Helping Students Elicit and Analyze Service-oriented Requirement Using SPOM (Subject-Predicate-Object Method) *

WAN Hai, LI Wen-jun
(Computer Science Department, Sun Yat-sen University, Guangzhou 510275, China)

Abstract: In order to design and provide good services, it is necessary for students to describe and understand the customer requirements for services rapidly, to design and plan the services and their behavior, and to schedule service processes effectively. This paper proposes a new service-oriented requirement elicitation and analysis method based on answer set semantics, sketches requirement Meta-model and correlative proofs and algorithms, which can help students represent requirement more declarative and precise. Based on the Meta-model, Subject-Predicate-Object requirement method and specification is presented, by which students can constitute a smooth, small-step, incremental and iterative development process cut in by forms. By applying this method to teach and help students develop service-oriented application, it is clear that it can help students eliminate the miscommunication between developers and users, assure the correctness of the artifacts and make the users active in development cycle.

Key words: Service computing and engineering; Requirement specification; Answer Sets Semantics; Meta-model; Subject-Predicate-Object method
CLC number: TP31 Document code: A Article ID: 0529-6579 (2007) S2-0130-07

1 Introduction

Services computing and engineering covers the science and technology that underlie Business Services and bridge the gap between Business Services and IT Services. The core technology suite includes Service-Oriented Architecture (SOA) and Web services, and business process integration and performance management[1]. In the developed countries, more than 70% labor force is working for service industry. In China, the above number is 35% in 2004 and will grow up year by year. The objectives of research on services are: in the right time schedules, allocate the right resources (people and various resources), to provide the right services to customers in "on demand" style to response to the requirements of customers[2].

However, there exist some problems for computing and engineering service when teaching and training students, either Undergraduate or Postgraduate students: Service requirements elicitation and analysis; Service models; Service modeling; Service building; Service evaluation; Support tools and platforms; etc.

This paper focuses on helping students elicit and analyze service-oriented requirement, i.e. How to describe and understand the customer requirements for services rapidly; How to design and to plan the services and their behavior, and to schedule service processes effectively?

Requirements Engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the business process in which it will be applied[3]. Requirement can be termed as mapping from informal requirement description to requirement definition, then to corresponding formal specification[4]. One reason that requirements engineering deserves special attention is its general lack of tools and effective procedures relative to later life cycle phases, especially in service computing and engineering.

defines five requirement processes: requirement definition and analysis[5], requirement decision, specification, implement and validation, and evolution management. Semiformal, formal, and executable (or informal) specification languages[6] have been developed for describing requirement specification.

(1) Informal specification: uses natural language to describe requirements, which is easy to describe and understand, but ambiguity and inconsistency.

(2) Semi-formal specification: means that the specification techniques have formal syntax without for-

*收稿日期: 2007-10-28
作者简介: 万海 (1976年生), 男, 讲师; E-mail: wanhai@mail.sysu.edu.cn

mal semantic including JSD, object-oriented visual modeling language[7-8] (such as: UML). point out the main drawbacks are① understanding of models can be more apparent;② wasting considerable time resolving disputes over usage and interpretation of notations;③ rigorous semantic analysis is difficult.

(3) Formal specification is composed of syntax, semantics. Currently, there are more than 80 formal specification languages. Formal specification language can be roughly divided into 5 categories[9]:

① Model based: Z, VDM, B, Object-Z, etc;② Algebraic based: Larch, OBJ, LOTOS etc;③ Process algebraic based: CSP, CCS;④ Logic or logic programming based: temporal logic, frame logic, action logic, etc;⑤ Network based: Petri, Predicate Tran. Network.

Although traditional logic programming languages provide a powerful tool for knowledge representation, they cannot deal directly with incomplete and uncertain information about software requirement engineering.

extended logic programming language to contain classical negation in addition to negation-as-failure not[10]. General logic programming language[11] provides negation information implicitly through closed world reasoning. The semantics of the extended logic programming language is base on the notion of answer sets. In [12] the authors consider primarily well-founded extended logic programming language. The answer such a program returns to a ground query Q is yes, no, or unknown, depending on whether the answer set contains Q, Q, or neither. The existence of several answer set indicates that the corresponding program P has several possible interpretation. That is, it is possible for a rational reasoner to construct several theories (called answer set[10]) satisfying P.

Aimed at the features of training students using service-oriented methods and technique, this paper proposes service-oriented requirement elicitation and analysis method based on answer set semantic, which can help students present requirement more natural than other formal method. Furthermore, the negation business logic can be represented and business logic can be reasoned by answer set programming solvers, such as DLV, Smodels, etc. In order to apply answer set semantic in requirement, service-oriented requirements Meta-model is presented, with which requirement elicitation is easily realized and integrated with Subject-Predicate-Object-Method (SPOM), requirement specification is precise and easy to maintain.

2　Answer Sets Semantic and Programming

A general logic programming can be defined as a set of rules of the form[13]:

$$A_0 \leftarrow A_1, \cdots, A_m, \text{not } A_{m+1}, \cdots, \text{not } A_n \quad (1)$$

where $n \geqslant m \geqslant 0$, and each A_i is an atom.

The word "general" is used for the fact that such rules may contain negation, and consequently, are more general than Horn clauses.

Using CWA (Closed World Assumption) can be viewed as a simple form of non-monotonic reasoning[14]. The answer set semantics of logic programming described below is an extension of the stable model semantics, whose semantics defines under which condition a set S of ground atom is a "stable model" of a given program. The stable model indicates that the program has several possible interpretations.

Gelfond and Lifschitz defined the answer semantics (also called the stable model semantics) for programs with negation as failure[10,15]. Let M be any set of atoms from P. The reduct P^M of the program with respect to M is obtained by deleting from P:

① Each rule that has a negative literal not a in its body when a belongs to M, and

② All negative literals in the bodies of remaining rules.

As P^M is negation-free, it has a unique minimal Herbrand model. If this model coincides with M, then M is a stable model of the program P. If M is the set of atoms that follow logically from P^M, we can say that our belief was consistent and "rational".

For instance, the reduct of the program

$$\begin{aligned} p &\leftarrow \text{not } q \\ q &\leftarrow \text{not } p \\ r &\leftarrow p \\ r &\leftarrow q \end{aligned} \quad (2)$$

relative to $\{p, r\}$ is

$$\begin{aligned} p &\leftarrow \\ r &\leftarrow p \\ r &\leftarrow q \end{aligned} \quad (3)$$

A set M of atoms is an answer set for if (2), it is the set of consequences of the reduct (3). For instance, the set of consequences of the reduct (3) is $\{p, r\}$. Then the set $\{p, r\}$ is an answer set for (2). similarly, the set $\{q, r\}$ is an answer set for (2).

Apparently, the answer set semantics belongs to a higher level of computational complexity than the stratified programs semantics and well-founded semantics. Thus the answer set semantics is more expressive and very useful for nonmonotonic reasoning.

Answer set programming（ASP）is a new form of declarative logic programming. ASP interprets a logic program as a constraint on sets of literals, just as a propositional formula can be viewed as a constraint on assignments of truth values to atoms. Currently, several systems have been developed, which can be applied to compute the answer sets of a logic program based on answer sets semantics, such as DLV and SMODELS[15]. SMODELS has two parts, models and lparse. The first part, smodels is the actual logic programming engine doing all the hard work and lparse work as variable grounding which maps a subset of variables of a rule into ground terms[16].

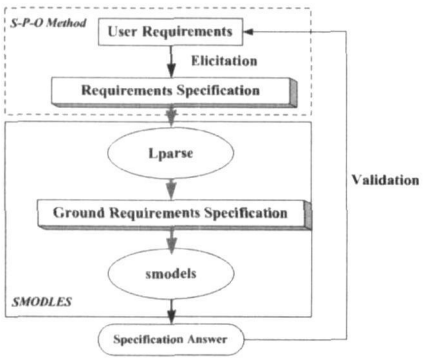

Fig. 1 Requirements Engineering based on SMODELS

How to apply answer sets semantic to requirement engineering depends on the principle of answer set programming（i.e. STABLES adopted in this paper）. The framework is shown in Fig. 1.

3 Service-oriented Requirement Meta-model

In this section, we try to build a meta model for Service-oriented requirement, which mainly consists of three parts: Roles, Actions and Forms. So we give the definition below:

Definition 1: Service-oriented Requirement Meta-model

Service-oriented Requirement meta-model can be specified by a triple $<R, A, F>$, where R represents roles in the system, A is a set of actions performed by roles, and F is the forms of various data resource that the system processes.

Given a software requirement engineering problem $<R, A, F>$, answer set programming represents it by translating it into a logic program $\Pi<R, A, F>$ (or Π, for short) consisting of domain facts and constraint rules that describe R, A, and F respectively, and then find the answer set（stable model）of the program which allow us to query and do some planning work.

3.1 Roles Representation

By investigating the system requirement, we will get the basic information of all the roles that plays a part in the system easily.

Definition 2:

$$\text{role}(r_i) \tag{4}$$

$$\text{superior}(r_i, r_j) \tag{5}$$

$$\text{subordinate}(r_i, r_j) \tag{6}$$

$$\text{samelevel}(r_i, r_j) \tag{7}$$

$$\text{work}(w_i) \tag{8}$$

$$\text{do}(r_i, w_i) \tag{9}$$

$$\text{cooperate}(\{r_i, r_j\}, w_i) \tag{10}$$

where $r_i \in \{r_1, \ldots, r_n\}$, which denotes the set of all the roles in the system, and $w_i \in \{w_1, \ldots, w_n\}$ corresponding to all the works in the system. (4) represents all the valid roles, and (5), (6), (7) together are used to describe the hierarchy of the system; (8) represents all the works that system does, and (9) states role r_i is associated with work w_i while (10) shows the relationship between roles upon work w_i.

And now we can get define some rules, for example, rules for system hierarchy:

Relationships:

samelevel(Ri, Rj) :- samelevel(Rj, Ri).

superior(Ri, Rj) :- subordinate(Ri, Rk), superior(Rk, Rj).

samelevel(Ri, Rj) :- samelevel(Ri, Rk), samelevel(Rk, Rj).

subordinate(Ri, Rj) :- subordinate(Ri, Rk), subordinate(Rk, Rj).

superior(Ri, Rj) :- subordinate(Rj, Ri).

subordinate(Ri, Rj) :- subordinate(Rj, Ri).

Constraints:

:- superior(Ri, Rj), superior(Rj, Ri).

:- superior(Ri, Rj), samelevel(Ri, Rj).

:- subordinate(Ri, Rj), samelevel(Ri, Rj).

:- subordinate(Ri, Rj), subordinate(Rj, Ri).

:- superior(Ri, Rj), subordinate(Ri, Rj).

In order to represent the relationship between role more clearly, we try to divide the roles into different domains and groups, like the domain mode and group mode in Windows Network Management.

Define the propositions below to describe the two different modes:

Domain mode:

domain($d_1, \ldots, d_n$): All the domains exists in the system.

authority($b_1, \ldots, b_n$): All the authorities exists in the system.

dconsistof($d_i, \{r_i, r_{i+1}, \ldots, r_j\}$): Domain d_i consists of roles $r_i, r_i+1, \ldots, r_j$.

authorized($d_i, \{b_i, b_{i+1}, \ldots, b_j\}$): d_i has authorities $b_i, b_{i+1}, \ldots, b_j$.

trust(d_i, d_j): The relationship between d_i and d_j is that d_i trust d_j.

subdomain (d_i, d_j): d_i is the subdomain of d_j

rbelongto (r, d_i): r belongs to d_i

administrator (r, d_i): r is the administrator of d_i

Group model:

group (g_1, ..., g_n): All the groups exists in the system

gconsistsof (g_i, { r, r_{i+1}, ..., r_j}): Group g_i consists of roles r, r_{i+1}, ..., r_j

rbelongto (r, g_i): r belongs to g_i

(For each role in the group, it will restrict the authorities that can be reached by others)

There are concrete rules for special occasions in practice which can be determined in the requirement analysis. And of course there exists evolvement in the system, such as CreatRole (r), DismissRole (r), etc. Such works involves actions, and we may need some knowledge about action representation discussed below.

3.2 Action Representation

An action theory consists of two finite, disjoint sets of names called actions and fluents. Actions transition the system from one state to another. Fluents are propositions whose truth value can change as the result of actions. Unless otherwise stated, a is used to denote an action, f and p are used to denote fluents.

Definition 3.

action (a)　　(11)

perform (r, a)　　(12)

caused ({ p_1, ..., p_n}, f)　　(13)

causes (a, f, { p_1, ..., p_n})　　(14)

executable (a, { p_1, ..., p_n})　　(15)

initially (f)　　(16)

where f and p_i s are fluent literals (a fluent literal is either a fluent g or its negation ¬ g) and a is an action. (11) and (12) show that there exists an action a in the process and role r performs action a. (13) conveys that whenever the fluent literals p_1, ..., p_n are true, so does f, which represents the ramification problem. (14) states that f will be true after the execution of action a in any state of world where p_1, ..., p_n are true. (15) guarantees action a is executable only if literals p_1, ..., p_n hold. Finally, propositions of form (16) describe the initial state of the world. (16) says f holds in the initial state.

Now we need some predicates that will be used in the rules of actions, and the main predicates in these rules are:

– holds (L, T): L holds at time T,

– possible (A, T): action A is executable at time T,

– occ (A, T): action A occurs at time T.

where L is a variable denoting the fluent literals, T is a variable of the sort time, and A is a variable of the sort action. In the following rules, L, T, and A remain the same as above, and S is a variable denoting the conjunction of all p_is ($1 \leqslant i \leqslant n$) in the set { p_1, ..., p_n}.

holds (L, T+1) : - occ (A, T), causes (A, L, S).　　(17)

holds (L, T) : - caused (S, L).　　(18)

holds (L, T+1) : - contrary (L, G), holds (L, T), not holds (G, T+1).　　(19)

possible (A, T) : - executable (A, S).　　(20)

holds (L, 0) : - literal (L), initially (L).　　(21)

Here, (17) encodes the effects of actions, (18) encodes the effects of ramification problem, and (19) is the inertial rule. (20) defines a predicate that determines when an action can occur and (21) encodes the initial situation.

3.3 Dealing with the Forms

Forms are the important data that the system processes. The forms here are alike the tables in database system, but not completely the same. They are the original forms used by a company or enterprise, and the relationships between them are not very clear. So our task is to formalize the forms and find their relationships. First we give following propositions for forms:

Definition 4.

attribute (c): An attribute of a form

value (f, c, x): A value of attribute c in form f

form (f, { c_1, c_2, ..., c_n}): A form consisting of attributes c_1, c_2, ..., c_n

authority ({ r_1, ..., r_n}, f): roles r_1, r_2, ..., r_n have the authority to modify the form

authority ({ r_1, r_2, ..., r_n}, f, { c_1, c_2, ..., c_n}). Attributes c_1, c_2, ..., c_n are visible to roles r_1, r_2, ..., r_n

use (r, f): Role r uses form f

primeKey (f, C): Primary key of form f

foreignKey (f_1, C, f_2): Foreign key of form f_2 using the attribute set C of form f_1.

After we get all the forms, we analyze their relationships using data source diagram. Data source diagram is a kind of static model, which try to specific the conceptual model of the system. It has several ways to represent the data source diagram, and in the following we will show these ways and their corresponding propositions.

Definition 5.

(1) original table and extended table

extend (f_1, f_2): f_1 is the original table, and f_2 is the extended table

Rules are:

primeKey (f_2, C) : - primeKey (f_1, C)

(InsertItem (f_2) : - InsertItem (f_1).

deleteItem (f_2) : - deleteItem (f_1).

The insert and delete operation of extended table can only be triggered by original table.

form (f, c_1, c_2) : - form (f, c_1), form (f_2, c_2),

extend (f_1, f_2).

Form f_1 and f_2 can be combined into a new form.)

(2) main form and sub form

subForm (f_1, f_2): f_2 is the sub form of f_1

Rules are:

foreignKey (f_1, c, f_2): -primeKey (f_1, c), subForm (f_1, f_2)

: -deleteItem (f_1, x), foreignKey (f_1, c, f_2), value (f_2, c, x), subForm (f_1, f_2)

(3) separated table and combined table

ComForm (f, f_1, f_2): f is combined by f_1 and f_2 and the rule:

primeKey (f, c_1, c_2) : -primeKey (f_1, c_1), primeKey (f_2, c_2), cpxForm (f, f_1, f_2)

(4) joint table and composite table

form (f, c_1, c_2, c_3): -form (f, c_1, c_2), form (f, c_2, c_3).

(5) list table and collective table

divAttribute (f, c)

collectForm (f, f_1)

Relationship between list table and collective table:

primeKey (f, c): -divAttribute (f_1, c), collectForm (f, f_1)

InsertItem (f_2): - InsertItem (f_1).

deleteItem (f_2): - deleteItem (f_1).

6. base table and divided table

divForm (f, f_1)

dividedAttribute (f_1, c): -divAttribute (f, c)

partofPK (f, c): - primeKey (f_1, c)

Now we have represented R, A, F of the triple $<R, A, F>$ which is the Meta-model we try to build for the software requirement engineering by answer set programming semantics. And with this Meta-model, we could write the software specification by the logic programs in ASP semantics whose answer set can be computed, and evolve the specification using the dynamic program update discussed in[17].

3.4 Properties of Meta-model

Since the software requirement is represented by the logic programs by ASP semantics, we can find some properties of our Meta-model. Limited by page size, we only present Theorem 1-3 without corresponding proofs.

Theorem 1. Let set S be the set of all the stable models of the triple $<R, A, F>$. If $S = \phi$, then some inference rules are wrong or missed. It is also possible that we get more than one stable model, which is one of the merits of ASP. When this occurs, we may wonder what cause it.

Theorem 2. Let set S be the set of all the stable models of the triple $<R, A, F>$. if $(S) \geqslant 2$, where (S) means the number of elements S contains, ambiguity may exists.

With the theory of dynamic program update proposed in[17], we deal with the specification evolvement by add new logic programs to it.

Theorem 3. The update of software specification, i.e. the update of logic program of $<R, A, F>$ represented by P, can be denoted by $P \oplus U$, where U is an update program and $P \oplus U$ is the update after doing the dynamic program update.

Till now, we have built the Meta-model for software requirement engineering, showed how to write the software specification in the form of logic programs with ASP semantics, and discussed the properties and merits of Meta-model. And this chapter ends here.

4 Subject-Predicate-Object Method (SPOM)

According to service-oriented requirement Meta-model $<R, A, F>$, we propose a new requirement elicitation and analysis method in this section, which can help students constitute a smooth, small-step, incremental and iterative development process cut in by FORMS, clewed and cored by Subject-Predicate-Object method (SPOM), where Subject stands for the users of a software system, Predicate stands for the operations available for the users to activate, and "Object" is the object manipulated by the Subject. SPOM mainly consists of the following ten steps (based on[18], we modified a lot). Note that to be brief, we omit some elements of a diagram, which is illustrated in[19].

Subject-Predicate-Object Method

(1) Obtain the user information, interpret the SUBJECTS and describe their relationship in Organization Structure Diagram (OSD);

(2) Start from FORMS, use Form Flow Diagram (FFD) to depict the flow direction of each concerned FORM and the key operations on each. The expressive relation among the FORMS can be illustrated in a table named Interpret Table of Rules (ITR);

(3) Based on FFD, we use layered Role Function Diagram (RFD) to describe the functional requirement;

(4) Integrating FFD and RFD, we get Business Flow Diagram with swim lanes (BFD) from different point of view to show the business procedure of every key function.

(5) According to FFD and BFD, fill in the Subject-Predicate-Object Table (SPOT);

(6) Recursively interpret the domain vocabulary, namely the OBJECT;

(7) Draw Activity Flow Diagram (AFD) to decompose those non-atomic and key activities in the BFD of step 4;

(8) Recursively interpret the PREDICATE gathered by step 4 and 7;

(9) Mark the dynamic changing states of the OBJECTS by Statechart Diagram (SD);

(10) Indicate the functional and data constraints of different users by Operational Right Diagram and Data Right Diagram.

As can be seen from the above steps, SPOM meth-

od really integrates as a whole by smooth and small steps from the beginning through to the end. Note that these ten steps are not fixed but extensible instead. Nor are they executed straight forth on the contrary they formulate an iterative refined step-by-step procedure.

Because of the limitation of space here we cannot make every step clear. The following is some key steps.

Step 1: Gather as many FORMS as possible and divide them into several groups by some criterion such as background FORMS business FORMS and statistical FORMS.

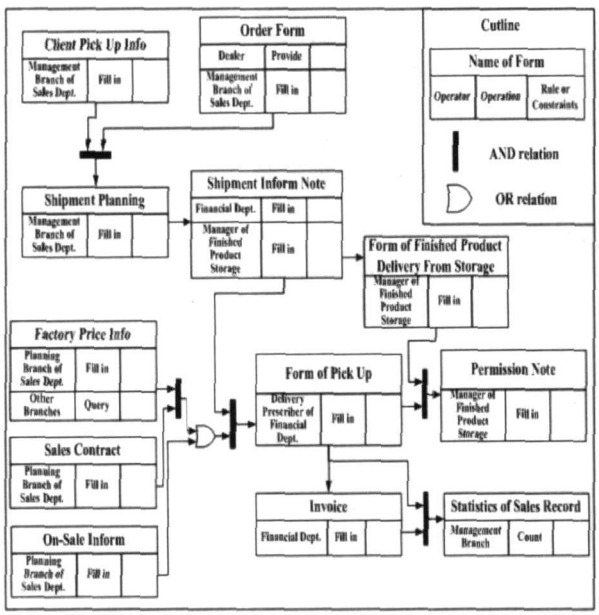

Fig. 2　This diagram is taken from Finished-product Storage of an ERP system. The arrows represent the relation, and the constraints are all omitted for brief.

Step 2: In this step we group the FORMS in FFD of last step by the operation on them, i. e. allocate certain business function (a function usually finishes a specific task) to certain FORMS. These operations can be regarded as the functional nodes in the functional tree which demonstrates hierarchically the functions of a system. In order to specify more clearly we present an example of BFD as follows:

In the last steps, FFD shows the operation on each FORM and the relationship among these FORMS, and RFD illustrates the possible function set of the system, which are derived from FFD. In this step we integrate this information together and view them from a different angle.

Step 3: The major effect of SPOT is to check if the antecedent steps are complete and correct by collecting the acquired results. Every line in the table shows

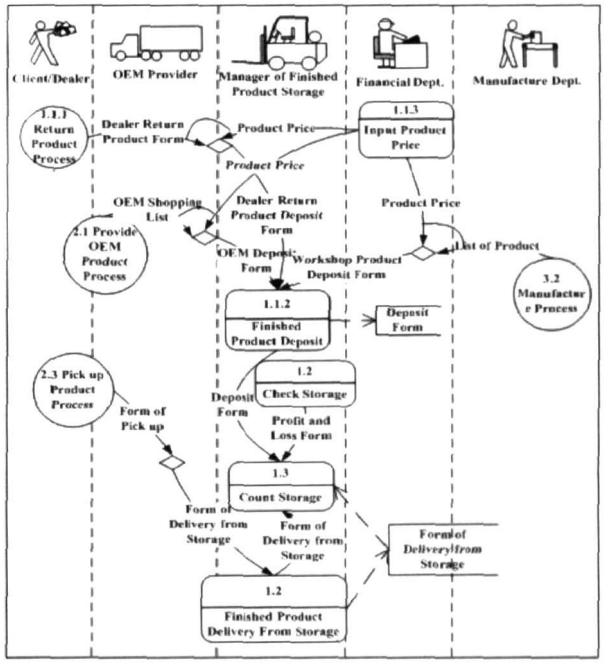

Fig. 3　Business flow of Finished Product Deposit and Delivery of the ERP system, where rectangle stands for functions in the process under consideration, round stands for outer process, diamond stands for a transformation, arc across arrows represents AND relation and rectangle with a right line open stands for data storage. Swim lanes are used to separate the territory of different roles.

the operations made to a certain Object, while every column displays the operations of a certain Subject, which relate to some Object. The table also makes preparation for the right allocation in step 10.

Step 4: The Predicates discussed in this step are the sub functions in RFD. The reason why we do not present them in BFD is that they "belong to" a single Subject (normally they construct the transaction of a given department), and it would distract BFD by adding too much detailed information into it. Thereby we pick them out here and express in AFD. It is essential to keep the involved Objects congruous as what it would like in BFD.

Through practice in the Vantage ERP (Enterprise Resource Planning) System of Zhongshan Digital Platform of Donghua Campus, and Resource Scheduling System of Guangdong Telecom we evaluate SPOM method as follows:

Here we present some comparisons of several major methods which are often adopted in teaching (such as Process-oriented, Object-oriented and Data-oriented) as follows:

Table 1. Requirement Method Comparisons

Types of Requirement Method	User Participant Degree	Functional partition	Development Cycle	Iteration	Completeness of Requirement	Adaptability	Combination with Authority Configuration	Auxiliary Tools
Process-oriented	B	B	M	B	C	B	C	B
Object-oriented	C	B	L	A	C	C	B	A
Data-oriented	C	C	M	C	C	C	C	A
SPOM	A	B	M	B	B	B	A	C

5 Conclusion and Future Work

We have presented a new approach and framework Service-oriented Requirement Metamodel based on Answer set semantic and Subject-Predicate-Object Method (SPOM) to help students elicit and analyze requirements and explained in detail.

There are several issues to be addressed. First of all we need to perfect our methodology including the requirement Metamodel such as Role in different phase Action corresponding to temporal information. Furthermore SPOM method and the framework are to be formalized and followed by the validation of requirement specification to some extent. Another interesting issue to design software as tools to help students directly with the ideas proposed in this paper.

Acknowledgments

This effort is supported by Guangdong Provincial Natural Science Foundation project (07300807).

References:

[1] http://domino.research.ibm.com/comm/research.nsf/pages/r.servcomp.html. 2007.

[2] XU Xiao-fei, WANG Zhong-jie, MO Tong. SMDA: a New Methodology for Service Engineering[C]. 2006 Asia Pacific Symposium on Service Science, Management and Engineering, 2006.

[3] SOMMERVILLE I. Software Engineering[M]. [s.l.]: Addison-Wesley, 2000.

[4] SIDDIQUI J, SHEKARAN M C. Requirements Engineering: The Emerging Wisdom[J]. IEEE Software, 1996, 23(3): 15—19.

[5] ABEN M. Formal Methods in Knowledge Engineering (PhD dissertation)[PHD]. University of Amsterdam, 1995.

[6] FRANCE R B, BRUELAND M, Larrondo Peter M M. An Integrated Object-oriented and Formal Modeling Environment[J]. Journal of Object-oriented Programming (JOOP), 1997, 10(7): 25—34.

[7] CHEN Yi-hai, MIAO Huai-kou. Integrating Object-oriented Methods and Formal Methods for Requirement Engineering[J]. Journal of Harbin Institute of Technology (New Series), 2004, 11(3): 296—299.

[8] GELFOND M, LIFSCHITZ V. The stable model semantics for logic programming[M]. Logic programming: Proc. of the Fifth Int. Conf. and Symp, 1988: 1070—1080.

[9] LLOYD J. Foundations of logic programming[M]. Springer, 1984.

[10] GELFOND M, LIFSCHITZ V. Logic programs with Classical negation[C]. Logic programming: Proc. of the seventh Int'l conf., 1990: 579—597.

[11] REITER R. Towards a Logical Reconstruction of Relational Database Theory[C]. M. Brodie, J. Myloppoulos and J. Schmit, editors. On Conceptual Modeling. Springer-Verlag Pub., New York, 1984: 163—189.

[12] IKKA Niemela, PATRIK Simons. Efficient implementation of the well-founded and stable model semantics[C]. In Proc. Joint Int'l Conf. and Symp. on Logic Programming, 1996: 289—303.

[13] ESRA Erdem. Theory and Applications of Answer Set Programming[D]. University of Texas at Austin, 2002.

[14] ALFERRS J J, LEITE J A, PEREIRA L M, et al. Dynamic Logic Programming[M]. APPIA-GULP-PRODE'98, 1998: 1—16.

[15] LU Mei, LI Ming-shu. Review of Methods and Tools of Software Requirements Engineering[J]. Computer Research & Development, 1999, 36(11): 1289—1300.

[16] HSIA Pei, DAVIS Alan M, KUNG David C. Statues Report Requirements Engineering[J]. IEEE Software, 1993: 75—79.

[17] VLADIMIR Lifschitz. On the Declarative Semantics of Logic Programs with Negation[C]. In Jack Minker, editor, Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, San Mateo, CA, 1988: 177—192.

[18] ZHENG Yun-xiang, LI Lei, FU Shao-chun. Requirements Capture and Analysis Method Based on Subject[C], Predicate and Object Logic. NASAC 2004: 158—162. (in Chinese with English abstract)

[19] HE Zhi-qiang, ZHENG Yun-xiang. Recursive Capture and Analysis Method on Software Functional Requirement[D]. Research report of The Software Research Institute of Sun Yat-Sen University, 2003.