Inf. Sci. Lett. 1 No. 1, 21-39 (2012)

Information Science Letters
An International Journal

A Domain Oriented Approach for Network Software Requirement Modeling

Chen Shu[1], Wu Guo Qing[2] , Ye Jun Min[1]

[1] Computer Science Department HuaZhong Normal University 430079, China
[2]School of computer science Wu Han University, 430072, China
Email :

Received March 02, 2012; Accepted March 25, 2012
Published online: 15 April 2012

Abstract: Requirement analysis on NS (network software) poses a lot of problems for the reason of dynamic deploying topology, dynamic proposing and dynamic changing of requirement. In this paper, we propose a unified solution for NS requirement modeling called PVSS. The purpose of PVSS is to provide a stepwise modeling framework on extracting common artifacts of families of NS applications from domain on constructing reusable domain models, then refined and customized for specific NS requirement model.

Keywords: Software requirement; requirement modeling; domain modeling

I. Introduction

Network software (NS for short) has introduced to us new approaches of software architecture, such as SOA (service oriented architecture) [1] to build applications. The architecture of modern NS has some characteristics and purpose in common with and inherited from traditional component oriented architecture [2], in order to provide the facility of reusability and expansibility, reducing the effort for effective management of complexity and raising competition in market for enhancing the quality of components [2,3]. NS as a newly software architecture, however, its requirement engineering (RE for short) is also an evolution from traditional RE paradigm, which means, they have some common aspects. The evidence is that they all have two phases:

- Requirement developing phase: It includes activities like, requirement elicitation, analysis, specification and verification.
- Requirement managing phase: It deals with managing and controlling changing and evolving requirements.

The result of requirement development phase is an agreement between stakeholders and designers about what the users acquire to build on applications and during requirement management phase, the changes to those agreed requirements are controlled and managed and reflected to final applications [4]. Simply speaking, for both of NS and traditional requirement, the source that generate the requirement information is from domain knowledge provided by domain experts and end users, and final application provided by designers is the solution of problems in domain [5].

However, NS has its own characteristics in architecture, which directly cause the different from traditions in developing life cycle, thus, RE for NS is also a shift from traditions. For example, in traditional waterfall model, major effort in requirement phase is to acquire, specify and analyze the requirement information during analysis phase, but in case of NS, requirements are taken only in informal function interaction scenarios as the main focus here is on coding phase. So the way in which the requirements are gathered, specified and used is different according to various software architectures. Considering the service oriented paradigm of software development life cycle [6, 7]. RE for SOA can involve activities of traditional process like modeling, specification, and analysis but the way in which these activities are carried out is different. The main focus here is to identify the services that meet the system requirements and then produce a composition on the basis of selected services. Keeping the above discussed view in mind we can clearly see that RE for NS has a partially different architectural style as well as designing lifecycle from traditional development for software application, hence, a complete new requirement process and methodology is required [6, 8]. For this purpose, we proposed a unified framework called PVSS for NS requirement modeling, and for the conclusion of the problem that occurs in RE of NS.

PVSS is a stepwise modeling and analyzing framework for NS requirement from domain to concrete function unit that distributed over internet. Function unit is a self-contained, function-independent, interactive and concurrently-executing software component that deployed on internet, typically, such as services in SOA. In order to give a unified glossary in this paper, we more than prefer to use service instead of function unit in the following sections. As PVSS has much difference from other approaches, thus, except for what we have stated above, we shall first discuss about our motivation before detailedly introduce PVSS: What can PVSS modeling framework do, what is the position of PVSS in NS requirement engineering.

2. What can PVSS Modeling Framework Do

In model driven requirement engineering, requirement model serves as a supplement of requirement specification, in order to analyze and organize informal requirement statements into a form that can be verified by users, further more, used as an input to design and implement process for more concrete designing models. According to the research in NS requirement engineering, the role of requirement model in NS engineering should focus more on reusable domain presentation for a family of similar applications, due to its characteristics and development life cycle that differs from traditions. According to our investigation on different network applications, NS is always a domain crossing, multiple roles mixing and large scaling complex system. This means the RE of NS emphasizes more on requirement assets reusing other than redesigning, in order to reduce unnecessary cost. So, when we are building a requirement model for NS, we should shift our mind to consider more about source reuse, which means to provide a mechanism for establishing a stable and reusable asset of the requirement model which building once for all that reflects the common aspects for a family of similar applications [9]. On the other side, consider the NS development life cycle, requirements of NS can be proposed and modified by users continuously, the time situation of the requirement that proposed and modified may be uncertain (random time sequence, fuzzy time sequence, etc). This means NS requirement differs with traditions in nature, such as uncertainty, high changeability, conflict, fuzzability, uncompleted, high performance requirement, etc. ,thus, attached on the backbone of the requirement model, the preference requirement information that specific application focus on should be self-adaptable and customizable.

More specifically, considering the characteristic of the development of NS application, instead of making requirement model as a reference for designing and coding, users in NS focus more on selection and composition of services, in order to directly deduce the final application. This means requirement model should also be able to directly reflect the structure of final application. On the other side, due to the self-topology-evolving properties of NS, except for what services are selected, how services are composed and evolved should also be represented in requirement model.

For those issues discussed above, we have summarized the following aspects that PVSS framework would concern:

- Support for model reusability, especially for large scale applications. The reason for this issue have discussed previously.
- Having innovative and creative methodology as integral part of the NS requirement process. Model driven process seems to have more practicability in NS development. To traditional software development, people always argue that, the models that built in requirement phase, totally lost their touch with implementation, so researches about transforming models to program code have proposed, however, most of these them are not practical. In NS development, designers do not have to care about how services are coded, thus requirement model can be a reliable bridge that connects abstract problems with final application.
- Offering high level language support for requirement specification. This issue is inherited from traditional model driven requirement process. Requirement model that elicited manually from requirement specification needs a well-defined, complete and visualized model specification language, which can also be graphical notation.
- Able to give orchestration mechanism for service composition and avoid flaws during service invocation. When services are determined according to their individual requirements, we need to see whether they will work properly in a workflow by making composition and whether the integrated system still meets the global requirements.
- Support for QoS modeling, and aggregating the QoS dimensions of the individual services. This allows verifying whether a set of services selected for composition satisfies the QoS requirements for the whole composition.
- Able to redesign and redeploy the composed services when new requirements are proposed and old requirement are changed over time.

The issue that listed above are depended and overlapped with each other. For example, how flaws of composite services are identified is depend on how model was constructed, and the definition of model specification language is also depended on applied requirement analysis methodology. Thus, it is not a good idea to separately doing research on these issues, which means, it is better for these issues to be considered in a same semantic scope. On the other side, researches to these new characteristics of RE on NS breaks through the initial condition and boundary of traditional requirement engineering. These researches trend in RE of NS has been approved in the IEEE Requirement Engineering Conference [10, 11, 12, 13]. In such a situation, an open, cooperative and reusable unified requirement modeling framework is needed urgently.

3. PVSS framework

In order to give a briefly but clearly view of using PVSS for domain oriented requirement engineering of NS, we summarize the main concepts including modeling phase, tasks and products of our approach from domain analysis to domain implementation in table 1.

Table 1 phase, tasks and products of PVSS methodology

Phase	Tasks		Products		
Domain Analysis	Domain concepts modeling		Concepts model	Domain model	Stable content
	Static view	Problem modeling	Problem model		
		Role modeling	Role model		
	Dynamic view	Roles Interaction modeling	Implicit Roles interaction model		
Domain Design	Knowledge of roles-multi-viewpoint modeling		Structured viewpoint model	Divide-and-rule of view point model	Changeable crust
	Architectural modeling		Activity model	Design model in each view point	
			Scenario model		
Domain imp	Mapping from design to implementation web services		Common services and orchestration model	Implementation model	Concrete mapping

Following the traditional domain oriented methodology [14], PVSS framework provided three phases for NS requirement modeling. Domain Analysis phase mainly focus on building a stable and reusable domain model, in order to give a static view of entities that from an organization, interactions of some certain entities and representing the problems about to solve. Domain Design phase offer a structure model in the view of pre-implementation design according to analysis solution, however, could either be redesigned for specific application or be modified according to requirement change. Domain implementation phase maps design solution to concrete services, as well as validating the requirement validity, more over, offered an reflect mechanism for self-adapting on design models.

For the sake of establishing a complete and reliable model, some techniques have been used on PVSS. For example, we have applied information filtering [15] and usage mining [16] on domain analyzing, in order to extract common concepts from domain knowledge. We also have applied design pattern techniques on domain design process. These issues are ignored in this paper because they are not our main concern in introducing PVSS, and we shall report them in our further papers. In the following sections, we will introduce PVSS in detail by following the three phases. In the following subsections, we will introduce the four layers of PVSS in detail.

4. Over view of PVSS

In PVSS framework, we offered a stepwise requirement modeling process from top to bottom through domain analysis, domain design and domain implementation as we can see in Fig 1.

The core content of NS PVSS model is a reusable domain model which is a product of domain analysis phase. A domain model in RE can be thought of as a shared conceptual model of a domain of interest (referred to as a problem domain) which describes the various entities, their attributes, roles and relationships, with the constraints that govern the integrity of the model elements comprising a problem domain for a family of similar NS applications. Domain model is thought to be a reusable backbone that with changeable requirement information that decided in domain design phase attached on it. Domain design supported by the PVSS technique provided the architectural and detailed design solution to the requirements of a family of NS applications specified in this phase, while Domain implementation approaches the mapping of design models to common services, behaviors and communication orchestrations and finally adopted in implementation platform.

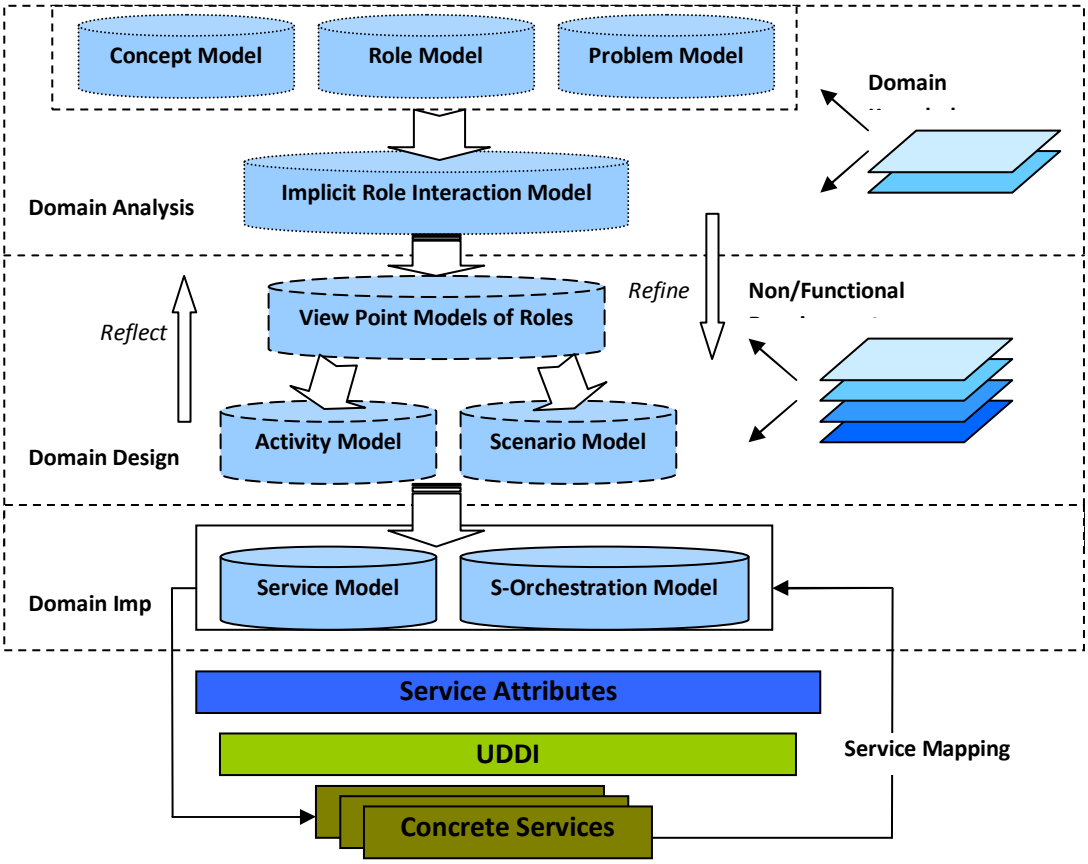

Fig 1 The Three Phases of PVSS Framework

5. Meta Model of PVSS Framework

In PVSS, we provide a unified process from domain analysis to domain implementation (Fig 1). In domain analysis level, a static view of organization with its roles and to-be-solved problems are represented. In domain design level, according to the problems that represented in the upper level, a design structure solution for NS application is constructed. Scenarios model that reflects the functional

requirement is decided in this phase, and decomposed into process which represents the orchestration of service composition. We also have applied Divide-and-rule multi-view modeling technique in order to reduce the analyzing complexity on large scaling system. In domain implementation level, the designing structure solution is finally mapped to structured common services distributed on internet.

5.1 Domain Analysis Phase

In domain analysis phase, the first step is to analysis the concepts that forms the organization in which the business process intended to run. An organization is composed of entities with their functions they would perform. The organization concept model provides a static view of entities it has and their relationships among them. Meta model of Concept model is shown in Fig2.

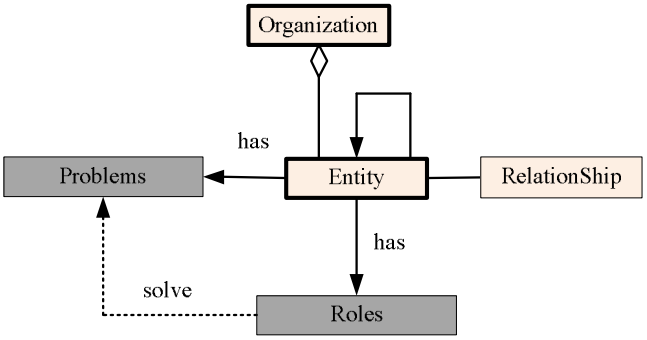

Fig 2. Concept model

Entity is an abstraction of the elements that compose an organization. In another word, anything that belongs to an organization which can be extracted from domain knowledge such as department, employees, equipment, actors can be defined as an entity. Relationships between entities that represented in this step are simply transplanted from ontology. Ontology formally represents knowledge as a set of concepts within a domain, and the relationships between those concepts. It can be used to reason about the entities within that domain and may be used to describe the domain, thus, it means that the semantics of concept model can easily be supported by ontology, and has a more clearly perspective in analyzing the following properties. PVSS framework in this lever gives minimal basic modeling terms to support the representation of the following contents:

- Organizational structure: definition of an organization, relations among organizations and entities, decomposition of sub organizations and entities, purpose of an organization.
- Location information: sites or buildings, locations within sites.
- Organizational history (merging, renaming, repurposing).

Besides, PVSS also allow domain-specific extensions to add classification of organizations and relations. The product of this step is to offer a reference on analyzing the roles and responsibilities, as well as problems that organization intended to solve. In our opinion, every single entity or a group of entities in organization is associated with one or several problems that the organization intended to solve, and these problems are formed into problem models, the Meta model of problem is demonstrated in Fig3. More over, these entities are also associated with some actors with their skills they possessed, to fulfill

their responsibilities in order to solve those problems. Actors with same responsibilities are abstracted into roles and formed into role model, the Meta model of role is shown in Fig 4.

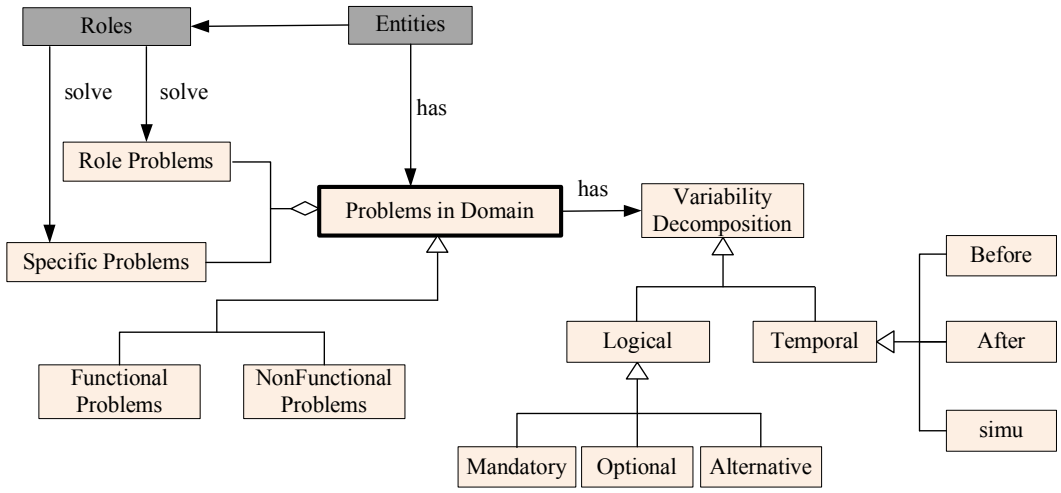

Fig 3 Problem model

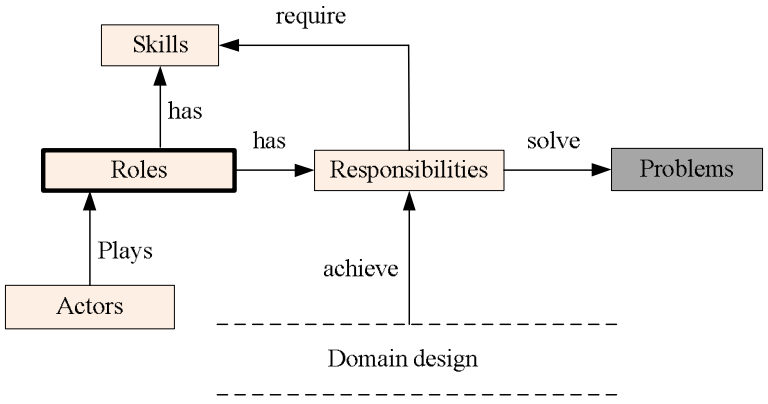

Fig 4 Role model

Problem model in Fig 3 and role model in Fig 4 are an analyzing solution from concept model. By analyzing concept model, serious of roles and problems that associated with each entity are extracted from domain knowledge. Each role has its own view point on a same problem. Thus, problems are decomposed into different role problems. Specific problems are a kind of special problems that should be solved by certain actors. For example, actors that take the responsibility of organizing system users, maintaining system circulation can be abstracted as the role of system manager, they have in facing of a manager-role problem in how to organizing and maintaining system correctly and efficiently. How ever, some kind of special actors such as novice managers might have problems in system using, thus they may have the preference of specific problem of “how to use the system for the first time”. By the concept and role analyzing, designers only should have to concern the problems in the scope of corresponding view point, and this strategy could be kept in implementation level, which means, designers only have to concern about what problems are, and how to solve these problems in the scope of view point. This approach provides us a divide- and- rule policy, see fig 5.

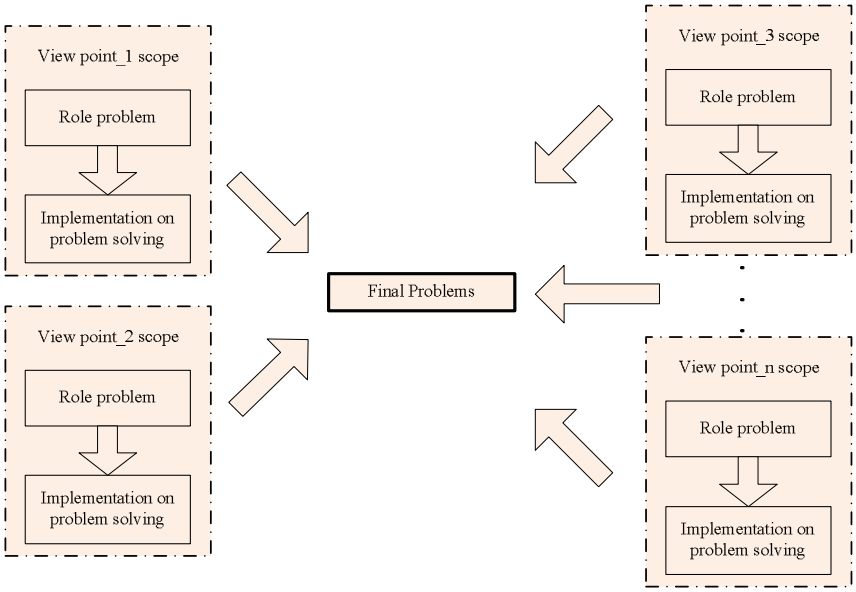

Fig 5 structure of divide-and-rule strategy

Notice that, In PVSS framework, role problems could be directly inducted from role model, and composed into organization problem, and role problems could be also obtained by decomposing organization problem. This means that "composed into" and "decomposed from", either direction is possible. This is due to the rational iterative software developing process. When more problems are discovered in organization, role problems are then decomposed from them. On the other side, when more problems are attached to roles, these problems should also be combined to the related organization problem. The relations between problem and decomposed problems are represented by variability decomposition. In PVSS, we provided two kinds of such relations: logical and temporal. Logical relation defines the static relation on problems such as mandatory, optional, alternative and or. The motivation is that, in practical system, it is not always necessary for all the sub-specific-problems to be solved for the contribution of solving the upper-general-problem. That is, some problems can be solved by alternative sub problems, and the optionally solvent of some sub problems that have different features may have the ability to enhance the solvent of the upper problem. For instance, when dealing with a family of information management system, we shall have a general problem such as "maintaining user data security" which may have two specific sub problems "maintaining permanent user data security" and "maintaining user session data security". Thus, these two problems are both mandatory if all such systems provide user information security. Temporal relations are used if the solving of sub problems has temporal sequence in the scope of the upper problem. For example, for a family of journey planning system, we may have a general problem of "provide journey planning assistance", we decompose this problem into "gathering trip preference" and "evaluate trip plans", the solvent of this two mandatory sub problems have a sequence in contributing solving the upper problem in "gathering trip preference first, evaluate trip plans then", which means, the solvent of "gathering trip preference" is a precondition for the solvent of "evaluate trip plans". Notice that, in practice, the temporal sequence of problem solving for some kinds of application is not obvious, or these problems are solved in an implied interleaving way. Furthermore, this may involve into another more complex that, the roles that taking the responsibility to

solve these problems may have dynamic interactions. Fortunately, in problem model, these issues could be left in an implicit way, as this task is left to domain design level. Thus in problem model, we could make an assumption that those interleaving problems are solved simultaneously. The two kinds of relations can be used in mixture. A simple example on decomposing the problem of "malicious code detection through taint analysis" is shown in Fig 6.

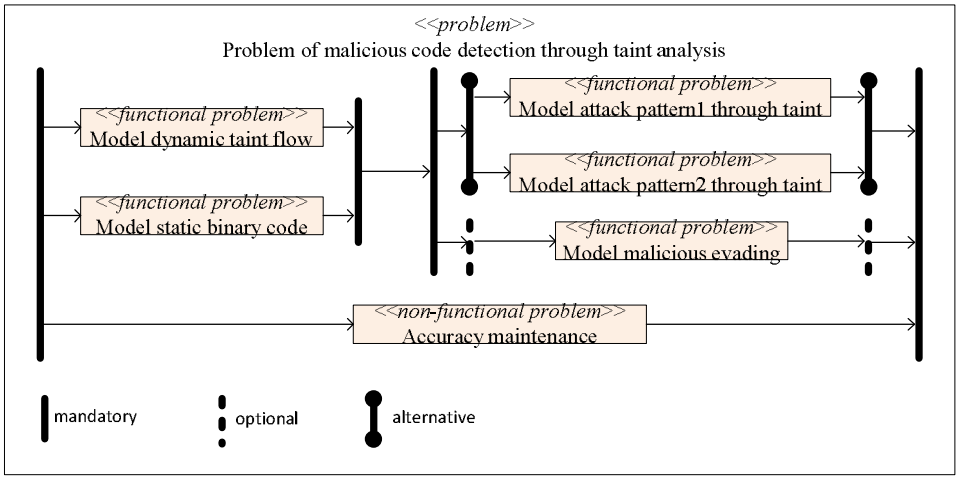

Fig 6 Decomposing of malicious code detection through taint analysis problem

The problem of malicious code detection in fig 6 could be decomposed into 6 sub specific problems. The horizontal line represents temporal relations, while the vertical line represents logical relations. Thus in fig 6, it means that, the problems of "model dynamic taint flow" and "model static binary code" are two mandatory problems, in this example, they should also be solved simultaneously. After they have been solved, alternative problems "model attack pattern1" and "model attack pattern2" are solved alternatively, because either of them could contribute to the solvent of upper problem. Besides, the problem "model malicious evading" could be solved optionally, because it would enhance the effect of malicious code detection, however, not strictly necessary. In the mean time, the problem "Accuracy maintenance" should also be solved.

Problems could also be extended into functional problems and non-functional problems. Functional problems are associated with functional requirement and represent the goals that the function units supposed to accomplish. In domain implementation level, functional problems are finally solved by the composition of function services. Non-functional problems are quality related problems such as dependability, security, usability, etc, and were determined by QoS properties. In the example of fig 6 problem "Accuracy maintenance" is a non-functional problem.

5.2 Domain Design Phase

Problems are solved by the responsibilities that possessed by roles (Fig 4). Some skills may be required to exercising the responsibility. Skills could be technology, algorithm or other abilities that may be needed to perform the responsibility.

Responsibilities are fulfilled by activities that modeled in domain design layer. The domain design process produces an architectural and detailed design solution to the problems that represented in domain

model. Each responsibility is associated with several activities, which are formed into activity models, see Fig 7. For instance, a role of "store manager" may have a responsibility to perform some certain activities, such as "register cargoes", "query certain cargoes", "remove cargoes" to solve the problem of "how to manage storage" that demanded by entity "warehouse", in order to satisfy the ERP needs of some organization. Activity model not only describe the own properties such as pre, post condition, I/O definitions, but also provides a static view of the relationship among roles, responsibilities and activities.

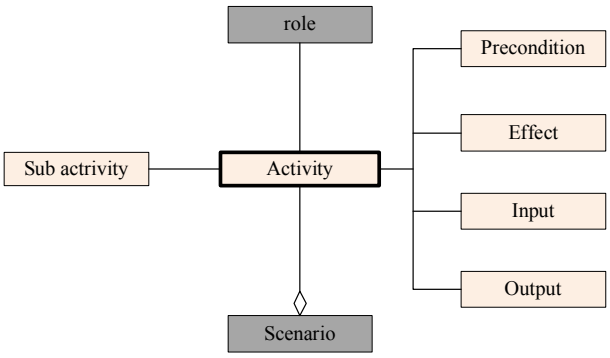

Fig 7 Activity model

The solvent of specific problems might require one or more roles to participant, on the other side, the solvent of two or more problems may act as an interleaving way. Thus both of these situations require a dynamic interaction of roles. These issues have been obscurely represented in domain analysis phase, however should have a clearer description in domain design level, and we represent this dynamic interaction in scenario model see Fig 8. Thus, the main goal of scenario model aims at identifying how roles should cooperate by taking the responsibility of executing certain activities to solve some certain problems.

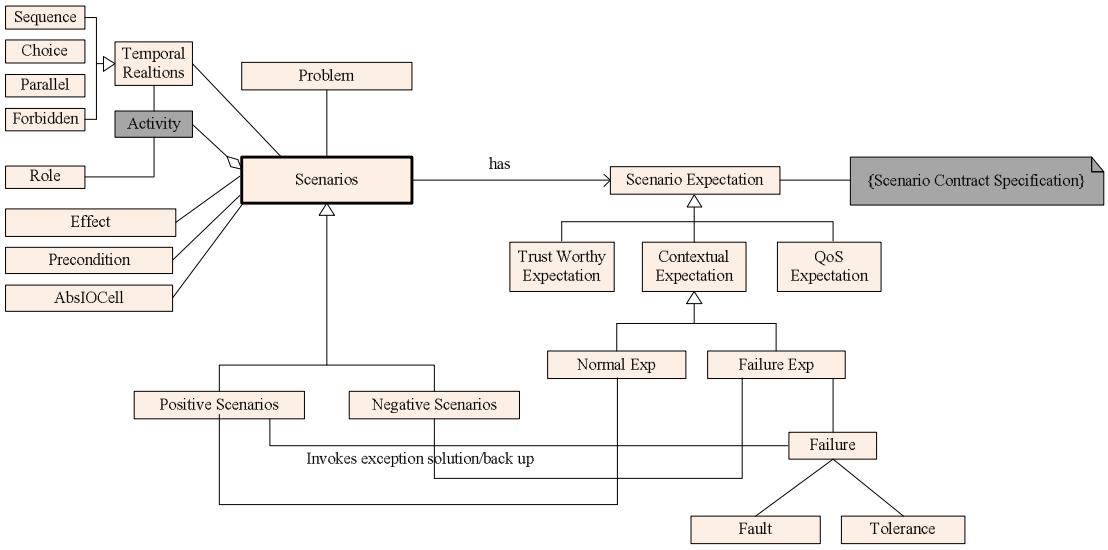

Fig 8 scenario model

Scenario can be recognized as a series of common activities within a certain time scope and with a certain temporal schedule of interaction. Two or more scenarios can also be assembled into a composed one. The following temporal relations are provided:

(1) Sequential: means two activities or scenarios are executed linearly. The execution of one is the precondition of another. Take another word, the latter can not be executed until the former one has reached the end.
(2) Choice: means two or more activities or scenarios can only have one to be selected to execute. Choice relation has two type, undetermined choice and conditional choice. The former one means a random selection from many candidates, thus is unstable for different executions. The latter one means that the one meets the given conditions can be selected to execute, thus is stable. Undetermined choice can be refined into conditional choice in step wise scenario modeling.
(3) Parallel: parallel means a concurrent execution among several activities or scenarios. We give its semantics on interleaving execution. Two activities can communicate with each other in paralleled executing.
(4) Forbidden: means a mutex execution. When one activity or scenario is selected to execute, the others should be discarded if they have forbidden relation.

As activities are executed by taking the responsibility of roles, thus scenario can also be considered as interaction of roles, and played as a basic element to fulfill the responsibility of roles which reflect the functional requirement of the target system. A scenario may have optional pre-conditions to be satisfied in order to have it triggered. Effect of a scenario demonstrates the result of the execution of the scenario. In scenario model, activities are communicated by send and receive output and input data. For more detail, the synchronization communication of activities are achieved directly by I/O definition of them, see Fig 9. The asynchronous communication of activities are represented by the help of abstract I/O cell that defined in scenario model in the way that activities produce input data to “AbsIOCell” and consume output data from “AbsIOCell” asynchronously, see Fig 10. In order to reduce the complexity in this level, the communication among scenarios in PVSS is limited in asynchronous. Thus, communicated through the “AbsIOCell” of scenario either, see Fig 11.

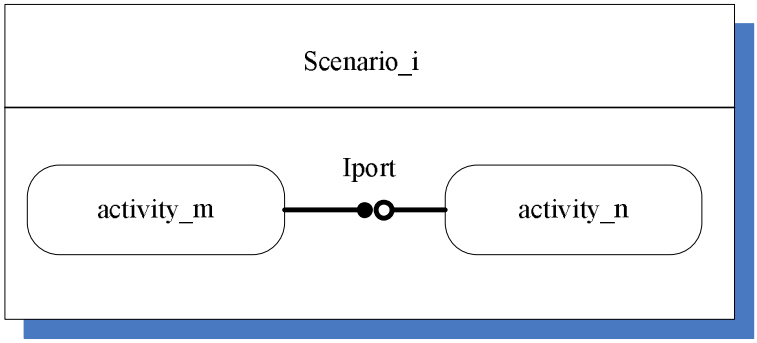

Fig 9 Synchronization communication of activities

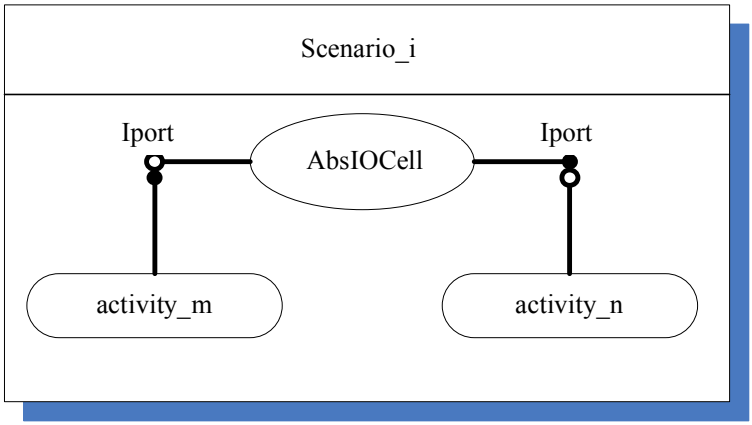

Fig 10 Asynchronous communication of activities

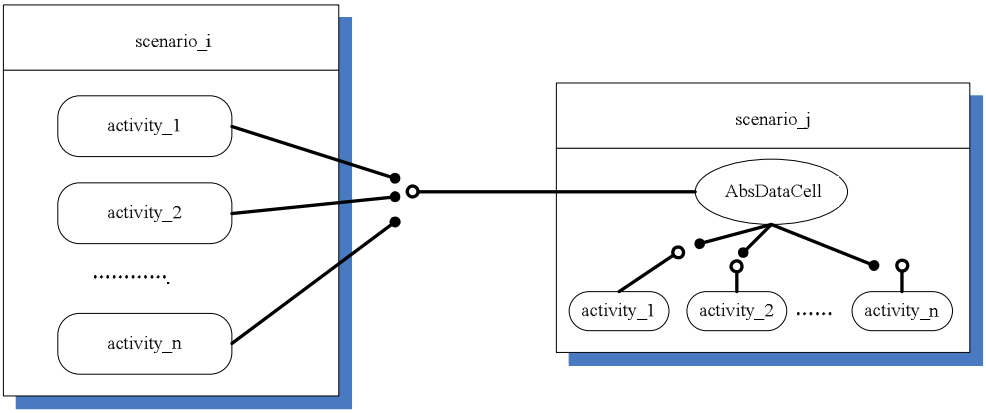

Fig 11 Asynchronous communication between two scenarios

Scenarios also have properties to be satisfied which we prefer to call it expectations in scenario model. Expectations can be refined into the combination of service properties in domain implementation. There are also two kinds of scenarios, positive scenarios reflect the normal execution that a system demonstrates, while negative scenarios simulate the abnormal executions, such as exception handling.

5.3 Domain Implementation Phase

Activities are implemented in domain implementation layer by service model, see Fig 12. Domain implementation supported by PVSS approaches the mapping of domain design models to concrete services in implementation platform (such as internet). Services represent the common dynamic characteristics and functions that NS exhibit. Service is defined as atom and composite service respectively. Atom service is considered as the minimal element to fulfill functional requirement of the system. Composite service is a composition of some other services by service temporal relations that defined by control structure. Atomic services are independent and rely on their operations only to fulfill their functionality while composite services rely on other services. Atom service communicate through messages that emitted by its operations. Messages are divided into input and output type and their value range are determined by message data type. Messages are the refinement of I/O property of activity in

scenario model. The functionality of each service is represented by operations. Operations are composed by control structures either to represent the functionality of its related service, and can be extracted by analyzing the verbs which describe the expected behaviors of system in domain knowledge.

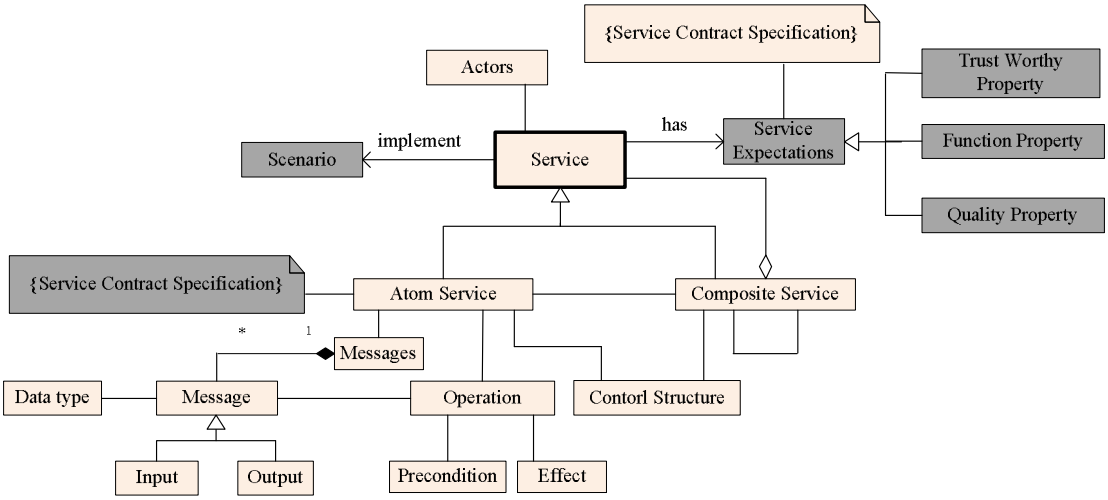

Fig 12 service model

Service model not only represent the static definition of each services, but also depict a predefined orchestration composition of services. The orchestration composition of services are defined by control structures, such as “sequence”, “choice”, “parallel”, which provides a convenience mapping to SOA specification standards, such as WSDL, WSDI. Besides, compared with WSDL, WSDI, PVSS in service model provides service compatibility and substitutability check, as well as service temporal correctness check. In order to fulfill those functionality, in the background of PVSS service model, we semantically considered it as a STS (state transition system) and represented as a quintuple T :

$$T = (S, A, \rightarrow S_0, Nil)$$

In which S is a finite set of states, A is a finite set of operations, $\rightarrow$ is a transition relation in the way that $\rightarrow \in S \times A \times S$, S_0 is the initial state, and Nil is the terminated state. Thus the dynamic evolution of service model is represented as:

$$P \xrightarrow{e_1} P_1 \xrightarrow{e_2} P_2 \xrightarrow{e_3} P_3 \rightarrow \ldots \xrightarrow{e_n} Nil$$

More precisely, when we interpreting the semantics of service interaction, the operations could be considered as message sending or receiving, and internal operations are unobservable from outside. The following example shows how a service defined in PVSS be represented by STS specification format.

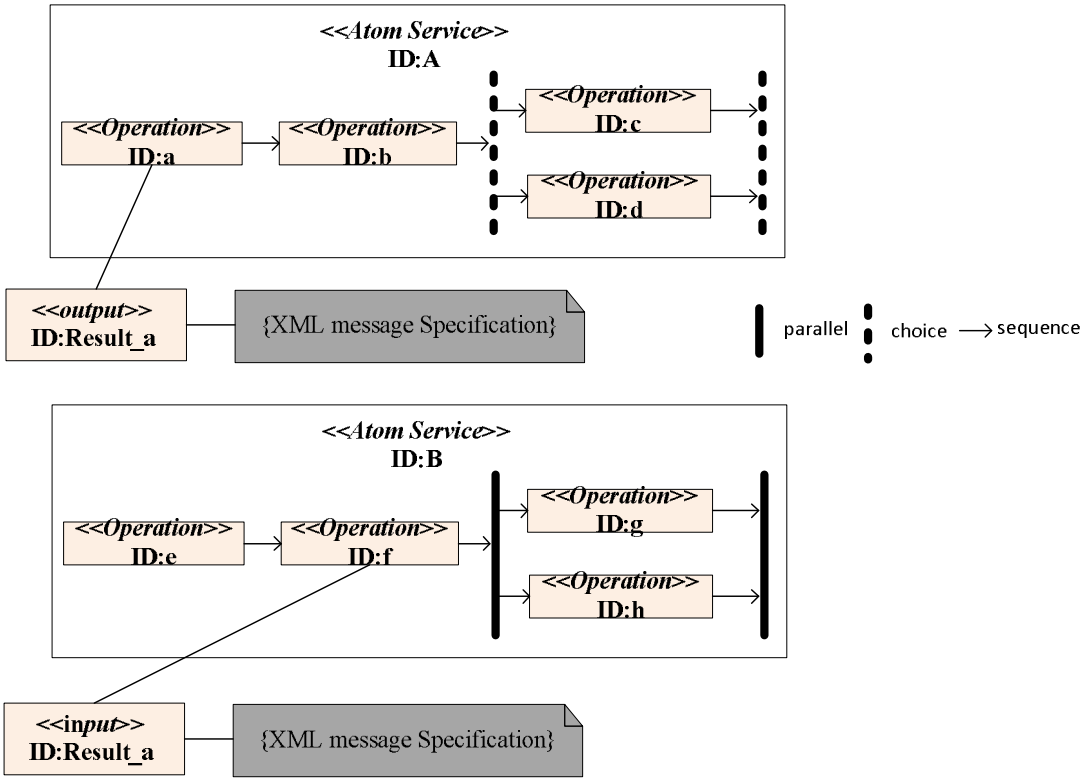

Fig 13 definition of two atom services A and B

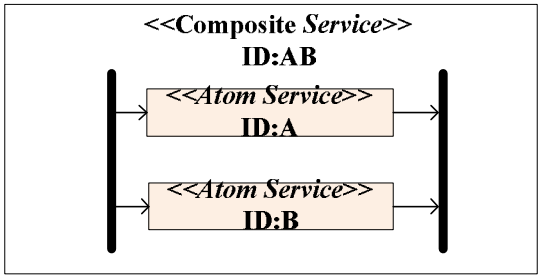

Fig 14 definition of the composition of two services

Fig 13 describe two atom services A and B, with four operations, the difference between them is that, *A* has operation *c* and *d* executed selectively, while *B* has operation *g* and *h* executed synchronously. Fig 14 represents the composition of serve *A* and *B*. thus, it can be semantically described as: $AB \square ((a!result\_a.b.(c+d).nil) \| (e.f\ ?result\_a.(g \| h).nil)).nil$, in which operation a of service *A* communicate with operation *f* of service *B* through message $result\_a$.

We also have designed the following axiomatic operation transition rules of STS to describe service derivation:

- General rule: a service is defined as a message based process in the way of $P = \sum e_i.P_i$, in which e_i is a pure operation or operation with message interaction and could either be a message sending: $e!m$, or a

message receiving: $e?m$, or internal interaction τ.

- Sequential execution rule: $\frac{}{e.P \xrightarrow{e} P}$;
- Non deterministic choice execution rule: $\frac{P \xrightarrow{e} P'}{P+Q \xrightarrow{e} P'}$;
- Parallel execution rule: $\frac{P \xrightarrow{e} P'}{P \mid Q \xrightarrow{e} P' \mid Q}$;
- Synchronization-communication rule: $\frac{P \xrightarrow{e?m} P', Q \xrightarrow{e!m} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$.

By the use of such definition, properties of services such as compatibility, substitutability, safety, liveness. Etc can be validated by finite state model checking technology. We give an example of compatibility and substitutability check as following, other properties check will be represented in our further report.

Compatibility of web service states the fitness of them, which means when one service emits something, the other one should finally receive it. Take another word, the messages that connects two services should not only grammatically matched, but also dynamically matched. The former problem is easy to solve, the letter one seems more complex. As we have discussed previously, a web service or a NS application could be described in an operation transition system, thus according to lacus [17, 18], the definition of compatibility in synchronous communication could be defined as: "Two services are compatible if they have opposite I/O behaviors". Thus we see that two services are compatible with each other if they are able to communicate with each other (surely they are compatible if they do not need to communicate). For example, service $(a!m.A)$ is compatible with service $(b?m.B)$ if $A = \overline{B}$. Internal operations are all excluded from outside. The message sending could be described as a transition: $A \xrightarrow{e!m} A'$ while message receiving is described as $A \xrightarrow{e?m} A'$, thus the collection of sent and received message of a service A is described as:

SentMsg(A) = { m | P = $\sum e_i.P_i$, m$\in${!e_1, !e_2, !e_3…!e_x} }

ReceivMsg(A) = { m | P = $\sum e_i.P_i$, m$\in${?e_1, ?e_2, ?e_3…?e_y} }

With the definition given above, the following algorithm could be used to check whether two services are compatible:

SComp(Service A, Service B)

BOOLEAN Res=TRUE;

IF(Res == TRUE &&

SentMsg(A)!=empty&& ReceivMsg (B)!=empty&&

SentMsg(B)!=empty&& ReceivMsg (A)!=empty)

{

$\exists$ m $\in$SentMsg(A) **AND** $\exists$ m $\in$ReceivMsg(B)

$A \xrightarrow{!m} A'$; $B \xrightarrow{?m} B'$;

$\exists$ n $\in$ReceivMsg(A) **AND** $\exists$ n $\in$SentMsg(B)

$A \xrightarrow{?n} A'$; $B \xrightarrow{!n} B'$;

The algorithm given above asserts that every message from a sending should correspond to a receiving, and so does the convert. Further more, the algorithm also satisfy the following definition of compatibility between two services, which means that every message that has ever been sent from one service should have participated in the design of another service:

$sended_P(P') \subseteq received_Q(Q')$ and $sended_Q(Q') \subseteq received_P(P')$

Substitutability property of service can be identified by the help of compatibility, however, the algorithm of "SComp" seems too strong, because it asserts that every message from a sending should correspond to a receiving and vice versa. So we add "weak-compatibility" to analysis substitutability. The "weak-compatibility" checking algorithm: "WComp" guarantees that every message from a receiving should correspond to a sending, but the convert is not necessary. And we may have the substitutability checking approach of two services that: Service B is substitutable with service A if B is observational equivalence with A, and B is "weak-compatibility" with every service existed in model that A is "strong-compatible" with. Formally defined as:

$$OE(A,B) \wedge (\forall T : SComp(A,T) \wedge WComp(B,T)) \Rightarrow Sub(B,A)$$

The proof of observation equivalent could be done by exhibiting a bi-simulation relation between the STS of two services and proofed by operation transition rules, which means one service is substitutable with the other if the former one could functionally simulate the latter one.

Service model stands on a conclusion of the upper models that it validates the correctness of domain analysis and domain design process and give a hint to correct the defects. It is not difficult to see from the statement above that, the models that belongs to different phases, has dependencies and refinement

relations such that: concept model has abstract elements such as problems and roles, which is refined into concrete problem models and role models, while activity models have abstract element of scenarios which can be refined into concrete scenario models. Service models also have implementation relation with scenario models. The clearly defined relations not only reflect the natural concept of the requirement evolution process, but also give a clear trace for designers to discovery the source of the flaws.

We have also implemented toolkit (called BDLMT) to support our approach. BDLMT was divided into two parts: graphical based modeling interface in front which was embedded in Microsoft Visio and model checking part in background. The former one gives a graphical interface for domain modeling, and the latter one validates the correctness of the model as well as generates hints for the model redesigning. If you are interest with this toolkit, please contact the author for a prototype copy.

6. Related works

Our work is based on the investigation of current researches of SOA, requirement modeling, problems domain, goals, business processes and model checking technologies. As these researches are independently or partially combined to solve some problems, thus our work partially aims at combined these concepts for domain modeling on NS.

FODA (Feature-Oriented Domain Analysis) [19] uses models represented by feature diagrams, to capture commonality and variability of domain at requirements level, and its intention is to support functional and architectural reuse. In PVSS framework, activities in domain design level can be viewed as the functional features of domain, thus the variability modeling of FODA are partially adopted in activity and scenario model. We also have concept, problems and roles to compensate for the loss of FODA on concept analysis.

MADEM (Multi-Agent Domain Engineering Methodology) [20] is a multi-agent software development methodology for domain engineering. The domain analysis process includes concept modeling, goal modeling, role modeling, and dynamic role interaction modeling. The main concept of PVSS in domain modeling process is very similar to MADEM. The difference between them is that, instead of modeling multi-agent, PVSS is intending to model the reusable chrematistic of web services and provided detail Meta model elements. To this purpose, we construct domain design and domain implementation process from the perspective of web services application, and we also provided service checking mechanism, which are not included in MADEM.

Furthermore, one major feature distinguishing PVSS from the existing domain modeling approaches is that the existing methods uses object-oriented paradigm as the basic modeling element, while PVSS approach uses web service-oriented paradigm, and offers service composition orchestration mechanism. With semantics that defined, the flaws of pre-designed service models can be discovered, and trace to requirement model for designers to correct. This was done by the help of the refined and connection relationship between different models that belongs to different phases. In our further papers, we shall discuss more details about refinement guarantee and error tracing among the models that belongs to different modeling processes.

7. Conclusion and further works

We proposed a domain oriented approach for network software requirement modeling. Our modeling approach provides three phases for modeling requirement of NS applications. Domain analysis phase mainly focus on building a stable and reusable domain model, in order to give a static view of entities that from an organization, interactions of some certain entities and representing the problems about to solve. Domain Design phase offer a structure model in the view of pre-implementation design according to analysis solution, however, could either be redesigned for specific application or be modified according to requirement change. Domain implementation phase maps design solution to concrete services, and validate the correctness of the upper works, and gives hint to designers for the model redesigning.

One difficult problem which not only for PVSS, but also for other requirement modeling approaches is that, whether the models that produced by designers can correctly and completely reflect the reality of software requirement? For this not yet been solved problem, our opinion is that: leave the judgment to practice and experience, and this is also what RE focus on. For our approach, we have tested it with many NS projects. By the investigation on these experiments, we found out that most of them can be correctly and completely modeled, however, we still found out some aspects to improve. One aspect is that, although some functional properties can be validated in services level, but the verification of non-functional factors is still in trouble. We have tried it with stochastic process algebra, such as semi-Markov processes, to estimate the performance of selected services, such as rate of resource utilize, throughput of critical services, but the practicality is still need to improve. On the other hand, during the practice of validating functional properties, we have used optimize approaches for the sake of states explosion with the help of the characteristic of PVSS Meta model, such as compositional deduction and proposition abstraction (details are not included in this paper for the space limits), but these approaches needs some manually configuration, and may not so cordial to amateurs. So develop a more easy-of-use validating approach is our other work.

8. ACKNOWLEDGMENTS

The work of this paper was supported by HuBei Provincial NSFC under Grant No. 2010CDB04001 and China Center College Independent Research Foundation for CCNU under Grant No. CCNU11A01012.

Reference:

[1] Huhns, M. N., & Singh M. P. (2005). Service-oriented computing: key concepts and principles., 1-2, 2-8, IEEE Internet Computing.
[2] Oscar Nierstrasz, Simon Gibbs, Dennis Tsichritzis. (1992). Component-oriented software development Communications of the ACM - Special issue on analysis and modeling in software development, Volume 35 Issue 9. ACM New York, NY, USA.
[3] Feiler, P., Goodenough, J., Linger, R., et al. (2006). Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute/Carnegie Mellon University, http://www.sei.cmu.edu/uls/uls.pdf.
[4] Siddiqi, J. (1996, March). Requirement engineering: The Emerging Wisdom. IEEE Trans on Software, Volume: 13, Issue: 2.
[5] Gilberto Filé, Roberto Giacobazzi, Francesco Ranzato. (1996 June). A unifying view of abstract domain. Journal of ACM on Computing Surveys design, Volume 28 Issue 2. ACM New York, NY, USA.
[6] W.T. Tsai. (2005). Service-oriented system engineering: a new paradigm. IEEE international workshop on service-oriented system engineering (SOSE), Beijing. pp. 3-8.
[7] H.P. Breivold & M. Larsson. Component-based and service oriented software engineering: Key concepts and

principles. SOFTWARE ARCHITECTURE EVOLUTION AND SOFTWARE EVOLVABILITY, p. 67.
[8] W.T. Tsai, Z. Jin, P. Wang, & B. Wu. Requirement engineering in service-oriented system engineering. proceedings of International Workshop on Service-Oriented System Engineering, Citeseer , pp. 661–668.
[9] Arango G Domain (1988). engineering for software reuse. Ph.D. Thesis, Department of Information and Computer Science, University of California.
[10] 13th IEEE International Requirements Engineering Conference (RE'05) proceeding, IEEE, 2005;
[11] 14th IEEE International Requirements Engineering Conference (RE'06) proceeding, IEEE, 2006;
[12] 18th IEEE International Requirements Engineering Conference (RE'05) proceeding, IEEE, 2010;
[13] 19th IEEE International Requirements Engineering Conference (RE'06) proceeding, IEEE, 2011;
[14] Gerhard Fischer. Domain-oriented design environments. AUTOMATED SOFTWARE ENGINEERING Volume 1, Number 2, 177-203
[15] Jin R, Chai JY, Si L (2004) Content-based filtering & collaborative filtering: an automatic weighting scheme for collaborative filtering. In: Proceedings of the 27th annual international conference on research and development in information retrieval, pp 337–344
[16] Cooley RW. Web usage mining: discovery and application of interesting patterns from Web data. PhD Thesis, Department of Computer Science, University of Minnesota 2000.
[17] Berardi & Massimo Mecella. (2005). When are Two Web services Compatible? TES 2004, Springer-Verlag, Berlin Heidelberg, pp. 15-28.
[18] ChenShu, WuGuoQin, XiaoJing. (2008). A Process Algebra Approach for the Compatibility Analysis of Web Services. Future Generation Communication and Networking, Volume: 1, On Page(s): 305 - 308.
[19] K. Kang, S. Cohen, J. Hess, W. Novak, & S. Peterson. (1990). Feature-Oriented Domain Analysis (FODA): feasibility study", Software Engineering Institute, Technical Report: CMU/SEI-90-TR-021.
[20] R. Girardi, & L. B. Marinho. (2007). A domain model of Web recommender systems based on usage mining and collaborative filtering" Requirements Engineering, vol. 12 , pp. 23-40