SREM: A Service Requirements Elicitation Mechanism based on Ontology

Jian Xiang Lin Liu Wei Qiao Jingwei Yang
School of Software, Tsinghua University, Beijing, China, 100084
{xiangj05, qiaow05}@mails.tsinghua.edu.cn, linliu@tsinghua.edu.cn

Abstract

The Service-Oriented Computing paradigm aims to support automated discovery and selection of web services according to user's requirements. At present, user's requirements are often represented in certain existing standard interoperable service description languages such as WSDL/OWL-S. However, general service requestors may find such languages hard to use directly due to the reason that service requirements are often partially elicited and fragmented.

In this paper, we propose an automated Service Requirements Elicitation Mechanism (SREM) to help extract and accumulate relevant knowledge on service requirements. First, the SREM elicitation approach proposes to use a list of questions to narrow generic service requirements down to specific expressions of user preferences. Then, a service requirements and capability ontology is adopted to capture services requirements in breadth and precision. By integrating service requirements issued by different requestors, SREM provides non-trivial requirements guidelines and heuristic rules on service publication and discovery, also provided is a service requirements analysis mechanism that improves the accuracy of service discovery and efficiency of service composition continuously.

1. Introduction

Requirements elicitation for services imposes different challenges from conventional software requirements elicitation process. In conventional software requirements engineering, elicitation takes a "face-to-face" mode. That is, there is a group of targeted customer, from whom the requirements engineers can obtain original requirements information. But during the services requirements process, there is often no fixed target, the requirements elicitation often takes a "back-to-back" mode, services providers and service requestors conduct double blind search. The success of services as a business and computational paradigm depends on how well the two sides understand requirements and constraints of each other. Ill-defined and misrepresented requirements of service may lead to unbalanced service-level agreement, or no agreement can be formed at all. Thus, Requirements engineering for services plays a definitive role during the service engineering life cycle. In order to achieve efficient service design, publication, discovery, binding and evolution, we need requirements facilities that can handle service requirements issues automatically and systematically.

Key research questions regarding requirements elicitation from the service provider's perspective should include: How a prospective service provider can map its core competence into a maximum set of users requirements that can be satisfied by it? Where and how can we acquire and accumulate valuable service requirements knowledge? Is it from existing published service profiles, from Service-Level Agreements, or from logs of user queries? Unfortunately, there is no systematic approach in requirements engineering or service engineering that can address these issues yet. We need a mechanism to guide the service providers through the process of transforming legacy systems into easily reusable and customizable services according to user's real needs, which should lead us into a win-win situation in the service-oriented world.

In this paper, we propose a service requirements elicitation mechanism, *SREM*, to facilitate the process of service requirements elicitation. *SREM* is based on a service requirements ontology *SRMO*, which inherits some basic concepts from the agent-oriented requirements modeling framework *i** for early-phase requirements analysis. *SREM* also includes a set of questions, which can be organized into a dynamic questionnaire to draw service requirements details from requestors. Based on answers to this questionnaire, a graphical requirements model describing key elements and structure of each service requirements can be built. Besides requirements modelling, *SREM* includes a mechanism to analyze and integrate individual requirements models. First, requirements models with related concepts are correlated and connected. Then such individual requirements models are merged to form domain-specific requirements network model based on consensus. This integrated requirements network model offers richer information than the original requirements fragments. It provides a service requirements analysis mechanism that improves the accuracy of service discovery and efficiency of service composition. By conducting the elicitation process iteratively for service in specific domains, requirements knowledge can be elaborated for not only one single service but the evolving service world. Moreover, through mashing-up requirements from various past, current and prospective service requestors, knowledge about the preference of service requestors can be accumulated. The provider who has the most popular components in the network will become the most competitive among other providers. By tracing upstream within the hierarchical service requirements model, service provider may figure out how to promote its position in the market place.

The structure of the paper is arranged as the following: Section 2 introduces *SRMO*, the ontology for service requirements elicitation, which is the foundation stone for *SREM*. Section 3 describes in detail the elicitation mechanism. Section 4 presents the requirements reconciliation process and related heuristics. Section 5 introduces experiments and example of applying *SREM*. Section 6 discusses related work. Section 7 concludes the paper.

2. Introduction to *SREM*

SREM includes three major components: Service Requirements Modeling Ontology (*SRMO*) [10], Service Requirements Elicitation Process (*SREP*), and Service Requirements Reconciliation Process (*SRRP*). *SRMO* defines the ontology for requirements modeling; it provides conceptual guideline for the other two. SREP directs the process for eliciting requirements details and structural information with service requestor; SRRP is the reconciliation process used to integrate all requirements instances to build a requirements knowledge repository which contain heuristics for service publication.

2.1 Service Requirements Modeling Ontology (*SRMO*)

Figure 1 shows the concepts in the proposed service requirements ontology. It models service at a higher level of abstraction than current service ontology such as, OWL-S, which focus more on implementation details irrelevant to general users. The level of abstraction adopted by the proposed mechanism aims to achieve more precise profiling of service requirements. At the same time, SRMO are designed to be easily mapped to OWL-S description when necessary. Individual service requests can be easily collected and crystallized as requirements knowledge applicable for future use.

Major concepts in SRMO include: *Actor*, *Goal*, *Task*, and *Quality*. Major properties of these concepts include:

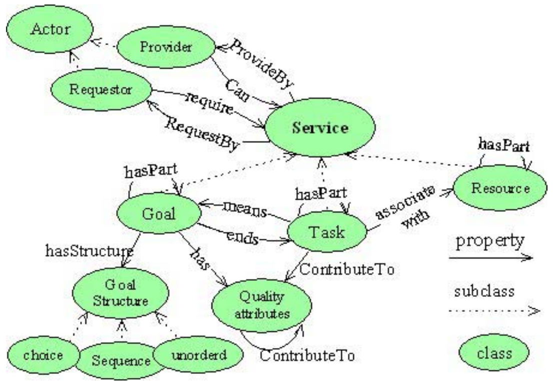


Figure 1. Service Requirements Modeling Ontology

2.2 Terminology and Definition

Definition 1. *$A = \{a_1, ..., a_n\}$ is a set of* ***Actors***. If a ∈ A, we write: *Actor (a).* An actor a_i is either a service requestor or a service provider.

Definition 2. $G= \{g_1, ..., g_n\}$is a set of ***Goal-states***. If $g \in G$, we write *Goal (g)*. A *goal-state* g_i is a condition or state of affairs in the world that a service requestor would like to achieve. A goal can be achieved in different ways, prompting alternatives to be considered.

Definition 3. $T = \{t_1,..., t_n\}$ is a set of *tasks*. If $t \in T$, we write ***Task** (t)*. A *task* t_i is used to represent the specific procedures to be performed by service provider, which specify particular ways of doing something. *Tasks* are used to incrementally specify and refine solutions in the target system. *Task* is the way for achieving *goals*.

Definition 4. $R = \{r_1,..., r_n\}$ is a set of *Service **Resources(r)***. *Resource* is a physical or informational entity, which may serve some purpose. Properties of an entity include whether it is available or not, what is the value its quality attributes, or non-intentional properties such as amount, producer, copyright owner, color, length, etc.

Definition 5. $Q = \{q_1,..., q_n\}$ 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 concerned by an actor requesting or providing a service, such as, cost of a service, performance, security/privacy assurance, easy-to-use, etc. In other words, anything within the scope of QoS can be described.

Definition 6. $S=G \cup T \cup R \cup Q$ is a set of *Services*. Textually, if $s \in S$, we can also represent it as: ***Service** (s)*.

Definition 7. $DC \subseteq (G \times G) \cup (T \times T) \cup (R \times R)$ is a set of ***decomposition*** relationships. If $dc(g_1, g_2) \in DC$, we write ***part-of*** (g_1, g_2), which is used to describe that g_2 can be achieved iff g_1 can be done.

Definition 8. The relations between *goals* are represented by class ***Goal Construct***. *Goal Construct* is used to describe the temporal or causal relation of *Goals*. So far, there are three structures: ***Sequence***: the *goals* connected by it must be executed one by one according to the order they appear in the goal sequence. ***Unordered***: the *Goals* connected by it can be executed in any order or concurrently; ***Selection***: only one *goal* among all alternatives can be executed.

Definition 9. $C \subseteq A \times \{S \cup SG\}$ is a set of *capable* relations. We use ***Can*** $_a$ ***s*** to denote that there exists an actor *a* can provide certain service or quality s, i.e., $c(a, s) \in C$.

Definition 10. $ME \subseteq T \times G$, is a set of *means-ends* relationships. If $me(t, g) \in ME$, we write ***Means-ends*** *(t, g)*. A means-ends relationship *me (t, g)* is used to describe that *g* can be achieved if the task *t* is performed. Each task connected to a goal by a means-ends link is one possible way of achieving the goal.

Definition 11. ***Contribution*** relationship: $CN = (SG \times (SG \cup S)) \rightarrow CT$ is a set of *Contribution* relationships. CT = *{positive, negative, unknown}* $\times$ *{full, partial, unknown degree}* is a set of *contribution types*. We use following alias for each possible value of the contribution type: *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 scale of contributions.

The SRMO has inherited its key modelling concepts from *i**, a widely adopted requirement modeling framework. Thus, SRMO can be considered as a service-flavored requirements modeling framework. *i** framework emphasize the actors or stakeholders distributed in different environments and the relationships among them, and is generally applicable to any distributed agent-oriented environments.

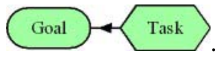
In other words, the requirement model defined here is generally applicable which means it can be applied to requirements settings other than the service-oriented paradigm. The SRMO ontology is also extensible to incorporate concepts representing other specific requirements modelling perspectives.

3. Service Requirements Elicitation Process

Getting the requirements right is a must in the development of complex, software-intensive systems. In conventional requirements engineering, we often use questionnaires, meetings, interviews to collect original requirements data. Such original requirements information are often fragmented like jigsaw puzzles, requirements analysts' major contribution is to recover the complete picture from such fragments, and making design decisions based on the information available. Service Engineering faces a similar situation. When consumer issues a service request, it expresses some intended need to be served. Our requirements elicitation mechanism is dedicated to make such service requirements explicit, and to apply it to conduct automated service discovery and composition.

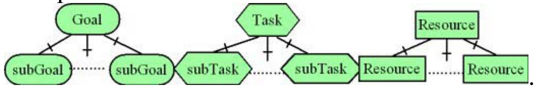
The Service Requirements Elicitation Process (*SREP)* helps generate a requirements model based on the concepts shown in the ontology above. Thus, it is inevitable for us to raise questions such as: How to extract the original requirements statements from a service requestor? Here, we present a series of questions and answer schemas to formulate the rough sketch of requirements model.

Q1: What is the service being required? The answer to this question could be a *goal* (Goal), which state a condition to be satisfied with the help of the service; or a *task* (Task), which specify a procedure or course of actions to be performed during the service execution; or a *resource* (Resource), which is made available by the service; or a quality (Quality Attribute), which is to be ensured by the required service.

Question 2: How can the state of the *Goal* be achieved? In what way can we fulfill requirements of the *Goal*? Answer to this question will identify a *Task* that defines a specific way for achieving the *Goal*. It adds a means-ends link between *Goal* and *task*, *task* is a means for achieving *Goal*, while *Goal* being the end of performing the *task*. We may have various *tasks* for one *Goal*; each *task* stands for one alternative ways of achieving this *Goal*. The answer to this question is modeled as 

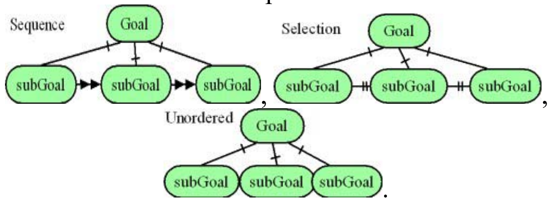
.

Question 3: What are the sub-components of this required *goal/task/resource*? What sub-*goals* are need to achieve for this *Goal*? Decomposition of *goal* is similar to business process modeling to some extent, but focuses on the requirements and desires from requestor. This step need to be done iteratively for each goal until all *goals* are refined into a structure which requestor considers as satisfactory. The decomposition process can also be applied to *task* and *resource*, such as, what sub-tasks need to be executed for the task? How can the resources be assembled? The answer to this question is modeled with structures as follows:

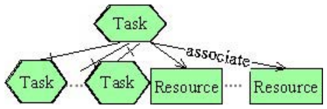


.

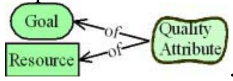
Question 4: Among the sub-*goals/tasks* identified, are there any ordering constraints? What is the temporal or causal relationship between these sub-*goals/tasks*? The relationship includes 3 kinds: *sequence*, *selection* and *unordered*. The relationships are modeled as:



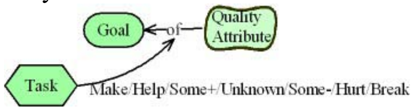
Question 5: What sub-*task* should be done to execute a specific *task*? What *resource* does this task need and associate with? The *resource* is often an entity, which can be a service also. The answer is modeled as:



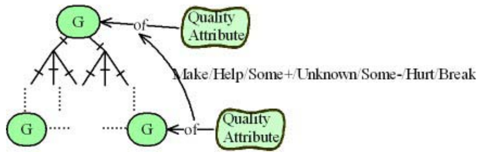
Question 6: What quality attribute associated with *goal/resource* is required? *Quality attributes* of a service are those judge rules for evaluating its quality. The answer to this question is modeled as:



Question 7: How does a specific *task* influence quality of service? *Task* has its contribution to attributes link specified in question 5. *Task* directly influences the *Quality attribute* of a *goal.* The answer to this question is graphically modeled as:



Question 8: How does one quality attribute influence another quality attribute? The answer to this question is graphical modeled as:



Questions 1 to 8 help us collect requirement fragments from requestors. Figure 2 shows an integrated view example for a service requirement, which if all question answered, can be constructed by those answers.

Potential inconsistencies between different requirement fragments are inevitable. For example, some consider that *sequence* is the feasible relationship between several *goals* while others prefer the *selection* relation. We solve this problem with a statistical approach. In the process of elicitation, all users' inputs will be maintained. We calculate the score of answers comprising each solution, and use the solution with highest overall score. This is a simple and straightforward approach, which is easily operable and effective. More comprehensive measure is to build several types of user profiles, and give different weight to different requirements parameters to reflect users' preferences and biases.

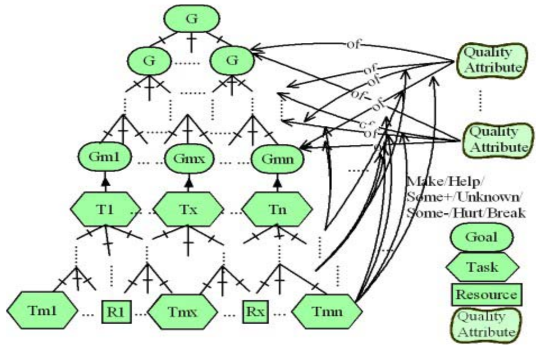


Figure 2. a hierarchical requirement model in *SRMO*

4. Service Requirements Reconciliation Process and Heuristics

Service requirements elicitation can help each requestor understand their requirements better and express their service desires more efficiently in future. As afore mentioned, for different requestors targeting at the same service *goal*, *SREM* can build up different requirements models using *SREP*. SREP was meant to bring benefits to not only service requestors, but also service providers. Through the integration of users' requirements models from different service requestors, this mechanism builds a requirements knowledge repository. Such a repository offers service providers requirements knowledge about the needs and preference of service requestors with regard to the service offered by him. In the meanwhile, the one who owns the most important components in the repository would find itself in an advanced position when competing with other service providers. Having received the corresponding position in the requirements model hierarchy, service providers could trace upstream the network to figure out what other influential requirements it can set hand in. This section discusses this requirements reconciliation process.

4.1 Matching Strategy

The key step in the *SRRP*, is to combine new incoming requirements model into the existing repository. When integrating a new requirements model into the knowledge base, SRRP first need to analyze all components in the new model to see if it has any similarity with any existing components of the concept diagram. First, we define the matching strategy. To understand better, the components are prioritized in a stack in figure 3.

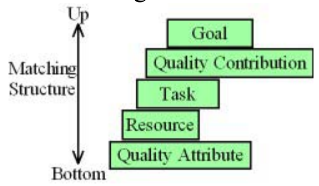


Figure 3. Matching priority

Quality Attribute Synergy rule: There are overlapping quality attributes for different services. Quality attributes can be domain-specific or domain-independent. Domain-specific attributes vary from one service requirements *Goal* to another. When matching *quality attributes*, we combine ranges of the domain independent attributes across services, and combine the ranges of domain-specific attributes only if their service *Goals also match.*

Resource combination strategy: *Resource* represents a physical or informational entity, which is the object of certain *task* for the achievement of the corresponding service *goal*.

Task combination strategy: *Task* is the specific procedures to be performed by service provider for achieving a *goal*. Two *tasks* are same if only they have the same capability and constraint; they operate on the same type of *resource* and provide the same function. Two same *tasks* may have different description, sub-structure and super-structure.

Quality contribution propagation strategy: Each *task* has its contribution link to *quality attribute* of *goal*. Only if the same *tasks* under the same *goal* have the same *contribution* link.

Goal grouping strategy: Goal matching often occurs at different structure levels. A top level *goal* for a new incoming service requirement may be the same with a sub-*goal* in another *goal* sub-structure. The matching rule for *goals* is based on the *task* connected to it by means-ends link. Two *goals* can be combined when they have matching means-ends *task* and matching *quality attributes*.

Other matching rules include:

R1: If two *goals*/*tasks*/*resources*/*attributes* have the same name, they can be matched. We can use edit distance and work Net to compare two name string.

R2: If two *goals*/*tasks*/*resources*/*attributes* have the same identifier such as URI, they can be matched.

R3: If *goals*/*tasks*/*resources* have the same direct sub-structure, they can be matched

R4: If two *goals*/*tasks*/*resources* have the same parent and siblings, they can be matched potentially.

R5: For all elements, if A matches with B, B matches with C, and then A matches with C.

R6: if two *goals* have the same means-ends *tasks*, they can be matched.

R7: If two *tasks* are the same, they have the same quality *contribution* link to *quality attributes*.

These rules will help us merging new service request model into existing one. This part analyzes the potential matching between requirement models. It is the foundation for requirements reconciliation process to be discussed next.

4.2 Merging requirements instances

SRRP aims to conduct a learning process based on a large number of requirements model instances to form a requirements knowledge structure. A possible scenario could be: SREP accepts a first request into an empty requirement model. When another request comes, SRRP reconciles the new one with the original model, construct a hierarchy structure. The learning procedure continues when new request comes.

We describe the process of requirements reconciliation in SRRP as follows:

1. Conduct concept learning, compare the elements in the new incoming model with existing ones, merge the models by identifying various *goal, task, resource, or attributes*.
2. Counting the frequency each element and its sub-structure link appeared in the requirements model when performing the merge operation.
3. The *reinforcement* of an element refers to how often it appears in the model network.
4. Redundant decomposition links are removed and the corresponding transitive links are reinforced.
5. Equivalence heuristics are applied for merging matching elements. Apply all matching rules mentioned above.

This requirements reconciliation process help generate an integrated requirements model. The more requests issued, the more refined the requirements are, and the richer is the knowledge.

4.3 Requirements Elicitation from statistics

When the number of service requests grows, the integrated model becomes more refined. Based on the integrated model and the *reinforcement* of each *goal/task/resource*, both service requestor and provider can benefit from the information. Typical scenarios are:

①Elicitation results from requestor's perspective:

The integrated requirements hierarchy provides more possible alternative implementations for a service goal, for one *goal/task*, it can have many different ways of decomposition; standing for different viewpoint for this *goal/task*. Instead of simply integrating all viewpoints from different requestors; integrated network has an important statistic variable, *reinforcement*, which consensus on how most people view the *goal/task*. For a service *goal*, we can find a relevance model which chooses the sub-component link with the largest reinforcement at every decomposition node. This model would reflect most requests' position on this *goal*.

② Elicitation results from provider's perspective:

Service providers will identify the *goal/tasks* with higher *reinforcement* as major revenue producers, and are the focus of requestors.

An important issue the requirements process can help service provider answer is how to map its core competence into a maximum set of users' requirements that can be satisfied by it. The result of the above learning process offers us a solution. Given a network, each *task* in the network requests a provider to execute, thus, provider first locates its core service capability on a *task* for one *goal*, then it trace upstream the network to find other requirements *goal* it can take part in. For instance, a provider for currency transformation service *goal* may find itself a role in a hospital financial service process, which is unknown before. Using this knowledge, provider may map its capability to any domain, find more potential business leads.

5. Example Case Study

We have applied the proposed approach on a simple web process of order-processing. When an order request is issued by a customer, the manufacturer, upon receiving the order detail, checks the inventory to verify if it has enough quantity of goods to fill up the order. In case there is enough stock then the manufacturer contacts its delivery partner to confirm shipping date and address. Based on the shipment fee returned by delivery partner and the products cost from order detail, the accounting partner, often a bank, calculate the total price and return the results to customer. When there is not enough stock in the inventory, the manufacture contacts its supplier first and then the delivery partners.

For each component service in the process, SREP can build a requirements model. Take *Account Service* as an example, its service *goal* is "total cost for the new order is explicitly calculated". The following figure 4, 5 presents the model built by SREP. They are constructed by different requirement fragments. Different figures reveal viewpoints from different prospective providers. For limitation of space, we use labels for model elements: (G for *goal*, T for *task*, R for *resource* and QA for *quality attribute*.)

G: total cost for the new order is explicitly calculated;
G1: Cost for supply service in business process is computed;
G2: Cost for delivery service in business process is computed;
G3: discount for consumer is computed;
G4: the tax for this order is computed;
G5: The finance of enterprise is balanced.
T1: Compute the fee for *Supply Service*;
T2: Compute the fee for *delivery Service*;
T3: check the discount for customer.
T4: Compute the tax cost for all cost;
T5: Bank help balancing the finance of the enterprise; **T6**: Basic computation capability;
T7: Check the reputation of the corresponding customer;
T8: Check the discount for the customer within the reputation level;
R1: supply service; **R2**: delivery service;
R3: customer information record;
R4: tax radio table; **R5**: money;
QA1: Accuracy; **QA2**: Security; **QA3**: Good reputation;

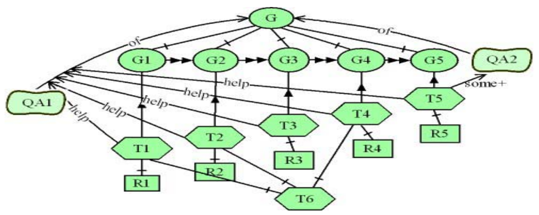


Figure 4. Model of request 1 for Account Service

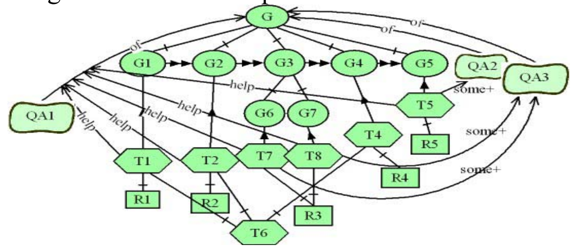


Figure 5. Model of request 2 for Account Service

The integrated model from the two requests for account service is shown in Figure 6, the reinforcement of each *goal*/*task*/*quality* is labeled on the upper right side of it:

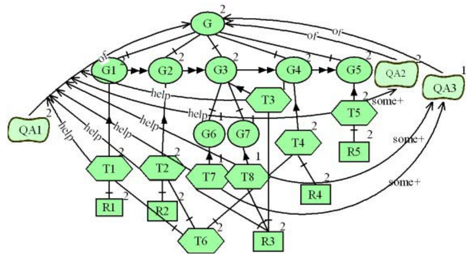


Figure 6. Merged model for Account Service

After building a model, when new request comes in, the process of SRRP merge it into the integrated model. As we always emphasize, requirements gathered from requestor are mostly fragments, suppose request 3 is as below which is only fragment:

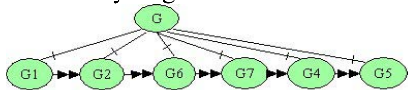


Figure 7. Model of request 3 for Account Service

According to the step 4 in SRRP, merging this request will not changed the model, but increases the *reinforcement* of the corresponding *goals*. Through SRRP process to merge requirement fragments into the model, SREM makes the requirement more and more refined and detailed.

Among the huge amount information provided by the model, the *reinforcement* number of elements presents us a relevant path to understand the requirement. For instance, if more requests issued like request 3, account service would be considered as shown in figure 8:

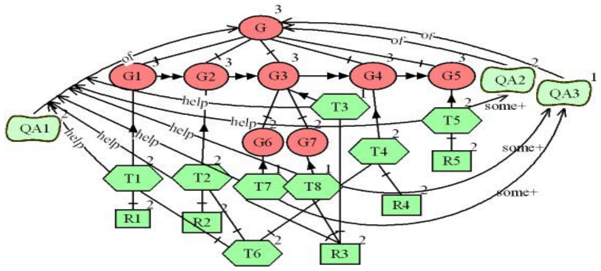


Figure 8. Reinforcement path for Account Service

An important application of this model is its can help relate requirements items, from which requestors can obtain necessary information they expect. The learning process can continue by referring to other requestor's model. And the optimal path indicated by *reinforcement* is a guideline for ordinary requestors.

On the other hand, provider expects to map its core competence into a maximum set of users requirements. Now we discuss another scenario, in which requirement fragments come from requests for other services. Figure 9 show the model for *delivery service*:

G': Goods provided by supply service delivered safely to target;
G8: Cost for delivery service is confirmed;
G9: The goods is received from supplier;
G10: The goods is delivered at target;
G11: The finance of delivery company gets balanced;
T9: Delivery company discuss cost;
T10: Delivery people pick up the goods;
T11: Delivery people transfer the goods to target.
T12: Financial institute does the balance for the company;

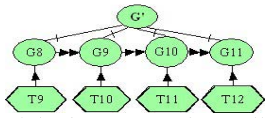


Figure 9. Model of a request for Delivery Service

After SRRP process this request, the integrated model becomes: G5, G11 are the same; it has two implementations, T5 and T12. Thus, the providers for T5 or T12 would be useful in both G and G'. Concretely, both bank and financial institute can perform the task for *Account Service* and *Delivery Service*. Thus, providers capable of a *task* would find their other capabilities related to the *task*. The more request integrated into this model, the providers would found more users' requirements its core capability can help performing. The model structure provides information for all service providers, to organize their legacy system to a more reasonable and comprehensive set of services.

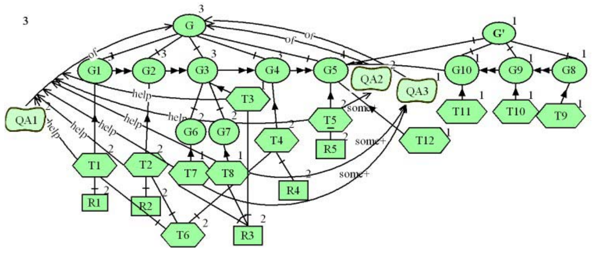


Figure 10. Integrated Model with delivery service

6. Related Work

Requirements elicitation for software systems has been focused primarily on the development, implementation, and evaluation of a variety of techniques, methods, and tools. Many of these were adopted from other disciplines such as the social sciences [2, 4] and knowledge engineering [9, 11]. Regardless of their origin, the objectives for these approaches were to reduce the complexity of the elicitation process and improve the quality of the requirements. In reality, there are about a hundred approaches that can be used for requirements elicitation. [7] has examined at a relatively high level a small number of the traditional techniques such as interviewing, observation, and task analysis. In a more recent survey on the theory and practice of requirements elicitation [6], more approaches were examined including those based on goals [5], scenarios [13], viewpoints [15], and domain knowledge [16].

We consider our approach a natural migration of the goal-oriented requirements elicitation approach into services-oriented computing. That is, in the service-oriented requirements engineering, the elicitation process has to be conducted within the service life-cycle, and with web-based elicitation tools. Instead of targeting at one particular service user, it is targeting at user clusters with similar needs and preferences. Thus, our major contribution is three folds: first, we identified a group of service requests fragments; then we have given heuristics on how to assemble such service request fragments from a same requestor into a comprehensive goal-structure representing service requirement. Finally, we propose a mechanism for reconcile service requirements knowledge on a same service from different requestors.

Thus, we consider work on consensus ontologies [8,12,14] relevant to this paper. Consensus ontologies assume that a multiplicity of ontology fragments, representing the semantics of different sources can be related to each other automatically without the use of an existing global ontology. As the readers will see, this paper take a moderated position by having the goal-oriented world view as the background, and assume that user's request are ontologies to consensus within this macro scope.

There are also other slightly related works such as [3] which targets at similar problems on collecting users' preferences in order to guide service composition. It suggests user to present a service request with an external e-Service schema of a finite state machine. We consider such work as targeting as similar problem using different conceptual mindsets and treatments. There was a homonymous requirement methodology SREM[1] in 1970s, designed for large embedded systems providing descriptions of real world objects, data requirements and message processing.

7. Conclusion

In summary, we present a mechanism to facilitate service requirements elicitation based on service requestors' answers to a group of queries designed based on a goal-oriented requirements language – *i**, called *SREM*. The *SREM* mechanism consists of three major components *SRMO*, *SREP* and *SRRP*. We have elaborated on each of these components and the requirements reconciliation heuristics to establish requirements hierarchical model based on it. The proposed elicitation mechanism will benefit all involved actors in the service world. Requestor can encode its request more efficiently, and provider can dig out the potential business leads through tracing over the requirements model. Our major contribution is three fold: first, we identified a group of service requests fragments; then we have given heuristics on how to assemble such service request fragments from a same requestor into comprehensive goal-structure representing service requirements. Finally, we propose a mechanism for reconcile service requirements knowledge on a same service from different requestors.

In the future, we will continue with the line of research in the following directions: first, to bind the proposed service requirements elicitation mechanism with a widely applied service computing platform, so that validity, advantage and limitations of the approach can be stressed. Second, we will extend the goal-oriented requirements ontology with other existing requirements frameworks such as scenarios-based, state-machines-based, object-oriented approaches, so that we can ladder user preferences, constraints, and requirements from any starting points.

Acknowledgement

Financial support from NSF China (no.60503030), National High Tech. 863 Program (no.2006AA01Z155), National Basic Research and Development 973 Program (no.2002CB312004), and Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList) is gratefully acknowledged.

References

[1] Alford M.W., Software Requirements Engineering Methodology (SREM) at the Age of Two, COMPSAC78. Proceedings: 332-339, 1978.
[2] Ball L.J., Ormerod T.C., Putting ethnography to work: the case for a cognitive ethnography of design. International Journal of Human–Computer Studies 53(1), 147-168, 2000
[3] Berardi D., Calvanese D., De Giacomo G. etc, Automatic composition of e-services that export their behavior, Proc. Of 1st ICSOC, 2003, pp. 43-58.
[4] Beyer H.R., Holtzblatt K., Apprenticing with the customer. Communications of the ACM, 38(5): 45-52, 1995.
[5] Dardeene A., van Lamsweerde A., Fickas S., Goal-Directed Requirements Acquisition. Science of Computer Programming 20(1-2): 3-50, 1993.
[6] Zowghi D., Coulin C., Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In Engineering and Managing Software Requirements, Springer: US, 2005.
[7] Goguen J.A., Linde C., Techniques for Requirements Elicitation. International Symposium on Requirements Engineering, 152-164, January 4-6, San Diego, CA, 1993.
[8] Hindriksv K. V., de Boer F. S., der Hoek V., etc, Agent Programming in 3APL, AAMAS, 1999, pp. 357-401.
[9] Hudlicka E., Requirements Elicitation with Indirect Knowledge Elicitation Techniques: Comparison of Three Methods. ICRE'96, 4-11, Colorado Springs, USA. 1996.
[10] Liu L., Chi C., Jin Z., Yu E., Strategic Capability Modelling of Services. The 2nd Workshop of Service-Oriented Computing Consequences and Experience of Requirements(SOCCER 2006). Paris, France.
[11] Maiden N.A.M., Rugg G., Knowledge Acquisition Techniques for Requirements Engineering. Requirements Elicitation for Systems Specification, July,Keele, UK, 1994.
[12] Munindar P. Singh, Michael N. Huhns. Service-Oriented Computing: Semantics, Processes, Agents. Wiley, London, UK, 2005
[13] Potts C., Takahashi K., Anton A., Inquiry-Based Requirements Analysis. IEEE Software 11(2): 21-32, 1994.
[14] Rao A. S., AgentSpeak(L): BDI Agent Speak Out in a Logical Computable Language, Proceeding. of the 7th EuropeanWorkshop on Modeling Autonomous Agents in a Multi-Agent World, 1996, pp. 42-45.
[15] Sommerville I., Sawyer P., Viller S., Viewpoints for Requirements Elicitation: A Practical Approach. International Conference on Requirements Engineering, 74-81, April 6-10, Colorado Springs, USA, 1998.
[16] Sutcliffe A, Maiden N., The Domain Theory for Requirements Engineering. IEEE Transactions on Software Engineering 24(3): 174-196, 1998.