Towards a service requirements modelling ontology based on agent knowledge and intentions

Lin Liu*, Qiang Liu and Chi-Hung Chi

School of Software
Tsinghua University
Beijing, 100084, China
E-mail: linliu@tsinghua.edu.cn
E-mail: liuqiang@tsinghua.edu.cn
E-mail: chichihung@tsinghua.edu.cn
*Corresponding author

Zhi Jin

Academy of Mathematics and System Sciences
Chinese Academy of Sciences
Beijing, 100080, China
E-mail: zhijin@amss.ac.cn

Eric Yu

Faculty of Information Studies
University of Toronto
Toronto, M5S 3G6, Canada
E-mail: yu@fis.utoronto.ca

Abstract: In this paper, we propose a formalism for service requirements and capability modelling. It adopts concepts from the agent-oriented requirements modelling framework *i**, which can be used as a means of studying the requirements and architecture for distributed agent systems. We argue that a social modelling framework such as *i**, extended with the necessary service-related concepts and formal reasoning mechanisms, offers a better understanding of the social/organisational relationship in an open services world. By explicitly representing the underlying assumptions and the essential factors of services, a semiformal requirements model in *i** can automatically evolve and be refined into a service requirements and capability reasoning framework. Eventually, it will assist intelligent agents with certain knowledge and intentions to make intelligent, rational decisions during service discovery, publication, selection and binding within an open services community.

Keywords: services; requirements; modelling; ontology; knowledge; trust; Quality of Service; QoS.

Reference to this paper should be made as follows: Liu, L., Liu, Q., Chi, C-H., Jin, Z. and Yu, E. (2008) 'Towards a service requirements modelling ontology based on agent knowledge and intentions', *Int. J. Agent-Oriented Software Engineering*, Vol. 2, No. 3, pp.324–349.

Biographical notes: Dr. Lin Liu is currently an Associate Professor affiliated with the School of Software, Tsinghua University, Beijing, China. Her current research interests include requirements engineering, services engineering and security modelling.

Qiang Liu is an Associate Professor at the School of Software, Tsinghua University, China. Her current research interests are software engineering, project management and collaborative working.

Dr. Chi-Hung Chi is currently a Professor at the School of Software, Tsinghua University, China. His research interests are internet services engineering, content engineering and network systems and architecture.

Dr. Zhi Jin is a Professor of Computer Science at the Academy of Mathematics and Systems Science, the Chinese Academy of Sciences, China. Her current research focuses on requirements engineering and knowledge engineering.

Dr. Eric Yu is an Associate Professor with the Faculty of Information Studies, the University of Toronto, Canada. His research focuses on requirements engineering, information systems engineering and business and organisation modelling.

1 Introduction

During recent years, the Service-Oriented Architecture (SOA) (Erl, 2005) has been recognised as the best way to build complex systems within a short time. It can adapt to changes rapidly and provide effective inter- and intra- organisation application integration. A general operational model includes three different kinds of actors: the service requestors searching for needed services, the service providers who publish and provide services, and the service registry that supports the match-making process between requestors and providers. This framework assumes that the service providers know what services should be offered to the requestors, while the service requestors would expect that the requested service is available from some of the providers. When there is no acceptable match between the service request and the published service description, there is no other action can be taken.

Recent development of service-oriented computing integrates more perspectives including generic service descriptions of semantics, allowing some service registry to take semantic measures to find nearby services that might be acceptable to the requestors. The potential to achieve dynamic, scalable and cost-effective infrastructure for electronic transactions in business and public services has driven many recent research efforts towards enriching web services with semantics. Ontology plays a key role in providing machine readable vocabularies used by applications to understand the shared meanings. So far, many service ontology description languages have been proposed, *e.g.*, SWSL (Battle *et al*., 2005), OWL-S, and earlier, DAML-S, and LARKS (Lu and Yu, 2007; Martin *et al.*, 2006; Mandell and McIlraith, 2003; McGuinness and da Silva, 2004; Mylopoulos, 1998; Sycara *et al*., 2002; Wang *et al*., 2006). These are active research projects to address the problem of services interoperability and match-making, but we feel that they are not yet adequate to solve the current problem of modelling services

requirements. The main issue is that in each of the ontology language, services are modelled and analysed at a certain level of abstraction, without a holistic treatment that could connect high-level abstract goals of services to concrete service operations. For instance, LARKS mainly handles service requests at the level of database queries, and OWL-S mainly deals with service requests at the level of operations with input, output, and pre- and post- conditions, while SWSL mainly focuses on the formalisation of services as objects with logic approaches.

In this paper, we are not proposing yet another ontology description language, but a modelling ontology for matching up service requests from the service requestor with the service capability description from the service provider within an open network of services. The proposed approach uses concepts and strategies from an agent oriented requirements modelling framework, *i**, which can potentially support a more flexible 'goal-driven' match-making process. Here 'goal-driven' match-making means that we view service requestors and providers as agents with intent who will look for alternative ways to find a suitable match if there is no immediate match. It is a generic modelling framework that allows representation of service requirements and capability at different levels of abstraction, which can be used to describe the rules and assumptions that drive the automatic service discovery and selection processes as well as the binding for the service level agreement. We consider this a critical step for the success of SOA.

It is developed based on the previous work in requirements engineering using a social ontology to model and analyse service relationships among strategic actors (Liu *et al*., 2006). That is, we consider that web services are software agents who have their own requirements to fulfil, and who have certain core competence, as well as common and special abilities or knowledge about how to fulfil another agent's requirements or to extend another agent's capability. As the agents form social networks to serve their own and others' interests, the issues of delegation, trust, security, and privacy have to be taken into consideration as well.

In this paper, we aim towards building an automated services representation framework based on *i** concepts. Its unique feature is that it allows the existing modelling constructs of *i** language to be mapped into elements of a service requirements and capability reasoning framework. More importantly, the relevant SOA activities from service publishing, requesting, discovery, and selection to binding will be captured by the framework to allow automatic service discovery and composition. This service requirements and capability modelling ontology contributes not only to the theoretical study of SOA but also forms the basis for a possible service application deployment structure.

2 Service requirements modelling ontology: a formal service requirements ontology based on *i **

In order to let heterogeneous service agents communicate with each other, we need a common service requirement and capability description language before service publication, request and match-making activities take place. The main desired characteristics of the language include: expressiveness, inference capability, ability to be easily understood and used and suitability for web-based open services environments.

2.1 i Framework*

The *i** framework is an agent-oriented requirements engineering approach (Yu, 2001; 1997). Agents attribute intentional properties such as goals, beliefs, abilities, and commitments to each other. Agents consider alternative configurations of dependency networks to assess their strategic positioning in a social context. The framework is used in contexts in which there are multiple autonomous units with strategic interests.

We adopted the basic modelling concepts and their definitions in *i**: actor, goal, task, resource and softgoal. An actor in *i** () is an active entity that carries out actions to achieve its goals by exercising its capability and knowledge. A *goal* in *i** () is a condition or state of affairs in the world that an actor would like to *achieve*, *maintain*, or *avoid*. Usually, a goal is only a rough sketch of the end result, leaving space for negotiation, elicitation and refinement. A *task* in *i** () is used to represent the specific procedures to be performed by actors, and the task specifies particular ways of doing something. A *resource* in *i** () is a physical or informational entity, which may serve particular purposes. Properties of an entity include whether it is available or not. A *softgoal* in *i** () is used to define a quality or non-functional requirements where subjective judgements of the modeller are needed.

We have also applied the following relationships and their definitions in *i**: means-ends, decomposition, and contribution. A *means-ends* relationship is used to connect a goal with a task, and indicates that the goal can be achieved after the task is performed. In *i**, connected to a goal by a means-ends link (), each task is one possible way of achieving the goal. A *decomposition* relationship () is used to connect a task with its sub-components. The subcomponents could be a goal, a task, a resource, or a softgoal. A *contribution* (→) in *i** describes the elaboration of a quality softgoal into more concrete softgoals, or the operationalisation of a softgoal into tasks having an impact on it. The impact can be positive or negative, partial or sufficient. The following aliases are used to represent possible types of contributions: *Make = (full, positive)*  *; Help =(partial, positive)*  *; Some+ = (positive, unknown degree)*  *; Undecided = (unknown, unknown degree)* *; Some- = (negative, unknown degree)*  *; Hurt = (partial, negative)**; Break = (full, negative)*  . The partial order of the above types is: *Make ≥ Help ≥ Some + ≥ Undecided ≥ Some – ≥ Hurt ≥ Break.* Other qualitative or quantitative measurements can be used as a scale of contributions (Chung *et al*., 2000).

In *i**, a *dependency* link () is used to describe a strategic dependency relationship. The relationship is strategic since the actors may decide to support the dependency or break the dependency according to their own interests. In *i**, a belief () construct is used to represent an actor's knowledge or perception of other actors, domain characteristics, design assumptions and relevant environmental conditions.

The *i** framework supports two kinds of modelling: Strategic Dependency (SD) modelling and Strategic Rationale (SR) modelling. The strategic dependency model assumes that actors form dependency networks to achieve their goals, perform their tasks, free desired resources, and thus the desired softgoals can be reached with the help of others. The strategic rationale model explores the internal reasoning structure of an actor to reveal how high-level goals and tasks can be decomposed, so that alternative solutions are identified and evaluated. Figure 1 shows an SD model on SOA, and Figure 2 shows an SR model of a service provider.

Figure 1 A strategic dependency model in *i** on SOA (see online version for colours)

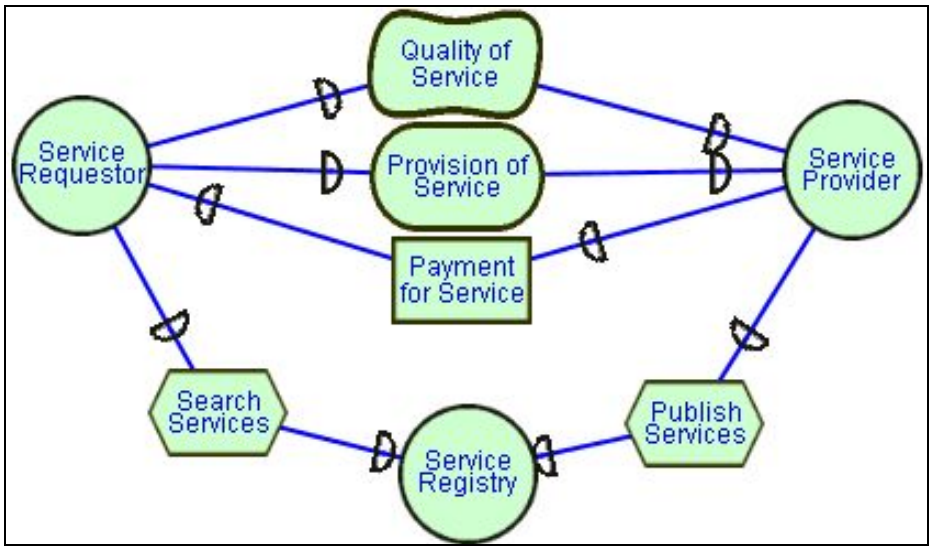
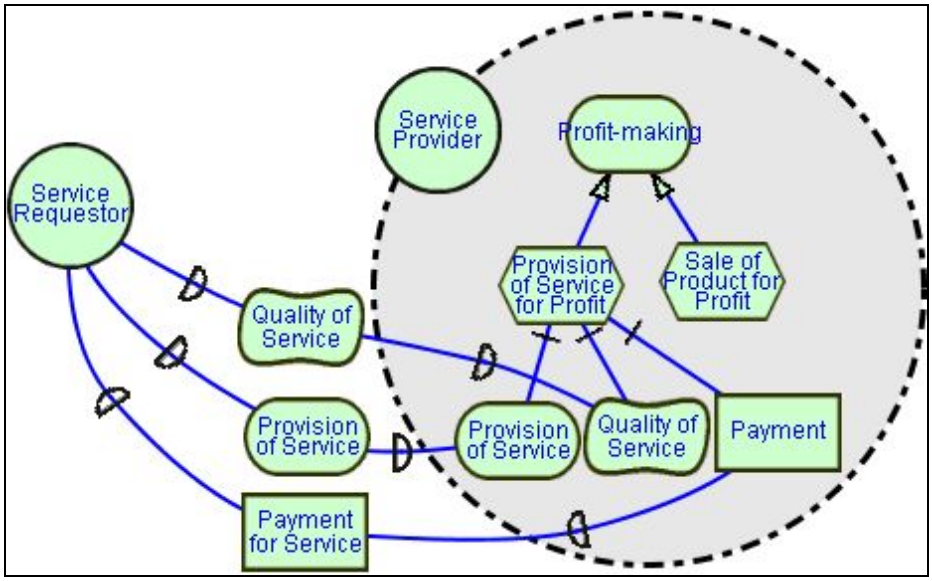


Figure 2 A strategic rationale model of a service provider in *i** (see online version for colours)

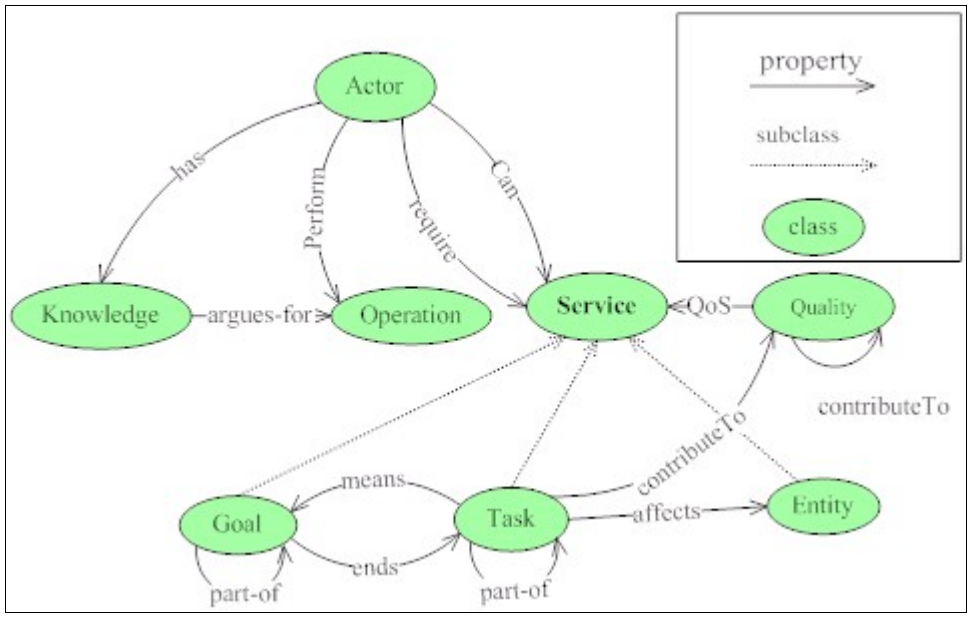


2.2 Ontology concepts in service requirements modelling ontology

Based on the concepts introduced above from the *i** framework, we may build models to analyse the strategic relationships between actors in the services world. Each actor has knowledge of the other actors' capabilities and requirements, and also knows how goals could be addressed by other tasks, how tasks could be decomposed, how resources can be obtained, and how quality attributes are assured. An actor can also delegate a service to

another actor, and inform other actors of its requirements, capability and knowledge. To move towards automated support for services manipulation, we built a formalism based on these concepts with necessary extensions to establish a services analysis mechanism.

Figure 3 Service Requirements Modelling Ontology (SRMO) (see online version for colours)



Definition 1 $A = \{a_1, \ldots, a_n\}$ is a set of *Actors*. If $a \in A$, we write: ***Actor*** (a).

Definition 2 $S = G \cup T \cup E$ is a set of *Services*. Textually, if $s \in S$, we can also represent it as: ***Service*** (s). There is no service concept in *i**. We also enforce that each instance of service should belong to one of the four subtypes of services. Services expressed as abstract objectives are modelled as goals; Services defined as concrete procedure and implementation can be modelled as tasks; Services related to content provisioning are modelled as resources; services related to non-functional, quality attributes are modelled as softgoals. Here we import these relevant concepts and redefine their meaning in the service setting.

- $G = \{g_1, \ldots, g_n\}$ is a set of *Service goals*. If $g \in G$, we write ***Goal*** (g).
- $T = \{t_1, \ldots, t_n\}$ is a set of *Service tasks*. If $t \in T$, we write ***Task*** (t). Tasks are decomposable based on process-related operators, including the ***parallel*** operator, ***sequence*** operator, and ***selection*** operator, to name a few.
- $E = \{e_1, \ldots, e_n\}$ is a set of *service entities* (*resources in i**). If $r \in R$, we write ***entity*** (r). Properties of an entity include whether it is available or not, the value of its quality attributes, and its non-intentional properties such as amount, producer, copyright owner, colour, length, *etc*.

Definition 3 $Q = \{q_1, \ldots, q_n\}$ is a set of *quality attributes*. If $q \in Q$, we can also write ***Quality*** (q). A quality attribute could be any attribute that is of interest to an actor requesting or providing a service, such as, the cost of a service, performance, security/privacy assurance, ease of use, *etc*. In other words, anything within the scope of QoS can be described.

2.3 Ontology relations in service requirements modelling ontology

Definition 4 $QoS \subseteq Q \times S$ is a set of *quality of service relations*. We use ***QoS*** (q, s) to denote the quality attribute *q of service s*, $q \in Q, s \in S$.

Definition 5 $R \subseteq A \times \{S \cup QoS\}$ is a set of *require* relations. We use ***requires*** $_a$ s to denote that there exists an actor a that requires certain service *or quality s*, *i.e.*, $r(a, s) \in R$. To represent this relation, we extend the *i** graphical notation as follows: position the service icon (of the goal, task, resource, or softgoal) within the actor's dotted boundary, and labelling it with a question mark (?).

Definition 6 $C \subseteq A \times \{S \cup QoS\}$ is a set of *capability* relations. We use ***can*** $_a$ s to denote that there exists an actor a that can provide a certain service or *quality s*, *i.e.*, $c(a, s) \in C$. To represent this relation, we extend the *i** graphical notation as follows: position the service icon (of the goal, task, or resource) within the actor's dotted boundary, and labelling it with a small 'c' on the top right corner. There is also $QC \subseteq A \times Q \times S \times Int$, representing the set of quality-related services that can be provided by *a.* It is often represented with a quality function $f: A \times Q \times S \rightarrow Int$.

Definition 7 $ME \subseteq T \times G$ is a set of *means-ends* relationships. If $me(t, g) \in ME$, we write ***means-ends*** (t, g). $DC \subseteq \{S \cup QoS\} \times S$ is a set of *decomposition* relationships. If $dc(s, t) \in DC$, we write ***part-of*** (s, t). *Contribution* relationship: $CN \subseteq \{QoS \cup S\} \times QoS \times CT$ is a set of *Contribution* relationships. $CT = \{$*positive*, *negative*, *unknown*$\} \times \{$*full*, *partial*, *unknown degree*$\}$ is a set of *contribution types*. Textually, we can represent it as: ***contributes-to*** (s, ct, qos).

Definition 8 $K \subseteq A \times \{A \cup S \cup R \cup C \cup ME \cup DC \cup CN\}$ is a set of *Knowledge*. We use ***Know*** $_a$ x to denote that there exists an actor a who has knowledge about a certain fact x, *i.e.*, $k(a, x) \in K$.

To represent this relation, we extend the *i** graphical notation as follows. We model such relationships in two subcategories: practical knowledge about means-ends, decompositions, and contributions relationships between services; knowledge about other actors' requirements, capability, and basic attributes of a service. Graphically, an actor's practical knowledge of a service relationship is shown as a link defined in Section 2.1. Knowledge about other actor's requirements and capability is represented as beliefs. Logic operators such as: $\neg$, $\wedge$, $\vee$ are applicable to construct more complex knowledge assertions.

Definition 9 $O = \{o_1, \ldots, o_n\}$ is a set of *Operations* applicable to a service situation $SC = <A, R, C, K>$. There are the following basic types of operation constructs: ***new***, ***remove***, ***delegate***, ***tell*** and ***commit***.

- ***newActor***(x)
- ***removeActor***(x)
- ***newRequests***(x)
- ***removeRequests***(x)
- ***newCapability***(x)
- ***removeCapability***(x)
- ***newKnowledge***(x)
- ***removeKnowledge***(x)
- ***delegate*** (x, y, z), represents that there is an inter-actor delegation, where $x, z \in A$, $y \in S$
- ***tell*** (x, y, z), represents an inter-actor *communication* sharing knowledge, where $x, z \in$ A, $y \in K$
- ***commit*** $_x y$, represents a *service agreement*, where $x \in A$, $y \in S$.

Similar to the dependency concept in *i**, for each delegation operation, we call the delegating actor the *delegator*, and the actor to whom the task is delegated is designated as the *delegatee*. By delegating a service to another actor, an actor (the delegator) can achieve goals that it could not have achieved without the delegation, or it could not have achieved as easily or as well. At the same time, the delegator becomes vulnerable. If the delegatee fails to deliver the service, the delegator will be adversely affected in its ability to achieve its goals. We extend the *i** graphical model with a notation (→◌→) to represent a communication action: ***tell***. Graphically in SRMO, a service commitment is represented by labelling the service icon with a check mark (✔).

Definition 10 $AR = K \times O \times CT$ is a set of *arguments*. Textually, we can represent it as: ***argues-for*** (k, o, ct), where $k \in K$, $o \in O$, $ct \in CT$. Graphically, we use dashed arrows links to express the argumentation relationship between an operation and the knowledge supporting it.

2.4 Service action rules based on service requirements modelling ontology

A world of services is an open environment, in which each of the above element sets can be updated dynamically. In other words, actors will join and leave the environment; new requests will be issued and removed by actors; capabilities will be obtained and will expire, and knowledge about service relationships will be obtained and discarded by actors. In such a highly dynamic and distributed environment, automated service discovery, service agreement formation, and service selection need to be manipulated by certain machine operable rules and policies. Below, the rules applicable to a service situation: $sc_i = <A, R, C, K>$ are defined.

Rule 2.1 Service commitment

If an actor a is capable of providing a service s, and it also needs the service, it should commit to the service.

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Task}}(s) \wedge \textbf{\textit{Can}}_{\ a}\ s \wedge \textbf{\textit{Requires}}_{\ a}\ s \Rightarrow \textbf{\textit{commit}}_{\ a}\ s \wedge \textbf{\textit{removeRequest}}\ (\textbf{\textit{Requires}}_{\ a}\ s).$$

Rule 2.2 Service composition/transformation

If an actor a is capable of providing a service s_1, and it also has knowledge of how to compose or transform it into another more complicated or simpler service s_0, then it will be able to provide the transformed service s_0.

(1) $\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}\ (s_0) \wedge \textbf{\textit{Know}}_{\ a}\ (\textbf{\textit{means-ends}}\ (s_1, s_0)) \wedge \textbf{\textit{Can}}_{\ a}\ s_1 \Rightarrow \textbf{\textit{newCapability}}(\textbf{\textit{Can}}_{\ a}\ s_0)$

(2) $\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}\ (s_0) \wedge \textbf{\textit{Service}}\ (s_1) \wedge \textbf{\textit{Know}}_{\ a}\ (\textbf{\textit{part-of}}\ (s_0, s_1)) \wedge \textbf{\textit{Can}}_{\ a}\ s_1 \Rightarrow \textbf{\textit{newCapability}}(\text{Can}_{\ a}\ s_0)$.

Rule 2.3 Request decomposition/transformation

If an actor a requires a services s_0, and it also has knowledge of how to decompose or transform it into another more concrete services s_1, then it can send a request for those transformed services or component services instead.

(1) $\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}\ (s_0) \wedge \textbf{\textit{Service}}\ (s_1) \wedge \textbf{\textit{Know}}_{\ a}\ (\textbf{\textit{part-of}}\ (s_1, s_0)) \wedge \textbf{\textit{Requires}}_{\ a}\ s_0 \Rightarrow \textbf{\textit{newRequest}}(\textbf{\textit{Requires}}_{\ a}\ s_1)$.

(2) $\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}\ (s_0) \wedge \textbf{\textit{Service}}\ (s_1) \wedge \textbf{\textit{Know}}_{\ a}\ (\textbf{\textit{means-ends}}\ (s_1, s_0)) \wedge \textbf{\textit{Requires}}_{\ a}\ s_0 \Rightarrow \textbf{\textit{newRequest}}(\textbf{\textit{Requires}}_{\ a}\ s_1)$.

Rule 2.4 Service publication constraints

An actor a may inform other actors about its request, and its capability of providing a service. The rules given below show a possible strategy an actor may take during the 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

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Service}}\ (s) \wedge \textbf{\textit{Requires}}_{\ a}\ s \wedge \textbf{\textit{Know}}_{\ a}\ (\textbf{\textit{Can}}_{\ b}\ s) \Rightarrow \textbf{\textit{tell}}\ (a, \textbf{\textit{Requires}}_{\ a}\ s, b) \wedge \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{\ b}\ \textbf{\textit{Requires}}_{\ \text{a}}\ \text{s}).$$

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 discover about his requirement for this service. This rule only considers the knowledge update from the publisher's side; the knowledge update on the receiver's side is addressed by Rule 2.5.

(2) Publish request to an expert on service transformation/composition/decomposition

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Service}}\ (s) \wedge \textbf{\textit{Requires}}_{\ a}\ s \wedge \textbf{\textit{Know}}_{\ a}\ (\textbf{\textit{Know}}_{\ b}\ (s)) \Rightarrow \textbf{\textit{tell}}\ (a, \textbf{\textit{Requires}}_{\ a}\ s, b) \wedge \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{\ b}\ \textbf{\textit{Requires}}_{\ a}\ s).$$

An actor *a* may publish a request to a known expert who has knowledge of a service's composition, decomposition, or transformation, with the intention of learning the relevant steps for fulfilling a service. A direct effect of this publication action is that the publisher knows that the receiver of the message will become aware of his need for this service.

(3) Publish request to service registry (or other information intermediary)

$$\textbf{\textit{Actor}}\,(a) \wedge \textbf{\textit{Actor}}\,(b) \wedge \textbf{\textit{Actor}}\,(x) \wedge \textbf{\textit{Service}}\,(s) \wedge \textbf{\textit{Requires}}_{a}\, s \wedge \textbf{\textit{Know}}_{a}\,(\textbf{\textit{Know}}_{b}\,((\textbf{\textit{Requires}}_{x}\, s) \vee (\textbf{\textit{Can}}_{x}\, s) \vee (\textbf{\textit{Know}}_{x}\, s))) \Rightarrow \textbf{\textit{tell}}\,(a, \textbf{\textit{Requires}}_{a}\, s, b) \wedge \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{b}\, \textbf{\textit{Requires}}_{a}\, s).$$

An actor *a* may publish a request to a known information centre, which might be a web services registry, or simply another actor, who has knowledge about the capabilities, requests, or knowledge of other unknown actors, with the intention of discovering relevant information to fulfil a service. A direct effect of this publication action is that the publisher knows that the receiver of the message will find about his need for this service.

(4) Request broadcasting

$$\textbf{\textit{Actor}}\,(a) \wedge \textbf{\textit{Service}}\,(s) \wedge \textbf{\textit{Requires}}_{a}\, s \Rightarrow \textit{For}\ \textbf{\textit{all}}\ a' \in \textbf{\textit{Actor}}, \textbf{\textit{tell}}\,(a, \textbf{\textit{Requires}}_{a}\, s, a') \wedge \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{a'}\, \textbf{\textit{Requires}}_{a}\, s).$$

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

(5) Publish service to known requestor

$$\textbf{\textit{Actor}}\,(a) \wedge \textbf{\textit{Actor}}\,(b) \wedge \textbf{\textit{Service}}\,(s) \wedge \textbf{\textit{Can}}_{a}\, s \wedge \textbf{\textit{Know}}_{a}\,(\textbf{\textit{Requires}}_{b}\, s) \Rightarrow \textbf{\textit{tell}}\,(a, \textbf{\textit{Can}}_{a}\, s, b) \wedge \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{b}\, \textbf{\textit{Can}}_{a}\, s).$$

An actor *a* may publish a service to a known requestor, 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 find out about his capability to perform this service.

(6) Publish service to known expert on service composition/transformation

$$\textbf{\textit{Actor}}\,(a) \wedge \textbf{\textit{Actor}}\,(b) \wedge \textbf{\textit{Service}}\,(s) \wedge \textbf{\textit{Can}}_{a}\, s \wedge \textbf{\textit{Know}}_{a}\,(\textbf{\textit{Know}}_{b}\, s) \Rightarrow \textbf{\textit{tell}}\,(a, \textbf{\textit{Can}}_{a}\, s, b) \wedge \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{b}\, \textbf{\textit{Can}}_{a}\, s).$$

An actor *a* may publish a service to a known expert, who has knowledge of the service composition, decomposition, or transformation, with the intention of discovering the 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 discover his capability to perform this service.

(7) Publish service to information intermediary

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Actor}}\ (x) \wedge \textbf{\textit{Service}}\ (s) \wedge \textbf{\textit{Can}}_{\ a}\ s \wedge \textbf{\textit{Know}}_{\ a}\ (\textbf{\textit{Know}}_{\ b}\ ((\textbf{\textit{Requires}}_{\ x}\ s) \vee (\textbf{\textit{Can}}_{\ x}\ s) \vee (\textbf{\textit{Know}}_{\ x}\ s))) \Rightarrow \textbf{\textit{tell}}\ (a, \textbf{\textit{Can}}_{\ a}\ s, b) \wedge \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{\ b}\ \textbf{\textit{Can}}_{\ a}\ s).$$

An actor a may publish a service to a known information centre, which might be a web services registry or simply another actor who has knowledge of the capabilities, requests, or knowledge of other unknown actors, with the intention of revealing relevant information of promote a service. A direct effect of this publication action is that the publisher knows that the receiver of the message will discover his capability to perform this service.

(8) Service advertising

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Service}}\ (s) \wedge \textbf{\textit{Can}}_{\ a}\ s \Rightarrow \textit{For}\ \textbf{\textit{all}}\ a' \in \textit{Actor}, \textbf{\textit{tell}}\ (a, \textbf{\textit{Can}}_{\ a}\ s, a') \wedge \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{\ a'}\ \textbf{\textit{Can}}_{a}\ s).$$

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

Rule 2.5 Knowledge update rule

$$\exists x \in R \cup C \cup K, \textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{tell}}\ (a, x, b) \Rightarrow \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{\ b}\ x).$$

An actor will update his knowledge upon receiving a message about a requirement, a capability, or a piece of information. A direct effect of this action is that the receiver of the message will discover the relevant information.

Rule 2.6 Knowledge contradiction resolution rule

An actor may receive contradicting knowledge from different sources. For more effective decision-making based on these knowledge, we need to resolve the contradictions listed below first.

(1) No contradiction

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Know}}_{\ b}\ (\textbf{\textit{Know}}_{\ a}\ x) \wedge \textit{no}\ \textbf{\textit{Know}}_{\ b}\ \textit{not}\ x \Rightarrow \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{\ b}\ x).$$

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

(2) Ignore the contradiction

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Know}}_{\ b}\ (\textbf{\textit{Know}}_{\ a}\ x) \wedge \textbf{\textit{Know}}_{\ b}\ \textit{not}\ x \Rightarrow \textbf{\textit{removeKnowledge}}(\textbf{\textit{Know}}_{b}\ x) \wedge \textbf{\textit{removeKnowledge}}(\textbf{\textit{Know}}_{\ b}\ \textit{not}\ x).$$

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

(3) Refer to public opinion about the contradiction

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Know}}_{\ b}\ (\textbf{\textit{Know}}_{\ a}\ x) \wedge \textbf{\textit{Know}}_{\ b}\ not\ x \Rightarrow \textbf{\textit{tell}}\ (b, not\ x, all).$$

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

(4) Check with the sender to confirm the contradicting knowledge

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Know}}_{\ b}\ (\textbf{\textit{Know}}_{\ a}\ x) \wedge \textbf{\textit{Know}}_{\ b}\ not\ x \Rightarrow \textbf{\textit{tell}}\ (b, not\ x, a).$$

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

(5) Accept the sender's knowledge although contradicting

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Know}}_{\ b}\ (\textbf{\textit{Know}}_{\ a}\ x) \wedge \textbf{\textit{Know}}_{\ b}\ not\ x \Rightarrow \textbf{\textit{newKnowledge}}(\textbf{\textit{Know}}_{\ b}\ x).$$

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

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 the decision the actor encounters.

Rule 2.7 Service agreement/delegation rule

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Service}}\ (s) \wedge \textbf{\textit{Requires}}_{\ a}\ s \wedge \textbf{\textit{Know}}_{\ a}\ (\textbf{\textit{Can}}_{\ b}\ s) \wedge \textbf{\textit{tell}}\ (b, s, a) \wedge \textbf{\textit{satisficing}}(a, f\ (b, q, s)\ ) \Rightarrow \textbf{\textit{delegate}}\ (a, s, b).$$

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

Rule 2.8 Service motivation propagation based on mutual benefit

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Service}}\ (s) \wedge \textbf{\textit{delegate}}\ (a, s, b) \wedge \exists cn \in CN, \exists s' \in S, \textbf{\textit{Requires}}_{\ b}\ s' \wedge \textbf{\textit{know}}_{\ b}\ cn(s, s', positive) \Rightarrow \textbf{\textit{newRequest}}\ (\textbf{\textit{Requires}}_{\ b}\ s).$$

A delegation will be applied to the delegatee only if it believes that the delegation is beneficial. That is, the delegation will occur when committing to the service can help him satisfy his own requirements. In a real world scenario, this required service could be a general one, such as payments, social benefits, *etc*.

Rule 2.9 Capability propagation through delegation

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Service}}\ (s) \wedge \textbf{\textit{delegate}}\ (a, s, b) \wedge \textbf{\textit{Commit}}_{\ b}\ s \Rightarrow \textbf{\textit{newCapability}}(\textbf{\textit{Can}}_{\ a}\ s).$$

A delegation will be applied to the delegator, only if the delegatee agrees to perform the service provisioning procedure. That is, if a delegatee does not deliver the expected services, the fulfilment 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 an action sequence such that for each $\boldsymbol{Requires}_{a\ s}$, eventually there is a $\boldsymbol{Commit}_{a'\ s}$.

The rules listed above illustrate how the proposed ontology can be used to represent the basic policies of an SOA setting. By exploring the possible usage of quality attributes, we can describe other service composition/decomposition rules, obtain a richer set of policies for establishing precedence, clarify ambiguity and resolve conflict. For instance, by explicitly representing quality requirements, we can logically determine how quality requirements can be used in service selection. By looking into scenarios in which an actor reveals false capabilities, knowledge, and beliefs, we will be able to model trust issues in the service world. In this paper, knowledge is managed and applied in a lighter-weight way comparing to the higher-order logic-based approach of agent knowledge and intent as in Hintikka *et al*. (1962), Levesque and Lakemeyer (2001) and Lausen *et al*. (2005).

3 Modelling social rationale and networking in the services world

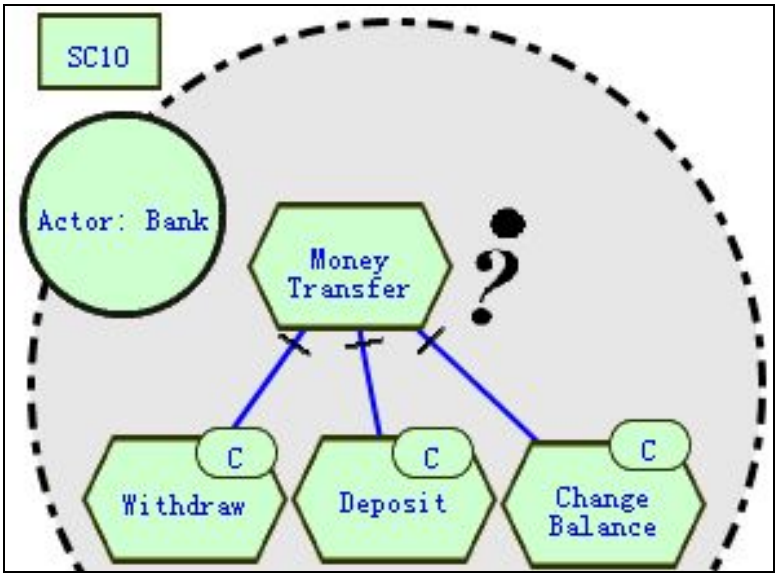
In an open service environment, a major issue to be addressed is how the actors can form social network to obtain required services and to make use of their own capability. This paper selects four typical stages of services networking to show how the proposed ontology and operation rules can be applied when a service network has one single actor, or a pair of actors, or actors with brokers or actors forming communities. The benefit of modelling different kinds of service worlds is that each of the stages covers one particular aspect of the service capability modelling framework. The service world in reality is usually a combination of all the scenarios. It often evolves from one world to another under different service relationships.

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 a money transferring service. The service actor has requirements to be fulfilled on its own, *e.g.*, 'transfer money from Buyer's bank to Seller's bank'. In the meantime, the actor possesses some abilities, such as Withdrawal, Deposit and Change Balance, *etc*. If this service is situated in the conventional closed enterprise mode, the organisation has no one else to rely on to fulfil its required services. Thus, it has to satisfy the requirements by itself. In such a single actor's world, the issue of service involves self-consciousness of the actor's own capability and knowledge. If the organisation's capability and knowledge are sufficient, its goals will be satisfied. One way to put this situation down in *i** graphical representation is shown in Figure 1, and the corresponding formal description and reasoning is as follows.

Under this situation, an actor Bank requests the Money Transfer service, and it can provide the Withdraw, Deposit, and Change Balance service at the same time. It knows that through the three component services, the Money Transfer service can be performed.

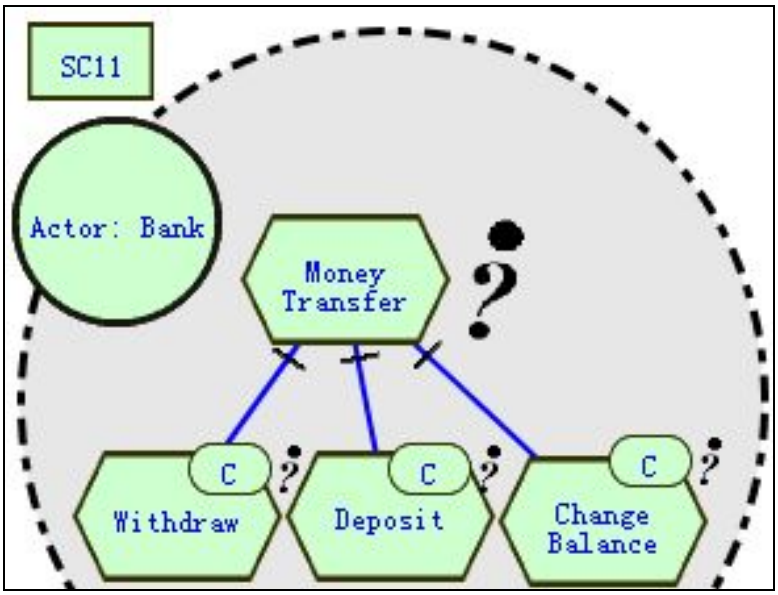
SC*$_{10}$**: = (newActor*** (Bank), ***newServiceRequirements***(***Requires*** $_{Bank}$ *Money Transfer*), ***NewServiceCapability***(***Can*** $_{Bank}$ *Withdraw*, ***Can*** $_{Bank}$ *Deposit*, ***Can*** $_{Bank}$ *Change Balance*), ***NewKnowledge***(***Knows*** $_{Bank}$ ***Part-of***({*Withdraw*, *Deposit*, *Change Balance*}, *Money Transfer*)). (see online version for colours)



Routine 1

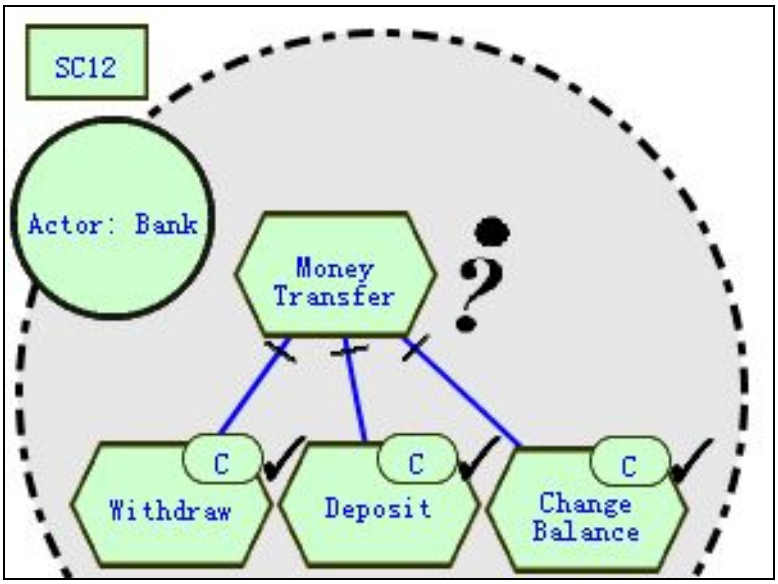
Step 1 Apply Rule 2.3(1) to SC_{10}: request decomposed.

SC*$_{11}$**: = (…, ***Requires $_{Bank}$ *Withdraw*, ***Requires*** $_{Bank}$ *Deposit*, ***Requires*** $_{Bank}$ *Change Balance*,…). (see online version for colours)



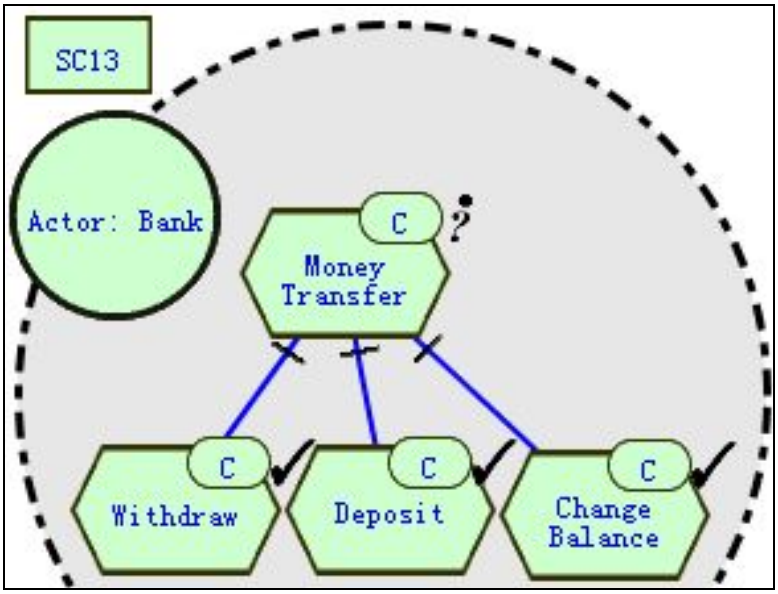
Step 2 Apply Rule 2.1 to SC_{11}: component services committed.

SC_{12}: = (… ***Commit***$_{Bank}$ *Withdraw*, ***Commit*** $_{Bank}$ *Deposit*, ***Commit*** $_{Bank}$ *Change Balance*, ***removeRequest***(***Requires*** $_{Bank}$ *Withdraw*, ***Requires*** $_{Bank}$ *Deposit*, ***Requires*** $_{Bank}$ *Change Balance*) …). (see online version for colours)



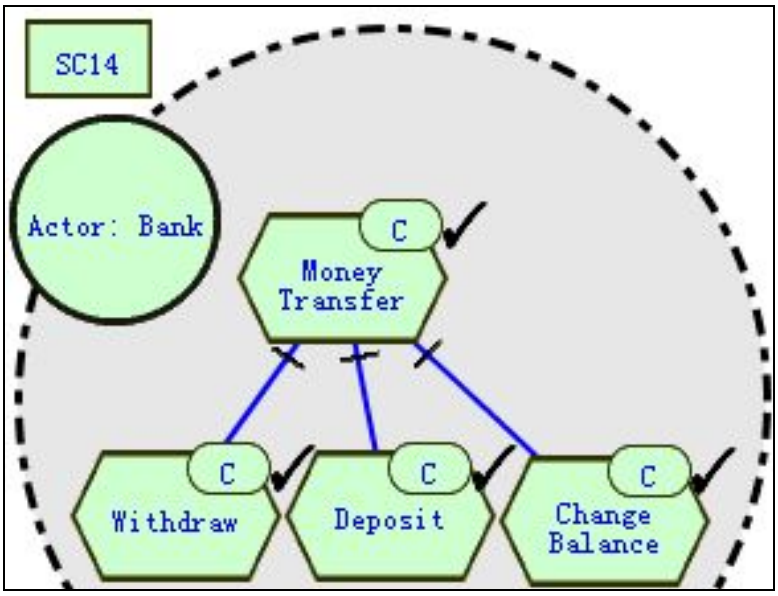
Step 3 Apply Rule 2.2(2) to SC_{12}: capability propagated.

SC_{13}: = (… ***Can*** $_{Bank}$ *Money Transfer*…). (see online version for colours)



Step 4 Apply Rule 2.1 to SC_{13}: requested composite service committed.

$\boldsymbol{SC_{14}}$: = (… ***Commit*** $_{Bank}$ *MoneyTransfer*, ***removeRequest***(*Requires* $_{Bank}$ *MoneyTransfer*)…). (see online version for colours)



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

The model above can be analysed by finding routines through which an actor can accomplish the required services by task decomposition of the 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, and the practical knowledge is represented as links. They can be organised into a rough action plan, and are related to the correspondence service requirements.

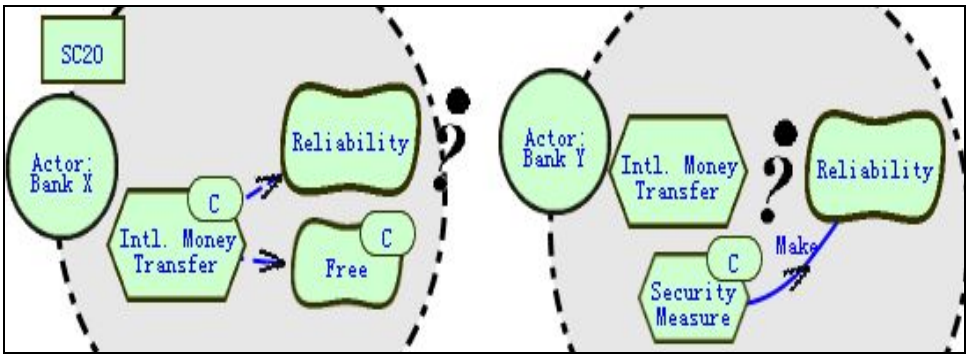
3.2 A world of partners: a service outsourcing model

Now consider the case in which a money transfer service cannot be provided by a single actor. In a world of partners, we assume that there is no trusted third party and advance knowledge is not available on either side. The purpose of this model 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 fulfilment of a service request from another actor. The model shows the reasoning procedures of the two actors regarding a service situation SC_{20}.

In the physical world, knowledge about the participants of a service relationship can be obtained easily; for instance, a local bank sees a foreign bank becoming popular worldwide, so it believes that the foreign bank has the capability of making a profit together with him. Such scenario works fine in a closed world where people can easily meet in person. However, when we come to an open world where direct observation and past experience are not available, how do we build a relationship among the service

participants? What new problems do we need to deal with? Assume a bank (Bank X) is facing a service query 'Find a bank that is both reliable and low cost in international money transfers'. Assume that money transfers at Bank X are free, but the bank is not reliable with respect to international transfers. On the other hand, Bank Y does not do international transfers, but has sufficient security and insurance to ensure reliable financial operation. The situation can be modelled as follows:

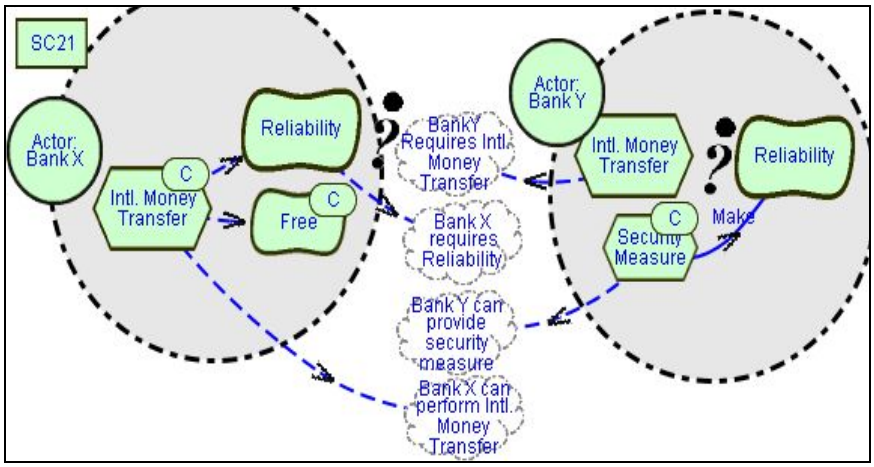
SC_{20}: = (***newActor*** (*BankX*, *BankY*), ***newRequest***(***Requires*** $_{BankX}$ *QoS*(*Intl. Money Transfer*, *Reliability*), ***Requires*** $_{BankY}$ *Intl. Money Transfer*), ***newCapability***(***Can*** $_{BankX}$ *Intl. Money Transfer*, ***Can*** $_{BankY}$ *Security Measure*), ***NewKnowledge***(***Know*** $_{BankY}$ ***Contributes-to***(*SecurityMeasure*, *Make*, *Reliability*))). (see online version for colours)



Routine 2

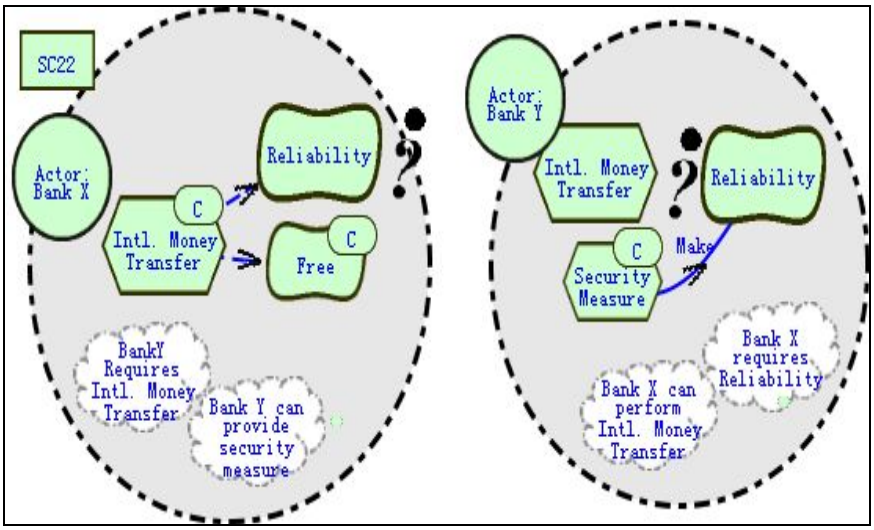
Step 1 Apply Rule 2.4(4), Rule 2.4(8) to SC_{20}: acknowledge service requirements and capability.

SC_{21}: = (…, ***tell*** (*Bank X*, ***Requires*** $_{BankX}$ *Reliability*, *Bank Y*), ***tell*** (*Bank Y*, ***Requires*** $_{BankY}$ *Intl. Money Transfer*, *Bank X*), ***tell*** (*Bank X*, ***Can*** $_{Bankx}$ *Intl. Bank Transfer*, *Bank Y*), ***tell*** (*Bank Y*, ***Can*** $_{BankY}$ *Security Measure*, *Bank X*) …). (see online version for colours)



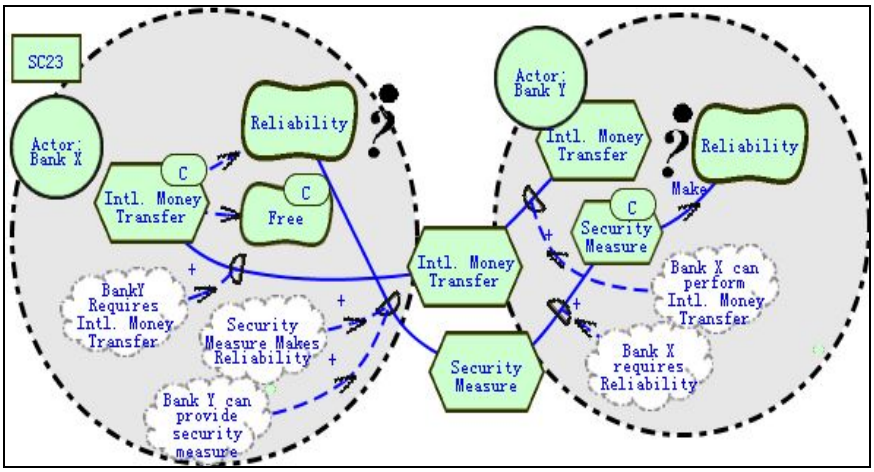
Step 2 Apply Rule 2.5 and Rule 2.6(1) to SC_{21}: knowledge update.

$\boldsymbol{SC_{22}}$: = (…, ***Know*** $_{BankX}$ ***Requires*** $_{BankY}$ *Intl. Money Transfer*, ***Know*** $_{BankY}$ ***Requires*** $_{BankX}$ *Reliability*, ***Know*** $_{BankX}$ ***Can*** $_{BankY}$ *Security Measure*, ***Know*** $_{BankY}$ ***Can*** $_{Bankx}$ *Intl. Bank Transfer*,…). (see online version for colours)



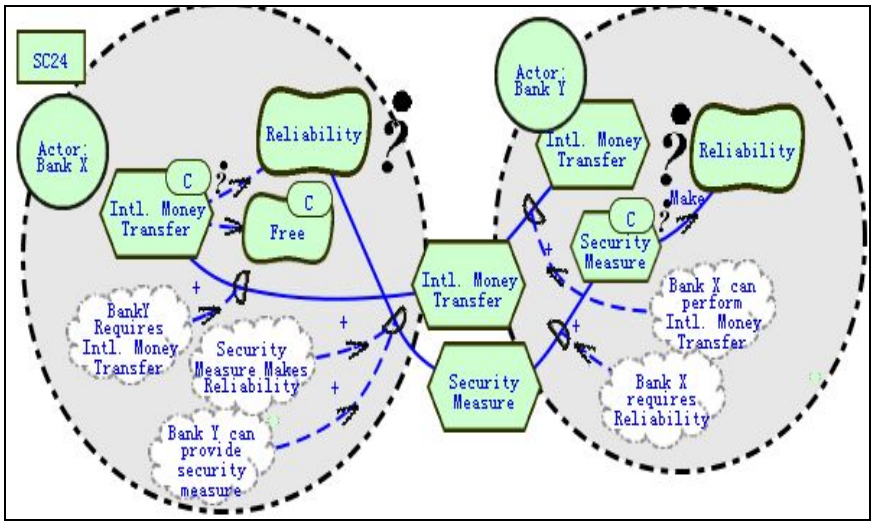
Step 3 Apply Rule 2.7 to SC_{22}: delegate services.

$\boldsymbol{SC_{23}}$: = (…, ***delegate***(*BankX*, *Security Measure*, *BankY*), ***delegate*** (*BankY*, *Intl. Money Transfer*, *BankX*), …). (see online version for colours)



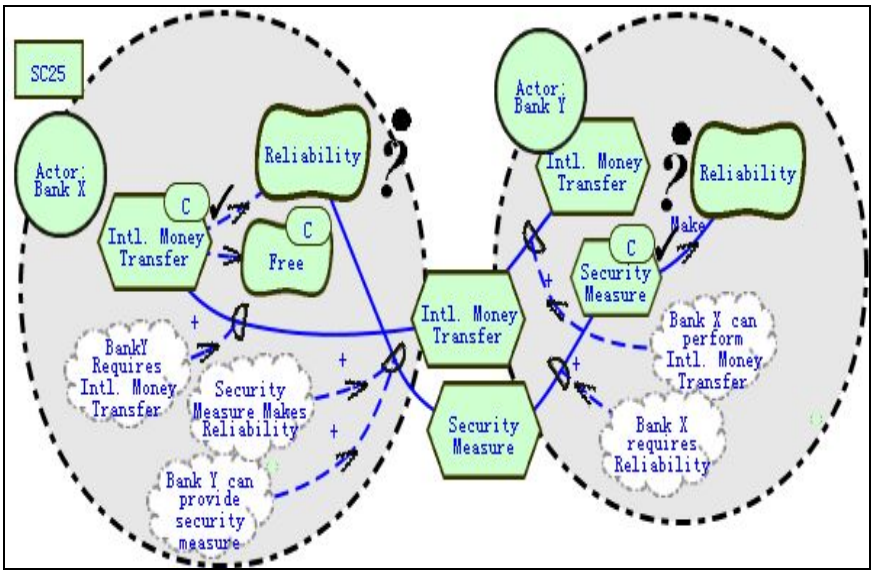
Step 4 Apply Rule 2.8 to SC_{23}: request transfer.

SC*$_{24}$: =** (…, ***Requires $_{BankY}$ *Security Measure*), ***Requires*** $_{BankX}$ *Intl. Money Transfer*, …). (see online version for colours)



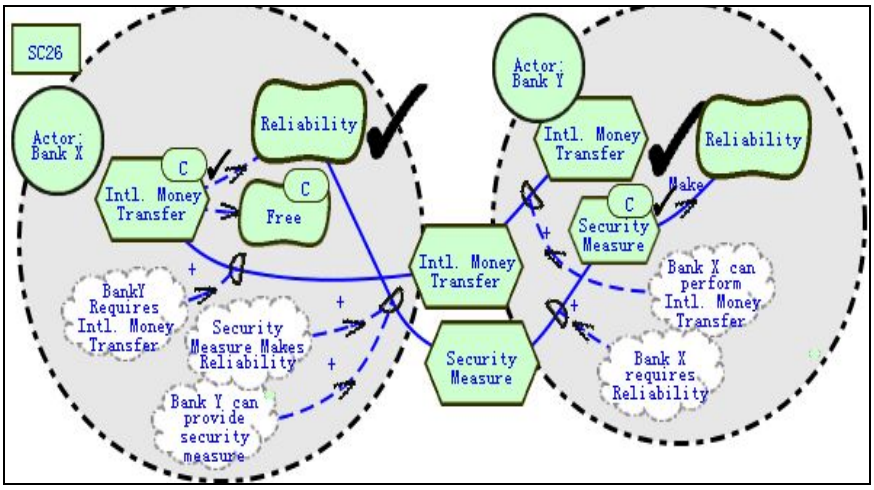
Step 5 Apply Rule 2.1(1) to SC_{24}: delegated service committed.

SC*$_{25}$: =** (**…** ***Commit $_{BankX}$ *Intl. Money Transfer*, ***Commit*** $_{BankY}$ *Security Measure*…). (see online version for colours)



Step 6 Apply Rule 2.9 to SC_{25}: service capability propagated through delegation.

SC*$_{26}$: =** (**…** ***Can $_{BankX}$ *Security Measure*, ***Can*** $_{BankY}$ *Intl Money Transfer*,…). (see online version for colours)



Step 7 Apply Rule 2.1(1) to SC_{26}: requested services committed.

SC*$_{27}$: =** (***…Commit $_{BankX}$ *Reliability*, ***Commit*** $_{BankY}$ *Intl. Money Transfer*,**…).**

No new applicable rule to SC_{27}. End of Routine 2.

After such capability outsourcing procedures, both Bank X and Bank Y have obtained the capability of providing reliable International Money Transfer service. The cost of Bank X could be lower than that of Bank Y, so it may have an advantage during service selection.

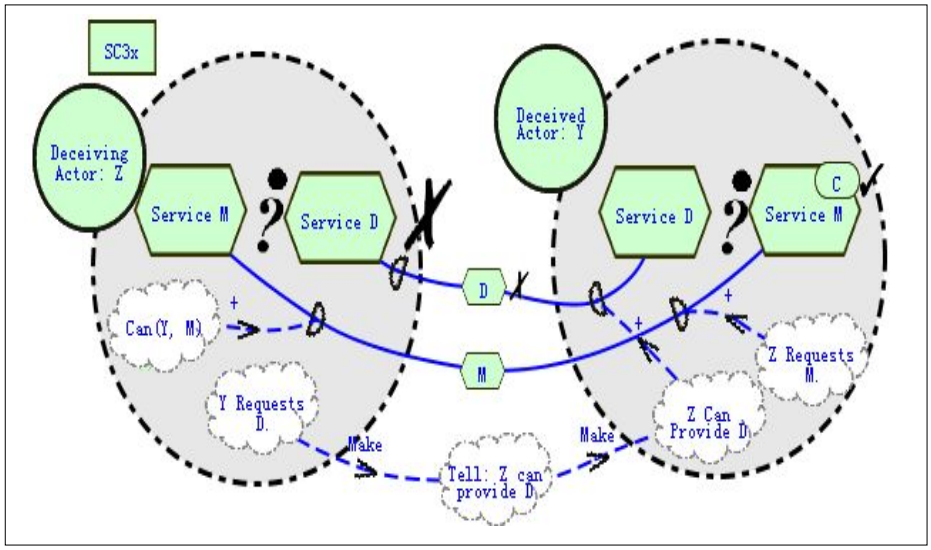
3.3 A world with possible deception: a service model of trust

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 in order to obtain another actor's service. We may extend the framework with action rules as follows.

Rule 3.1 Publish false capability

Actor $(a) \wedge$ ***Actor*** $(b) \wedge$ ***Service*** $(s) \wedge$ *no* ***Can*** $_a$ $s \wedge$ ***Know*** $_a$ ***Requires*** $_b$ $s \Rightarrow$ ***tell*** $(a,$ ***Can*** $_a$ $s, b)$.

The service situation can evolve into the one represented by the following graphical model: (see online version for colours)



Rule 3.2 Establish black list

$$\textbf{\textit{Actor}}\ (a) \wedge \textbf{\textit{Actor}}\ (b) \wedge \textbf{\textit{Service}}\ (s) \wedge \textbf{\textit{delegate}}\ (a, s, b) \wedge no\ \textbf{\textit{Commit}}_{\ b}\ s \Rightarrow \textbf{\textit{newKnowledge}}\ (\textbf{\textit{Know}}_{\ a}\ not\ \textbf{\textit{Can}}_{\ b}\ s).$$

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 analysing the differences between systems showing desired properties, and those allowing undesirable behaviours, a designer will be able to build mechanisms reflecting the right control schema.

3.4 A world with a circle of trust: service selection based-on community feedback

As mentioned in the previous sections, in an open environment, direct knowledge about other actors is very hard to obtain. And sometimes, deciding the trustworthiness of another actor cannot be described by a simple true or false assertion, but vectors representing varying levels of confidence. In this case, it is best to adopt a trust scoring schema to quantify the confidence level of beliefs circulated within the service network.

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

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

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

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

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

6 The trust level of a recommendation is based on the recommender's confidence of the content, and the recommender's level of confidence in 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 $A \cup B$, with the range being an integer.

Rule 3.3 Trust function management

1 Set initial Trust value (Rule 1 above):

$$no\ know_{\ a} f\ (a, Trust, b) \Rightarrow \textbf{\textit{newKnowledge}}(know_{\ a} f\ (a, Trust, b) = 0)$$

2 Compute Trust value of a received recommendation (Rule 6 above):

$$\exists x \in K, a, b \in A, \textbf{\textit{tell}}\ (a, x, b) \Rightarrow f(b, Trust, x) = f\ (b, Trust, a) \times f\ (a, Trust, x).$$

$$\exists x \in K, a, b \in A, \textbf{\textit{tell}}\ (a, x, b) \wedge \textbf{\textit{Know}}_{\ b}\ x \Rightarrow f'(b, Trust, x) = f\ (b, Trust, x) \times f\ (b, Trust, a) \times f\ (a, Trust, x).$$

3 Compute Trust after a service (Rules 2, 3, 4, and 5 above):

$$\exists a, b \in A, s \in S, \textbf{\textit{delegate}}\ (a, s, b) \wedge \textbf{\textit{Commit}}_{\ b}\ s \Rightarrow f'(a, Trust, \textbf{\textit{Can}}_{\ b}\ s) = f\ (a, Trust, \textbf{\textit{Can}}_{\ b}\ s) + 1.$$

$$\exists x \in K, a, b \in A, s \in S, \textbf{\textit{tell}}\ (a, x, b) \wedge no\ \textbf{\textit{Commit}}_{\ b}\ s \Rightarrow f'(a, Trust, \textbf{\textit{Can}}_{\ b}\ s) = f\ (a, Trust, \textbf{\textit{Can}}_{\ b}\ s) - 5, \text{ if } f\ (a, Trust, \textbf{\textit{Can}}_{\ b}\ s) \geq 4; f'(a, Trust, \textbf{\textit{Can}}_{\ b}\ s) = -1, \text{ otherwise.}$$

$$\exists x, a, b \in A, s \in S, \textbf{\textit{delegate}}\ (a, s, b) \wedge \textbf{\textit{Commit}}_{\ b}\ s \wedge \textbf{\textit{tell}}\ (x, \textbf{\textit{Know}}_{\ x}\ \textbf{\textit{Can}}_{\ b}\ s, a) \Rightarrow f'(a, Trust, x) = f\ (a, Trust, x) + 1.$$

$$\exists x, a, b \in A, \text{s} \in \text{S}, \textbf{\textit{delegate}}\ (a, s, b) \wedge no\ \textbf{\textit{Commit}}_{\ b}\ s \wedge tell\ (x, \textbf{\textit{Know}}_{\ x}\ \textbf{\textit{Can}}_{\ b}\ s, a) \Rightarrow f'(a, Trust, x) = f\ (a, Trust, x) - 1.$$

4 Select a service according to trust level:

$$\exists x, a, b \in A, s \in S, \textbf{\textit{Requires}}_{\ a}\ s \wedge \textbf{\textit{Know}}_{\ a}\ \textbf{\textit{Can}}_{\ b}\ s \wedge \textbf{\textit{Know}}_{\ a}\ \textbf{\textit{Can}}_{\ x}\ s \wedge \textbf{\textit{tell}}\ (b, s, a) \wedge f(a, Trust, \textbf{\textit{Can}}_{\ b}\ s) \geq f\ (a, Trust, \textbf{\textit{Can}}_{\ x}\ s) \geq 0 \Rightarrow \textbf{\textit{delegate}}\ (a, s, b).$$

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

4 Related work

The approach proposed in this paper mainly synergises ideas from three major areas: knowledge representation and reasoning in autonomous agent systems, requirements modelling and analysis, and semantic web services. In conventional knowledge engineering and AI, various subjective logic approaches and social ontologies to represent belief, knowledge, desire, and intention of autonomous agents have been proposed (Castelfranchi *et al*., 2003; Hintikka *et al*., 1962; Rao and Georgeff, 1991; Wooldridge and Jennings, 1995; Yu, 1997; Yu and Liu, 2001). Our work aims to

adopt theoretical results from these areas and build a practical framework for the service-oriented computing paradigm. Thus, we mainly focus on the specific needs, assumptions, rules and reasoning mechanism for the service requirements and capability setting. Existing requirements modelling frameworks (Kethers *et al*., 2005; Mylopoulos, 1998; Penserini *et al*., 2006) emphasise capturing and eliciting the requirements in the problem domain. They usually include top-down refinement processes. However, the open, dynamic, continuous system environment needs to have a model that seamlessly integrates high-level abstract requirements models with concrete executable service manipulating mechanisms. By representing service request and service capabilities in a compatible ontology, we aim to provide a holistic solution to the problem.

The Web Service Modelling Ontology (WSMO) (Lausen *et al*., 2005) provides a conceptual framework focusing on the functional and behavioural aspects of a web service. Compared to the WSMO framework, the concepts and reasoning mechanism proposed in this paper emphasise a strategic actor's knowledge of and intention regarding the capability of other actors, rather than a straightforward description about web services behaviours and constraints. This is based on the assumption that actors involved in a service are strategic. That is, an actor has his own intended requirements of service quality to fulfil, which may only partially be made known to other actors. The ontology proposed in this paper is a natural complementary to DAML-OIL (Heflin and Hendler, 2000), 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.

Discovering and assembling individual web services into more complex new and user-centric web processes is an important challenge. In Arpinar *et al*. (2004), web services composition techniques 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. By adopting this kind of two-way thinking, alternative ways to satisfy the user's service requirements can be taken into consideration.

Inference rules of semantic web services ontology are key to dynamically discovering, selecting, and binding the services that best meet user needs. In order to address the lack of an effective means to formally specifying individual services and their interactions, comprehensive formal languages for services have been proposed in several references, such as Maximilien and Singh (2004) and Battle *et al*. (2005), which include SWSL-FOL, a full first-order logic and SWSL-Rules, a rule-based language designed to provide support for a variety of tasks that range from service profile specification to service discovery, contracting, policy specification and so on. These, in essence, have the same goal as our approach; building a comprehensive logic system from very fundamental concepts, such as symbols, strings, and number values, to provide a solid foundation for the knowledge and quality evaluation rules in the proposed SRMO framework. Compared to the SWSL effort, the work in this paper specifies services at a higher level of abstraction.

Penserini *et al*. (2006) proposes using the Tropos requirements methodology (Castro *et al*., 2002) to support service 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 and dependency

handshakes, are now considered as the 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 has the same concept base as this paper. The major difference is that capabilities are defined as links in their work, while capabilities in this paper correspond to the concept of tasks in *i**, while links are considered as knowledge. Also we hope to provide a capability reasoning framework that can handle requirements analysis and design work at run-time. The incorporation of capability and knowledge has better potential in addressing uncertainty and partial knowledge, and conflict of interest of actors.

5 Conclusion

In this paper, we propose a formal service requirements ontology framework that is based on the actors' knowledge and intention. Unlike most other work on service ontology, our proposal focuses on explicitly representing knowledge and allowing subjective decision-making about 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 usage of the proposed approach.

The results from our study are important because they contribute not only to the theoretical study of SOA but also form the basis for its future deployment. In the future, the proposed modelling ontology will be implemented and extended to support different kinds of automatic reasoning for qualitative or quantitative QoS-based service selection including reliability, availability, and request-to-response time, and user experience and preferences.

Acknowledgement

The authors wish to thank the anonymous reviewers and the associate editor for their constructive comments and suggestions. This work was supported financially by the National Natural Science Foundation of China (Grant No. 60503030), the National Basic Research and Development 973 Program (Grant No. 2002CB312004), the National 863 High-tech Project of China (Grant No. 2006AA01Z155, 2007AA01Z122), the National Natural Science Fund for Distinguished Young Scholars of China under Grant No. 60625204 and the Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList).

References

Arpinar, I.B., Zhang, R., Aleman, B. and Maduko, A. (2004) 'Ontology-driven web services composition', *Proceedings of the IEEE E-commerce Technology*, San Diego, California, 6–9 July.

Battle, S., Bernstein, A., Boley, H., Frosof, B., Gruninger, M., Hull, R., Kifer, M., *et al*. (2005) *Semantic Web Services Language, W3C*, September.

Castelfranchi, C., Falcone, R. and Pezzulo, G. (2003) 'Cooperating through a belief-based trust computation', *WETICE*, pp.263–268.

Castro, J., Kolp, M. and Mylopoulos, J. (2002) 'Towards requirements driven information systems engineering: the tropos project', *Information Systems*, Vol. 27, No. 6, pp.365–389.

Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J. (2000) *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers.

Erl, T. (2005) *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall, August.

Heflin, J. and Hendler, J. (2000) 'Dynamic ontologies on the web', *In Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*', Menlo Park, CA: AAAI/MIT Press, pp.443–449.

Hintikka, J., *et al*. (1962) *Knowledge and Belief: An Introduction to the Logic of the Two Notations*, Cornell University Press.

Kethers, S., Gans, G., Schmitz, D. and Sier, D. (2005) 'Modeling trust relationships in a healthcare network: experiences with the TCD framework', *Proceedings of the 13th European Conference on Information Systems*, Germany: Regensburg, May.

Lausen, H., Polleres, A. and Roman, D. (2005) *Web Service Modeling Ontology (WSMO)*, W3C Submission, June.

Levesque, H.J. and Lakemeyer, G. (2001) *The Logic of Knowledge Bases*, MIT Press.

Liu, L., Liu, Q., Chi, C., Jin, Z. and Yu, E. (2006) 'Towards a service requirements ontology on knowledge and intention', *QSIC*, pp.452–462.

Lu, J. and Yu, Y. (2007) 'Web service search: who, when, what, and how', Proceedings of Workshop on Human-Friendly Service Description, Discovery and Matchmaking (Hf-SDDM@WISE 2007), Nancy, France, 03 December, pp.284–295.

Mandell, D. and McIlraith, S. (2003) 'A bottom-up approach to automating web service discovery, customization, and semantic translation', *Proceedings of the 12th International Worldwide Web Conference, Workshop on E-services and the Semantic Web (ESSW'03)*, Budapest.

Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D., Sirin, E. and Srinivasan, N. (2006) 'Bringing semantics to web services with OWL-S', Special Issue on Web Services: Theory and Practise, *Worldwide Web Journal*.

Maximilien, E.M. and Singh, M.P. (2004) 'A framework and ontology for dynamic web services selection', *IEEE Internet Computing*, September–October, pp.84–93.

McGuinness, D.L. and da Silva, P.P. (2004) 'Explaining answers from the semantic web: the inference web approach', *Journal of Web Semantics*, October, Vol. 1, No. 4, pp.397–413.

Mylopoulos, J. (1998) 'Information modeling in the time of the revolution', *Information Systems*, June, Vol. 23, Nos. 3–4, pp.127–156.

Penserini, L., Perini, A., Susi, A. and Mylopoulos, J. (2006) 'From stakeholder intentions to software agent implementations', *Proceedings of the 18th Conference on Advanced Information Systems Engineering (CAiSE'06)*, *LNCS*, Springer-Verlag, No. 4001.

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.

Sycara, K.P., Widoff, S., Klusch, M. and Lu, J. (2002) 'Larks: dynamic matchmaking among heterogeneous software agents in cyberspace', *Autonomous Agents and Multi-Agent Systems*, Vol. 5, No. 2, pp.173–203.

Wang, P., Jin, Z. and Liu, L. (2006) 'An approach for specifying capability of web services based on environment ontology', *ICWS*, pp.365–372.

Wooldridge, M. and Jennings, N.R. (1995) 'Agent theories, architectures, and languages: a survey', in M. Wooldridge and N.R. Jennings (Eds.) *Intelligent Agents, Lecture Notes in Artificial Intelligence*, Berlin: Springer Verlag, Vol. 890, pp.1–39.

Yu, E. (1997) 'Towards modeling and reasoning support for early-phase requirements engineering', *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, Washington, DC, 6–8 January, pp.226–235.

Yu, E. (2001) 'Agent orientation as a modeling paradigm', *Wirtschaftsinformatik*, April, Vol. 43, No. 2, pp.123–132.

Yu, E. and Liu, L. (2001) 'Modeling trust for system design using the i* strategic actors framework', in R. Falcone, M. Singh and Y.H. Tan (Eds.) *Trust in Cyber-Societies*, *LNAI-2246*, Springer, pp.175–194.