. In such a single party's world, the issue of service turns into self-consciousness to one's own capabilities and knowledge. If its capability and knowledge are sufficient, its goals will be satisfied. One way to put this situation down as an *i** graphical representation is

in [21], and the corresponding formal description and reasoning is as follows:

SC_{10} := (Actor (IKEA), Requires $_{IKEA}$ *Make Profit* , ***Can*** $_{IKEA}$ *Design*, ***Can*** $_{IKEA}$ *Manufacture*, ***Can*** $_{IKEA}$ *Marketing*, ***Knows*** $_{IKEA}$ *{Designing, Manufacturing, Marketing,...}* $\rightarrow$ *Make Profit* ***).***

Applicable rules to SC_{10}: rule 2.3 (1)--new, rule 2.2 (2)--new

Routine 1: step 1 - apply rule 2.3(1) ,

SC_{11} := (..., ***Requires*** $_{IKEA}$ *Design*, ***Requires*** $_{IKEA}$ *Manufacture*, ***Requires*** $_{IKEA}$ *Marketing...).*

Applicable rules to SC_{11}: rule 2.3 (1)--loop, rule 2.2 (1)--new

Routine 1: step 3 - apply rule 2.1,

SC_{12} := (… Perform $_{IKEA}$ *Design*, ***Perform*** $_{IKEA}$ *Manufacture*, ***Perform*** $_{IKEA}$ *Marketing* ***…).***

Applicable rules to SC_{12}: rule 2.3 (1)--loop, rule 2.2 (1)--new, rule 2.1--new

Routine 1: step 4 - apply rule 2.2(1),

SC_{13} := (…Can $_{IKEA}$ *Make Profit…).*

Applicable rules to SC_{13}: rule 2.3 (1)--loop, rule 2.2 (1)--loop, rule 2.1--new

Routine 1: step 4 - apply rule 2.1,

SC_{14} := (… Perform $_{IKEA}$ *Make Profit…).*

No new applicable rule to SC_{14}. End of Routine 1.

Conducting analysis to the model above is to find routines through which an actor can accomplish his required services by means-ends reasoning on required services. As we can see, Routine 1 is one possible answer returned by the service reasoning procedure. A routine consists of services that the actor is capable of performing and the know-how knowledge represented as links in *i**, they can be organized into a rough action plan, and related to the correspondence service requirements.

3.2 From Informal to Formal Strategic Delegation

A Strategic Dependency (SD) model in *i** consists of a set of actors linked together with dependency links. Each dependency link between two actors indicates that one actor depends on the other for certain service such that the former may attain some goal. By depending on another actor, an actor (the depender) is able to achieve goals that it was not able to without the dependency, or not as easily or as well. At the same time, the depender becomes vulnerable. If the depended actor fails to deliver the service, the depender would be adversely affected in its ability to achieve its goals. We are to model generic patterns of service relationships in the following, and study the different situations in the different service-oriented computing environments.

A world of partner: A Service outsourcing model

Now consider the case, in which IKEA expands its business abroad. In a world of partners, we assume that there is no third party and zero advance knowledge is available to either side. Conducting analysis to such models is to find another actor through whom the required services of an actor can be accomplished through *delegation*. The basic assumption is that a capable and trusted actor can be depended on for the fulfillment of a service request an actor has. The model shows the reasoning procedures of the two actors regarding a service situation SC_{20}:

In a physical world, knowledge about the participants of a service relationship can be obtained easily, for instance, Local Furniture Factory sees IKEA getting popular world wide; so it believes that IKEA has the capability of making profit with him together. Such scenario works fine in a closed world where people can meet face-to-face easily. However, when we come to an open world where direct observation and past experience are not available, how do we build a relationship between the service participants? What new problem do we need to deal with?

SC_{20} := (Actor (IKEA), Actor (Local Factory), Requires $_{Local\ Factory}$ *Design*, ***Requires*** $_{IKEA}$ *Manufacture*, ***Can*** $_{IKEA}$ *Design*, ***Can*** $_{LocalFactory}$ *Manufacture)***.**

Applicable rules to SC_{20}: rule 2.4 (4)--new, rule 2.4 (8)--new

Routine 1: step 1 - apply rule 2.4 (5),

*SC_{21} := (…, **tell (**Local Factory, **Requires** $_{Local\ Factory}$ Design, IKEA**), tell (**IKEA, **Requires** $_{IKEA}$ Manufacture, Local Factory**)** …).*

Applicable rules to SC_{21}: rule 2.4 (4)--loop, rule 2.4 (8)--new, rule 2.5-new

Routine 1: step 2 - apply rule 2.5 and rule2.6 (1) ,

*SC_{22} := (…,**Know** $_{IKEA}$ **Requires** $_{Local\ Factory}$ Design**, Know** $_{Local\ Factory}$ **Requires** $_{IKEA}$ Manufacture**,** …).*

Applicable rules to SC_{22}:, rule 2.4 (8)--new,

Routine 1: step 3 - apply rule 2.4(8) ,

*SC_{23}:= (…, **tell (**Local Factory, **Can** $_{Local\ Factory}$ Manufacture, IKEA**), tell (**IKEA, **Can** $_{IKEA}$ Design, Local Factory**),** …).*

Applicable rules to SC_{23}: rule 2.4 (5)--loop, rule 2.4 (9)--loop, rule 2.5--new

Routine 1: step 4 - apply rule 2.5 and rule2.6(1) ,

*SC_{24}:= (…, **Know** $_{IKEA}$ **Can** $_{Local\ Factory}$ Manufacture**, Know** $_{Local\ Factory}$ **Can** $_{IKEA}$ Design**,**…).*

Applicable rules to SC_{24}: rule 2.7--new

Routine 1: step 5 - apply rule 2.7,

*SC_{25}:= (…, **delegate(**IKEA, Manufacture, Local Factory**), delegate (**Local Factory, Design, IKEA**),** …).*

Applicable rules to SC_{25}: rule 2.8--new

Routine 1: step 6- apply rule 2.8,

*SC_{26}:= (…, **Requires** $_{Local\ Factory}$ Manufacture**), Requires** $_{IKEA}$ Design**,** …).*

Applicable rules to SC_{26}: rule 2.1(1)--new

Routine 1: step 7- apply rule 2.1(1),

***SC_{27} := (… Perform** $_{IKEA}$ Design, **Perform** $_{Local\ Factory}$ Manufacture…).*

Applicable rules to SC_{27}: rule 2.9--new

Routine 1: step 8- apply rule 2.9,

SC_{28} := (… Can** $_{IKEA}$ Manufacture, **Can** $_{Local\ Factory}$ Design,…**).

Applicable rules to SC_{28}: rule 2.1(1)--new

Routine 1: step 8- apply rule 2.1(1),

SC_{29} := (…Perform** $_{IKEA}$ Manufacture, **Perform** $_{Local\ Factory}$ Design**,…).

No new applicable rule to SC_{28}. End of Routine 1.

3.3 A World with Deception: A Service Model on Trust

The publication rules set given in Rule 2.4 is based on an assumption that the actors in the system are telling the truth, but this may not be the case in the real world. Assume that there is an actor who lies about his capability to obtain another actor's service. We may extend the framework with action rules such as the following:

Publish false capability:

***Actor** (a) ∧ **Actor** (b) ∧ **Service** (s) ∧ no **Can** $_a$ s ∧ **Know** $_a$ **Requires** $_b$ s ⇒ **tell** (a, **Can** $_a$ s, b).*

The service situation can evolve into the one represented by the following graphical model:

Establish black list:

***Actor** (a) ∧ **Actor** (b) ∧ **Service** (s) ∧ **delegate (**a, s, b **)** ∧ no **perform** $_b$ s ⇒ **Know** $_a$ not **Can** $_b$ s.*

From this model we can see that the proposed formalism can be used to describe different domain assumptions, operational rules in a service environment. By analyzing the differences between systems showing desired properties, and those allowing undesirable behaviors, a designer will be able to build mechanisms reflecting the right control schema.

3.4 A World with Circle of Trust: Service Selection based-on Community Feedback

As mentioned in the previous sections, in an open environment, direct knowledge about others actor is very hard to obtain. And sometimes, decide if another actor is trustworthiness on providing a service are not two value black-or-white assertions, but vectors using discrete values to represent varying levels of confidence. For instance, we may adopt a trust scoring schema to quantify the confidence level of beliefs circulated within the service network.

i. At the beginning, the trust level of all actors is 0.

ii. Whenever an actor successfully delivers a service, its trust level to the service user will be increased by 1.

iii. When an actor fails to deliver a delegated service, its trust level will be decreased by 5 or to –1 whichever is higher.

iv. Whenever an actor recommends a provider who delivers a service successfully, its trust level to the service requestor will be increased by 1.

v. Whenever an actor recommends a provider who fails to deliver a service, its trust level to the service requestor will be decreased by 1.

vi. The confidence level of a recommendation is based on the recommender's confidence to the content, and the recommender 's confidence level to the receiver of the recommendation.

Naturally, we may consider defining a function of each of the knowledge in *K* of a service situation *SC*, whose domain is A ∪ B, with range being Integer:

Rule 2.10: **Trust Function Management Rules**

1. Set initial Trust value between actors (in response to rule (i) above):

Actor *(a)* ∧ ***Actor*** *(b)* ∧ *no f (a, Trust, b)* ⇒ *f(a, Trust, b) = 0*

2. Compute Trust value of a received recommendation (in response to rule (vi) above):

∃*x*∈ *R*∪*C*∪*K*∪*B,* ***Actor*** *(a)* ∧ ***Actor*** *(b)* ∧ ***tell*** *(a, x, b)* ⇒ *f(b, Trust, x) = f(b, Trust, a)* × *f(a, Trust, x).*

∃*x*∈ *R*∪*C*∪*K*∪*B,* ***Actor*** *(a)* ∧ ***Actor*** *(b)* ∧ ***tell*** *(a, x, b)* ∧ ***Know*** $_b$ *x* ⇒ *f'(b,Trust, x) = f(b, Trust, x)* × *f(b,Trust, a)* × *f(a, x).*

3. Compute Trust after a service (in response to rule (ii, iii, iv, v) above):

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ ***perform*** $_b$ *s* ⇒ *f' (a, Trust,* ***Can*** $_b$ *s) =f (a, Trust,* ***Can*** $_b$ *s) +1 .*

∃*x*∈ *R*∪*C*∪*K*∪*B,* ***Actor*** *(a)* ∧ ***Actor*** *(b)* ∧ ***tell*** *(a, x, b)* ∧ *no* ***Perform*** $_b$ *s* ⇒ *f' (a, Trust,* ***Can*** $_b$ *s) = f (a,Trust,* ***Can*** $_b$ *s) –5, if f (a, Trust,* ***Can*** $_b$ *s)* ≥ *4; f' (a,Trust,* ***Can*** $_b$ *s) = -1 , otherwise.*

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Actor*** *(x)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ ***perform*** $_b$ *s* ∧ ***tell*** *(x,* ***Know*** $_x$ ***Can*** $_b$ *s, a)* ⇒ *f' (a, Trust,* ***x****) = f (a, Trust,* ***x****) +1 .*

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Actor*** *(x)* ∧ ***Service*** *(s)* ∧ ***delegate*** *(a, s, b)* ∧ *no* ***perform*** $_b$ *s* ∧ ***tell*** *(x,* ***Know*** $_x$ ***Can*** $_b$ *s, a)* ⇒ *f' (a, Trust,* ***x****) = f (a, Trust,* ***x****) - 1 .*

4. Select a service according to trust level:

Actor *(a)* ∧ ***Actor*** *(b)* ∧ ***Actor*** *(x)* ∧ ***Service*** *(s)* ∧ ***Requires*** $_a$ *s* ∧ ***Know*** $_a$ ***Can*** $_b$ *s* ∧ ***Know*** $_a$ ***Can*** $_x$ *s* ∧ ***tell*** *(b, s, a)* ∧ *f (a, Trust,* ***Can*** $_b$ *s)* ≥ *f (a, Trust,* ***Can*** $_x$ *s)* ≥ *0* ⇒ ***delegate*** *(a, s, b).*

The rules defined above are to illustrate that the proposed formalism can be easily used and extended to represent a qualitative trust management mechanism. Other qualitative or quantitative mechanisms for service representation, evaluation, or management, can be modeled and analyzed by similar means.

4. Related work

The approach proposed in this paper mainly synergizes ideas from three major areas: knowledge representation and reasoning in autonomous agents systems, requirements modeling and analysis, and semantic web services. In conventional knowledge engineering and AI, various subject logics and social ontologies to represent belief, knowledge, desire, and intention of autonomous agents have been proposed [10, 11, 12, 13, 14, 17]. Our work aims to adopt theoretical results from this area and build a practical framework for the service-oriented computing paradigm. Thus, we will mainly focus on the specific needs, assumptions, rules and reasoning mechanism for

the service setting. Existing requirements modeling frameworks [15, 16] emphasize on capture and elicit the requirements in the problem domain. It usually takes a top-down refinement way of thinking. However, the open, dynamic, continuous system environment needs to have a model integrating high-level abstract requirements models with concrete executable service manipulating mechanisms seamlessly. By representing service request and service capability in a compatible ontology, we aim towards a holistic solution to the problem.

The Web Service Modeling Ontology (WSMO) [3] provides a conceptual framework focusing on the functional and behavioral aspects of a Web service. Comparing WSMO, the concepts and reasoning mechanism proposed in this paper emphasis on strategic actor's knowledge and decision making about the capability of other actors, rather than a straightforward description about web services behaviors and constraints. This is based on the assumption that actors participate a service are strategic. That is, an actor has his own intended requirements on service functionality and quality to fulfill, which may only partially knowable to other actors. The ontology proposed in this paper is a natural complementary to DAML-OIL [4], since it describes web services in a higher level of abstraction. Instead of focusing on the static structure of a service implementation, it describes service from a service requestor's perspective, i.e., from the intended usage angle.

QoS attributes are the key to dynamically selecting the services that best meet user needs. In order to supplement the deficiency of lacking effective means for expressing its quality of service, quite a few QoS ontologies have been proposed in recent literature, such as [5], which address dynamic service selection via an agent framework coupled with a QoS ontology. With these approaches, participants can collaborate to determine each other's service quality and trustworthiness. This, in essence, targets at the same goal with our approach. Another related work on non-functional aspects of web services is DAML-QoS [2], which is a complementary to DAML-S ontology for providing a better QoS metrics model. The difference is similar to our analysis above, i.e., their ontologies look service as passive objects, but we consider services as active agents with intentions and preferences. [7, 8, 9] examine the development of generic ontologies for Quality of Service (QoS) by consensus, which can be considered as knowledge and quality evaluation rules in the framework proposed in this paper.

Discovering and assembling individual Web Services into more complex new and user-centric web processes is an important challenge. In [6], Web Services composition techniques by using their ontological descriptions and relationships to other services are proposed. An automatic composition technique is used to check semantic similarities between interfaces of individual services while taking the service qualities into consideration. The ontology proposed in this paper can be used to help the composition of individual services, and also the decomposition of service requirements. Taking such a two-ways thinking, alternative ways to satisfy user's service requirements can be taken into consideration.

In [18], Penserini et al. propose to use the Tropos requirements methodology to support services design, identification, composition, and binding. The concept of service capability is defined as Means-ends links and Contribution links in the *i** framework. Tropos design steps such as goal-decomposition, dependency handshake, are now considered as service-agents' decision making actions. Specifically, top-down goal analysis is used for service identification; bottom-up goal analysis is used for service composition. The idea of using Tropos in service requirements engineering is promising, and having the same basis with this paper. The major difference lies in that capabilities are

defined as links in their work, while capabilities in this paper correspond to the concept of task in *i**, links are considered with knowledge. The incorporation of capability and knowledge have better potential in addressing uncertainty and partial knowledge, and conflict of interest of actors.

5. Conclusion

In this paper, we propose a service requirements ontology that is based on the actors' strategic capability. Although it is a preliminary proposal explaining our ideas for the basic conceptual structure, we feel that unlike other work on service ontology, our proposal focuses on represent explicitly the knowledge and subjective decision-making on service publication, discovery, negotiation, and selection rather than the traditional concept decomposition. Both the formal service requirements ontology and its automatic reasoning rules are given. Example models and reasoning traces are also given to illustrate the usefulness of our proposal. Results from our study are important because it contributes not only to the theoretical study of SOA but also forms the basis for its future implementation and deployment. The proposed model ontology can be easily implemented and extended to support most kinds of automatic reasoning for qualitative or quantitative QoS-based service selection, which including those objective ones, encompassing reliability, availability, and request-to-response time, or those that are fairly subjective focusing on user experience, and preferences.

Acknowledgements.

This work receives financial support from the National Natural Science Foundation of China (Grant no. 60503030), Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList), and the National Basic Research and Development 973 Program (Grant no.2002CB312004).

References

[1] T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Published by Prentice Hall, August 2005.

[2] C. Zhou, L.T. Chia, B.S. Lee. DAML-QoS Ontology for Web Services. Proceedings of the International Conference of Web Services, 2004.

[3] H. Lausen, A. Polleres, D. Roman. Web Service Modeling Ontology (WSMO). W3C Submission, June 2005.

[4] Jeff Heflin, James Hendler. Dynamic Ontologies on the Web, Proceedings of 17th National Conference on Artificial Intelligence (AAAI-2000). 2000.

[5] E. Michael Maximilien, Munindar P. Singh. A Framework and Ontology for Dynamic Web Services Selection. IEEE Internet Computing, September/October 2004, pp.84-93.

[6] I. B. Arpinar, R. Zhang, B. Aleman and A. Maduko. Ontology-Driven Web Services Composition. Proceedings of the IEEE E-Commerce Technology, July 6-9, San Diego, CA, 2004.

[7] G. Dobson, R. Lock. Developing an Ontology for QoS. Proceedings of the 5th Annual DIRC Research Conference, 2005, pp. 128-13

[8] G. Dobson. QoSOnt: an Ontology for QoS in Service-Centric Systems. Proceedings of the Conference on e-Science All Hands Meeting 2005, Nottingham, September 2005.

[9] C. Zhou, L.T. Chia, B.S. Lee. Service Discovery and Measurement based on DAML QoS Ontology. Proceedings of the World Wide Web Conference 2005, pp. 1070-1071.

[10] Rao, A.S. and Georgeff, M.P. (1991). Modeling rational agents within a BDI architecture. In: R. Fikes and E. Sandewall (eds.), Proceedings of the Second Conference on Knowledge Representation and Reasoning, Morgan Kaufman, pp.473-484.

[11] Shoham, Y. and Cousins, S.B. (1994). Logics of mental attitudes in AI: a very preliminary survey. In: G. Lakemeyer and B. Nebel (eds.) Foundations of Knowledge Representation and Reasoning, Springer Verlag, pp. 296-309.

[12] Wooldridge, M. and Jennings, N.R. (1995). Agent theories, architectures, and languages: a survey. In: M. Wooldridge and N.R. Jennings, Intelligent Agents, Lecture Notes in Artificial Intelligence, Vol. 890, Springer Verlag, Berlin, pp. 1-39.

[13] Jaakko Hintikka, et al. Knowledge and Belief: An Introduction to the logic of the two notations. Cornell University Press, 1962.

[14] Yu, E. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97) Jan. 6-8, 1997, Washington D.C., USA. 226-235.

[15] Yu, E. & Liu, L. Modeling Trust for System Design Using the i* Strategic Actors Framework. In: Trust in Cyber-Societies - Integrating the Human and Artificial Perspectives. R. Falcone, M. Singh, Y.H. Tan, eds. LNAI-2246. Springer,2001. pp.175-194.

[16] S. Kethers, G. Gans, D. Schmitz, D. Sier: Modelling Trust Relationships in a Healthcare Network: Experiences with the TCD Framework. In Proceedings of the Thirteenth European Conference on Information Systems, May 2005, Regensburg, Germany.

[17] Cristiano Castelfranchi, Rino Falcone, Giovanni Pezzulo: Cooperating through a Belief-based Trust Computation. WETICE 2003: 263-268.

[18] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. From Stakeholder Intentions to Software Agent Implementations.In *Proceedings of the 18th Conference On Advanced Information Systems Engineering (CAiSE'06)*, number 4001 in LNCS. Springer-Verlag, 2006.

[19] J. Spohrer, D. Riecken. Services Science. *Communications of the ACM, SPECIAL ISSUE: Services science*, Vol. 49 , no. 7 (July 2006), 30 – 32.

[20] Cohen, P. R., & Levesque, It. J. Speech Acts and the Recognition of Shared Plans. *Proc. of the Third Biennial Conference, Canadian Society for Computational Studies of Intelligence*, Victoria, B.C., May, 1980, 263-271.

[21] E. Yu。Strategic Modelling for Enterprise Integration。*Proceedings of the 14th World Congress of International Federation of Automatic Control (IFAC'99)*, July 5-9, 1999, Beijing, China. pp. 127-132. Permagon, Elsevier Science.