A Scenario-based Requirement Model for Crossover Healthcare Service

Meng Xi*, #Ying Li*, Yongna Wei*, Naibo Wang*, Yuyu Yin†,
Zhiling Luo*, Shuiguang Deng*, Yihua Mao*, Jianwei Yin*
*Zhejiang University, Hangzhou, China
†Hangzhou Dianzi University, Hangzhou, China
corresponding author
{ximeng, cnliying, weiyongna, wangnaibo, luozhiling, dengsg, maoyihua, zjuyjw}@zju.edu.cn
yinyuyu@hdu.edu.cn

***Abstract*—As the population ages, eldercare and healthcare have become major issues in recent years. Crossover healthcare services, instead of individual ones, have become the main form of service provision. In this work, a scenario-based requirement model (SBRM) is proposed for crossover healthcare service. A DSL and a prototype system are designed based on the model as well. Our model defines the requirements as: WHO, in what SCENARIOs, what PROCESSes need to be performed, and what RULEs need to be satisfied. We verify our model in the real case of the MEH (medical, eldercare, healthcare) crossover service. SBRM supports the service better in our cases and shows satisfactory efficiency, effectiveness, and reusability.**

***Keywords*-scenario; healthcare; crossover service; requirements modeling;**

I. Introduction

Requirements engineering has always been an important research direction. As the population ages, healthcare requirements elicitation becomes one of the core factors in the healthcare service designing for service suppliers. Researches about requirements engineering for kinds of services have been ongoing [1], [2]. Kinds of requirement oriented DSLs have been proposed in recent years [3], [4], [5]. Though these studies proposed valuable ideas and methods to describe the requirement, challenges arise when it comes to crossover healthcare service.

Crossover healthcare service is an increasingly popular pattern for healthcare services. Crossover healthcare service integrates services in medical, eldercare and healthcare fields and provides them to the elders. In crossover healthcare service, challenges mainly come from crossover data fusion and heterogeneous service integration. But before that, we need to describe the requirements well and quickly.

The purpose of this work is to propose a requirement modeling method which could help the product manager (PM) or middle-end to describe and locate the requirements more quickly and accurately.

We have cooperated with several internet healthcare companies and get real cases.

We summarized the problems exist in the requirement engineering of the modern healthcare industry and proposed a scenario-based requirement model(SBRM) for crossover healthcare services. The contributions of this paper are as follows:

- A requirement modeling method called SBRM is proposed. It formally defines terms that appear in requirements such as nouns like roles and their attributes, scenarios, attributes, rules, etc. SBRM owns the ability to express business process on two levels: the transition between scenarios and business processes in scenarios.
- A kind of DSL for the requirement design called SBRM-DSL is defined which enables the PMs and the elders to design requirements together. The form of SBRM-DSL is similar to natural language and it can be transformed into executable code in a semi-automated way, which could help shorten the development cycle as well.
- A prototype system is developed to collaborate with SBRM-DSL in a visual way. The system can automatically check the grammar mistakes and make formal inferences to identify the logical errors in the requirements, which could help to improve the development efficiency and business security.

The rest of this article is organized as follows. Section II is some related works about domain specific language and requirement engineering. Section III illustrates a motivation case of MEH(medical, eldercare, healthcare) crossover service and our research task. Detailed introduction and definition of SBRM is shown in section IV. Section V is a case study and a demonstration of our prototype system. Section VI is the discussion. Finally, the conclusions are given.

II. Related Works

In 2001, Halle et al. [6] show a way to construct an application by using conceptions and techniques of business rules through a book, which has become an authoritative guide for system designers. Ten years later, Business Vocabulary and Business Rules (SBVR) was proposed, which is an adopted standard of the Object Management Group (OMG) [7]. The goal of SBVR is to provide a standard way to describe business rules through a set of formal and detailed natural language. Based on SBVR, Bernotaityte et

al. [8] made some efforts to produce a methodology to create vocabularies and rules of SBVR from Web Ontology Language *OWL2*, which could help SBVR get a wider range of applications. In 2013, Feuto et al. [9] presented a Domain Specific Language (DSL) to express business rules in a more natural and friendly way. They also built a tool to provide auto-completion, automatic highlighting, and other useful functions. Chittimalli et al. [10] proposed a method to detect inconsistencies amongst the rules based on SBVR and First Order Logic (FOL). Brzostowski et al. [11] apply SBVR to the transportation system to help model the rules in the domain. Mohanan et al.[12] presented a way to transform natural language requirements to object-oriented models, which is rather useful.

Except for SBVR, there are kinds of DSLs as well. In 2009, Dias et al. [13] used goals to treat requirements and propose a DSL in the domain of Goal-Oriented Requirements Engineering (GORE). Within the same year, Mellegard et al. [3] presented another DSL which can help to specify and visualize requirement. Their method was evaluated via a pilot controlled experiment and showed a satisfactory result. Later in 2013, Buchmann et al. [14] illustrated the challenge of the domain in the case of ComVantage EU and proposed a modeling language to describe the requirements sources, their definition methodology and other involved components. Textual domain-specific modeling notation for requirement specification is also studied and presented [4]. Ajit et al. [15] did some work to transform requirements expressed in DSL to a formal specification language. Visic et al. [5] proposed a language draft through a meta2 layer of abstraction, and this method can be defined in a declarative manner. Pescador et al. [16] proposed a notation inspired by mind-maps called DSL-maps to alleviate the situation that the automated transition from requirements to design is largely neglected.

Requirements definition is also a research hotspot in requirements engineering. Alqrainy et al. [17] introduced 28 risk factors and proposed a set of risk management strategies through system analysis and requirements definition phase. They confirmed their assumptions and method through a web-based survey. Evaluation framework for requirements definition was proposed as well based on the relationships between the components of request [18]. Elansary et al. [19] developed a Behavioral Pattern Analysis (BPA) modeling methodology and developed an interactive software tool. Requirements definition is also used to define a new development method to speed up the software lifecycle and increase efficiency [20], [21]. Requirements definition based technologies used to guide users define their products or experience are studied and presented as well, which could be applied in many different domains.

These studies are valuable and meaningful and resolved the problems in some specific situations. However, crossover data fusion and heterogeneous service integration are not considered in these works. In the following section, we will illustrate the challenges with MEH crossover service.

III. Motivation Case & Research Task

MEH crossover service is a typical case of crossover healthcare service. MEH crossover service is a kind of crossover service which integrates medical service, eldercare service, and healthcare service. The services from those three domains are amalgamated through a service center and deployed to the community service stations (see Figure 1). Service integration will bring many conveniences to healthcare services. For instance, the eldercare service would not provide sugary food to an elderly person who has been diagnosed with diabetes in medical service.

When MEH crossover service was just starting to build, there were some problems appeared especially in the process of requirements analysis. A same thing may be called differently in different domains. Since MEH crossover service is an integration of online and offline binding services from different domains, it is hard to match each online service with offline scenarios. Moreover, the requirements now are more like a stack of business rules, which makes it hard to see the business processes clearly. We summarized some features in MEH crossover service requirement management and analysis.

Feature 1: The requirements from different domains are amalgamated. The requirements of MEH crossover service are taken from the domains of medical, eldercare, and healthcare. It could be difficult to unify the expressions. All the nouns or verbs appeared in the requirements need to be predefined.

Feature 2: The requirements need to cover both online and offline scenarios. In MEH crossover service, online business and offline scenarios are strongly bound.

Feature 3: The requirements are iterated continuously. The iteration of requirements may occur in every domain. How to keep requirements readable and reusable has become an important issue.

These features are increasingly prominent in healthcare companies nowadays, especially when involved with crossover services such as MEH crossover service.

On the basis of those features, we position the problems SBRM needs to solve below.

Problem 1: The concepts in requirements of crossover healthcare service are usually ambiguous and inconsistent, which should be managed formally. A model needs to be proposed to describe the attributes and properties of these concepts appeared in the requirements to ensure the concepts' consistencies.

Problem 2: The iterations become frequent, while the requirements are hard to reuse and extend. Data, process and business logic are coupled together in traditional requirement description, and minor modifications may affect all requirement documents.

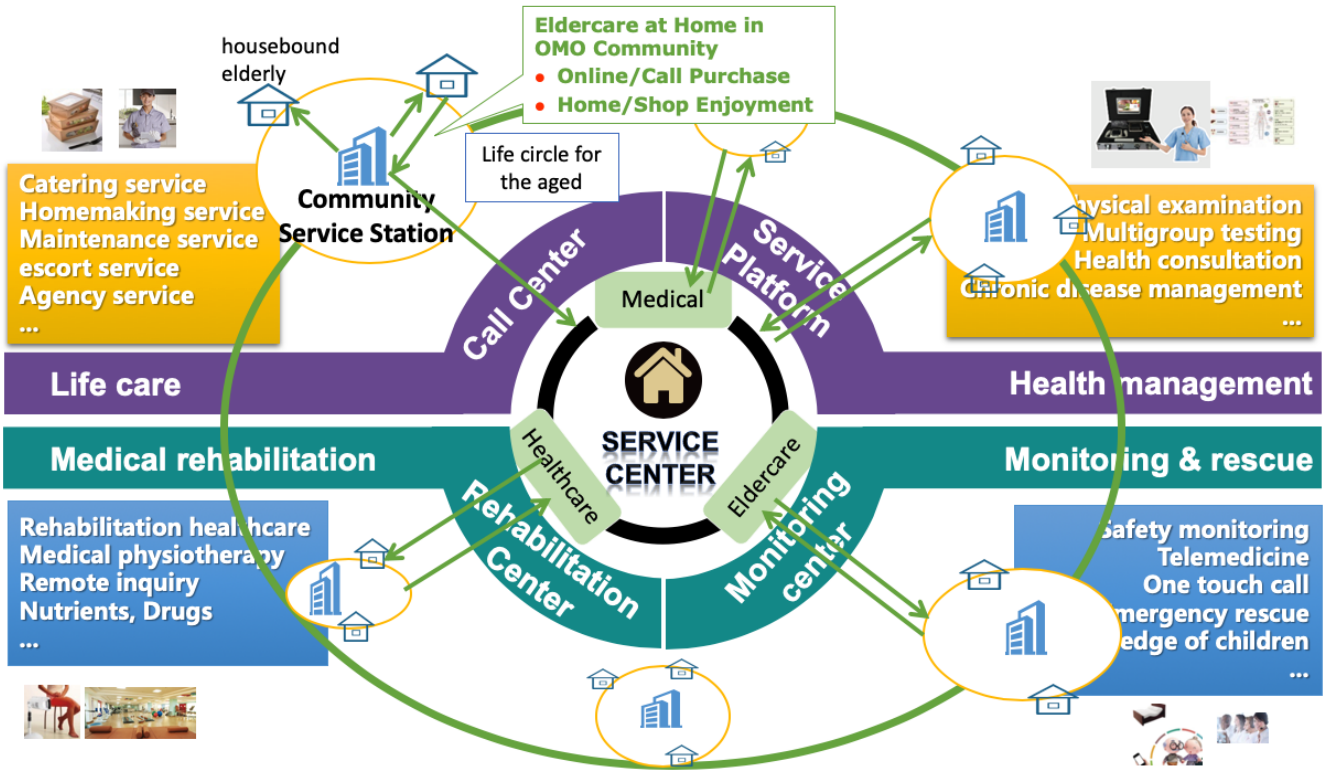


Figure 1: Architecture of MEH crossover service.

Problem 3: The integration of crossover services makes logical errors in requirements occur frequently. So in addition to describing requirements in a formal way, we also need to validate these requirements through logical inferences.

To solve the problems above, SBRM is proposed. A requirement organized by SBRM is mainly composed of 4 parts: concept, scenario, process, and rule. The concept is an abstraction of a role or action. The scenario is a kind of service operation block which bounds to offline scenes. The process is an operation sequence in a scenario, and rules are constraints for a process in a scenario. Meanwhile, a kind of DSL is proposed to embody SBRM. The detailed definitions are given in section 4.

IV. SBRM

This section formulates the key syntaxes and basic notations of SBRM. As mentioned above, there are four basic elements in SBRM: concept, scenario, process, and rule. The concept is constructed and maintained throughout a project or within an alternative domain like a company. And for a piece of requirement, there is a scenario, a process, and several rules. The relation of the elements and basic notations is presented in Figure 2 by the Unified Modeling Language (UML) diagram.

For convenience of later interpretation, we expect the existence of the following pairwise disjoint countable infinite sets: $\mathcal{T}_p$ of primitive types, $\mathcal{T}_{ac}$ of action type, $\mathcal{T}_{ru}$ of rule type, $\mathcal{C}$ of classes (names), $\mathcal{AT}$ of attributes, $\mathcal{AC}$ of actions (names), $\mathbf{ID}_C$ of identifiers for each class $C \in \mathcal{C}$. A type is an element in the union $\mathcal{T} = \mathcal{T}_p \bigcup \mathcal{T}_{ac} \bigcup \mathcal{T}_{ru} \bigcup \mathcal{C}$.

The domain of each type τ in $\mathcal{T}$, denoted as $\mathbf{DOM}(\tau)$, is defined as follows:

1) if $\tau \in \mathcal{T}_p$ is a primitive type, the domain $\mathbf{DOM}(\tau)$ is some known set of values (integers, strings, etc).
2) if $\tau \in \mathcal{T}_{ac}$ is an action type, $\mathbf{DOM}(\tau)$ = positive/passive.
3) if $\tau \in \mathcal{T}_{ru}$ is a rule type, $\mathbf{DOM}(\tau)$ = TriggerRule/ AttributeRule/ OperationRule.
4) if $\tau \in \mathcal{C}$ is a class (name), $\mathbf{DOM}(\tau) = \mathbf{ID}_t$.

In order to have a specific understanding of the entire requirement in SBRM, we give the definition of requirement first.

Defintion 1. *A requirement is a five-tuple (C, Γ, S, P, R), C is the identifier of the requirement, Γ is a schema of concepts, S is a set of scenarios with respect to Γ, P is a process with respect to Γ and S, R is a set of business rules with respect to S.*

In this article, we give our definitions of the rest notations in SBRM through a top-down approach. So you will see definitions of each part first, and following is their components.

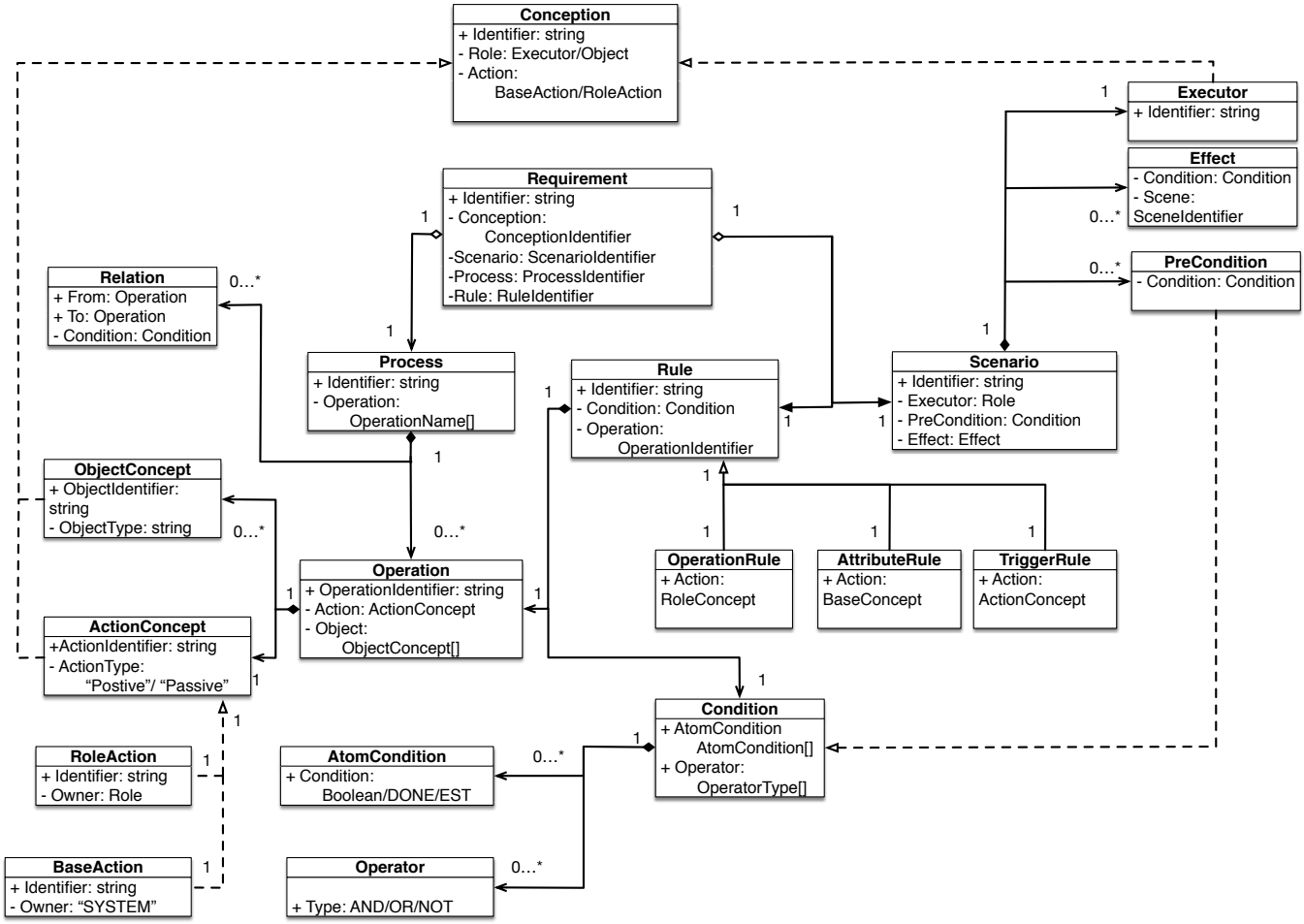


Figure 2: The UML diagram of basic notations in SBRM.

A. Concept

Concept here is a data structure to express the nouns and verbs appeared in the requirements. In a concept, the nouns are called roles and the verbs are called actions. Each action is associated with a role, which means an action can only be performed by a specific role.

Defintion 2. *A concept is a four-tuple (C_{co}, Ro, A, μ_{ra}), where C_{co} is the identifier with respect to Γ, Ro and A are the set of roles and actions in this concept respectively, μ_{ra} is the partial mapping that assigns each action to role.*

In SBRM, a role could be a participant of the business like a customer or it could be just an object involved in the business like a piece of goods or an order. The role does not correspond to data directly but attributes instead. A role could be the executor of a scenario. For instance, for the scenario *submit order*, the corresponding executor could be *customer*. There is a reserved role called *SYSTEM* in our model, the action associated with it is called a *base action*. Otherwise, the action is a *role action*.

Defintion 3. *A role is a four-tuple (C_{ro}, **AT**, τ, μ_{at}), where C_{ro} is the identifier, **AT** is the attributes of this role, τ is types of the attributes, μ_{at} is the partial mapping that assigns each **AT** in **DOM**(τ(**AT**)).*

Defintion 4. *An action is a binary (C_{ac}, τ, **D**), where C_{ac} is the identifier, τ is the action type, **D** is the description of the action.*

The relationships of each part in the concept are shown in figure 3.

B. Scenario

A scenario could be an abstract situation of the business. It may appear as the form of a web page or just a popup module. A scenario is bound with an executor who would use and handle all the functions in the scenario. Meanwhile, the precondition is also provided because there are usually conditions you need to satisfy first to perform your operation in the scenario. An effect of a scenario indicates the subsequent operations after the execution of the main process of

Table I: Grammar of the SBRM-DSL

Type	Statement Specifications in EBNF	Meaning
General	note_statement = '%', sentence;	The way to write annotations.
Condition	atom_statement = [NOT], < 'TURE' | 'FALSE' | 'DONE(' operation_statement ')' | 'EST(' identifier ',' identifier ',' action ')' >; condition_statement = 'atom_statement', { < 'AND' | 'OR' >, condition_statement };	The way to describe the condition ϵ mentioned in Definition 11.

Requirement	concept_statement = 'import', concept_file_name;
	concept_file_name = identifier, '.cpt';
	requirement_statement = 'requirement', requirement_name, 'is in scenario', scenario_name, 'with process', process_name, { 'and rule', rule_list }, '.';
	rule_list = rule_name, { ',', rule_name };
	requirement_name = identifier;
		The definition of requirement, which first introduces concepts' file and then specifies an actual requirement by combining a scenario, a process, and a series of business rules.

Role	role_statement = 'role', role_name, 'has', attribute_list, 'with action', action_list, '.';
	attribute_list = attribute_statement, { ',', attribute_statement };
	attribute_statement = identifier, ':', attribute_type;
	attribute_type = 'integer' | 'string' | 'boolean' | 'object';
	action_list = identifier, { ',', identifier };
	role_name = identifier;
		The definition of the role. Note that a role usually contains a list of attributes for a role and a series of actions.

Action	action_statement = 'action', action_name, 'is', action_type, 'which', sentence, '.'; action_type = 'positive' | 'passive'; action_name = identifier; identifier = 'a..z,$,_', { 'a..z,$,_,0..9' }; sentence = { character }; character = 'based on the unicode character set';	The definition of the action. An action is composed of its type and description sentence.

Scenario	scenario_statement = 'scenario', scenario_name, 'has executor', executor_list, ['with precondition', condition], 'and link to', effect_list, '.';
	executor_list = role_name, { ',' , role_name };
	effect_list = effect_statement, { ',' , effect_statement };
	effect_statement = scenario_name, 'if', condition;
	scenario_name = identifier;
		The definitions of the scenario and effect. An executor will enter into a scenario when it satisfies the scenario's precondition and jumps to the next scenario according to the conditions of the effects in the effect list.

Process	process_statement = 'process', process_name, 'is', operation_list, '.';
	operation_list = operation_statement, { ',' , operation_statement };
	operation_statement = action_name, ':', [object_name];
	process_name = identifier;
	object_name = identifier;
		The definitions of the process and operation. An operation will include an action, and an object when the action's type is *positive*.

Rule	operation_rule_statement = 'rule', rule_name, 'means', operation_statement, 'can be performed if', condition, '.'; trigger_rule_statement = 'rule', rule_name, 'means', operation_statement, 'will be performed if', condition, '.'; attribute_rule_statement = 'rule', rule_name, 'means', condition, 'should be true.'; rule_name = identifier;	The definitions of the three types of rules. Note that different types of rules provide different kinds of constraints according to their condition.

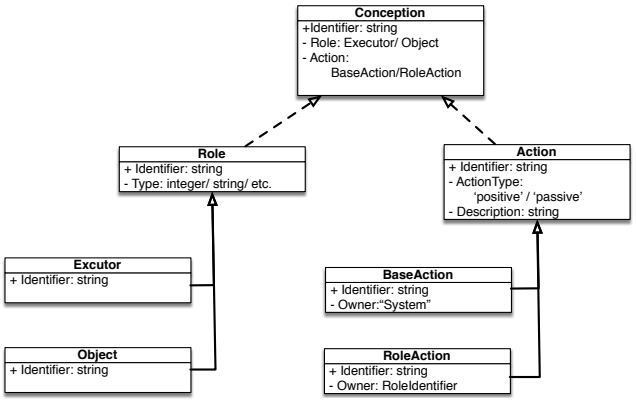


Figure 3: The UML diagram of concept.

the scenario. Here, the effect is mainly used to indicate the logical method to find the next scenario.

Defintion 5. *A scenario is a four-tuple (C_s, e, P, E), where C_s is the identifier, e is the executor of the scenario with respect to role in Γ, P is the precondition, and E is the effect.*

Defintion 6. *An effect is a triple (ε, ρ, μ_{ep}), where ε is a set of conditions, ρ is a set of scenarios with respect to C_s, μ_{ep} is a onto mapping that assigns each condition to scenario.*

C. Process

In SBRM, a process always corresponds to a scenario. And it is, in fact, a sequence of operations. The subject of the operations is the executor of the relative scenario by default. A process indicates what the user (executor) should do under a scenario orderly. We use relations to keep the order, and conditions bound on them can ensure the operations are taken properly.

As we have mentioned at the beginning of this section, there are two types of actions: positive and passive. When the type of the action is passive, the object of the operation is *NULL* by default.

Defintion 7. *A process is a triple (C_p, O, R), where C_p is*

the identifier, O is the set of the operations, and R is the relations among the operations.

Defintion 8. *An operation is a binary (C_{op}, A, σ), where C_{op} is the identifier, A is the aciton of the role Ro with respect to scenario C_s, σ is the objects with respect to Γ.*

Defintion 9. *A relation is a triple (C_{re}, from, to, ϵ), where C_{re} is the identifier, from and to are names of the operations and ϵ is the condition.*

D. Rule

Rule is used to define business rules. There are three types of rules in SBRM:

- *OperationRule :* Operation rule is used to express an operation can be performed if the corresponding condition is true,
- *AttributeRule :* Attribute rule is used to express that attributes with respect to Γ must meet the constraints if the corresponding condition is true,
- *TriggerRule :* Trigger rule is used to indicate that operations would be performed automatically as long as the corresponding condition is true.

Defintion 10. *A rule is a four-tuple (C_r, ϵ, o, τ), where C_r is the identifier, ϵ is the condition of the rule, o is the corresponding operation, and τ is the type of the rule.*

Defintion 11. *A condition ϵ could be one of the following:*

1) *Boolean,*
2) DONE*(A, O), where A is an action of C_{ro} and O is the target object of the action,*
3) EST*(a_1, a_2, comp), where a_1, a_2 are two attributes of C_{ro} and comp is comparison operation,*
4) *ϵ_1 AND ϵ_2, where AND is a reserved word which means that the condition is true iff so is both ϵ_1 and ϵ_2,*
5) *ϵ_1 OR ϵ_2, where OR is a reserved word which means that the condition is false iff so is both ϵ_1 and ϵ_2,*
6) *NOT ϵ, where NOT is a reserved word which means the condition is opposite to ϵ.*

In definition 11, DONE(A, O) is true when executor of the scenario has taken action A to object O. If the type of A is passive, then O will be *NULL* by default. And EST(a_1, a_2, comp) is true when a_1, a_2 satisfy the comparison operation comp.

E. SBRM-DSL

We design a kind of DSL called SBRM-DSL with the syntaxes and notations in SBRM to express requirements in the field of e-commerce. We follow the EBNF standard to implement our language here. Table I introduces our language in the similar order as above, it describes the metasyntax notations of the requirement, concept, scenario, process, rule and their auxiliary elements one by one. Then PMs can describe their requirements by our language very clearly and efficiently as long as they follow the grammar. In the next section, we illustrate the effectiveness of our DSL by a practical scenario in Alibaba.

V. Case Study

To illustrate SBRM, we apply it to an example of constructing a requirement of MEH crossover service. In this case, we will construct a requirement of the service *visiting healthcare examination* through SBRM. And this procedure should be performed by PM or middle-end staff.

Build the concepts involved in the requirement. This step determines all participants in the service as well as all their attributes and possible operations. All the concepts of the services in one system would be merged and generate an overall concept library. A concept example is shown in Example 1. The reserved words in SBRM-DSL are shown in bold.

Example 1. role doctor **has** name:string, age:int, major:string, community:string **with action** draw_blood, take_pulse, check_blood_pressure, check_eye, check_tooth, call_relative.

Formalize the scenario through SBRM. As we have built the concept used in this requirement, the scenario needs to be defined next. In this step, the name, executor, precondition and effect of the scenario should be confirmed so that subsequent steps could be taken (see Example 2).

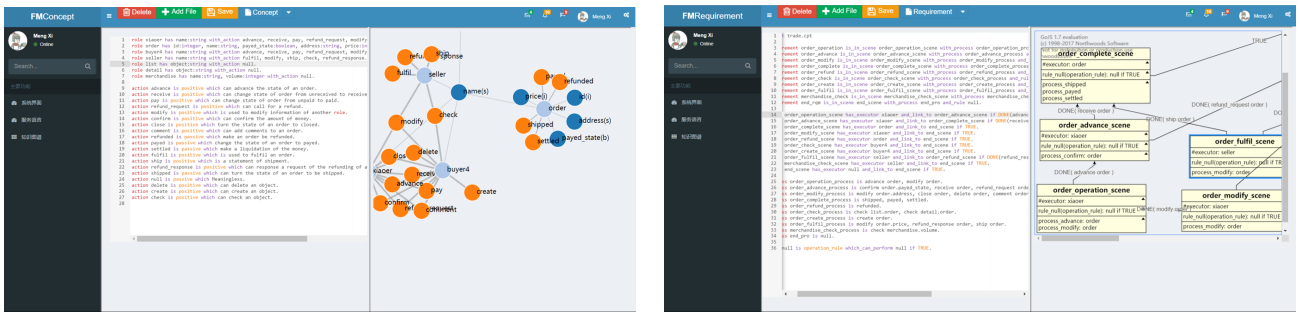
Example 2. scenario visiting_healthcare_examination_scenario **has executor** doctor **with precondition** EST(patient, home, at) **and link to** DONE(draw_blood):blood_test.

Construct the business process of the scenario. After the formalization of the scenario, the business process needs to be constructed. In healthcare services, the business process logics are always simple. Therefore in SBRM, a process consists of a sequence of operations (see Example 3).

Example 3. process healthcare_examination_process **is** draw_blood:patient, take_pulse:patient, check_blood_pressure:patient, check_eye:patient, check_tooth:patient.

Configure the corresponding business rules. With the concept built, scenario defined and process constructed, rules need to be configured now. As defined in section IV, there are three types of rules in SBRM: OperationRule, AttributeRule, and TriggerRule. Each of these three types can express one type of business constraint. Example 4 is a set of rules in this case.

Example 4. rule at_home_rule **means** EST(patient, home, at) **should be true**.
rule call_relative_rule **means** call_relative **can be performed if** EST(patient, draw_blood, refuse).



(a) Definitions of roles and actions.

(b) Definitions of requirements, scenarios and processes.

Figure 4: A demo of MEH crossover service with SBRM-DSL.

Generate the requirement of the service. A requirement is a combination of concepts, scenarios, processes, and rules. This makes the four notations decoupled and could be reused easily (see Example 5).

Example 5. import visiting_healthcare_examination_concepts.cpt
requirement visiting_healthcare_examination **is in scenario** visiting_healthcare_examination_scenario **with process** healthcare_examination_process **and rule** at_home_rule, call_relative_rule.

To illustrate our SBRM-DSL, we construct a tool with the GUI to show the relations between the roles as well as their attributes and actions, as shown in Fig. 4a. Also, our tool can display the relations among the 4 parts in SBRM in the form of UML, as shown in Fig. 4b. Actually, Fig. 4 demonstrates a closed loop requirement flow of the MEH crossover service. To meet these requirements, we design 7 roles with 19 actions to execute a total of 9 processes in 9 scenarios. We can see that with our customized language, the division of tasks becomes more specific and roles' ambiguities are eliminated. And because of the conciseness of our language, the PMs can easily write similar statements to describe their actual requirements. Also, the statements can be semi-automatically converted into executable code, which speeds up the business process.

Until here, we have got the concept including roles and their actions, scenario with process and rules in it. And you will find that a piece of requirement has been fully defined by SBRM. But we still do not know whether SBRM is able to solve the problems proposed in section three, so it is discussed in the next section.

VI. Discussion

In this section, we will discuss whether SBRM solve our research problems proposed in section three.

All the concepts are managed formally by our model. By using concept, all the nouns appeared in the requirements, like executor and objects are abstracted as roles in concept. And the verbs are abstracted as actions in concept. Classified according to whether the relative role of the action is *SYSTEM* or a role, there are *base actions* and *role actions*. Classified according to the relation between action and relative, there are *positive actions* and *passive actions*. All the nouns and verbs are managed formally and well. Also, our model ensures these concepts' consistencies between different requirements.

Requirements can be reused by later PMs and extensible by the scenario part in SBRM. By dividing a whole business requirement into several scenarios, processes and rules in requirements are divided as well. On one hand, the model becomes more flexible and owns strong scalability, because a new requirement could be a new scenario and so is the change. On the other hand, this could be a division of development task for different departments at the same time, which could help save some time from requirement analyzing.

We can validate the requirements through logical inference in our visual tools. Like other IDEs, our visual tools can also check grammar mistakes before the statements are compiled by the compiler. When the action or data abuse occurs, or undefined concept appears, our tools can detect these errors automatically. This increases the rigor of the requirements. Meanwhile, as we can see that our tools can convert the statements into relation graphs and UML diagrams so that if there are some logical errors in the statements, the conversion process will be terminated.

VII. Conclusion

In this work, we construct SBRM to deal with the problems of requirements engineering in crossover healthcare service. We give the definitions and examples in the study case. We locate research tasks and discuss how they are met. Though SBRM may not be perfect now, we believe that we have achieved some relevant results.

For future works, as illustrated in the article, SBRM defined requirements should be compiled to the formal language, through which the requirements could be converted into applications or services automatically.

ACKNOWLEDGMENT

This work is supported by the national key research and development program of China under grant No.2017YFB1401202 and the key research, development program of Zhejiang Province under grant No.2017C01013 and Model Information Service Industry Program of Guangdong Province(GDEID2010IS049).

REFERENCES

[1] D. Georgakopoulos and M. Papazoglou, "Requirements engineering techniques for e-services," pp. 331 – 352, 2017.

[2] L. Kolos-Mazuryk, G. J. Poulisse, and E. V. Pascal, "Requirements engineering for pervasive services," *Acm Sigops Operating Systems Review*, 2017.

[3] N. MellegÃěrd and M. Staron, "A domain specific modelling language for specifying and visualizing requirements," vol. 457, 2009.

[4] O. Olajubu, "A textual domain specific language for requirement modelling," in *Joint Meeting on Foundations of Software Engineering*, 2015, pp. 1060–1062.

[5] N. Visic, H. G. Fill, R. A. Buchmann, and D. Karagiannis, "A domain-specific language for modeling method definition: From requirements to grammar," in *IEEE International Conference on Research Challenges in Information Science*, 2015, pp. 286–297.

[6] B. V. Halle and R. G. Ross, "Business rules applied: Building better systems using the business rules approach," *John Wiley & Sons Inc*, 2001.

[7] C. C. Eglantine and business, *Semantics of Business Vocabulary and Business Rules*. TypPRESS, 2011.

[8] G. Bernotaityte, L. Nemuraite, R. Butkiene, and B. Paradauskas, *Developing SBVR Vocabularies and Business Rules from OWL2 Ontologies*. Springer Berlin Heidelberg, 2013.

[9] P. B. Feuto, S. Cardey, P. Greenfield, and W. E. Abed, "Domain specific language based on the sbvr standard for expressing business rules," in *Enterprise Distributed Object Computing Conference Workshops*, 2013, pp. 31–38.

[10] P. K. Chittimalli and K. Anand, "Domain-independent method of detecting inconsistencies in sbvr-based business rules," in *International Workshop on Formal Methods for Analysis of Business Systems*, 2016, pp. 9–16.

[11] K. Brzostowski, "Modeling business rules for transportation systems," 2016.

[12] M. Mohanan and P. Samuel, "Software requirement elicitation using natural language processing," 2016.

[13] A. Dias, V. Amaral, and J. Araujo, "Towards a domain specific language for a goal-oriented approach based on kaos," in *International Conference on Research Challenges in Information Science*, 2009, pp. 409–420.

[14] R. A. Buchmann, D. Karagiannis, and N. Visic, "Requirements definition for domain-specific modelling languages: The comvantage case," in *International Conference on Business Informatics Research*, 2013, pp. 19–33.

[15] S. Ajit, O. Olajubu, S. Thomson, and M. Edwards, "Model transformation of high-level requirements in a domain specific language into a formal specification language," 2015.

[16] A. Pescador and J. D. Lara, "Dsl-maps: From requirements to design of domain-specific languages," in *Ieee/acm International Conference on Automated Software Engineering*, 2016, pp. 438–443.

[17] S. Alqrainy and H. Hijazi, "Managing risks in the system analysis and requirements definition phase," *International Journal of Computer Applications*, vol. 99, no. 3, pp. 23–29, 2014.

[18] M. Alnabhan, A. Haboush, A. Albadareen, and M. Alnawayseh, "An evaluation framework for requirements definition of software development," 2014.

[19] A. Elansary, "Intelligent agent travel reservation system requirements definitions using the behavioral patterns analysis (bpa) approach," 2015.

[20] T. R. Silva, "Definition of a behavior-driven model for requirements specification and testing of interactive systems," in *Requirements Engineering Conference*, 2016, pp. 444–449.

[21] L. Hannola, K. Elfvengren, and M. Tuominen, "A group support system process for the definition of software requirements," *International Journal of Innovation & Learning*, vol. 7, no. 2, pp. 171–186(16), 2017.